2004 Command and Control Research and Technology Symposium

The Power of Information Age Concepts and Technologies

An Exposé of Autonomous Agents in Command and Control Planning

Author: Christopher Matthews

RDECOM CERDEC
Command and Control Directorate,
Battle Command Division,
Integrated Battle Command Branch

HQ CERDEC
AMSRD-CER-C2-BC-IBC
Fort Monmouth, NJ 07703

Voice: (732) 427-3978
Fax: (732) 427-2685
Claw.Matthews@us.army.mil

An Exposé of Autonomous Agents in Command and Control Planning

Christopher Matthews
Communications Electronics Command
Command & Control Directorate
AMSRD-CER-C2-SS-A
Fort Monmouth, NJ 07703

Abstract

Autonomous agents are a value-added technology to facilitate integrating logistics into the in-theater maneuver planning processes. Conventionally, logistics impacts that can affect the successful prosecution of a maneuver plan are only analyzed or discovered after the maneuver course of action (COA) has been finalized. Logistic oversights, assumptions, or faulty assessments can lead to an unsupportable maneuver plan that must be re-planned or thrown away; a situation that may require unafforded time. We implemented autonomous agents to provide a near-real-time, logistics feedback loop to augment the on-going, dynamic battle planning processes. As the battle plan is progressing, a logistics plan, too, is being formulated whereby logistic assets are being tasked to support the tentative maneuver plan, dynamically. This logistics plan that maps to the current battle plan is the mechanism whereby resource management and allocation impacts are discovered and, thus, can alert the command and control (C2) planners of logistics issues. The coupling of logistics and C2 planning saves time, effort, and better utilizes resources and personnel. But, the major benefit is the adaptation of this model when applied to the execution-monitoring and real-time prosecution of a maneuver plan – using unfolding battlefield events as re-planning inputs to the current, executing plan.

1. Preamble

This paper is not intended to discuss the formal definition of what constitutes an intelligent agent (IA). Nor does it address how IA concepts differ from conventional programming – object-oriented, process-oriented, or otherwise. Suffice it to say that we recognize the intent of what many advocates of intelligent agency proclaim – that substantive differences exist in composition, usage, and the deployment environment for which these "agents" exist. We consider the debate as to what constitutes an agent a philosophical one, at least for the time being. And, one that is best left to message board debates or college-level courses. We are primarily concerned with the viability of this technology for its use in military applications and how it can be exploited to increase effectiveness and efficiency on the battlefield in the planning and execution phases of C2.

2. Objective

We wanted to learn how to use intelligent agents effectively in the military command and control (C2) domain, and we wanted to gain experience with agent frameworks. We sought to develop a relevant, prototypical system that tried to solve a critical mission need and use this experience to surface the transition-strategy issues and the maturity of agent technology for present and future C2 programs that build decision-aid software for the in-theater battle commander and staffs.

2.1 Relevance to C2
Information battlefield management has become significantly more complex; deciphering the relevant data from the minutia can be a daunting, never-ending job. Using IA mitigates the risks associated with meeting current and future information battlefield management needs. IA technology allows the commander to more efficiently track objectives of a battle plan, manage battlefield resources, and utilize unfolding, battlefield events and external conditions as inputs to the warfighter's dynamic planning process. Using agents as the core technology to supplant the current C2 planning paradigms empowers the commander to her job better – commanding the battlefield vis-à-vis analyzing mountains of data.

3. Background
We investigated agent technology as part of the raw research and development (R&D) effort under the Logistics Command and Control Advanced Technology Demonstration (LogC2 ATD) program. As stated above, we developed agents to tightly couple the logistics planning process with the maneuver planning process. This prototype system, or agent application, was to satisfy a critical mission need and to meet the exit criteria for the program.

