Cougaar: A Scalable, Distributed Multi-Agent Architecture*

Aaron Helsinger, Michael Thome, Todd Wright
BBN Technologies
10 Moulton Street
Cambridge, MA 02138 USA
{ahelsing, mthome, twright}@bbn.com

Abstract – *Distributed multi-agent systems have become more mature in recent years, with the growing potential to handle large volumes of data and coordinate the operations of many organizations. However, widespread adoption by industry and government has been blocked in part by concerns about scalability and survivability, especially in unpredictable environments of attacks and system failures. In this paper, we present Cougaar, an open-source Java-based agent architecture that provides a survivable base on which to deploy large-scale, robust distributed applications. We define the challenging problem of the UltraLog project; a distributed logistics application comprised of more than 1000 agents distributed over 100 hosts, which guided the design of the Cougaar architecture to ensure scalability, robustness, and security. We conclude with a survey of Cougaar uses as the preferred agent platform for a variety of applications.*

Keywords: Agents, architectures, scalability, survivability, distributed systems.

1 Introduction

Distributed agent systems have become increasingly popular in recent years. However, many agent platforms consider agents to be relatively trivial models for single-platform execution. Others focus on developing complex features for specific research applications, without attempting to provide usability or reliability. The success of peer-to-peer systems and negotiating agents has engendered a demand for more flexible, robust agent platforms.

Cougaar has been developed to meet the stringent scalability, reliability and survivability needs of these users, as outlined in Section 2. The resulting architecture is feature-rich, while remaining easily configurable, usable, and most importantly, very reliable under stress. Section 3 outlines the key features of Cougaar.

Cougaar is one of many documented agent architectures. To better understand the relative merits of Cougaar, in Section 4 we compare it with three other platforms: JADE, Grasshopper, and Aglets. A key strength of Cougaar is the rigorous stress and scalability testing that it has undergone. Section 5 highlights the key results demonstrating the survivability of Cougaar. Finally, we highlight some of the current uses of Cougaar, and possible future directions for Cougaar.

2 Problem

Cougaar is the product of an eight-year US Defense Advanced Research Projects Agency (DARPA) funded effort to explore the potential of distributed multi-agent systems for military logistics (expanding on other works, such as [3]). Under the Advanced Logistics and then UltraLog [4] programs, DARPA challenged a collection of companies and universities to build a detailed US military logistics plan for a major 180-day deployment, and then to execute that plan, including irregular changes to that plan. This data intensive, security sensitive, time critical application is required to be robust to expected failures and adversary attacks compromising 45% of infrastructure, with minimal performance and capabilities degradation. [5] To support such a significant application, the currently completing UltraLog program distributed over 1000 medium weight agents across nearly 100 machines.

Although Cougaar can be configured to support small-scale embedded applications, Cougaar applications are typically large-scale, complex, and data intensive and must be efficient, distributed, and secure. Cougaar problems are in general far too difficult to solve on a single computer. These problems require user visibility into system operations, and occasional human feedback. While important for policy direction and exception handling, human input must not be a required part of application performance: Cougaar applications are typically autonomous and may not depend on timely response from a human operator to produce timely results.

Cougaar applications are most often naturally distributed, often for domain reasons beyond simple load balancing of hardware resources. The applications must maintain, at

* **0-7803-8566-7/04/$20.00 © 2004 IEEE.**

minimum, points of presence with the component organizations being modeled. Many such organizations have high-bandwidth data requirements, have their own semi-private databases, and require high levels of interactivity with their local users.

Cougaar systems must be continuously available, and must degrade gracefully if any component is missing, disconnected or damaged. In particular, we assume that the network over which Cougaar operates is not wholly reliable; connectivity and bandwidth are limited, and latency is high. Cougaar also assumes that all hardware is inherently unreliable, and any security or survivability mechanism may fail. In this context, the architecture should be self-maintaining and provide robust distributed failure recovery. Cougaar systems should be amenable to security in all its forms: integrity, confidentiality, authentication, and automatic response to attack.

