Extending the Limits of DMAS Survivability: The UltraLog Project

Marshall Brinn, Jeff Berliner, Aaron Helsinger, Todd Wright, *BBN Technologies*; **Mike Dyson**, *Schafer Corporation*; **Sue Rho**, *Cougaar Software*; **David Wells**, *Object Services and Consulting*

Assuring the survivability of a complex distributed multi-agent system (DMAS) in a stressful environment is an extremely difficult, yet important, challenge. More important, however, may be providing this survivability by a methodology that is applicable to general classes of DMAS.

The challenge of assured survivable DMAS

The stated challenge was survivability. The unstated, greater challenge was relevance.

In 2000, we finished developing a sophisticated large-scale complex distributed multi-agent system (DMAS) for developing military logistics plans. In order to transition this capability for use in the military, we knew it needed to be hardened to survive in stressful, even wartime, environments. Assuring and demonstrating the survivability of our DMAS was the stated goal of the DARPA UltraLog program.

This goal is challenging enough; despite their promise, DMAS are often unsurvivable. Their distributed nature fills them with potential 'soft spots' ripe for attack and failure; their autonomous nature results in chaotic emergent behavior that is difficult to predict, let alone control [2,8]. How can we make our complex DMAS survivable?

In addition, many organizations inside and out of the military have been reluctant to embrace DMAS approaches, despite the promises of reduced development and maintenance costs and improved performance, due to survivability concerns. To address this concern, DARPA challenged the UltraLog program to not only make our specific system survivable, but to achieve it in a way that could convince other potential customers that *their* application could be made survivable in a similar way. Can we develop a generic methodology for survivable DMAS? What can we claim about its applicability?

In this paper, we will describe our successful efforts in the UltraLog program to assure the survivability of our logistics application under stress. We will also describe a general methodology derived from our experiences, and discuss its applicability to other DMAS.

Survivability concepts and definitions

Before we describe the UltraLog survivability approach, application and architecture in detail, some definitions are in order. We assume that a DMAS presents some function, the qualities of which can be measured. These qualities may be along several dimensions, including performance (e.g., timeliness, completeness, correctness, or precision), usability (e.g., availability, responsiveness), and integrity (e.g., accountability, confidentiality). We call these qualities *measures of performance* or MOPs. For different applications and contexts, different MOPs will be crucial while others may be insignificant or meaningless [5].

The utility that an overall system provides is captured by some rollup utility function of these MOPs, a scalar representing how well the system is providing its aggregate system function. We call this the *multi-attribute utility* (MAU) or *system utility* of the application.

The environment in which the application operates provides many challenges that may undermine the ability of the application to maximize its system utility. Such challenges or *stresses* may fall under the category of *load* (significant work demanded of the system relative to its capabilities), *resource degradation* (attacks or failures having the effect of degrading the computation or network resources available to the system), or *information attacks* (directed attacks with the express purpose of degrading system utility, particularly in areas of integrity and confidentiality).

The *survivability* of a system is the degree to which system utility is maintained in the face of environmental stresses. A system is said to be *robust* if no permanent degradation of system utility is imposed by application of some stress. That is, a robust system will eventually restore an acceptable level of performance after the application of a stress. By *predictable*, we mean that a given system can provide a reliable expectation of the level of system utility that will be exhibited in the face of a given stress environment. *Assured survivability* characterizes a system that is both robust and predictable.

The UltraLog experience

UltraLog Context

The application whose survivability we have worked to assure in UltraLog performs planning and simulated execution of military logistics operations. The DMAS is composed of hundreds (537 in 2003, 1092 in 2004) of autonomous agents, each representing a combat or logistics organization. As operations plans are received by the system, demands for particular quantities of particular goods are computed by combat organizations and the logistics organizations determine how they can best satisfy this demand. The resulting logistics plan or TPFDD contained 250K individual plan elements representing demand and transportation for 34000+ entities of 200+ various types of major end items. The demand covered ammunition, fuel, and food; and the trans-

portation covered end-to-end movement schedules for personnel, unit equipment and ammunition resupply. Once the operation is planned, we simulate execution of the plan, with associated changes in requirements and deviations between expected and observed behaviors.

The original application was developed by a previous DARPA program, the Advanced Logistics Project (ALP), and is built in the Cougaar agent architecture. For more details on Cougaar, see "*Cougaar Essentials*".

[Sidebar 1 "Cougaar Essentials" about here.]

The UltraLog program has been a broad effort enlisting the contributions of 15-20 companies and universities, collaborating to make this application survivable. The component-based nature of Cougaar enabled a successful distributed development environment. The program managed a large test and integration facility containing hundreds of computers and VPN access in support of this large distributed team. For more details on the UltraLog program, see [4].

The stated goal of UltraLog is to "operate with up to 45% information infrastructure loss with no more than 20% capabilities loss or 30% performance loss in an environment characterized by wartime loads and directed adversary attack". This goal requires some clarifying definitions.

First, the goal characterizes the stress environment in which the application must be survivable:

- *Wartime Loads*. These represent large quantities and frequencies of tasks, plan perturbations and queries to which the system must respond.
- *Information Infrastructure Loss*. These represent a combination of loss/degradation of nodes (processing platforms) or of connectivity.
- *Directed Adversary Attacks*. These represent attempts to undermine system confidentiality or integrity by attempting to corrupt or read without authorization system code, messages, or internal or persisted state.

Second, the goal characterizes the aspects of system function that must survive:

- *Capabilities/Performance Loss*. This describes the degradation of system function as measured in terms of quality, completeness, correctness and timeliness of planning and execution functions. See "*UltraLog MOPS/MAU Overview*" for more details on our composed approach to quantifying system function.

[Sidebar 2 "MOPS/MAU" about here.]

Over the course of the UltraLog program, we iteratively identified and integrated features to improve system survivability. We will describe these in two categories, namely *survivability-enabling*, features to protect or recover processing and *survivability-enhancing*, features that improve exhibited system function in a given environment.

UltraLog survivability-enabling features

The early versions of UltraLog were simply not survivable. These versions lacked the basic ability to provide any system function under basic categories of kinetic and information attacks. Accordingly, we developed survivability enabling robustness and security defenses.