3.1 Sponsorship
The LogC2 program commenced in FY98 and concluded in FY03. Our agent development effort began in mid FY02. As with many programs budgetary constraints and reallocation of program funds had severely curtailed our grand plans of funding agent research and application. Despite the funding cuts we were still able to pursue this R&D effort, albeit, on a much smaller scale. We originally had planned to include many facets of logistics planning, and at a fidelity that could provide a micro-view of the scheduled logistics assets to support a maneuver plan as well as the modeled consumption estimations for both the maneuver and logistics domains. The adage of biting off more than one can chew was realized when we tried to tackle this big, extremely complex, cross-functional problem. For a two-man development team that had zero agent development experience this goal was pretty ambitious if not downright unrealistic. So, we scaled our objectives accordingly and focused only on the fuel modeling and resource management aspect of logistics to support a maneuver plan. We came to find later that the agent architecture that we used promotes domain-specific reusability of the business logic components and is amenable to developing a system in a piecemeal fashion. We could essentially build a system that focused solely on a one domain and integrate later any other domains as they became available.

3.2 The Agent Framework: an Executive Summary
The agent architecture that we used to develop our agent society is the Defense Advanced Research Project Agency (DARPA) Cognitive Agent Architecture (Cougaar). Cougaar is and open source, Java-based agent framework from which one typically develops large-scaled, long-lived, distributed agent applications. Cougaar provides the underlying infrastructure to create and manage agent applications while abstracting much of the underlying architecture from the developer. The agent creator need not necessarily

concern herself with the inter-agent communications layer, message pooling, or threading models used to manage computer resources among agents and the like. A developer can focus on the design and implementation of her agents and how they are used to solve complex, real-world tasks using the agent-ontology and tools provided by the infrastructure.

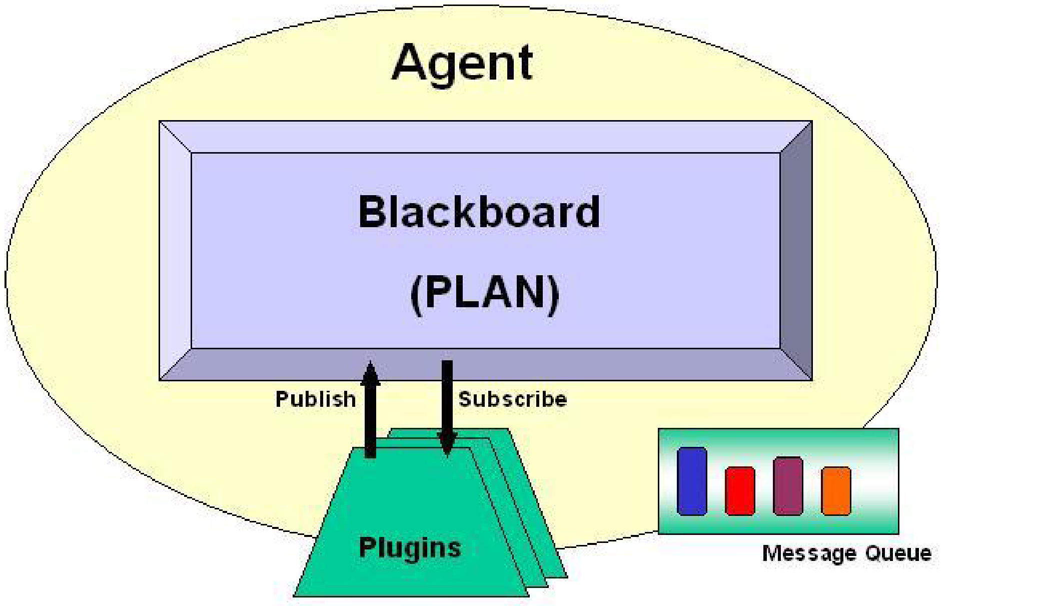
The Cougaar framework was borne from the Advanced Logistics Program (ALP). ALP was a DARPA program, initiated in FY96 and completed in FY01, which sought to revolutionize military logistics planning by exploiting novel information-age technologies. Conventional software solutions proved too cumbersome when managing the vast numbers of unique items to model as per the C2 planning process. At the completion of the ALP program DARPA had successfully developed an agent architecture that makes it much easier to build decision-aid software for the logistics community that provides dynamic planning and execution-monitoring capabilities. And, removing the military-specific functionality, the benefits of this agent architecture could be realized in any domain desiring a dynamic planning capability. Cougaar is simply the generic version of the ALP agent framework minus the embedded military logistics domain properties and logic providers. Follow-on enhancements to security, scalability, and survivability of Cougaar are the objectives of another on-going DARPA program, UltraLog.

