

A Framework to Control Emergent Survivability of Multi Agent Systems

Aaron Helsinger, Karl Kleinmann, Marshall Brinn
BBN Technologies
10 Moulton Street
Cambridge, MA 02138 USA
{ahelsing, kkleinmann, mbrinn}@bbn.com

Abstract

As the science of multi-agent systems matures, many developers are looking to deploy mission critical applications on distributed multi-agent systems (DMAS). Due to their distributed nature, designing survivable resource constrained DMAS is a serious challenge. Fortunately, the intrinsic flexibility of DMAS allows them to shift resources at runtime between dimensions of functionality such as security, robustness, and the primary application. In this paper we present an algebra for computing overall survivability from these dimensions of success, and a control infrastructure that leverages these degrees of freedom to make run-time adaptations at multiple hierarchical levels to maximize overall system survivability. We have implemented this survivability control infrastructure on the Cougar agent architecture, and built a military logistics application that can survive in chaotic environments. Finally, we present results from assessing the performance of this application, and discuss the implications for future deployed DMAS.

1. Introduction

Distributed multi-agent systems (DMAS) cannot universally guarantee survivability in hostile, dynamic environments. The distributed, asynchronous nature of such systems makes situation assessment and response difficult to coordinate in a scalable, efficient manner. Moreover, the potential is great for conflict and instability among different agents independently seeking to contribute to survivability. Finally, we require *dynamic survivability*, so that the survivability posture and responses adapt to changes in requirements and desires as well as the threat and computation environments. What is required is a framework to embed survivability within the fiber of the DMAS itself, so that survivability emerges in a reliable, responsive, controllable manner from the system just as the application function does.

By *survivability*, we mean the extent to which application function is available in the face of stress. These *stresses* may take the form of extra loads (work to be performed) or degradation of computation infrastructure, and may come from directed adversary attacks, or result from particularly dynamic periods of operation (e.g. wartime imposes much higher requirements for processing than does peacetime).

1.1. Approach

Survivability can be treated as an auxiliary aspect of application function. The primary application function is composed of the operation of many independent components and agents, each operating autonomously to represent a portion of the desired functionality. Similarly, survivability is composed of the operation of many components and agents, each with a particular task or scope of operations. Our experience indicates that survivability, built in such a manner, can be an *emergent* yet *controllable* property of DMAS. By *emergent* in this context, we mean observable macro behaviors arising from the composition of controllable micro behaviors. This paper describes establishing a multi-tiered control framework between high-level observable metrics and low-level control actions that enables control of the emergent survivability of the overall system function.

Overall system function is determined by a series of measures of performance (MOPs), which are assumed to be observable in real-time. A well-behaved system should restore full function when a given stress is relieved. Equally important, however, is that the system should attempt to provide as much function as possible during the application of the stress. To that end, the system must affect tradeoffs among application functions and survivability capabilities to make best use of available resources. As the availability of resources changes, as the relative importance of MOPs changes, and as the perceived threat environment changes, the system should modify its actions and resource allocations accordingly.

Our DMAS are composed of distributed autonomous *agents*, interacting on a peer-to-peer basis. These agents are themselves composed of components. Multiple agents typically co-reside on a single platform in a *node*, which manages platform-level resource allocations. Further, agents typically are grouped into *communities* that manage higher-level function and interface to sets of cooperating agents. We call an overall application a *society* of agents, consisting, potentially, of multiple communities.

Our approach is to impose control elements at each of these tiers. Noting that “one man’s macro is another man’s micro”, we observe the behaviors at one level and seek to manage these behaviors by taking control actions at a lower level. We impose intermediate MOPs at each level so that we can measure the contribution of different components and actions to higher-level functions. Further, we apply manager agents that impose policies and coordination to keep emergent behaviors within reasonable bounds. In fact, the control we achieve in our approach can be characterized as bounding the emergent behavior, rather than tightly, explicitly controlling it.

1.2. Context

There is significant other research on control architectures, and on the chaos of distributed systems. The complexity of DMAS is well documented, and the futility of attempting to exert absolute central control over complex systems is well understood [9, 13, 14]. Our effort seeks to add some control to DMAS, without sacrificing the power of the emergent behaviors possible from DMAS. This compares reasonably with the holonic control architecture of HDCA, with however a greater emphasis on explicit reasoning about survivability [14]. Other approaches to self-controlling systems are described in [11]. The approach to survivability described here aims to achieve reasonable sustainable stable behavior rather than seeking global optimal behaviors in our DMAS. We assume the likely loss or compromise of system capabilities, and do not focus on building impenetrable defenses or six-sigma reliability and redundancy [16]. Instead, we rely on adaptivity in function and resource allocation, so that the system can ‘bend without breaking’, rather than using policy or constraint-based control [1]. We consider sufficiently complex systems such that they cannot be quantitatively modeled to predict performance or designed to avoid loss [4]. (For one attempt to model MAS, see [12].) Our approach treats the primary application as one player in survivability along with other system attributes, rather than treating these as independent problem layers [17]. For a theoretical discussion of assuring survivability in DMAS, see [2].