In order to achieve these goals, we quickly focused on a particular class of multi agent systems. Cougaar agents are composed from multiple components, such that their own behaviors are emergent from the interactions of those components. Each agent is complex, long-lived and heterogeneous (in both behavior and software), and runs autonomously and asynchronously with respect to its peers.

The system architecture cannot impose any scalability limits, nor include any single points of failure. To allow efficient parallelism and distributed operations, communication between agents (at least over distributed links) must be message based and asynchronous. Finally, we require that Cougaar agents support run-time mobility and state persistence. The architecture must be modular, supporting heavy code re-use, replacement as necessary, and parallelism. Together these characteristics reflect the attributes of our target application, for which the Cougaar architecture was tuned.

3 Cougaar

Cougaar has been developed following these principles and successfully proven to meet our objectives. The result of implementing these principles is available as open source software under a BSD-equivalent license [2].

Cougaar, in fact, is composed of multiple layered applications interacting – a system of systems. These layers include several logistics applications, and a collection of support infrastructure applications: distributed naming and directory services, robustness and survivability infrastructures, and a military-grade security sub-system. Each of these is a scalable distributed multi- agent application in its own right. They all share the same underlying infrastructure for loading, communicating, and surviving. As a result, each application builds on the strengths and features of the rest of the infrastructure.

To provide the reuse and survivability our problem area requires, we built several key features into the Cougaar architecture. Cougaar uses a flexible component model to dynamically load components, inserting component "binder" proxies between components to mediate interactions with system services. To permit scalability, Cougaar uses a multi-tiered interaction model. Within agents, components interact via a local publish-and-subscribe mechanism, while inter-agent communication is via message passing. To handle the large volumes of data necessary, we developed a flexible data model that characterizes objects by their attributes, increasing code and memory re-use. Agent relationships are dynamically negotiated, using a hierarchical service discovery mechanism. Agents organize themselves into communities to monitor security conditions and agent availability, allowing them to adaptively control their behaviors.

The following sections detail the major components of Cougaar, as illustrated in Figure 1. We start with the overall organization of a Cougaar agent and its component pieces ("plugins"). Then we outline the Cougaar message passing mechanism, intra-agent communications, and persistence mechanisms. Cougaar also has a number of support services: naming, communities, service discovery, and per-JVM web servers. Finally, we outline the Cougaar support for representing assets and supporting task planning and other applications, as well as the security and robustness enhancements provided by the UltraLog program.

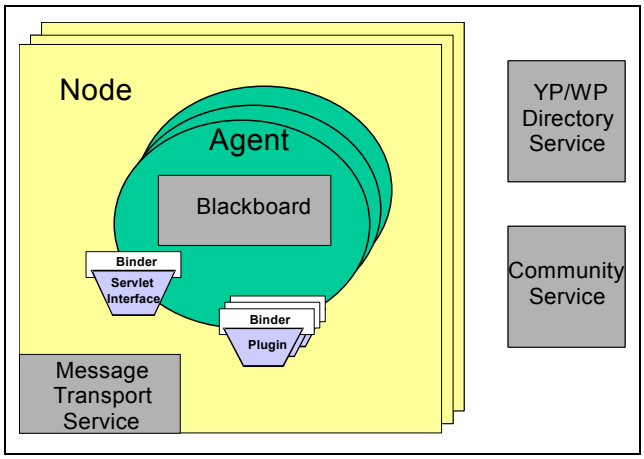

Figure 1. Cougaar components schematic.

3.1 Component Model

The Cougaar Component Model (CCM) [7] is a framework that loads and manages software units called *Components* that connect to and interact with one another through abstract interfaces called Services. The design is patterned after JavaBeans [6], re-factored to remove Swing dependencies and add the features below.

The CCM adds to traditional component models a strong service isolation and encapsulation model with an abstraction we call *Binders*. A binder is essentially a wrapper around a component that may audit, authorize or modify (similar in function to aspects [8]) any communications between a component and any services it interacts with. Binders properly nest so that a hierarchy of components may impose several layers of binder on a component that uses a high-level service. Binders are themselves components, so are subject to the same rules and have the same flexibility as other components.

The components that comprise a Cougaar agent are dynamically loaded and connected together at application start time, very much like most modern GUI-based applications. Cougaar uses this flexibility to enable high levels of customization to ease application construction.

