

An Infrastructure for Adaptive Control of Multi-Agent Systems

Karl Kleinmann, Richard Lazarus, Ray Tomlinson
BBN Technologies
10 Moulton St
Cambridge, MA 02138
{kkleinmann, rlazarus, rtomlinson}@bbn.com

Abstract— *In this paper, we present the control infrastructure of the Cougar distributed agent system that was developed under the DARPA ALP and UltraLog programs. We motivate its design from a control theory perspective and discuss the characteristics of an agent system as the controlled process. These characteristics are arguably the reason why formal methods of control theory are rarely applied in software engineering for agent systems.*

1. INTRODUCTION

Large, distributed, multi-agent systems (DMAS) have a huge number of internal states and many degrees of freedom. While these characteristics provide great benefits, like flexibility for system configuration, they also impose a complex multivariable control problem. The control goal is usually not only the optimization of the actual application but also the containment of hardware or software-related failures, which can be viewed as stochastic disturbances. In the area of survivable systems that are designed to operate under warlike conditions, robustness (which addresses intentional network and platform disruptions) and security (which addresses intentional software intrusions) become control goals of equal importance to the primary application (e.g., a planning system).

In this paper, we present the infrastructure elements of the control architecture of Cougar [1] that address the challenges above. Cougar is an agent architecture for large-scale DMAS that has been sponsored by DARPA through the former ALP program (1996-2001) and the current UltraLog program (2001-2004). In addition, Cougar is open source and enjoys a worldwide user community. Under the UltraLog program [2], the Cougar software is extended to inherently ensure survivability under extremely chaotic and high-stress environments, with particular focus on robustness, security, and scalability.

Survivability is predicated on maintaining the highest quality of service across many dimensions based on mission or application objectives. Hence, it is essential that the agents

are aware not only of their own performance, but also of the externally available resources. In addition, adaptive control must encompass optimizing cost functions describing varying emphasis across the multiple quality of service dimensions at various points in the control hierarchy.

As a primary application, UltraLog showcases a military logistics planning and plan-execution system that implements over 500 distinct agents running on more than 100 machines.

Section 2 of this paper discusses DMAS from a control theory perspective, motivated in part by [5]-[8], and suggests why their characteristics constitute a hard and unusual control problem, and what makes DMAS unique as a controlled process. We also point out analogies where control theory and software engineering use different terminologies for the same abstractions.

Section 3 outlines the control objectives of DMAS, using the various levels and types of requirements of the UltraLog application as an example.

Section 4 briefly introduces the Cougar agent architecture and describes in detail its control infrastructure elements and their interactions.

Section 5 gives some examples for implemented control strategies, one of which comes with the Cougar open source distribution, and an interpretation of the approach, comparing it with other adaptive control designs.

Section 6 summarizes the current design status and discusses what we hope to accomplish with future research.

2. SOFTWARE AGENT SYSTEMS AS CONTROLLED PROCESSES

Control theory captures the fundamentals for three activities:

- *Design of the control system.* This activity characterizes the system boundaries of the process to be controlled (also called controlled system or plant), the control inputs by which the behavior of the process can be changed, the structure of the controller that generates these inputs, and the sensor data as input for the controller reflecting current and desired performance.

- *Initialization of control parameters.* For the controller, they are either derived from an analytic or data-driven model of the controlled process, or based on experiments and heuristics.
- *Tuning of control parameters during operation.* In an adaptive control system, the controller can be continually adjusted to cope with changes of the inherent process behavior over time.

Whereas parts of almost every control loop are implemented in software, in cases where the controlled process is the software system itself, the means of control theory are rarely applied. As [7] points out, “the basic paradigm of control has not found its place as a first-class concept in software engineering.”