This paper describes a framework for measuring and controlling the emergent survivability of DMAS. Section 2 details our approach to measuring overall system survivability and weighting its dimensions. The key element of our approach is to implement a hierarchy of controllers as described in Section 3. In Section 4 we describe our experimental results from implementing this framework on the Cougar agent architecture [5], under the DARPA-sponsored UltraLog program [7]. Section 5 discusses implications of this approach to other DMAS applications. Section 6 discusses future work to be done in this area.

2. Metrics for survivability

There is a large class of applications for which there are multiple dimensions to the overall “success” or “survivability” of the application, and complete success in all areas is not possible given unpredictable threats, complex emergent system behavior and finite resources. But a DMAS that could measure its success in terms of these dimensions could instead achieve partial success in each of multiple dimensions, giving maximal survivability for minimal resources.

To this end, the UltraLog program has worked out a system of multi attribute utility (MAU) algebra for performing evaluations with multiple objectives. MAU models reflect explicitly the relative importance of attributes, and the degree to which system design impacts attributes [15]. Overall system performance is broken down iteratively into component dimensions, each given a normalized swing weight. At the leaves are the repeatable measures of performance (MOP). Each MOP is paired with an MAU curve, indicating the utility score given for a particular measurement on that MOP. For an example of this algebra, see section 4.1 on the UltraLog MOPs.

A system’s MOPs will tend to fall into several categories. Some MOPs will reflect the core application, while others will measure the supporting survivability aspects such as confidentiality, integrity, and availability. Some MOPs exist at a user level, reflecting the aggregate behavior of the society, while others measure some sub-system. By translating the attributes of survivability in UltraLog into measurable scalars, we allow the DMAS to reason about the dimensions of its own success at runtime.

3. The survivability control approach

Given this approach for measuring its own survivability, a DMAS needs a way to reason about and react to those numbers. This section highlights the principles for applying a hierarchy of control loops to guide survivability. At each level, an abstraction of the

MOPs is fed in, and the controller guides the operation of the lower level components towards the higher level goals.

Figure 1 looks at the survivability problem from a control perspective. The DMAS delivers the application function as its primary output (e.g., the plan in a planning application) to the users. They in turn may change MAU curves or application input at runtime, closing the loop.

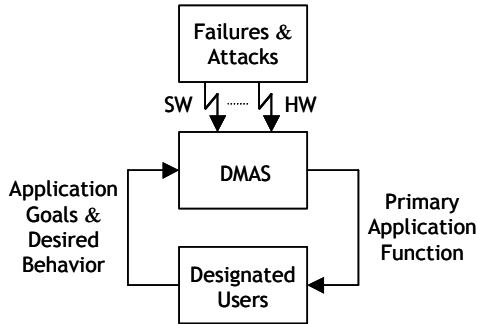


Figure 1. Control of DMAS survivability

During system operation a variety of hardware- and software-related disturbances affect the system. In addition to random failures that occur in every technical system, survivable systems face a broad class of intentional disruptions and intrusions (“attacks”) that are difficult to model and predict.

Resource degradation attacks can be handled like random failures (“repair if it’s worth it or perform your job as well as you can despite the disturbance”). The containment of information attacks (security) imposes an additional control dimension on the system, resulting in coupled control loops that cannot be independently optimized, assuming shared and constrained resources.

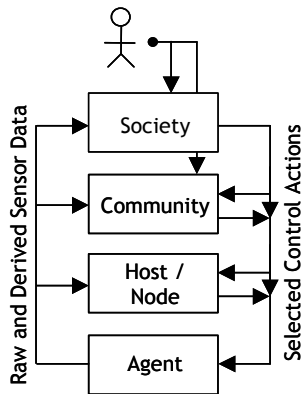


Figure 2. Hierarchy of control loops

A standard approach for the control of large complex systems is to set up a hierarchy of controllers, thereby cascading the lower level control actions. Figure 2 shows the principal levels for survivability control in a DMAS that correspond to the levels of resource contention where tradeoffs occur:

- *Agent-level control.* This level controls the operation

of individual agent sub-components. Whereas the controllers on all higher levels can be assumed to be agents, the form of resource contention and therefore the design of the controller within an agent depend on the specific agent framework used.