3.2 Plugins

A *plugin* is a software component that is added to an agent to contribute a specific piece of application business logic. Each plugin adds domain-specific behavior to the agent. The interesting or emergent behavior of an agent depends primarily on the set of plugins that are loaded into it. The aggregate behavior of all the plugins in the agent determines how the agent responds to the messages received from its peers. In most Cougaar applications, the agents differ only by data and the set of component plugins.

Once loaded into an agent, a plugin instance is a conceptually independent entity from the agent, acting through the agent's blackboard. The plugins of an agent, in effect, form a smaller agent-like network that is data-driven via a publish-subscribe mechanism rather than event-driven through messages (see [10]).

3.3 Message Transport System

The Cougaar Message Transport Service (MTS) is an adaptive JVM-level service that handles all inter-agent communication in a Cougaar society. The MTS is componentized and includes adaptive features that can be selected at runtime. Asynchronous transport protocols can be plugged into the MTS as components, where standard protocols include RMI, CORBA, HTTP, and UDP. Developers can insert Aspect components to control message handling, including security control over protocol selection. The MTS includes a robust retry mechanism that guarantees correct in-order delivery in case of transient communication failures or unplanned agent mobility.

UltraLog developers have enhanced the MTS by creating adaptive transport protocols and aspects. By default, the MTS is configured for maximum performance, with minimal resource allocated to robustness or security. UltraLog components can be added to adaptively enable alternate protocols that maintain function in stressed environments, such as compression for reduced bandwidth usage, SSL for security, or SMTP for store-and-forward messaging in intermittently partitioned networks

3.4 Blackboard

The components of a Cougaar agent communicate via a fairly traditional-looking *Blackboard* with standard publish/subscribe semantics. Cougaar blackboards are not themselves distributed – while the Cougaar data space could be viewed as the union of the blackboards of all connected agents, remote blackboards are never *directly* accessible. Blackboard modifications are transaction controlled using a membership model: that is, precisely the facts of add/remove/modify events are transaction controlled, never changes to referenced or inner structure.

For interactions with other agents, blackboard objects are transformed into messages by domain-specific *LogicProviders*, small pieces of code that allow messaging semantics to be assigned to classes of blackboard objects. For instance, in a planning domain, publishing a *Task* assigned to an *Asset* that is represented by a different agent will result in a message containing that task being sent to that agent. Developers can create and dynamically load new LogicProviders to support custom domains.

3.5 Persistence

Persistence is the mechanism whereby Cougaar agents can recover from many types of failures. Persistence saves the state of every published object and the state of the subscribers relative to those objects on non-volatile media. In addition to saving the primary blackboard-related information, the state of communication with other agents and the current microstructure of the agents (e.g., dynamically loaded components) is also saved.

When an agent is started, its state is restored from persisted data if available. Most components can continue executing as if there had been no interruption: If a component maintains no internal state except that which is published to blackboard with each transaction, it can continue execution without regard for whether a restore occurred or not. If a component does autonomously publish objects to the blackboard then it must exercise care to not repeat such actions if the objects are already on the blackboard. Inter-agent blackboard state reconciliation is used to recover from lost persistence data.

3.6 Naming Services

The Cougaar *White Pages* (WP) service is a distributed table that maps agent names to network addresses. The primary function of the WP is to support the MTS and other network-aware components.

While the interface and implementation of the WP is similar to DNS, the focus in Cougaar is on support of relatively rapidly changing names in parallel namespaces rather than relatively slowly changing names in a single, universal namespace. The WP supports replicated (peer) servers and hierarchical names (and servers). It is implemented entirely in-band as a Cougaar application itself, fully leveraging the security and robustness capabilities of the Cougaar defense applications [12].

The Cougaar *Yellow Pages* (YP) service is a directory service that supports attribute-based queries. This service allows agents to register themselves based upon their capabilities, and to discover other agents using queries for these capabilities. The YP is actually a fully independent Cougaar application which implements a complete UDDI [11] service through the standard message transport. An alternative server implementation is supplied to allow use of external UDDI servers instead of or in addition to the internal server(s). Cougaar also supplies a hierarchical YP registration and search facility, mirroring the hierarchical structure of most applications (e.g., spatial or organizational). The use of hierarchical servers increases the overall system scalability and often generates more applicable locality-based directory query results.