There are certain characteristics in DMAS that distinguish them from the examples of controlled processes commonly considered in control theory:

- *Dynamic system boundaries.* DMAS have typically many degrees of freedom, e.g., mobile agents can reside on various nodes over time, new communication channels are added and removed again, agents can be rehydrated elsewhere, the distribution of the application over the agents can change. This desired flexibility leads to a constantly changing topology and does not allow to assume constant system boundaries and to partition the system statically.
- *System size.* DMAS can consist of hundreds of agents containing thousands of components to be controlled. Thus, a huge number of internal states need to be measured or monitored by separate instrumentation code. Additional sensor data are generated while monitoring the communication between peers in the network and storing these matrices including their history.
- *Type of cost functions and performance criteria.* As opposed to processes in the physical world, where every change of a control input variable adds cost, many control actions in the logical world have a strongly non-linear impact on the cost function (e.g., up to a certain point and under certain conditions, using more CPU can be done for free). Furthermore, control goals are initially described in a symbolic way, and there is often no analytic way to transform these into numeric values and map them into set points for internal states. Therefore, the set points are mostly step functions, not trajectories.

These characteristics, originating both in the nature of software and the particular design of agent-based systems, make an approach to the control of DMAS based on control theory very complex. Because of the dynamic system boundaries and the system size, it is hard to build a model of the process that is smaller than the DMAS itself. There are no good abstractions that can be analytically derived from

internal states and capture the desired behavior. In addition, the number of internal states and their couplings impose a complex multivariable control problem. Since control inputs often lead to structural changes in the system, the system dynamics become nonlinear.

On the other hand, the software system can be used as a perfect model for itself and, given an automated testing environment, control approaches and control parameter variations can be simulated at almost no additional cost. The use of feed-forward controllers often avoids stability problems, with experimentally determined control parameters.

This heuristic approach is often taken in software engineering. In Cougar, we have created a set of terminology for describing control components and measures that possess analogs in traditional control theory. These abstractions include: control input vs. actions or operating modes; sensor input vs. sensor conditions; disturbance vs. stress; controller vs. engine; and control algorithm vs. rules or plays.

3. CONTROL OBJECTIVES IN DMAS

Besides the primary system function, a DMAS has to accommodate various requirements in parallel (e.g., usability, reliability, fidelity, and stability) that become additional control goals in the multidimensional control and optimization problem. Because of the distributed nature of DMAS, these generic requirements become a special meaning since communication over wide area networks, memory, CPU resources, and participating platforms are not only variable and unpredictable, but also vulnerable against kinetic or information attacks.

The primary system function of the UltraLog system is the planning and plan-execution of military deployment operations, the control goal is to build a timely plan in the face of varying workloads and system conditions. Besides this logistics application, the system has to accommodate extreme hardware- and software-related failures, motivated by the operational scenario of operating under warlike conditions. These requirements are captured under the functions of robustness and security. The control goal of robustness is to maintain a processing infrastructure despite the loss of processing resources (caused by intentional network or hardware platform disruptions). The control goal of security is to maintain system integrity despite information attacks (intentional software intrusions).

In the control hierarchy, there are two levels specifically designed to achieve these control goals:

- *Application level control.* The control inputs on this level are typically complex actions or sequences of actions composed of control primitives and designed as specific defenses against certain stresses. Examples are variable fidelity processing that requires less computing resources;

load balancing by moving agents to different hosts; or reconstituting agents that were residing on destroyed hosts. These control actions are mostly initiated and implemented by manager agents within a local scope, but often have global impact. Since some complex actions can have conflicting impacts, an additional control layer for deconfliction is required that arbitrates the action selection [4]).

- *Agent infrastructure level control.* The control inputs on this level are the parameters of the components within an agent, providing the agent with the autonomy to make local decisions. Examples are lowering the rate of status reports or turning on message compression when the local network load is high. In the following section, the agent-level control mechanisms of the Cougaar agent infrastructure are described in detail.

4. THE COUGAAR CONTROL INFRASTRUCTURE

The adaptive control mechanisms described here are part of Cougaar, a 100% Java agent architecture for building large distributed multi-agent systems, comprising around 500,000 lines of code. The prototype application uses over 500 distinct agents distributed over a 5-LAN network of over 100 machines. Cougaar was designed to support data intensive, inherently distributed applications, where application scalability is paramount. Intra-agent communication is accomplished via publish and subscribe to a local blackboard to reduce latency for tightly coupled component interaction. Inter-agent communication transfers locally published objects to targeted recipients to allow wide distribution of loosely coupled interactions. Communities of agents form to manage resources and provide scalable services.