Robustness Management. The UltraLog approach to robustness consists of several pieces:

- *Persistence and Rehydration*. UltraLog agents support state persistence and rehydration. The infrastructure further supports reconciliation of restarting agents and pre-existing agents to allow agents to re-enter an existing society on the fly. The application code must participate in supporting these features.
- *Community-based Monitoring/Restart*. We established a service in which agent or node failure is detected and agents are restarted on the same or different platforms. This mechanism relies on a set of low-level infrastructure services supporting sets of agents forming a community that commonly promotes and protects particular attributes (liveliness, in this case).
- *Distributed Robust YP/WP*. UltraLog uses a white-pages and yellow-pages infrastructure to register and lookup agent and community services by name and attribute. These registries are distributed and redundant so that the service is robust to the loss of significant pieces of the underlying infrastructure.

Security management. In response to the scoped set of information attacks to which UltraLog will be subjected, we have established a set of security services that allow the system to adapt against constantly evolving cyber attacks by i) providing the administrators with a real-time understanding of the operational threat situation; ii) allowing automation of rapid reconfiguration of the existing countermeasures; and iii) providing the ability to deploy new countermeasures:

- *Secure Execution Environment*. We protect the integrity of loaded code and static data by means of a secure bootstrapping class loader, a Cougaar-specific Java Security Manager and a jar file verifier.
- *Certificate Authority*. We establish a hierarchy of redundant authorities to ensure trusted identification of users and agents within the system, including coordinated certificate update and revocation.
- *Preventative Measures*. We have implemented a series of protections against malicious information attacks, including user and component access control to UltraLog services, and encryption-based protection of messages, blackboards and persisted data.
- *Policy Management*. We use the KAOS policy management system [1] to establish fixed policy-based constraints on component behaviors, including what messages may be transmitted between which entities. Policies may be changed by users or by adaptivity rules indicating a change in defensive posture. To enforce these policies, we leverage the Cougaar component model to establish binders on the blackboard, messaging and user access.
- *Monitoring and Response Service*. We collect evidence of potential attacks, faults and errors, and take corrective action in response. Gathering Intrusion Detection Message Exchange Format (IDMEF) data from various components, we invoke rules to determine what response to take ranging from logging and alerts to adaptive policy changes including changing the encryption level.

Table 1 summarizes these survivability-enabling features.

[Sidebar 3 "Table 1" about here.]

UltraLog survivability-enhancing features

Having provided UltraLog the basic enabling features necessary, we focused in later iterations on raising the level of survivability possible under stress. Enhancing the measurable utility of UltraLog required adding generic features in the infrastructure as well as specific features in the logistics application.

Infrastructure enhancements.

- *Adaptivity Engine*. UltraLog encourages components to provide multiple implementations of the same interface, providing different trade-offs of resource consumption versus utility. These different implementations are called OpModes (operational modes) and are available for selection for real-time adaptivity. Each agent has an Adaptivity Engine that runs a set of adaptivity rules called Playbooks to support adaptive selection of the desired OpModes in particular environmental conditions [7].
- *Defense Coordinator*. UltraLog has a variety of defenses that sense system state and attempt to repair or to proactively defend the society, thus allowing a society to survive in the presence of a large number of physical and cyber threats. Unfortunately, a given situation may cause multiple symptoms, and these may in turn automatically trigger one or more defenses, which may cause conflicts due to resource limitations or destructive interaction. The *defense coordinator* oversees and orchestrates the process of correlating and organizing sets of defensive actions that may pertain in a given situation. The goals of the coordinator are to prevent undesirable combinations of actions, to make intelligent use of information about current status and expectations of future behavior when selecting actions, to cope with uncertainty in the form of conflicting or missing diagnoses and the potential for incorrect diagnoses, and to consider the importance placed on various survivability criteria (application completeness, timeliness, and security) when making resource tradeoffs in the process of action selection. To achieve the desired flexibility, the coordinator relies on models of asset types (e.g., agents, hosts, networks), assets, sensors, actuators, and threats from the environment [10].

Application enhancements.

- *Opportunistic Planning*. One example of an attribute enhancing system survivability is opportunism. In this context, opportunism means taking advantage of available resources to improve performance, but making do with whatever resources are available. An example of this is an approach we took in UltraLog logistics planning. We develop a quick plan (labeled 'level 2') that requires relatively little resources to compute and store. As resources are available, we compute a more detailed plan incrementally in the background. Whenever we need the plan, we can provide a complete plan with as much detail as possible (some mixture of level 2 and 6) without having to explicitly determine how good a plan we can support currently.
- *Enhanced Stability*. One area leading to considerable unstable and unpredictable behavior is the presence of 'ringing' or feedback loops among agents within the application. While these interactions may reflect actual interactions in the modeled domain, it is desirable to identify and eliminate these where possible. In certain cases, we were able to streamline protocols between agents to a single deterministic transaction allowing for reliable and more efficient behavior.
- *Enhanced Granularity*. We identified several agents whose processing was beginning to represent a memory and processing bottleneck. These bottlenecks reflected the centrality of these agents in the application workflow. By re-factoring the application-level roles and relationships, we were able to break up these agents into sub-agents and distribute their functions across other processing platforms, thus providing greater utility from the same hardware baseline.
- *Predictors*. The application provides many different predictors, which are components providing low-level models of downstream processing. Using these, the system can continue processing and providing some degree of function with 'rule of thumb' estimates where connectivity to other service-provider agents is temporarily unavailable.

Our integration and assessment process

UltraLog operated on an annual cycle of design, build, integrate and assess. Each year, we would take the outcome of the previous year's assessment and attempt to improve the survivability of the system based on the most critical vulnerabilities and inefficiencies found.

Critically, one cannot assess the survivability of an application without the ability to provide a stress environment of known characteristics and measure system utility in a reliable efficient manner. We built an elaborate test and evaluation framework and lab environment in which to perform a broad set of experiments to determine system utility in response to different stresses. See *"UltraLog Test/Assessment Environment"* for more details.

[Sidebar 4 "Test/Assessment" about here.]