- *Host/Node-level control.* All agents on the same host share that host’s resources and are controlled by one distinct node controller, managing resource allocations among enclosed agents.
- *Community-level control.* Hosts on the same network partition share the same network resources, building a community (e.g., LAN) in which interactions and constraints are managed among the contained agents.
- *Society-level control.* The overall system is called a society of agents, consisting, potentially, of multiple communities on multiple networks. The society controller is maximizing the overall scoring function that integrates the various control dimensions.

Several general characteristics of a hierarchical control structure also apply to our DMAS case:

- Raw sensor data are condensed and propagated up.
- In connection with a slower sampling rate, controllers on higher levels can afford more planning and reasoning (vs. reacting).
- Control inputs (decisions) on higher levels utilize underlying capabilities of higher complexity, having a more global rather than local impact.

However, there are a number of specifics that make the survivability control of a DMAS especially challenging:

- The control infrastructure is part of the DMAS and therefore exposed to attacks, and requires additional logic, sensing, and replication.
- Users are also distributed and work at multiple levels of the system. The DMAS must therefore provide a means for mixed initiative control that allows users to override control inputs from higher levels.

The rest of the section will discuss the characteristics of control actions, sensors, and controllers over the various levels of this control hierarchy in more detail.

3.1. Control actions

Control actions that are selected, triggered, and parameterized by a controller are the capabilities that allow a system to dynamically respond to stresses. We note in particular those designed as explicit “defenses” against specific “attacks”. Defenses can be classified according to various attributes, such as:

- Targeted stress category (information attack; resource degradation attack; etc.).
- Complexity of defense (function primitives vs. sequence of actions).
- Causality of defense (proactive vs. reactive).

Table 1 categorizes a few prototypical defenses that are independent of the primary application function.

Table 1. Example defense classification

| Defense | Classification |
|---|--|
| <i>Reconstitution</i> (restarting dead agents from persistence data on a different host) | <ul style="list-style-type: none"> • Resource degradation (e.g., killed host) • Sequence of actions • Reactive |
| <i>Load Balancing</i> (moving agents between hosts) | <ul style="list-style-type: none"> • Increased workloads • Sequence of actions • Reactive |
| <i>Encryption</i> (of messages sent between agents) | <ul style="list-style-type: none"> • Information attack • Underlying function primitive • Proactive or reactive |

3.2. Sensor data

The control hierarchy requires the system to provide the following basic types of information by analyzing raw and derived sensor data:

- *Resource consumption.* This includes all data describing the status of low-level resources shared among the agents, such as CPU utilization, network throughput, and memory consumption.
- *Stress detection.* The defense mechanisms presented in the previous section are designed as a response to certain stress scenarios, requiring sensors and algorithms that can provide a diagnosis (e.g., tasks counts; aggregated message traffic data; health check).
- *System Performance.* This includes all data for the MOPs described earlier in section 2.

A detailed description of techniques for acquiring these sensor data within DMAS is given in [8].

3.3. Controller design

Since the control actions are the designated ways the system defends itself, the development of a survivable system typically starts with modeling attack scenarios and designing matching defenses. However, complex environments guarantee that this process never completes, and therefore the control infrastructure must provide both the hooks to integrate as well as the means to coordinate a large number of defense variations.

Complex, cross-host defenses are coordinated by the community level controllers, which include various responsibilities (Figure 3):

- *Diagnosis to disambiguate situations.* This addresses the problem that different sensors often deliver different and contradicting outputs. Therefore, intelligent algorithms based on decision theory need to

be leveraged in order to provide an unambiguous assessment of the situation.

- *Deconfliction of actions.* Due to the ongoing and independent development of defenses, there may exist redundant or even mutually incompatible defenses. The challenge for the controller is to select the optimum action leading to the best cost-benefit ratio, given the current MAU curves. Therefore, the defenses need to publish a model of their behavior (TechSpecs) that allows the controller to reason over the consequences of its selection.
- *Selection and Sequencing of actions.* There are instances in which multiple defenses are applicable and their combined execution desirable. For these cases the controller needs to invoke the defenses in a compatible order.