Cougaar has several subsystems for collecting and measuring overlapping sets of performance data, each with different usage requirements and quality of service characteristics. These include the Metrics Service and various domain-specific sensor groups. The instrumentation is dynamic (sensor values are only measured when needed) and built into the architecture [3].

One innovation of Cougaar is its hierarchical component model, based on the JavaBeans API. This model provides unique security and composition properties for Cougaar agents. All internal system functions and application functions are added at configuration or run time into a Cougaar agent as components, where one or more binders wrap each component to mediate and secure component access to system functions.

Each Cougaar node (agent container, one per JVM) contains a Tomcat web server that provides access to the agents blackboard to external clients such as user interfaces and status monitors. Data access is provided by servlets, dynamically loadable plugins provided by the client. These servlets have full access to agent-state and load/execute only

when invoked. For example, the CSMART UI for Cougaar configuration uses servlets both for system control, and runtime monitoring.

Under the UltraLog project, access control systems have been added as Cougaar components (binders) to limit component access to agent-internal data (restricted subscriptions), e. g., to restrict servlet access to Blackboard data and to restrict messaging between agents. The security subsystem can both provide performance metrics and use such metrics at runtime to tune access control policies. This security system, thereby, adaptively controls access to system data using the Cougaar agent-level control infrastructure based on feedback from performance measurements.

Adaptive control in Cougaar can be implemented using the inherent Adaptivity Engine (AE) mechanisms and associated components. Cougaar services are expected to have Operating Modes-modes of operation that provide increased Quality of Service (QoS) with increased resource consumption or with particular dependencies on other QoS providers. The AE provides the mechanisms by which control actions (Plays) can specify QoS in multiple dimensions (Operating Modes) based on measured operating state (Conditions). Figure 1 illustrates these components and their associated data flow.

The following discusses the key components, services, and objects, as well as their interactions, of the Cougaar agent-level control infrastructure. These include:

- *Operating Modes.* An operating mode is created and published by a component representing one control input dimension (out of many) of the component. An Operating Mode is a data structure with a list of ranges of values that it is allowed to have, as well as a current value. They are the control inputs (“knobs”) by which the component can be controlled (“tuned”).
- *Conditions.* Conditions are the generalized form of any (sensor) input information used by the controllers. Sensors can publish conditions that reflect their run-time performance measurements, and other components can aggregate measurements and state information to provide QoS values.
- *Plays, Playbook, Playbook Manager.* Plays represent the control laws; they specify restrictions or constraints on one or more Operating Modes and the Conditions under which those constraints are to be applied. A Playbook has a list of Plays that are tested in succession for applicability to the current conditions. The Playbook manager is a component that maintains the Playbook and provides the services needed to manipulate and use the Playbook.
- *TechSpecs.* TechSpecs are published by components as a high-level model (description) of their behavior. They allow the controller (AE) to reason and predict the