Finally, we ran a formal assessment of the UltraLog prototype to determine its survivability as well as to identify areas where more 'armor' may be required. This assessment process has been conducted annually on the evolving prototype, allowing us to measure progress as well as determine areas of development focus for the subsequent year. The assessment consisted of three independent efforts:

- A 'red team' assessment to identify absolute vulnerabilities in areas of security and robustness,
- A functional assessment to determine how UltraLog functionality (accuracy, adequacy of planning, usability of interfaces) degraded under attack, and
- A quantitative assessment to determine levels of survivable system function under stress.

[Figure 4 "Robustness" about here.]

We collected baseline measurements for comparison using the UltraLog system in an unstressed environment. As stresses were applied, we measured each parameter and calculated a survivability score using the algebra described above. Figure 4 shows

results of over 200 experiments run in the early 2004 timeframe. Stresses spanned a range of computer and network failures where computers and nodes were destroyed, communications were cut, or bandwidth, CPU and memory were degraded.

[Figure 5 "Technical progress" about here.]

Figure 5 shows our annual technical progress under the UltraLog program towards our goal of a survivable DMAS under harsh environments of wartime loads and directed kinetic and information attacks. Detailed assessment results can be seen at [4].

A survivable DMAS methodology

From our experiences in UltraLog, we have abstracted our approaches and lessons-learned into a methodology for iteratively assessing and improving the survivability of DMAS. This methodology is illustrated in Figure 6. At a high level, it forms a simple iterative loop; namely, to identify the problem (environment and metrics), and iteratively identify improvements and assess their impact. As we will see, underlying the simplicity of this iterative loop is considerable complexity; each of the steps will be described in detail. But the iterative nature of the approach is essential. Our experience tells us that, in general, DMAS survivability cannot be engineered and assured in a single design pass. Rather, we support an approach that allows for incremental identification of inevitable shortcomings, and integration of potential corresponding enhancements.

[Figure 6 "DMAS methodology" about here.]

Further, we note that Figure 6 illustrates two higher-level iterative loops (in dotted arrows). The process of defining measurements is an iterative one, in that the process of capturing the appropriate high-level utility from low-level observables is often imprecise and requires repeated revisiting. Further, the process of defining the stress environment is, itself, an iterative process, as the ability to characterize the stresses, add new stress types or change expected frequencies and magnitudes of expected stresses requires revisiting the entire survivability strategy.

Define stress environment

We characterize the environment in which an application must operate in terms of the work the system must do and resources available to accomplish that work. Work to perform or *load* may be expressed in terms of frequency and magnitude of tasks to perform, queries to answer, etc. Resources represent available computation and communication capabilities either at a low-level (memory, bandwidth, CPU), or more specialized high-level requirements (server access, for example).

Stresses represent changes to the required workload or available resources relative to some assumed 'unstressed' baseline. For our purposes, we do not and need not distinguish among stresses whose causes are benign or due to adversary action (e.g. normal upsurges in load due to wartime or occasional hardware failure, versus denial-of-service motivated false-tasking or kinetic attack on network resources). From a survivability point of view, we need to understand how these changes in load and resource availability impact system function in the short and long term.

We must characterize the dimensions of load and resources to be considered. For the purposes of assuring survivability, we must be able to declare certain attacks, loads, and failures to be in scope and others out of scope. It is not feasible to expect survivability against arbitrary unanticipated (even unimagined) stress types. As new stress types are discovered, they should be added into the iterative loop for consideration.

Further, we must characterize the distribution frequencies and magnitudes of these stresses. In order for DMAS to be configured in a survivable manner, there must be some expectation of the bounds on these stresses so that hardware resources and redundancy can be deployed accordingly. No particular DMAS configuration can be expected to survive an arbitrarily large set of stresses (e.g., a stress that kills all its resources at once or an infinitely long queue of work to perform). The appropriate characterization of the stresses should be sufficient to support a full simulation of the stress environment, including instantaneous magnitudes and average bounds.

Define MOPs, MAUs, and goals

Perhaps the most important step towards assuring survivability is defining it properly. The nature of the function that we wish to protect in the face of stress must be adequately characterized so that we can assess our progress and the contribution of different enhancements towards that survivability. These definitions must be made with great care and revisited periodically, as they determine all future efforts to achieve survivability. This point deserves emphasis; only those attributes that are measured and contribute to the overall survivability score are considered relevant for the purposes of assuring or improving survivability. An attribute that is degraded or destroyed but has no measurable impact is of no importance to overall system survivability.

Measures of performance (MOPs) must be defined that characterize how well the system is doing in some dimension that leads directly to user utility. Beyond a description of the measure (e.g., the time to complete a particular task), a procedure for computing this measure from available observable data within the running DMAS must be detailed. This is often not a trivial task, as the observable data may not provide adequate insight into the intended behavior, and either the MOP or the system visibility may need to be adjusted to assure that the procedure actually captures the spirit of the MOP.

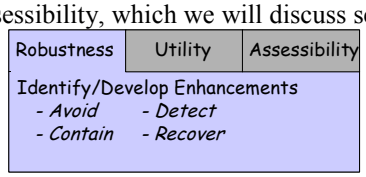
These individual agent-level MOPs need to be rolled up into a single metric for overall system utility. This must occur along several dimensions. First, the score must be aggregated across multiple agents to present a single value for the MOP over the whole DMAS. This may require weighting or synchronization among agents to achieve a meaningful combination. Second, a utility function must be ascribed to each MOP indicating how desirable/undesirable a given MOP value is within the space of possible values. These may be described in terms of absolute MOP values or relative to performance in some baseline environment. Third, these utility scores must be rolled up into a single scalar, requiring some weighted combination of the utilities provided by the different MOPs. Finally, some level of 'adequate' overall system performance must be established so that the system can be said to have 'survived' a given stress or not.

Here are some lessons learned from our experience in defining successful (and unsuccessful) MOPs:

- MOPs should be, as much as possible, measurable and visible to the system in real-time (as opposed to requiring off-line post-processing). In this way, real-time adaptivity can be supported.
- We highly recommend developing MOPs that capture 'macro' or aggregate behaviors as opposed to 'micro' or individual behaviors. The nature of DMAS is such that the processing of individual tasks or messages may vary dramatically within a run or, for a given task, from run to run. MOPs that attempt to capture how well a particular task was processed may suffer from non-determinism. Macro MOPs that capture the behavior of a processing *stream*, however, will tend to be stable and repeatable.
- In generating MAU curves, it is advisable to avoid 'stepwise' functions. In such cases, additional MOP value is generated for no additional utility, making small iterative improvements through a sensitivity-analysis approach difficult.