In short,

> "*Cougaar is a large-scale workflow engine built on a component-based, distributed agent architecture. The agents communicate with one another by a built-in asynchronous message-passing protocol. Cougaar agents cooperate with one another to solve a particular problem, storing the shared solution in a distributed fashion across the agents. Cougaar agents are composed of related functional modules, which are expected to dynamically and continuously rework the solution as the problem parameters, constraints, or execution environment change.*"(BBN Technologies, 2003)

3.3 The Agent Framework: a Detailed Look

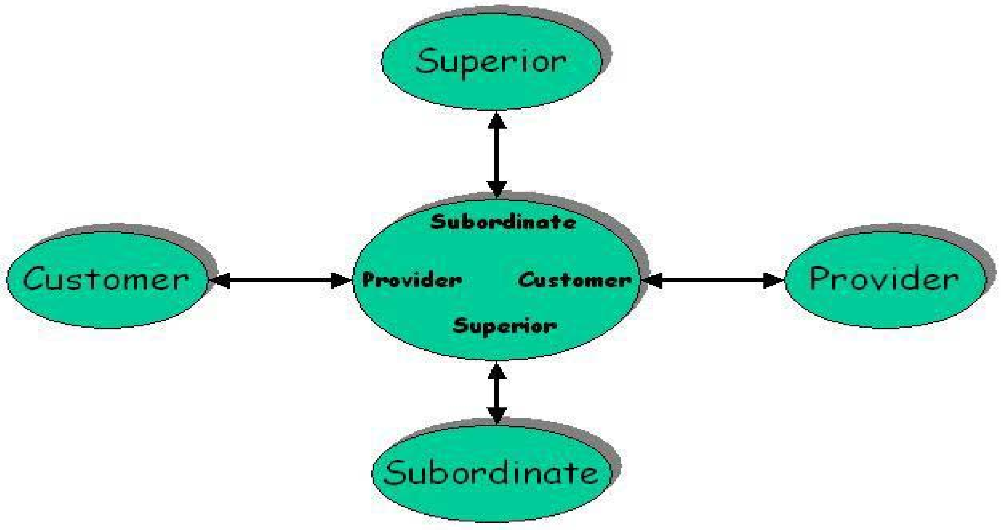
The Cougaar agent is comprised of *Plugin* components and one *Blackboard*. The *Blackboard* performs two essential functions. First, it is the communications channel for the agent; all of its communications will originate or terminate here. Second, it serves as the repository for any information needed by that agent to participate in the distributed planning process. A *Plugin* is a software component that provides each agent with its unique, domain-specific behavior. *Plugins* exchange information with each other, asynchronously, through publish/subscribe transactions to the *Blackboard*. When "subscribed-to" objects are introduced onto the *Blackboard* the *Plugin* is awakened by the infrastructure and executed.



(Cougaar Training slides, 2004)

Figure 1. Cougaar agent anatomy

Agents collaborate through pair-wise relationships, or roles, with other agents. Agents are aware of each other through these specialized relationships. Cougaar defines these relationships as *Customer* ←→ *Provider* and *Superior* ←→ *Subordinate.* Cougaar allows the developer to define his or her own pair-wise relationships if need be. This inter-agent relationship is the way agents are aware of other agents that may be available to collaborate on a given task. Any agent, at any time, may be participating in its deployed environment in any role, and most likely an agent will be simultaneously operating in a multi-role capacity with various agents.