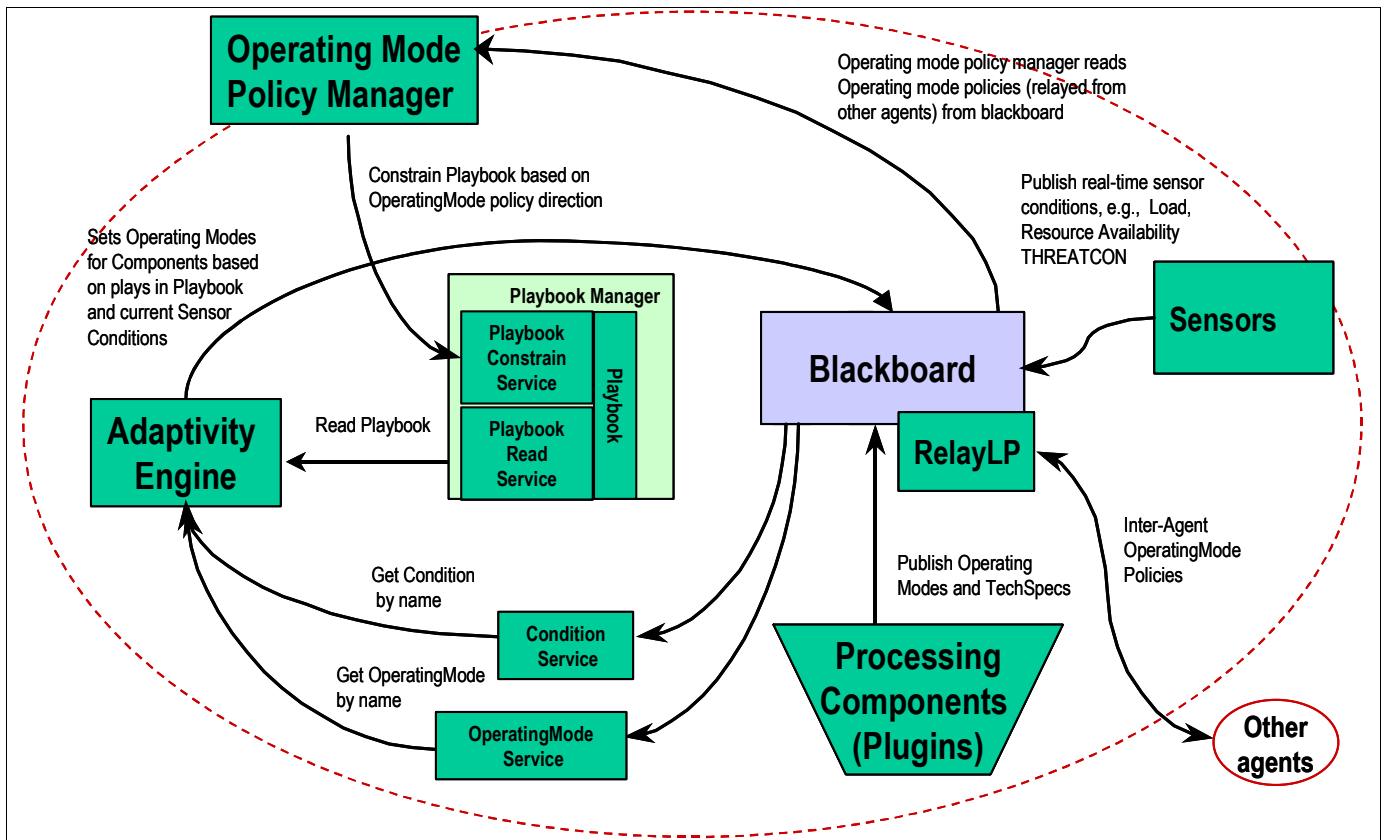


Figure 1 – Cougaar Agent-level Control Infrastructure

consequences of alternate Operating Mode settings of the Component.

- *Adaptivity Engine.* Each agent contains a component named Adaptivity Engine that acts as the controller for that agent and certain other external components. The Adaptivity Engine responds to changes in Conditions and modifications of the Playbook, evaluates the Plays, and sets new Operating Modes accordingly. By observing the system behavior and using the TechSpecs as a system model, one can use the Adaptivity Engine and associated components to implement an adaptive control approach that optimizes system behavior in accordance with specified objectives (Plays in the Playbook).
- *Operating Mode Policies and Operating Mode Policy Manager.* Higher-level system policies (set dynamically by other agents, or by a human operator) may restrict valid plays that the agent may enforce. In this way, Cougaar agents use sensors to adapt to changes in the environment and to optimize across application goals. Policies are communicated between agent blackboards via the Cougaar relay mechanism. As Operating Mode Policies can be disseminated across agents, it is the mechanism by which one can implement hierarchical control within a Cougaar agent society.

In addition to these conceptual aspects, there are various technical implementation aspects considered that are less

relevant for the control issues discussed in this paper. Examples are, access to plays, conditions, and operating modes via service providers; processing order of plays; or detection of missing or overly constrained operating modes.