Identify/Develop enhancements

Having defined the problem, the next step in the methodology is to iterate through a development cycle: develop, test, enhance and repeat. In each iteration, we identify the most important factors requiring additional enhancements. The factors underlying the survivability of DMAS fall into the categories of Robustness, Utility, and Assessibility, which we will discuss separately.



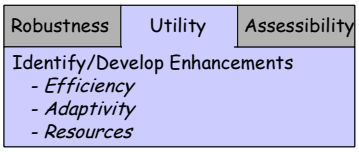
Robustness. The robustness of a given DMAS is limited by *vulnerabilities*. These represent 'holes' or failure modes that cause the system to fail to be robust; that is, to permanently lose some or all of its function due to application of a stress. Establishing the survivability of a system requires the development and deployment of a series of *defenses*. These defenses are responsible for maintaining an environment in which the application can operate under stress. They tend to fall into the following overall categories:

- *Avoid*: Take preventative action to prevent a given stress or attack from having any negative impact on system utility.
- *Detect*: Determine that a given stress or attack is underway.
- *Contain*: Dampen the impact or spread of the stress or attack across the DMAS society.
- *Recover*: Restore the computation environment to the application.

One essential attribute of a robust system, of course, is that it avoid any single-points-of-failure (SPOF's). To that end, a theme within UltraLog has been that there should be multiple ways of achieving a given goal, be it through multiple distributed implementations of a given capability, or different implementations of a similar capability. Our service discovery and leasing capability allows for continually finding live services without dependency on particular fixed servers. Further, our community structure allows for dynamic manipulation of configurations in a manner that is transparent to the rest of the society.

Asynchronous messaging, as in Cougaar, allows for reliable communications in the face of agent loss or periodic communications failures. We interpose predictors at time-critical communications points so that low-fidelity solutions are always available in a timely manner when more detailed answers may be temporarily unavailable.

We further note the importance of deconfliction as an enabling technology for robustness. The development of defenses is typically done with a focus on a particular threat or stress condition. The potential exists for simultaneous deployment of multiple uncoordinated defenses that conflict with one another to the point of undermining their effectiveness or opening up yet other vulnerabilities. Our experience with the defense coordination infrastructure indicates the value of limiting the strict autonomy of defenses by determining which defenses should fire in which circumstances.



Utility. By identifying opportunities to marginally improve utility, the overall system survivability may be raised. These enhancements include the following categories:

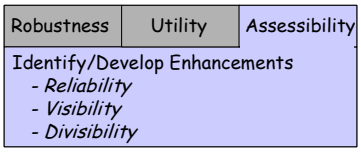
- *Efficiency*. Take better advantage of the resources available to provide measurable utility. Further, establish mechanisms to avoid wasted effort due to inter-defense conflicts, or mechanisms providing no contribution to a MOP.
- *Adaptivity*. Add new options for resource/utility trade-offs, and new capabilities to manage these trade-offs in different circumstances.
- *Resources*. Identify points wherein the application is hardware-bound and augment where possible.

The availability of many ways to do the same thing is a key enabler of adaptivity. Where possible, multiple implementations of the same interface (within the same component or multiple components) allows for different trade-offs of resources consumed for function produced in different circumstances. Of course, the infrastructure must be able to efficiently manage these multiple candidates for adaptivity, selecting and invoking the right mode in the right circumstance.

Increased granularity of the application is an additional adaptivity enhancer. This granularity may be in the dimension of agent configuration (breaking up the agent into smaller discrete pieces) or workflow (breaking up tasks into multiple sub-tasks). In either case, this enhanced granularity allows for more efficient utilization of resources by load-balancing and up-front configuration optimization.

One area of focus for utility enhancement is eliminating serial operations. The natural parallelism of DMAS should be leveraged as much as possible. Agents waiting for other agents to perform work in a serial workflow may pose significant bottlenecks. Another area in which waste can be eliminated is deconfliction. Unintended interactions among different adaptive operations may cause resource conflicts that degrade the overall system performance. Coordinated adaptive response and infrastructure-managed resource allocation can mitigate these effects. At the very least, it is imperative that the infrastructure assures non-destructive (if

not constructive) interactions among simultaneously operating entities.



Assessibility. We establish the following requirements for DMAS to allow their essential assessibility:

- *Reliability*. The system must support repeatability to allow for assessing the system function in a given environment over a long set of runs.
- *Visibility*. The system must provide defined and required MOPs and other key internal state.
- *Divisibility.* The system must allow for subdivision into subsystems and sub-problems for further analysis.

Reliability of DMAS is a known hard problem. The chaotic interactions of multiple queuing systems make reproducible predictable behavior difficult to achieve. We have attempted to address this in several ways. First, we attempt to measure macro-level MOPs, which tend to have greater stability. Second, we acknowledge the existence of low-level chaotic behaviors but attempt to minimize the impact of this chaos on overall system performance. We seek to eliminate component behaviors that act as amplifiers of small variability in the system into large fluctuations. Another source of non-deterministic behavior is timer-based batching of task flows, which we have attempted to limit.

In the area of visibility, we seek to assure the scalability of metric gathering by limiting the number and scope of aggregate (cross-agent) MOPs. Further, we seek to perform aggregation into macro MOPs (as noted above) prior to cross-agent aggregation. We make available a wide variety of low-level system performance and state metrics to all component levels in support of different forms of real-time adaptivity [2].

In the area of decomposition for scalability, we note that such substructures can and should exist in multiple dimensions. The application often presents multiple threads representing particular workflows or domain roles and relationships (e.g., the ammunition distribution thread within UltraLog). The infrastructure presents different threads of support services (e.g., the yellow pages or policy subsystems). Further, the component model of Cougaar imposes a natural heterarchical breakdown of the system into communities, nodes, agents and ultimately components.

Finally, we note that some problems and corresponding solutions may occur at the infrastructure layer, others at the application layer, and yet others may require coordination between these layers. Often an issue requires that the infrastructure establish a new capability and dictate requirements for the application to follow.