3.7 Communities

A Cougaar *Community* is a group of agents with some common functional purpose or organizational commonality. This notional grouping is made concrete by combining a membership list service (the Community Service) with a mechanism for community-based broadcast messaging.

A Cougaar application using communities will typically group agents with similar roles (e.g., "provider of X"), physical attributes ("running in Datacenter Y") and/or organizational features ("works for Z"). Members of a community may also be assigned particular roles therein ("Agent P provides YP Services for Community Q").

Attribute-based messages facilitate the delivery of messages to partially specified targets by resolving community-scoped and role-filtered specifications to sets of specific agents. Developers can use this feature to easily broadcast messages to a whole community or to agent(s) with a known role in the specified community scope.

3.8 Service Discovery

Service Discovery (SD) in Cougaar supports the efficient rendezvous between application providers and customers. In addition to hiding the intricacies of UDDI registry traversal and maintenance, the SD framework adds a high (semantic)-level query abstraction on top of plain YP queries, simplifying best-match provider discovery across multiple YP registries. To discover a suitable provider, SD queries are scoped by community relevance and incrementally broaden the search area, ensuring scalability.

3.9 Servlets

Since Cougaar has been designed as a large-scale *distributed* system infrastructure, it would be difficult to provide a matching user interface toolkit based on traditional Java UI libraries, especially per-agent pop-up UIs. Instead, Cougaar provides a suite of services based on Java Servlets. Cougaar components can register Servlet paths within their agent for HTTP-based web presence. The infrastructure handles cross-agent references transparently though HTTP redirects based on the WP, creating an interwoven distributed user interface through web links that requires nothing more on the human side than a standard web browser. Servlets are not limited to HTML: XML, text, spreadsheet data, and even serialized Java objects have been used to provide external views into running Cougaar applications. Standard Cougaar provides a large set of servlets for debugging, profiling, control, and analysis.

3.10 Logical Domain Model

The Cougaar Logical Domain Model (LDM) supports the development and implementation of application data ontologies (*Domains*) along with the associated logic to translate between domains and the messaging system.

A domain is composed of the classes and objects that define the language itself; an object factory to construct the terms of the domain; and a set of business logic objects (LogicProviders) that act as translators between the domain language, other domains, and/or the MTS. A domain language may implement an externally defined ontology.

To support domain development, Cougaar supplies a facility to efficiently represent real-world objects. *Assets* are objects that are identifiable as unique instances and have well-defined properties. Property values for a given instance are usually partially described by a prototypical asset instance. This facility is described in detail in [9].

3.11 Applications

A Cougaar *Application* is constructed from:

- The Cougaar infrastructure: the default configuration of infrastructure components will serve most applications, though the application developer has considerable latitude to add, modify or remove from the standard set.

- A set of domains: Cougaar supplies one domain for workflow-based planning, and another for logistics applications. Most developers will use custom domains.

- A set of plugins: plugins usually implement the entire business logic of an application.

- A network of agents: a breakdown of the application's function into named entities with assigned tasks. This could be highly specific for organization simulations, or without a-priori structure for stateless distributed computing tasks.

- A *Society* configuration: assigning agents to hosts and plugins to agents along with any linkages to other applications, databases, and support structures (e.g., the Adaptive Defenses).

The Cougaar community has developed a large suite of tools for application configuration, deployment and experiment design which may be useful for Cougaar application developers. These tools are published alongside the core Cougaar on the open source website [2].

3.12 Adaptive Defenses

The charter of the UltraLog program has been to extend the basic Cougaar infrastructure with a set of Adaptive Defenses to a variety of stresses. The major categories of defenses include:

- Robustness: enabling Cougaar applications to operate acceptably even when hardware infrastructure has been lost or degraded. Specific stresses include the unexpected loss of agents (fully automatic monitoring and restart of agents), Denial of Service (DOS) attacks (compression, alternate messaging, etc.), and automated application maintenance (e.g., Load Balancing).