5. EXAMPLES AND INTERPRETATION OF THE CONTROL APPROACH

This section provides some examples that demonstrate the effective control of a DMAS using these mechanisms. More specifically, we have included two examples. Example 1 is a “toy” control problem that was developed as a “plumbing” test and demonstration of the Cougaar adaptivity engine and its associated components (and is included in the open source Cougaar distribution). Example 2 is one application of controlled system adaptivity that has been implemented for our military logistics application.

Example 1: Demonstration of the Cougaar Adaptivity Engine

Example 1 is a two-agent system consisting of a task generator agent and a provider agent. Using a single play in its playbook, the adaptivity engine of the provider agent can modify the way tasks are processed within the provider agent. The sensor conditions are the available CPU resources (inverse to the CPU load) of the provider agent’s host and the rate by which new tasks are arriving from the generator agent at the provider. The Operating Mode used in

this playbook tunes the algorithm by which incoming tasks are allocated by the provider agent’s allocator plugin. The quality of the allocations done by this plugin depend on how many iterations the allocation algorithm can afford. The play connects the two conditions by dividing the task rate by the CPU value, and maps this input to the Operating Mode determining the number of iteration cycles in the allocator plugin. The play represents the heuristic that if the number of incoming tasks is low and enough CPU resources are available, the task allocation can be done more precisely using many iterations of the allocation algorithm. On the other hand, if the number of incoming tasks is high or limited CPU resources are available, the allocation should be done fast using less computing resources. Figure 2 shows this nonlinear control algorithm implemented as a play in the playbook.

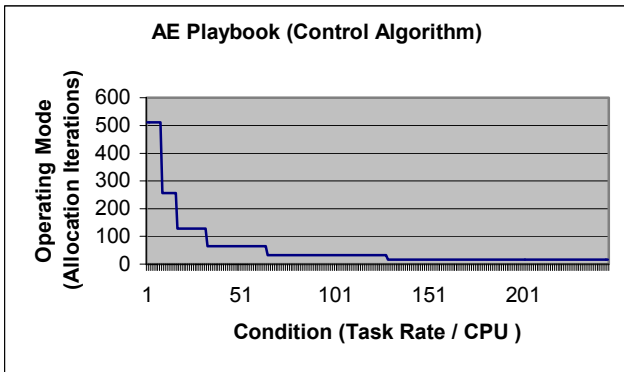


Figure 2 – Control Algorithm of the Adaptivity Engine Demonstration Example

Figure 3 shows the dynamic behavior of the example after stimulating the generator agent via user interface (servlet). Due to the high number of incoming tasks and the decreasing CPU resources, the adaptivity engine modifies processing in the allocation plugins so that tasks can be processed faster (trading off the quality of these allocations). Once the task load is reduced and the CPU resources increase, the operating mode is set to its initial value again.

Example 2: Utility-based Control for Adaptive Logistics

In our military logistics application, the completion of tasks vary in their utility to military planners based on 2 basic factors: task planning fidelity and task planning horizon. These are simplifying assumption that we have made, based on our simplified planning model that is used for researching system survivability. Of course, a real planning system would have to satisfy a larger set of planning requirements. Our objective is to maximize utility under system conditions of varying workload or processing resource availability. Assuming that high fidelity tasks require a high level of computational resources, but provide a high utility, we must trade-off computing high and low fidelity tasks to achieve the maximum utility.

To further simplify our control problem, we have quantized control regions into 2 levels of task fidelity (high and low fidelity) and 2 planning horizons (the next 6 days and the time period thereafter). These control regions correspond to the Operation Modes of the logistics application components. For each of these 4 control regions, we have specified unique utility curves. Figures 4 and 5 illustrate the utility functions for each planning horizon, respectively:

- 1) high fidelity within the 6 day planning horizon
- 2) low fidelity within the 6 day planning horizon (no utility)
- 3) high fidelity within the subsequent planning horizon
- 4) high fidelity within the subsequent planning horizon