(Cougaar Training Slides, 2004)

Figure 2. Agent in multiple roles

Conceptually, a Cougaar plan is distributed throughout the agent society. Each agent contains its portion of the overall plan on its blackboard in the form of *PlanElements*. A *PlanElement*, in Cougaar-speak, is simply one piece of the overall plan, and typically, agents will have many *PlanElements* on their blackboard. A workflow is just the description and details of a process. The notion of *PlanElements* is how Cougaar represents a workflow throughout the agent society. This specification is what allows agents to share information and understand the context of the distributed plan. A *PlanElement* is always associated with a task. The *PlanElement* provides traceable information on the task's progress and how it relates to other tasks or agents, for example, they denote whether the task has been decomposed into subtasks or allocated to another agent.

Tasks are the cornerstone of this workflow specification. Tasks provide the 'who, what, where, when, how' problem statement definition. Tasks are defined by their grammar elements (subject, verb, direct object, prepositions, etc.) For example, one may assign a task to a transportation company to: *Transport 6000 gallons of JP8 fuel to Bravo Co, Bn 1-8 Armor at 1600 hrs on D+5 day.* And, an agent designed to handle transport tasks would process this task request using the task grammar elements and task constraints as to how it will process this unique directive.

Cougaar blackboard objects are used to define and distribute the plan throughout the society. While any object may be posted to the *Blackboard*, typically, the only necessary object types needed for collaborative planning are *Tasks*, *Assets*, and *PlanElements*. Assets, while not covered in detail here, are simply the consumers of tasks. When tasks are created or decomposed they eventually must be allocated to an asset. When tasks are allocated to an asset Cougaar provides a mechanism for tracking estimated, reported, and observed allocation results (projected or actual task completion status). This feedback capability is provided by the underlying architecture and makes dynamic re-planning and execution monitoring of a plan possible. This, too, is how the continuous refinement of a plan can transcend from being merely feasible to (more) optimal (see figure below).

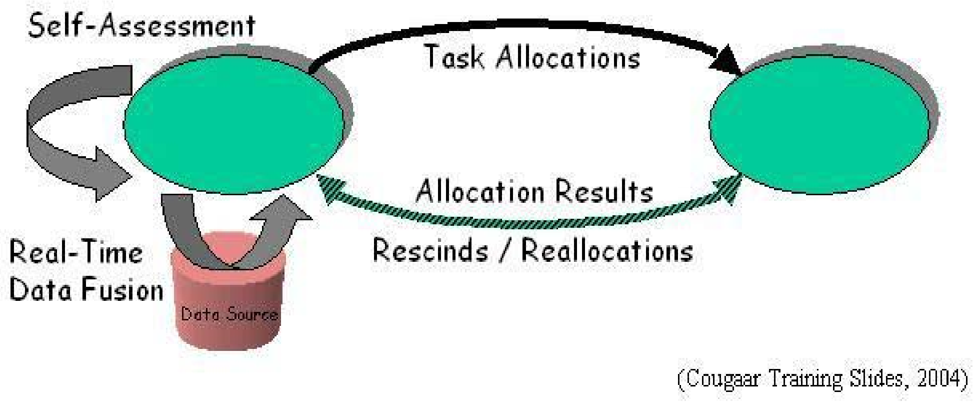


Figure 3. Dynamic re-planning and execution-monitoring concept

The dynamic replanning and execution-monitoring mode as described by the Cougaar Architecture Document is the "dynamic negotiation between Agents and Plugins to generate a feasible and ultimately optimized cooperative solution. The plan is based on real world requirements, situation information and asset availability: what do I need to do, what is the state of the environment and what can I use to accomplish my task? As these variables change, the solution becomes stale and Cougaar forces replanning to determine how to adjust the plan to compensate for those changes, if possible. Further, Cougaar continually monitors the plan as it is executed, and forces replanning as assumptions are modified in real-time."(2003)

4. C2 Logistics-Planning Tool

Based on the capabilities of Cougaar and it being tailored for military applications, our decision to use Cougaar for developing our agents was a natural fit. The Cougaar agent framework provides the logical separation of the business processes from the data. This separation of data and the development environment makes it easy to build distributed, scalable, maintainable, upgradeable, interoperable, and most importantly, reusable and sharable software components.