- Security: Augmenting standard Cougaar services with message and data encryption and authentication, adding a policy-driven service, servlet, and blackboard access control system, verifying jar files, and supporting IDMEF event based system auditing and monitoring.

- Adaptive Logistics: an example of modifying an application to adapt to its environment. This application uses variable fidelity processing to ensure partial solutions are available if needed. Additionally, this team uses "predictors" which model down-stream agents where connectivity is temporarily disrupted. The agents respond to changing service availability by adapting relationships.

The majority of the UltraLog developed software will be released to the open source Cougaar community after the program ends, in late 2004. Together, these features provide a rich basis for complex application development.

4 Comparison with Other MAS

In this section, we compare Cougaar to other agent platforms. Three platforms are evaluated, with an overview of their features and an examination of the different design choices that distinguish the platforms from one another.

Recent years have seen a rapid growth in the number of platforms, with a current total of over 100 products [15]. These platforms vary greatly in their features, maturity, and community acceptance. Of significant concern is active support, as many agent platforms are created as short-term research tools, where active development ends after only a couple years. A detailed comparison is beyond the scope of this paper, but we found several recent surveys ([16], [17]) that helped us narrow our comparison down to three agent platforms: JADE [18], Grasshopper [19], and Aglets [20]. These platforms are mature, widely accepted, support agent mobility, and offer many comparison points with Cougaar.

JADE is a highly popular FIPA-compliant agent platform. JADE is actively maintained and supported, where the current open-source version was released December 2003. Inter-agent communication messages are formatted in ACL or XML, and pluggable transport protocols include RMI, IIOP, and HTTP. Java utility classes simplify the construction and handling of FIPA-compliant ACL messages. Security is supported through SSL and socket-based proxy agents. Helpful GUIs are provided to debug agent communications and registries.

Grasshopper was favored in many of the surveys we consulted. Grasshopper is actively maintained and supported, where the current version was released July 2002, and is free for non-commercial use. Both FIPA and MASIF standards are supported, including ACL-formatted messaging. The message transport supports synchronous and asynchronous messaging, multicast, and alternate protocols (sockets, RMI, and IIOP). For security, SSL and X.509 certificates are supported, with the Java security manager protecting intra-host resources. There are several GUIs for development and debugging.

The Aglets platform was one of the first Java-based agent platforms. Main development ended in 1998, but the open-source community has maintained the Aglets platform, with patch release 2.0.2 released on 2/2002. The Aglets platform is MASIF compliant, using a custom message format based on Java serialization. Security is integrated into a custom design, using roles, proxies, and the Java security manager. The Tahiti GUI allows runtime debugging. The ample documentation includes a book.

For comparison, Cougaar is an agent platform with a focus on scalability and modularity. Cougaar is actively maintained and supported, where the current open-source release is version 11.0, released February 2004. The

message transport supports pluggable link protocols, including RMI, IIOP, SMTP, and UDP. Cougaar is not standards compliant, and messages are encoded using Java object serialization. Security is pluggable and highly flexible, featuring SSL, X.509 certificates, and role-based authentication. Agents can host HTTP-based Servlets to serve browser-based UIs and external GUI clients.

Cougaar's goal of high performance and scalability can be seen in a detailed comparison of these platforms. In JADE each agent has its own thread, and in Grasshopper there is a shared thread pool. Cougaar uses a thread pool with a powerful resource monitoring and allocation interface. Cougaar's message transport and many pieces of its infrastructure are asynchronous and have been optimized for maximum performance and extensively tested for scalability, as we present in our results section. Many of the design choices that were made to ensure scalability, performance, and robustness are quite different from those of current FIPA implementations, which have not (to our knowledge) been applied to an UltraLog-sized challenge problem. While performance drives Cougaar design, it seems less important for others (however, see [21]).

Another distinction between Cougaar and other platforms is the basic unit of development. In other platforms, developers subclass an agent class and provide message delivery and behavior callbacks, whereas in Cougaar, developers create one or more plugins that run within a standard agent and communicate through a blackboard. Cougaar's component-based architecture allows developers to incrementally add components, "wiring" them together to enhance the infrastructure. This design adds complexity to tutorials but provides greater flexibility and performance in more complex agent designs.