Integrate / Assess

Once a candidate enhancement or set of candidate enhancements to the application or infrastructure is available, these must be integrated into the full DMAS application, deployed to the distributed resource environment and assessed. To assure reliable performance, the assessment must allow for conducting multiple runs of the DMAS in an environment representative of the stress environment. In this environment, it must be possible to capture MOP and MAU rollups to assess overall system survivability in that environment. Additional logged information pertaining to failures, resource consumption or system performance needs to be captured to allow for post-analysis of the performance of the integrated system and allocation of credit/blame to subsystems and components. With these results in hand, we iterate back to identification of vulnerabilities and bottlenecks as described above.

Our experience with the ACME testing framework showed that the ability to perform multiple runs with the same parameters was invaluable in characterizing DMAS behaviors. We were able to refine our MOP definitions and measurement procedures by determining which behaviors were stable and reliable across many runs and which were not. Further, we were able to perform sensitivity analysis by modifying stress values and configuration parameters to measure corresponding survivability behaviors, thus pointing to areas for further analysis and likely performance enhancement.

Conclusion: Applying the methodology beyond UltraLog

Having described this methodology, we return to the original question of relevance. For what DMAS will this methodology work? A high-level review of the methodology may help illustrate the key attributes required of a DMAS in order for the methodology to be applied successfully.

Our methodology is, essentially, a DMAS realization of the standard debugging and tuning process, familiar to developers in single-threaded applications. This process depends on the ability to establish reliable *expectations*, compare those with system *observations* and *subdivide* the problem into smaller sub-problems for further analysis and repair of failure or inefficiency.

We have required these very attributes above as *reliability*, *visibility* and *divisibility*. Without some minimal level of these, such analysis is not possible. Above that, as DMAS exhibits greater reliability, visibility and divisibility, it can participate more readily and efficiently in this process. Requiring reliability, visibility and divisibility of DMAS is easier said than done, to be sure. But our experience in making UltraLog survivable shows that it can be done.

To be clear, all we claim is that such an application is able to participate in this process of iterative improvement. The ability to actually identify and implement solutions to shortcomings rests with the developers, and may further be bounded by attributes of the application, the hardware configuration or the stress environment.

Our future efforts under the remainder of UltraLog will focus on developing quantitative predictive models by which we can determine how survivable a given DMAS configuration can become in a given stress environment. In addition, we are exploring the use of logical models to identify vulnerabilities in robustness and security protocols that may lead to failure modes. Further, we have one final iteration of improvements and assessment still slated for our prototype, from which we expect to draw another round of lessons learned for broader DMAS survivability.

Finally, we note that a substantial amount of our survivability work was done at the Cougaar layer, leaving relatively well-defined tasks at the application layer to establish its survivability. We therefore recommend Cougaar and its UltraLog extensions

and tools as a solid platform on which to build applications and iterate towards enhanced survivability.

Acknowledgements

This work was sponsored by the Defense Advanced Research Projects Agency under contracts MDA972-01-C-0025, MDA972-01-C-0028, NBCHC010011 and N00174-02-D-0015. The work described here represents the efforts of the entire UltraLog team.

References

[1] Bradshaw, J., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkami, S., Lott, J. "KAOS Policy and Domain Services: Towards a Description Logic Approach to Policy Representation, Deconfliction and Enforcement", IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, 2003.

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

[3] Cougaar Web Site (http://www.cougaar.org)

[4] DARPA UltraLog Web Site (http://www.ultralog.net)

[5] Dietrich, S., Ryan, P. "The Survivability of Survivability". Fourth Information Survivability Workshop, (ISW-2001/2002).

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

[7] Kleinmann, K., Lazarus, R., Tomlinson, R., "An Infrastructure for Adaptive Control in Multi-Agent Systems", IEEE Conference on Knowledge-Intensive Multi-Agent Systems (KIMAS), Cambridge, 2003.

[8] Stafford, J, McGregor, J. "Issues in Predicting the Reliability of Composed Components". Submitted to 5th ICSE Workshop on Component-based Software Engineering.

[9] Ulvila, J., Gaffney, J. Boone, J., *A Preliminary Draft Multi-attribute Utility Analysis of Ultra*Log Metrics*. Vienna, VA: Decision Science Associates, Inc., 2001.

[10] Wells, D., Pazandak, P., Nodine M., Cassandra A., "Adaptive Defense Coordination for Multi-Agent Systems", submitted to 2004 IEEE Multi-Agent Security and Survivability Conference.

Sidebars

Sidebar 1

Cougaar Essentials

Cougaar, the Cognitive Agent Architecture, is an Open Source DMAS framework implemented in Java (see [3] for the source and the active user community). It has been developed under the DARPA Advanced Logistics (ALP) and UltraLog programs from 1998 to the present.

Cougaar is a modular infrastructure, in which entities are composed of cooperative sub-components. The essential autonomous entity is the *Agent*, which is composed of many sub-components including *Plugins*. Multiple agents co-reside on a single *Node* (Java VM), which manages common platform resources and message traffic. A fully running DMAS *Society* consists of many agents, typically organized in sub-societies or *communities* of agents.

Cougaar has been engineered for both scalability and flexibility, presenting a modular interface to support readily configuring and composing Cougaar systems with as many agents as desired, or as few services as required. The resulting architecture therefore supports running as many as 10,000 lightweight agents on a single host, or just one fully featured agent per host, with each component encapsulated by enclosing "Binder" components that monitor, control, or restrict access to system resources.

Components within an agent communicate via the blackboard, for efficient asynchronous data exchange when components require heavy interaction. Components within an agent may also share significant volumes of data, for which Cougaar supplies a novel logical data model that allows efficient representation of many items in memory for local processing. Figure 1 provides a high-level Cougaar component schematic.

[Figure 1 "Schematic" about here.]

Inter-agent messaging in Cougaar is asynchronous, and is handled by a full-featured quality-of-service enabled and componentized message transport service. This service supports a lightweight "gossip" service for inter-agent metrics exchange. Additionally, every agent may use an embedded Tomcat web server to provide lightweight servlet interfaces. Finally, agents locate each other using a DNS-style white-pages and locate services using a hierarchical yellow-pages directory service for dynamic relationships. Together these features provide a scalable, configurable, robust and transparent infrastructure for developing DMAS.