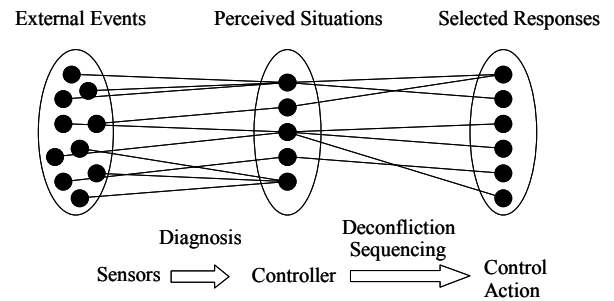


Figure 3. Community controller reasoning

The actual implementation of the controller may leverage any of a number of standard AI techniques [14]. Additionally, the necessary level of complexity varies by level in the hierarchy. Because of the independent development of the defense mechanisms, the functionality of a controller is typically distributed over several agents, which adds a significant complexity to the above task. Our implementation is described in section 4.

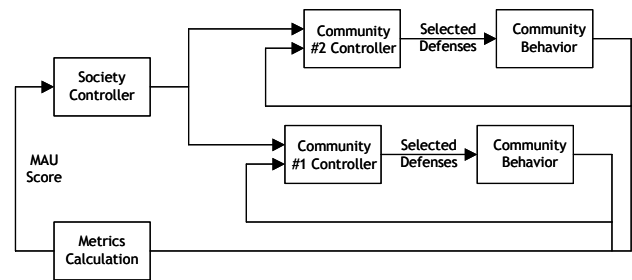


Figure 4. Society level control loop

Figure 4 shows an abstraction of the interaction between the community-level and the society controller. For space reasons, details of the node- and agent-level control loops are not described here, for which see [10].

The overall MAU score is calculated from derived sensor data collected across the society (cf. section 2) and provided as input to the society controller. This top-level

controller may then modify the behavior of the underlying communities, by modifying the cost-benefit functions used in the controller, the allowed range of actions, or the probabilities of various threat conditions. This closes the outermost loop, and allows the society as a whole to balance the various dimensions of system survivability.

These are generic design considerations for building defenses, sensors, and a control hierarchy that reasons about survivability metrics. The following section will show how UltraLog implemented this approach and improved the survivability performance of a large DMAS.

4. Experimental results

In the UltraLog program, we have built pieces of this control architecture, and a wide variety of defenses. We have then systematically stressed the system, and assessed its survivability, measuring the results using the MAU approach described above.

4.1. Implementation of the approach

UltraLog has extended the Cougaar open source Java agent architecture to support its requirements. Cougaar uses a flexible JavaBeans™-like component architecture, providing multiple control and access points. Additionally, Cougaar provides a robust persistence and recovery functionality, reliable efficient inter-agent message passing, and a powerful blackboard based intra-agent component communication and data sharing mechanism [5].

The UltraLog program is the culmination of an 8-year DARPA funded effort to explore the potential of DMAS for military logistics (expanding on other works, such as [6]). This application requires a complete model of the US military logistics supply chain to 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 [7]. To support such a significant application, UltraLog distributed over 500 medium weight agents across nearly 100 machines in 2003. This year the system comprises over 1000 logistics entities, plus control and management agents.

As described in section 2, UltraLog convened a panel of logistics experts to define appropriate measures of performance, and their relative contribution to the overall survivability of the system. Table 2 summarizes these MOPs, with the swing weights for each showing the percentage contribution of each to the overall survivability of the system (swing weight). The actual

score given the system for each MOP is calculated using an MAU curve.

Table 2. UltraLog MOPs and their weights

| <i>UltraLog Survivability</i> | <i>MOP Name</i> | <i>Swing Weight</i> |
|-------------------------------|----------------------------------|---------------------|
| MOP 3 | Timely Results | 0.42 |
| MOP 3-1-1 | Time to plan | 0.80 |
| MOP 3-1-3 | Time to present | 0.20 |
| Capability | | 0.58 |
| MOP 1 | Planning | 0.71 |
| MOP 1-1 | Completeness | 0.51 |
| MOP 1-2 | Correctness | 0.49 |
| MOP 2 | Confidentiality & Accountability | 0.29 |
| 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 ... | <Others> | 0.37 |

For example, figure 5 indicates that the score given the system for MOPs 3-1-1 and 3-1-3 is relative to the unstressed performance; if the system takes 1.5 times as long to present the completed plan under stress than in the baseline, the system is given a score of 70 for MOP 3-1-3. By then multiplying this score by the appropriate swing weight in Table 2 (0.20 for 3-1-3, then 0.42 for MOP 3), we find that this slower presentation time under stress adds only 5.88 to the final survivability score, instead of the maximum of 8.4. Put another way, presenting the final logistics plan 50% slower reduces the overall survivability score by 2.52. With this algebra, a DMAS can use knowledge of resource costs (stipulated or learned), to weigh the benefit of faster data display vs., for example, increased messaging encryption.