Cougaar's approach to UIs is also unique. The built-in support for HTTP-based Servlets simplifies the creation of custom agent-based UIs, configured as plugins within the agent. To ensure scalability, no single agent or UI has a complete picture of the agent society, but HTTP queries can be used to piece together partial or full views. Many low-level debugging and monitoring UIs that are present in other platforms are absent in Cougaar, primarily because these UIs do not meet the challenges of large-scale or rapidly communicating agent applications. The lack of such tools may dissuade new Cougaar developers, but is compensated for by more scalable tools that are crucial for large-scale application development.

Adoption of Cougaar has perhaps been slowed due to the lack of standards compliance, which is a prominent feature in many of the other agent platforms we surveyed. In part this was due to Cougaar's complex planning language, which does not easily fit into the ACL format, and Cougaar's research focus of a highly scalable and robust systems as opposed to interoperability. In future releases we plan to explore XML-based messaging, with a focus on web services interoperability, and ACL-level interoperability is a potential enhancement.

In our survey, we found that JADE is best suited to developers of relatively simple agent applications and developers that require FIPA compliance. JADE's debugging and monitoring tools, and FIPA's ACL message format, are ideal for low-volume agent interactions that require cross-platform interoperability. Grasshopper and Aglets share JADE's tool support and ease of development for simple applications. Cougaar is best suited to developers that want to customize the agent framework's core services or create complex, large-scale, robust, or highly secure agent-based applications.

5 Scalability & Survivability Results

The Cougaar architecture has been successfully used [1] to build our challenge application of an automated military logistics planning and execution system. As part of the UltraLog program, we have subjected our 1100+ agent application to a barrage of stresses and attacks, measuring the ability of the system to survive in chaotic environments.

Each year, the UltraLog program has followed a cycle of design, development, integration, engineering testing (system capability validation), and assessment (independent system function testing under stress). For example, in the 2003 program year, engineering test conducted over 190 test runs over the course of one month.

The assessment team has conducted numerous tests to measure the ability of UltraLog to meet its survivability goals (>80% capability, >70% performance) over a range of computer and network failures totaling 45% infrastructure loss. A key test is to kill various combinations of agents, at various points during system operation. Figure 2 shows the results from a set of these stress tests.

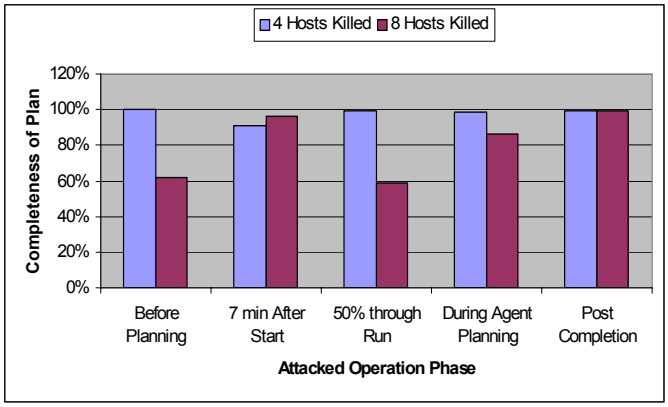

Figure 2. Sample 2002 UltraLog results.

In these runs, assessment killed two different combinations of agents at various points during the run. The first attack killed 15 selected agents (on 4 hosts). In the second attack 37 agents (8 hosts) were killed. Without any defenses, the UltraLog logistics application would be halted, so that agent kills earlier in the run would give increasingly worse completeness scores. Despite some anomalous results (second attack point), the chart shows relatively stable completeness, regardless of attack severity.

As seen in Figure 3, in 2003 we faired even better (the horizontal shaded area indicates acceptable performance, and the vertical line indicates 45% infrastructure loss). UltraLog has done well, although un-shaded data points below 45% indicate that additional defenses are required.