Sidebar 2

UltraLog MOPS/MAU Overview

The UltraLog program takes a unique approach to measuring system survivability. Most traditional information assurance techniques view the individual computer and network as the critical components requiring protection and thus, base their measures of performance on how much of a network survives or how much data on a computer is damaged. Typical security products are designed to thwart attacks by detecting and combating them before they can cause damage. In UltraLog, we assume that hardware components will fail, that aggressors can and will penetrate the network, damage or destroy computers, acquire passwords, and stress the system in malicious ways. UltraLog protects the logistics application function, meaning the vital information and critical processes necessary to conduct the business of logistics.

Measuring the logistics function required non-traditional measures of performance. For the massive logistics problem space in which we were operating, we encountered unique problems in determining the accuracy of a logistics plan of the magnitude of a major military operation. No plan had ever been constructed to that scale (180 days of deployment) or fidelity (detailed down to the individual box and truck), nor was there one to use as a baseline for comparison.

[Figure 2 "MAU rollup" about here.]

The solution was to build a hierarchical measurement framework based on multi-attribute utility (MAU) functions [9]. Using a panel of logistics experts, we built a survivability hierarchy from two attributes: capability and performance, as shown in Figure 2. Capability represents how complete, correct, and secure the logistics plan was. Performance represents the time it took to create or revise the plan and to present it to an operator. The functional expert panel further subdivided the hierarchy to account for attributes deemed essential by logistics experts, such as the completeness and correctness of supply and transportation tasks, and confidentiality and accountability. At the lowest level of the hierarchy, MAU curves were developed to establish the utility of each measurement to the logistics operator. Figure 3 shows a representative curve.

[Figure 3 "Sample MAU Curve" about here.]

The panel then considered tradeoffs between measures and assigned a swing weight to each to roll up into an overall score. By this process of expert-driven MOP definition and utility assignment, we were able to extract a measurable, objective definition from extremely subjective notions of defining and measuring system utility.

Sidebar 3

Table 1. Summary of UltraLog stresses, impacts and corresponding 'survivability-enabling' features

Stress/Attack	Impact	Defenses
Loss of nodes or agents	Loss of timely processing, loss of data associated with node and agents within node.	Automated fault detection, health checking, and recovery of state from a secure, persistent data store. Service discovery of alternative providers.
Loss of connectivity	Loss of timely, accurate detailed processing, inability to continue processing while inter-agent connectivity is down.	Messages are queued up to allow re-establishment of dialogs. Store-and-forward message-transport protocols. Predictors of downstream processing permit tentative solutions to continue to flow while waiting for downstream refinements.
Insertion of rogue software or agents into society	Corrupt integrity and degrade performance of system processing, allows for unauthorized access to data and services.	Trusted bootstrapping class loader and signed jar files assure that no Java classes are run without trust verification. All inter-agent communications are signed, and restricted by policy.
Corruption or loss of persisted state	Inability to restore state, or restoration of incorrect state on fault recovery, delays in processing.	State is stored in check-summed, encrypted format. If state is corrupted, the system can recover from no state.
Attempt to read or modify agent-internal data by unauthorized entities	Corruption of internal data integrity and correctness of processing.	All access to data and services are checked for proper permissions. Further, anomaly detectors may detect corruption and trigger the clearing and restarting of selected portions of processing.
Attempt to read or modify inter-agent messages by unauthorized entities	Compromise of application confidentiality; ability to snoop on details of internal processing.	Inter-agent messages are encrypted based on policy and posture. Messages are check-summed to assure no tampering of messages; corrupt messages are discarded and resent.
Attempt to read persistent state by unauthorized entities	Compromise of application confidentiality; ability to snoop on details of internal processing.	State is stored in a check-summed, encrypted format, limiting access.

Sidebar 4

UltraLog Test/Assessment Environment

Once the DMAS software components are developed and integrated, measuring the performance of a large system of distributed agents is a challenging task in itself. Our work in testing and evaluating systems of hundreds of agents on dozens of computers would not be possible without extensive automation support for running a large number of repeatable experiments in a controlled environment. This allows us to make a statistical evaluation of the experimental results. To this end, we developed the Automated Configuration Management Environment (ACME) [6].

ACME is a distributed agent system that runs in parallel to the UltraLog DMAS and is responsible for managing the computers and networks on which the UltraLog system runs. It manages the initialization of the UltraLog DMAS, applies survivability stresses, and collects measurements of the DMAS under test. Each computer used for UltraLog testing runs an ACME agent that is populated with Plugins that provide the services necessary for testing and measurement. These services include Cougaar agent initialization, log file management, and network bandwidth shaping. The ACME agents participating in a given experiment take their instructions from a single script, which defines the order of events in the experiment and issues the appropriate commands to ACME agents at the appropriate times. After each experiment, ACME passes the measurement and log file data to an automated analysis system called the Polaris Reporting Framework, which creates a set of reports and posts them to a central web site.

ACME agents and scripts are implemented using a lightweight language allowing testers, developers, and managers to all understand the experiment without the ambiguity common in large-system testing. Further, we have developed an UltraLog project "dashboard" continuously showing the status of all software components in terms of compilation status, code size, static code quality and design quality metrics (see http://pmd.sourceforge.net). Taken together, the UltraLog dashboard, ACME and Polaris give all the UltraLog participants visibility into the workings of the UltraLog project and the experimental DMAS.