4.1 System Overview

Our system imports a digitized maneuver plan as the primary input to initiate the logistics planning process in our logistics-planning tool. Our system, using planning factors, the plan's unit task organization, a tailorable force structure, and equipment usage profiles, produces the fuel consumption and demand generation model. This model is used to determine maneuver sustainment feasibility and to generate an initial log plan depicting all logistic asset allocations to support the maneuver plan. Once the plan enters the execution phase, changed external-events, conditions, and results of allocated tasks will be used to modify and adjust the on-going plan as it is unfolding.

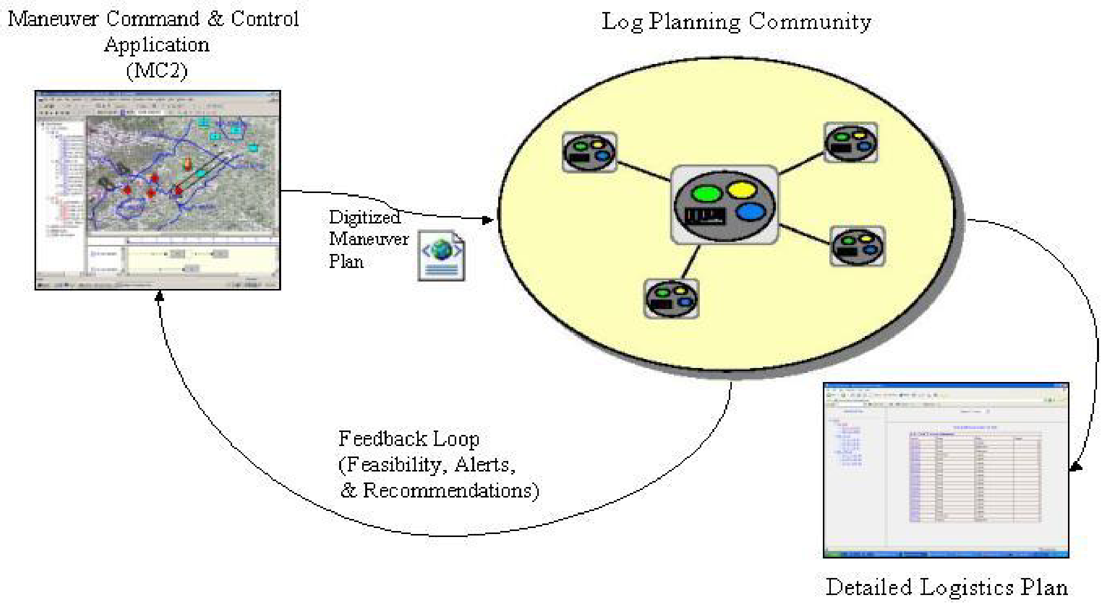

Maneuver Command & Control Application (MC2)
Log Planning Community
Digitized Maneuver Plan
Feedback Loop (Feasibility, Alerts, & Recommendations)
Detailed Logistics Plan

Figure 4. System overview of the C2 logistics-planning tool

The initial maneuver plan is parsed to extract units, tasks, planning factors, etc. The maneuver plan is modeled using the information from the maneuver plan, yielding the demand generation for the fuel needed to perform the mission. If the logistics infrastructure can support the maneuver plan then a detailed logistics plan is developed that depicts the logistic asset utilization needed to meet the plan objectives. If the maneuver sustainment requirements cannot be met then, using a feedback loop to the C2 planning application, alerts and recommendations are exported to depict the shortfalls and reasons for being unsupportable.

4.2 System Components

The components that make up our system are a collection of agents referred to as a community. Our community of agents collectively models the maneuver plan to formulate a corresponding logistics plan (LOGPLAN).