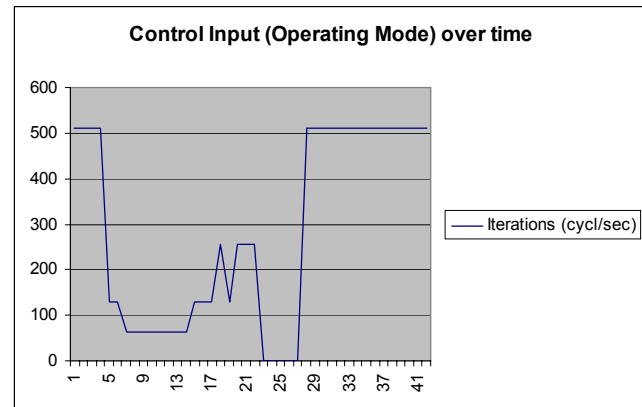
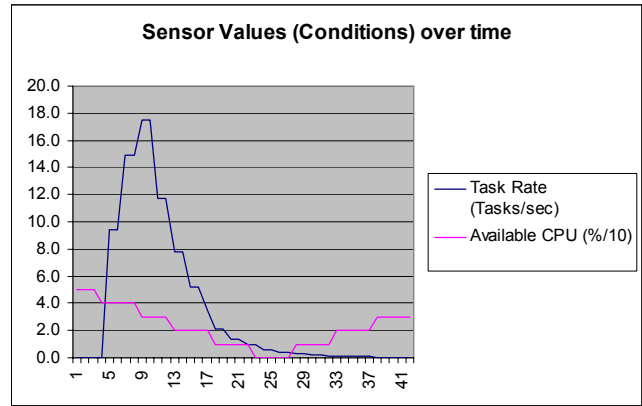


Figure 3 – Experimental Conditions and Operation Mode Values

In order to maximize utility with respect to system load, we need a measure of system load to drive our control actions. To quantify system load, we determine a quantized “falling behind” measure that is driven by a measure of task backlog. This falling behind measure is quantized into 3 levels: severe, medium and none; and provides the Condition that drives Adaptivity Engine play selection.

Once our Operating Modes and Conditions are defined, we construct a Playbook that represents the choice of control actions for the Adaptivity Engine. As a first attempt, we constructed a set of Plays based on the defined planning horizons. Table 1 describes this Playbook. Experimental

results were used to define the thresholds for the falling behind measures that maximized our utility score.