Figures

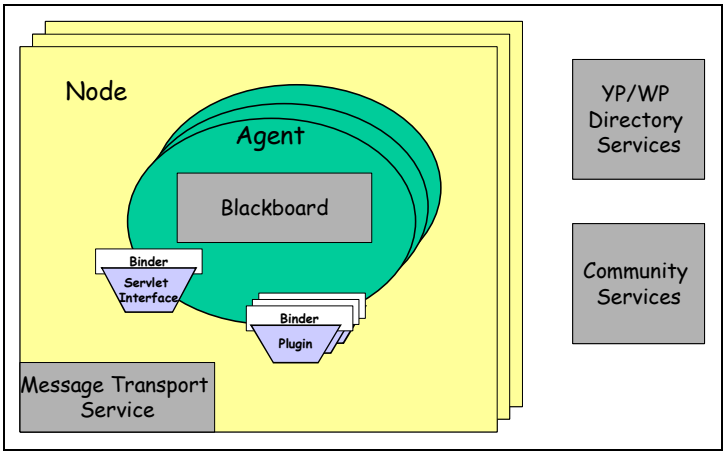


Figure 1. Cougaar components schematic

UltraLog Survivability	1.0		
MOP 1: Planning and Re-planning		0.41	
MOP 1-1: Completeness of Plan			0.41
MOP 1-2: Correctness of Plan			0.39
MOP 1-3: Completeness for Presentation			0.10
MOP 1-4: Correctness for Presentation			0.10
MOP 2: Confidentiality and Accountability		0.17	
MOP 2-1: Memory data available			0.16
MOP 2-2: Disk data available			0.16
MOP 2-3: Transmission data available			0.31
MOP 2-4: User actions counter to policy			0.21
MOP 2-5: User actions recorded			0.04
MOP 2-6: User violations recorded			0.12
MOP 3: Performance		0.42	
MOP 3-1: Time to compute plan or re-plan			0.80
MOP 3-2: Time to present			0.20

Figure 2. Summary of high-level UltraLog MOPs and associated swing weights

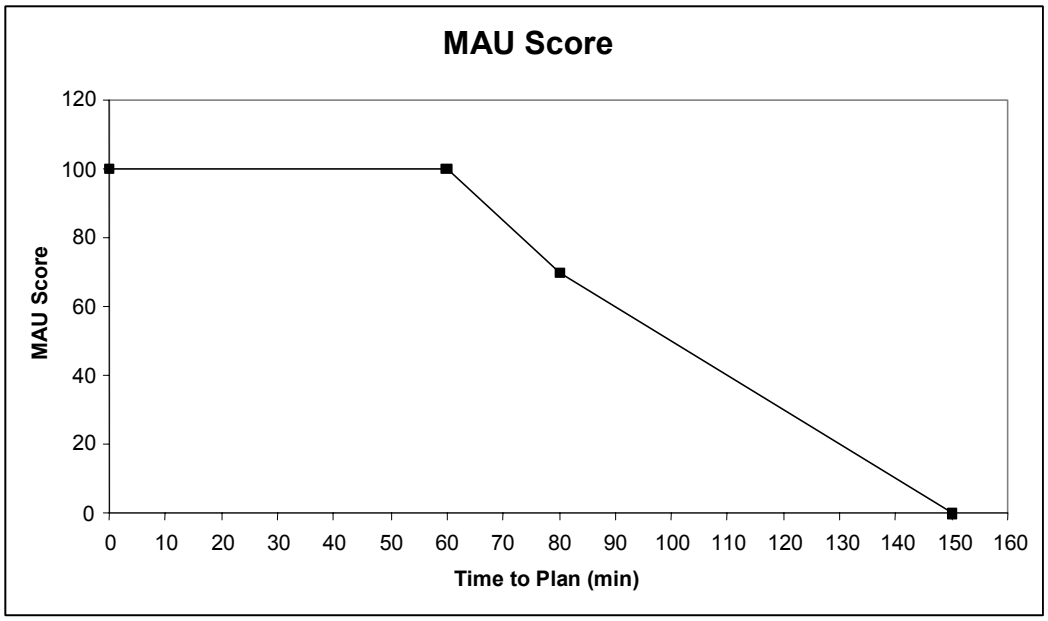


Figure 3. Sample MAU curve

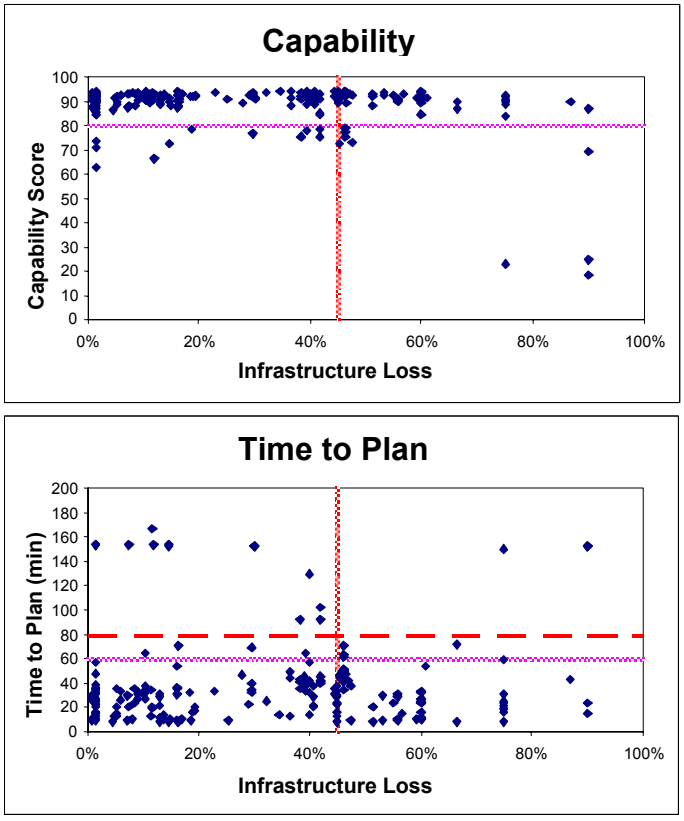


Figure 4. Robustness under attack: UltraLog demonstrates continuous performance (>80% capability, >70% performance) over a range of computer and network failures of over 45%.