There are basically two types of agents that comprise our logistics-planning tool. First, there are administration or management agents that handle system initialization and synchronization of the entire system at startup. These agents coordinate the system at startup to allow all the agents to initialize to completion before embarking on any problem solving. This must be done to ensure that the agent society is accurately represented before beginning. Once initialized, the system can elegantly handle changes in the environment and incorporate these changes into the planning process. After initialization these agents are no longer used.

The second and most important type of agent represents a military organization or unit. These agents model the collective behavior of military units in the context of how they behave in the given force structure in which they are operating. For our prototype, the agents represent Task Force XXI (TFXXI) force structure military units. These agents are loaded with business logic (plugins) representing the tactics, techniques, and procedures as outlined in TFXXI doctrine. These agents are the workhorses of the system. They provide the units with the capability to cooperate and collaborate to negotiate a valid LOGPLAN. Unlike the management agents these agents are meant to be active the entire time that the system is in the planning or execution-monitoring mode.

4.3 Developmental Items

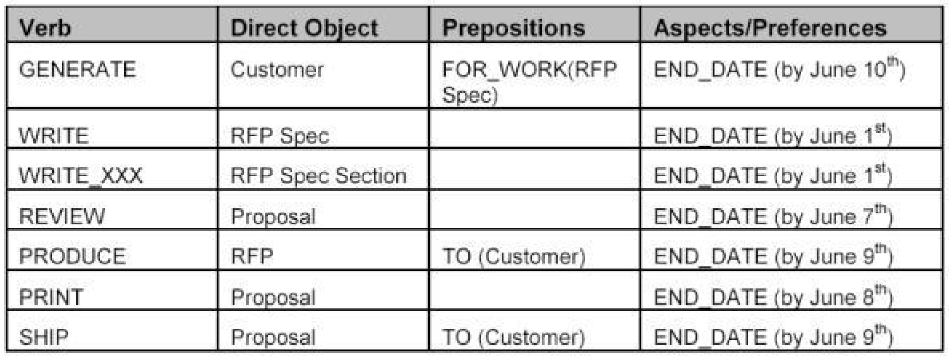
The *plugins* that make up an agent and define its behavior are the only items that the developer must create (or reuse). This paradigm is precisely why Cougaar is so attractive as a development framework. The developer need not concern herself with developing the intricate, underlying management infrastructure to make such a distributed system possible. She only needs to build the business logic modules that make up an agent and then deploy that agent. Building *plugins* to model the cognitive thinking process is a difficult task in and of itself. The arduous task of modeling the command and control domain is made orders of magnitude simpler through the use of Cougaar agents. Consequently, the major effort in developing our tool was *plugin* development. The remainder effort was simply configuring our environment for which to develop software and verifying that the agents behave correctly.

Agent *plugins* encapsulate a unit's behavior. A forward support company agent, for example, may have a *plugin* that processes incoming refueling requests. Depending on the unit and the appropriate doctrine, the unit may expand the request into subtasks or allocate the task to another agent or asset (e.g. fuel truck and/or personnel). The designed behavior of a *plugin* is always user-defined and will typically be consistent across units of the same type or domain – making them reusable, sharable, and easily maintainable.

Plugins typically should be lightweight components that perform a specialized function (e.g. processing specific incoming tasks, or managing a pool of assets.) A *plugin* can comprehensively model all of an agent's behavior but will likely be complex, inflexible, and hard to maintain. A *plugin* will most likely fall into one of the following categories: a graphical user interface (GUI) *plugin* that allows the user to interact with the system, a domain *plugin* that models a portion of how agents do business within the community, or a logical data model (LDM) *plugin* that is used to incorporate system-external data into the agent society. Our tool implemented several *plugins* of each type in order to model the processes, interoperate with other systems, and to display the state of the plan to the user.

4.4 User defined ontology

Cougaar provides the infrastructure to help you model and develop a complete planning system. Cougaar, however, is just a template for doing generic planning. It is still up to the users to define things like: task grammar, asset classes, asset properties, task decomposition, and how to score allocation results. Once the specialized vocabulary has been developed you can begin to model the workflow management process in your agents. Below is a sample of what a simple task definition table would look like.