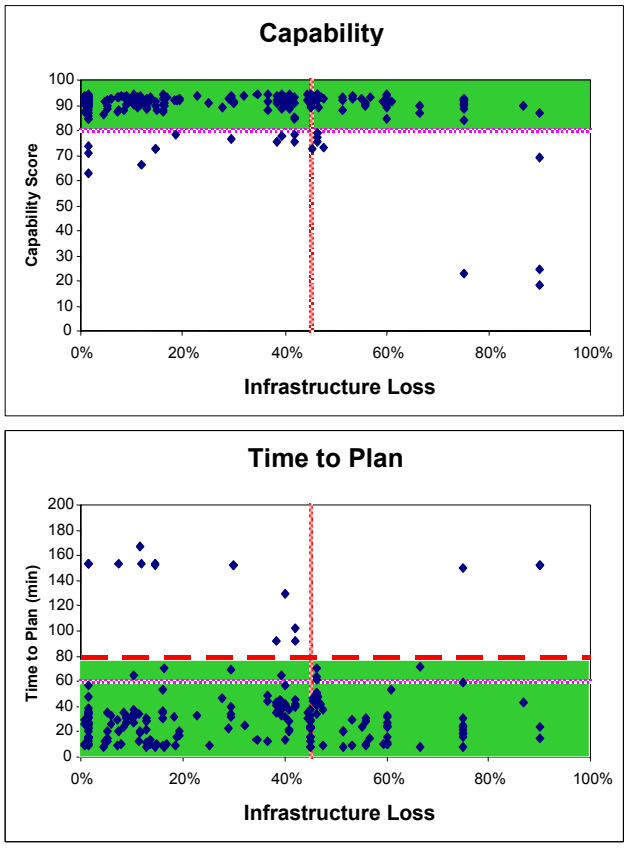

Figure 3. UltraLog survivability results, 2003.

Over the years, the assessment results have been collected and compared. As Figure 4 indicates, the system now copes with most categories of stresses (PASS>OK>FAIL). Together, these results indicate that the Cougaar architecture provides a measurably reliable, secure and survivable platform for application development.

6 Acceptance & Future Directions

This feature rich, well-stressed architecture has understandably been appealing to a range of users. Several users are expanding on the military logistics heritage of Cougaar and the UltraLog program, while others explore commercial and research avenues.

The US Army is planning to include Cougaar as a central design point in a new logistics decision support system. Another US government program built a toolkit for using service discovery and XML to dynamically reconfigure and execute disparate architectures. The US Army is also exploring using Cougaar to build a military maneuver decision support system. Others have used CougaarME, Cougaar tuned to small devices [14]. For example, one program used CougaarME to control semi-autonomous robots over a wireless ad-hoc network. Several commercial ventures use Cougaar as an integrating architecture. These include IT management software and a system that models critical infrastructure such as electrical grids for vulnerability analysis.

	2000	2001	2002	2003	Stress
Scalability	FAIL	OK	OK	PASS	Wartime loads1
	FAIL	FAIL	OK	OK	Wartime loads2
	FAIL	FAIL	OK	PASS	Wartime loads3
	FAIL	FAIL	OK	OK	Thrashing
	FAIL	OK	OK	OK	Scaling of nodes and agents
	FAIL	FAIL	OK	OK	Scaling logistics problem
Security	OK	PASS	PASS	PASS	Fraudulent, untrusted code
	FAIL	PASS	PASS	PASS	Untrusted communications
	FAIL	PASS	PASS	PASS	Insecure / dangerous code
	FAIL	FAIL	OK	PASS	Corruption of persisted state
	FAIL	OK	OK	OK	Unauthorized processing
	OK	OK	OK	PASS	Unexpected plugin behavior
	OK	PASS	PASS	PASS	Component masquerade
	FAIL	OK	OK	PASS	Compromised agents1
	FAIL	OK	OK	PASS	Compromised agents2
	FAIL	OK	OK	OK	Intrusion
	FAIL	FAIL	OK	OK	Compromised communications
	FAIL	FAIL	OK	OK	Snooping
	OK	PASS	PASS	PASS	Message intercept
Robustness	FAIL	OK	PASS	PASS	Processing failure1
	FAIL	OK	OK	PASS	Processing failure2
	FAIL	OK	PASS	PASS	Network failure1
	FAIL	OK	PASS	PASS	Network failure2
	FAIL	OK	OK	PASS	Processing contention
	FAIL	OK	OK	PASS	DOS attack

Figure 4. Annual UltraLog survivability improvements.