	2000	2001	2002	2003	Stress	Technology Approach
Scalability	FAIL	OK	OK	PASS	Wartime loads1	Variable fidelity processing
	FAIL	FAIL	OK	OK	Wartime loads2	Load balancing
	FAIL	FAIL	OK	PASS	Wartime loads3	Dynamic reconfiguration and problem
	FAIL	FAIL	OK	OK	Thrashing	Stability guards, distributed control
	FAIL	OK	OK	OK	Scaling of nodes and	Distributed adaptivity engine
	FAIL	FAIL	OK	OK	Scaling logistics problem	Service discovery
Security	OK	PASS	PASS	PASS	Fraudulent, untrusted code	Secure Java class loader, signed class
	FAIL	PASS	PASS	PASS	Untrusted communications	Prohibition policies, PKI infrastructure
	FAIL	PASS	PASS	PASS	Insecure / dangerous	Java Security Manager, M&R subsystem
	FAIL	FAIL	OK	PASS	Corruption of persisted	Encryption, certificate infrastructure
	FAIL	OK	OK	OK	Unauthorized processing	Access control binders, policies, rovers,
	OK	OK	OK	PASS	Unexpected plugin	Tech spec binder protection
	OK	PASS	PASS	PASS	Component masquerade	Certificate-based authentication
	FAIL	OK	OK	PASS	Compromised agents1	Component isolation
	FAIL	OK	OK	PASS	Compromised agents2	Identity and certificate leasing
	FAIL	OK	OK	OK	Intrusion	Cert revocation, component isolation, M&R
	FAIL	FAIL	OK	OK	Compromised	Random routing and active port
	FAIL	FAIL	OK	OK	Snooping	Traffic masking, trust models, user
	OK	PASS	PASS	PASS	Message intercept	Encryption, authentication, PKI
Robustness	FAIL	OK	PASS	PASS	Processing failure1	Liveliness checking, persistence,
	FAIL	OK	OK	PASS	Processing failure2	Role-based restoration
	FAIL	OK	PASS	PASS	Network failure1	Predictors
	FAIL	OK	PASS	PASS	Network failure2	Multiple message transports
	FAIL	OK	OK	PASS	Processing contention	Thread/Resource management,
	FAIL	OK	OK	PASS	DOS attack	Compression, multiple communication

Figure 5. UltraLog technical progress

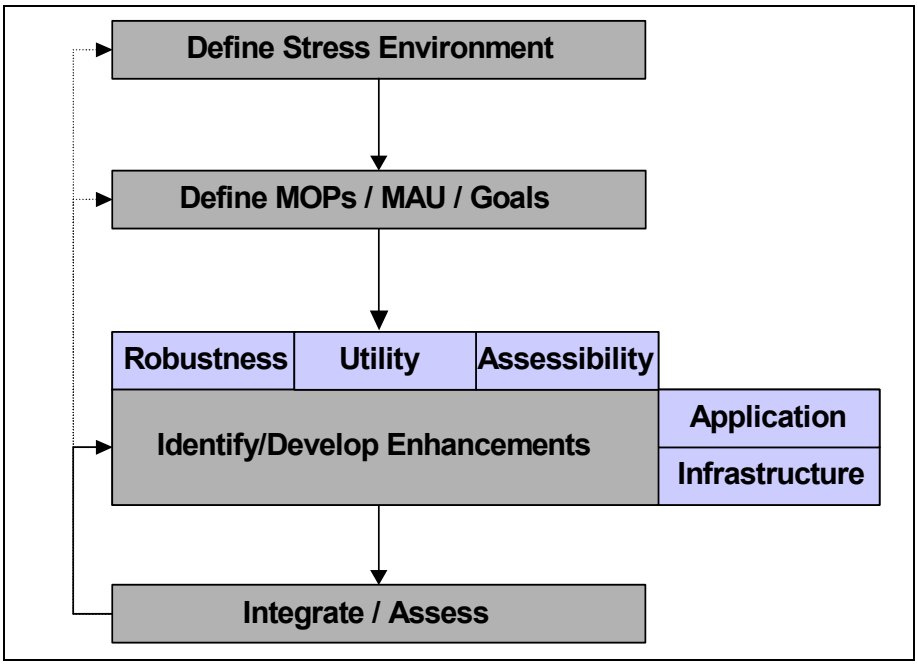
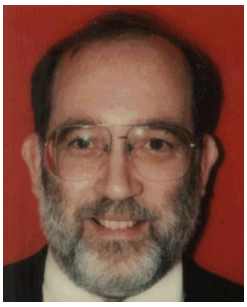


Figure 6. Iterative survivable DMAS methodology schematic

Authors

Marshall Brinn is is a Principal Engineer at BBN Technologies. He is the Survivability Architect on the DARPA UltraLog program and his key areas of interest are the control and modeling of emergent behaviors. He holds a B.S. and M.S. from Harvard in Applied Mathematics.



Jeff Berliner is a Division Scientist at BBN Technologies, where he is the Technical Director of the Distributed Systems and Logistics Department. His key areas of interest are modeling and predicting emergent behaviors of individuals and organizations, and in building applications which are useful in managing and controlling these behaviors. He holds a B.S. in Electrical Engineering from The Cooper Union and a M.S. and Ph.D. in Electrical Engineering from MIT.

Aaron Helsinger is an engineer at BBN Technologies, focusing for the past four years on developing the Cougaar agent architecture. His areas of current and past interest include management and configuration of complex systems, distributed system security, and data systems integration and scalability. He holds a B.A. in Physics from Harvard University.

Todd Wright is a Software Engineer in the Distributed Systems and Logistics department at BBN Technologies. He is a key infrastructure contributor to the Cougaar P2P multi-agent system and UltraLog distributed-logistics application. Mr. Wright holds M.S. and B.S. degrees in Computer Science from the University of Massachusetts, Amherst.

Mike Dyson is a program manager at Schafer Corporation. He currently supports development and execution of research projects within the Defense Advanced Research Projects Agency (DARPA). He is supporting the UltraLog program in the assessment and testing of survivable distributed agent software architectures for military logistics. Mr. Dyson holds a B.S. in Electrical Engineering from the University of Maryland.

Sue Rho is a director at Cougaar Software Inc. She has over 25 years in computer security, ranging from formal verification metholodogy to developing multi-level secure communication systems. Currently, she is a Principal Investigator for the Ultralog program, responsible for incorporating adaptive security into the Ultralog system. Her primary interest is in providing adaptive security for highly distributed systems.

David Wells is a founder of OBJS and was previously on the faculty of Southern Methodist University and was a Member of the Techical Staff at Texas Instruments. Wells has over twenty-five years of experience in Computer Science, having worked in the areas of cryptography, computer security, database systems, agents, and self-healing systems. He has received four U.S. patents, and was the architect of the Open Object-Oriented Database system. He earned his B.S. in applied mathematics, M.S. in Computer Science, and D.Eng. from the University of Wisconsin – Milwaukee.