Verb	Direct Object	Prepositions	Aspects/Preferences
GENERATE	Customer	FOR_WORK(RFP Spec)	END_DATE (by June 10th)
WRITE	RFP Spec		END_DATE (by June 1st)
WRITE_XXX	RFP Spec Section		END_DATE (by June 1st)
REVIEW	Proposal		END_DATE (by June 7th)
PRODUCE	RFP	TO (Customer)	END_DATE (by June 9th)
PRINT	Proposal		END_DATE (by June 8th)
SHIP	Proposal	TO (Customer)	END_DATE (by June 9th)

(Cougaar Architecture Document, 2002)

Table 1. Sample Cougaar task grammar table and decomposition

4.5 Synopsis

The end of the LogC2 ATD signified the end of our development of the C2 Logistics-Planning Tool. By program end we had developed a prototype that could model a small set of maneuver plan activities, generate the consumption demand for the fuel needed to prosecute the plan, and manage the resources needed to schedule fuel deliveries (retail and wholesale planning) on the battlefield using Forward Support Company and Base Support Company combat trains.

The user could navigate the entire plan across the distributed society and trace workflow results using our web-based GUI. She could visually inspect plan shortfalls, task results, and task schedules (sync matrices) at all levels of the force structure (down to Company-level) and assets within the units as well. Our plan to incorporate user interaction to trigger dynamic replanning was omitted due to time constraints. However, dynamic replanning concepts were demonstrated using computer stimulated changes to the agent environment (we had introduced modifications to unit inventory by simulating fuel truck breakdowns during the execution phase of a plan).

5. Results

What we have found through our somewhat limited exposure to Cougaar is that there are some situations where using the architecture is indispensable and there are other situations where using Cougaar may be overkill. We outline from our experience where Cougaar is a value-added development environment and where it may provide more features than what is needed for a particular application.

There are many features of Cougaar that we have not utilized nor explored. Agent mobility, for instance, was not a concern for our application so we did not use it. However, Cougaar provides this capability. Likewise, we did not explore the data persistence and fault-tolerant features of Cougaar either. We invite the reader to investigate Cougaar further for consideration of using it for their agent development framework or planning applications. We only provided an overview of the architecture to extract the salient points of what benefits the architecture provides.

5.1 Benefits

The benefits of using Cougaar to build your C2 planning applications are many. Depending on your planning needs and intricacy of the planning processes will determine if Cougaar is right for your program. But, on the whole, Cougaar is tailored for developing agent-based planning applications, and if your application domain is military planning then Cougaar is really the only logical choice afforded to you.

Cougaar utilizes the classic superior/subordinate or customer/provider relationship model. Any planning application or real world business process will map elegantly to Cougaar for this reason, and the underlying architecture automatically provides you with a wealth of features through these relationships (discovery, task allocation, scheduling, etc.)

Cougaar is ideal where many agents can use a modeled process, or *plugin*. Cougaar promotes software reuse and makes maintenance of code simpler. You can upgrade, swap out, or fix one plugin and alter the behavior of all agents using it. This capability is attractive when modeling military units in various operating environments or force structures. A unit's behavior can be changed to reflect its operating environment simply by exchanging plugins.

Cougaar promotes reuse and information sharing across functional domains as well. Highly detailed, specialized, planning systems can be developed and made available to any other agent community that needs that planning service. For instance, our logistics-planning tool could utilize an agent community developed elsewhere that specializes in maintenance planning. When logistics assets break down in the field we could request support from this maintenance provider. The ubiquitousness of Cougaar can lead to a tightly coupled user library of planning services available to many agent-based applications.

Cougaar facilitates the design process needed to develop planning systems. The hard part is already done. The agent and workflow management infrastructure is provided. The only design and development left to the software engineer is modeling and coding the planning processes. While this is still no trivial task, designing and implementing a monolithic, planning application that can do dynamic replanning and execution monitoring from design to completion could take years longer. Cougaar, after hurdling its learning curve, can decrease the time-to-market deployment of your system.