With such a broad user community, the open source development of Cougaar promises to move in several directions. Alternate message transport systems are likely, for example multicast and broadcast mechanisms, or XML-based, or standards-compliant transports (e.g., FIPA ACL, OWL). We anticipate newer revisions of CougaarME. Additionally, we expect to refactor Cougaar agents to facilitate alternate intra-agent services (i.e., blackboards or persistence mechanisms). These efforts will be driven by the lively Cougaar open source community, and supported by revisions of the tutorials, documentation, and debug tools [13] that have made Cougaar so popular.

The Cougaar agent architecture has spawned a lively open source community with a broad range of military, research and commercial users because of its flexibility, reliability, scalability, and broad support. It provides commercial grade robustness and security support, a highly flexible configuration mechanism, and proven scalability and survivability results. Because of these features, Cougaar promises to contribute strongly to the further adoption of agents for complex applications.

Acknowledgements

The work described here was performed under the US DARPA UltraLog contract #MDA972-01-C-0025. These ideas and results represent contributions by the many participants in the DARPA ALP and UltraLog programs, and in no way represent the opinions of DARPA.

References

[1] M. Brinn, and M. Greaves, "Leveraging Agent Properties to Assure Survivability of Distributed Multi-Agent Systems," *2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, 2003.

[2] Cougaar website, http://cougaar.org.

[3] P. Davidsson, and F. Wernstedt, "A multi-agent system architecture for coordination of just-in-time production and distribution." *Proceedings of SAC 2002*, Spain. pp. 294-299.

[4] UltraLog website, http://ultralog.net.

[5] A. Helsinger, R. Lazarus, W. Wright, and J. Zinky, "Tools and Techniques for Performance Measurement of Large Distributed Multiagent Systems," *2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Melbourne, 2003.

[6] L. Cable, "Extensible Runtime Containment and Server Protocol for JavaBeans Version 1.0", *JavaBeans Glasgow Specification*, December 3, 1998.

[7] M. Thome, "Managing Applications Comprised of Untrusted Components,", *JavaOne 2002*.

[8] Aspect Oriented Programming Tools for Practitioners, http://aosd.net/technology/practitioners.php.

[9] J. Berliner, M. Thome, and D. Cerys, "Multi-Resolutional Knowledge Representation Using Prototypes and Properties", *IEEE Conference on Knowledge-Intensive Multi-Agent Systems (KIMAS)*, Cambridge, 2003.

[10] M. Thome, "Multi-Tier Communication Abstractions for Distributed Multi-Agent Systems", *IEEE Conference on Knowledge-Intensive Multi-Agent Systems (KIMAS)*, Cambridge, 2003.

[11] UDDI Standards body website, http://uddi.org.

[12] T. Wright, "Naming Services in Multi-Agent Systems: a Design for Agent White Pages," *3rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, New York, 2004.

[13] Java Memory Profiler and Cougaar IDE Cougaar projects, http://cougaar.org.

[14] Cougaar Micro Edition project website, http://cougaar.org/projects/micro.

[15] UMBC AgentWeb website, http://agents.umbc.edu.

[16] G. Nguyen, T. Dang, L. Hluchy, Z. Balogh, M. Laclavik, and I. Budinska, "Agent Platform Evaluation and Comparison". Technical report for Pellucid 5FP IST-2001-34519. Jun 2002, Bratislava, Slovakia.

[17] J. Altmann, F. Gruber, L. Klug, W. Stockner, and E. Weippl, "Using mobile agents in real world: A survey and evaluation of agent platforms," *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS* at the 5th International Conference on Autonomous Agents, pages 33-39, Montreal, Canada, June 2001. ACM, ACM Press.

[18] JADE website, http://jade.cselt.it.

[19] Grasshopper website, http://www.grasshopper.de.

[20] Aglets website, http://aglets.sourceforge.net.

[21] G. Vitaglione, F. Quarta, and E. Cortese, "Scalability and Performance of JADE Message Transport System," *Proceedings of AAMAS Workshop on AgentCities*, Bologna, July 2002.