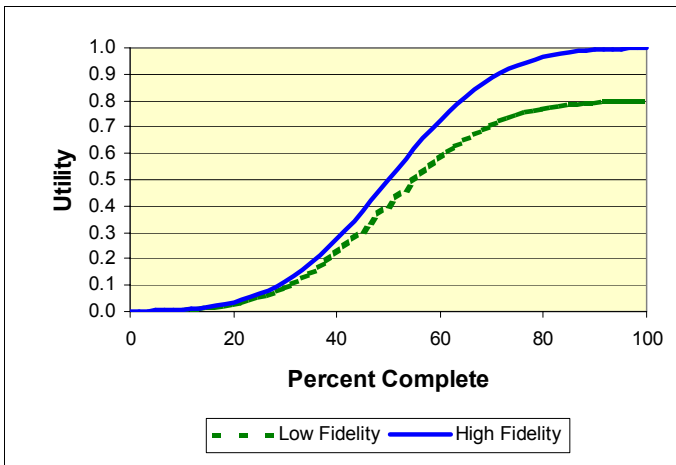


Figure 4 – Task Planning Utility Curves for Six-day Planning Horizon

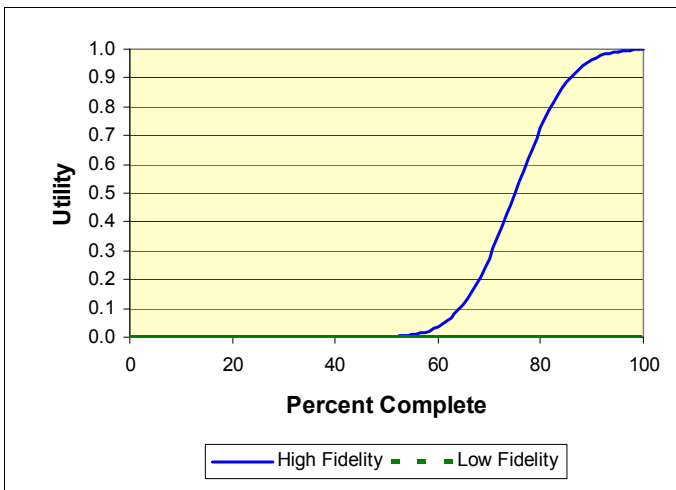


Figure 5 – Task Planning Utility Curves for Subsequent Planning Horizon

Table 1: A Playbook Example

Play	Condition	Operating Modes
1	None	high fidelity = end of period low fidelity = none
2	Medium	high fidelity = 6 days low fidelity = 7 days to end of period
3	Severe	high fidelity = 6 days low fidelity = none

Utilizing the Conditions, Operating Modes, and Plays described above, we were able to maximize the utility of our application. Based on dynamic system measurements under varying system stresses (affecting available computational resources), the adaptive controller performed well, selecting the appropriate control action for the measured system condition. This initial implementation represents our first attempt at closed loop control of a DMAS using the Adaptivity Engine mechanism. As part of our continuing research, we plan to use the utility functions and a more

granular measure of system load to construct a Playbook as a closed form solution of the utility functions.

6. CONCLUSION

We discussed several characteristics of DMAS that make them a special case of a controlled process, to which the conventional means of control theory are hard to apply. We argued that, instead, software engineering uses a more experimentally driven approach to control, often leading to rule-based controllers parameterized by heuristics.

From a control theory perspective, the control infrastructure presented in the previous section has the following properties:

- The architecture allows both feedforward and feedback control, depending on the selection of conditions and operating modes.
- The modification of the control algorithm (Plays), either by using a model (TechSpecs) or by policies constraining existing plays, constitutes for an adaptive control system.
- The Adaptivity Engine in conjunction with the Playbook constitute a rule-based controller. Theoretically, the control algorithm could be linear or nonlinear; as soon as policies impose constraints, it becomes nonlinear.

The Cougar open source agent architecture provides a rich set of sensor instrumentation and a control infrastructure that is easy to adapt to various applications. Our results obtained under the UltraLog program have shown that this infrastructure is suited to support the various control goals of a survivable system.

However, in this area of multivariable control, there are still many open issues to be solved by our future research. Examples are, the proper distribution of control knowledge in order to avoid single points of failure; the systematic optimization of control parameters; or the wide use of small models for components (TechSpecs) that are currently still in the early stages of development.

7. ACKNOWLEDGEMENTS

This work was sponsored, in part, by the DARPA UltraLog contract #MDA972-01-C-0025. These ideas represent contributions by the many individuals who participated in the DARPA ALP and UltraLog programs.

8. REFERENCES

- [1] The Cougar Open Source Website: <http://www.cougaar.org>
- [2] The DARPA UltraLog Program Website: <http://www.ultralog.net>

- [3] Helsinger, A., Lazarus, R., Wright, W., Zinky, J., "Tools and Techniques for Performance Measurement of Large Distributed Multiagent Systems," accepted for: 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Sidney, 2003.
- [4] Brinn, M., Greaves, M., "Leveraging Agent Properties to Assure Survivability of Distributed Multi-Agent Systems," accepted for: 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Sidney, 2003.
- [5] Combs, N., Vagle, J., "Adaptive Mirroring of System of Systems Architectures," 1st Workshop on Self-healing Systems, Charleston, NC, 2002.
- [6] Valetto, G., Kaiser, G., "A Case Study in Software Adaptation," 1st Workshop on Self-healing Systems, Charleston, NC, 2002.
- [7] Kokar, M., Baclawski, K., Eracar, Y., "Control Theory-Based Foundations of Self-Controlling Software," IEEE Intelligent Systems, May/June, 1999.
- [8] Selfridge, O.G., Feurzeig, W.: Learning in traffic control: adaptive processes and EAMs", Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on , Volume: 3 , 12-17 May 2002 page(s): 2598 –2603