5.2 Weaknesses

There are a few drawbacks to using Cougaar. For one, it has an extremely steep learning curve coupled with lengthy, esoteric documentation. However, being open source software one can always delve into the code to understand the concepts, but this too sometimes can be overwhelming. Unfortunately, in our experience we have found this to be a major detractor for many would-be agent developers to use Cougaar.

Secondly, Cougaar is not well suited to low-bandwidth environments like tactical radio networks. Kevin Barry of Lockheed Martin's Advanced Technology Labs writes "nothing is particularly lightweight. Many performance management issues seem delegated. The event-driven behavioral model will start breaking down under heavy event traffic." (Barry, 2002) Cougaar does have a MicroEdition version that can be found on their web site (at www.cougaar.org) that may alleviate this problem, but we have not investigated this version.

Lastly, we found Cougaar to be too much overhead for simple planning modules. Cougaar is not meant for simple planning applications where the powers of Cougaar could not be fully realized. It still could be done and if proficient at Cougaar it may be the approach to take. But, considering the learning curve and its heavyweight framework it may not be prudent as the architecture to use in this instance.

5.3 Technology Readiness

The technology readiness of Cougaar is subjective. The viability of deploying agent-based systems using Cougaar will vary depending on an organization's policy regarding their technology readiness level (TRL) guidelines. For military use the TRL guidelines tend to be more restrictive than other organizations. For the most part, though, Cougaar is just a utility framework that uses the Java programming language with some remote method invocation (RMI) features of the language. The TRL issue and security implications are the subject of the DARPA UltraLog program. The results of this program will provide a more, in-depth maturity assessment.

6. Conclusion

In the real time execution and monitoring aspect, highly parallel applications involving the generation and maintenance of dynamic plans with relatively loose-coupling and low-bandwidth communications between parallel streams are too complex to model monolithically (CAD, 2003). Cougaar was developed specifically to address this difficulty in building these types of systems. Cougaar provides a way to incorporate complex workflow management

In closing, we have found Cougaar to be a premier framework for developing any software application where complex process modeling is needed, regardless of the intended planning domain. Military logistics planning, or the difficulty in modeling it in software, was the impetus for DARPA developing Cougaar. Our tool implemented Cougaar agents as a logistics-planning tool to augment the in-theater C2 battle planning process. After building our system, we could see how Cougaar could be applied generically to any application needing a workflow management capability.

Understanding Cougaar's concepts takes some time. It has a very steep learning curve. However, the investment of time and effort needed to grasp the concepts is recouped as increased productivity when one begins to design and build the implementation of your planning application. And from our assessment, the more complex your planning process is the more productivity you can expect to leverage – the architecture is *that good*.

We like the architecture so much that we have lobbied successfully to use Cougaar to implement the dynamic planning and execution-monitoring capability of the embedded sensor-planning engine for the Network Sensors for the Future Force (NSfFF) ATD. This application will import sensor information needs and develop a sensor emplacement strategy by mapping sensor objectives to sensor asset capabilities. Once the plan has begun the real-time execution monitoring capabilities will incorporate environmental changes, modified sensor objectives, and exploitive opportunities to update the current sensor plan.

References

1. Barger, Mark, & Wong, Jason. (2004). *Cougaar training slides*, (Available from http://www.cougaar.org)

2. Barry, Kevin. (2002) Cougaar *and EMAA*. [white paper]. Camden, NJ: Lockheed Martin – Advanced Technology Labs.

3. BBN Technologies. (Version 9.2). (2002). *Cougaar Architecture Document*, (Available from http://www.cougaar.org)

4. BBN Technologies. (Version 10.0). (2003). *Cougaar Architecture Document*, (Available from http://www.cougaar.org)

5. BBN Technologies. (Version 9.4). (2002). *Cougaar developer's guide*, (Available from http://www.cougaar.org)