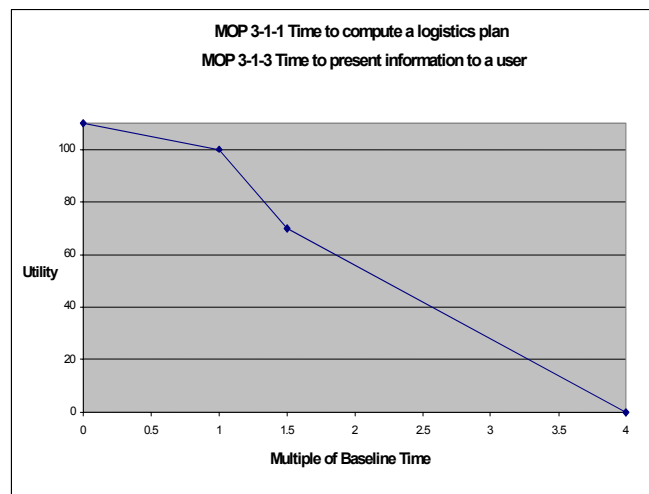


Figure 5. Sample UltraLog MAU curve

While it is impossible to anticipate the arbitrary stress types that a deployed system may face, it is possible to respond successfully to arbitrary combinations of known classes of stresses. To that end, UltraLog provides a library of defenses, and a control architecture that can selectively deploy and combine these defenses, allowing new survivable behaviors to emerge in response to new stress or attack conditions.

To support the hierarchical control required to achieve emergent survivability, we are implementing a control architecture following the principles in section 3.

- *Society*: At the society level, we intend to have a defense coordinator to propagate society-wide changes to expected threat conditions (changing situation diagnosis) and MAU curves. There is also a society-level security policy manager.
- *Community*: At the community level there are manager agents that control security (propagating policy, detecting security breaches, and coordinating security posture changes) and robustness (load balancing et al). These managers collect sensor data from hosts and agents and deploy community-wide defenses. We are currently adding a defense coordinator capable of adjusting to changing MAU curves, and doing better tradeoffs between security and robustness.
- *Node/ Agent*: At the agent level, Cougar provides an Adaptivity Engine, which executes a playbook of rules; given a specific set of agent-local sensor inputs, the engine sets operating modes for local components, and sends sensor data to the community manager.

Multiple organizations have added to the UltraLog system components that embody best practices of three “threads”: security, robustness (load balancing, restart, etc), and adaptive logistics (variable planning, predictors, etc.). While it is impossible to itemize all of the design decisions, features, components, preventative measures, redundancies, and defensive components that the team has built in UltraLog, an example will be illustrative.

As UltraLog agents each do unique portions of application function, they must be restored if they are lost (killed, cut-off, compromised, etc.). And as threats cannot be anticipated, UltraLog assumes that any combination of agents may be lost at any time. Recovering from this stress requires the intelligent interaction of multiple components implementing the approach from section 3:

- *Sensors*. We built various sensors to detect problems indicating a lost agent: secure messaging failures, network communications loss, machine loss, or planned network disconnects. For details, see [8].
- *Agent-level Control*. These agent level readings are sent via the agent Adaptivity Engine (which may take short-term actions, like increasing measurement sensitivity), and then relayed to the community manager agents. For details, see [10].

- *Community-level Control*. A defense deconfliction engine at the robustness manager agents (the predecessor to the coordinator described above) uses Partially-Observable Markov Decision Processes [3] to diagnose the issue, and then a cost/benefit calculator to select the appropriate response.
- *Defense Actions*. Available responses include moving agents (in case of network problems), shifting load (in case of overloaded hosts), switching to store-and-forward messaging (i.e. email, if the disconnect is planned), or restarting a dead agent.
- *Coordinating Actions*. Restarting the agent requires re-issuing crypto keys (coordinating with the security manager), and then decrypting previously stored agent state persistence data.
- *Infrastructure Support*. The Cougar architecture then performs state reconciliation among the agents.

This UltraLog implementation overview is necessarily both brief and incomplete. However, this indicates the framework in which UltraLog deploys defenses, and the hierarchical control structure that coordinates them. The results of using this structure have been quite positive.

4.2. Assessment results

Each year the UltraLog program has followed a cycle of design, development, integration, engineering testing (component testing by the integrator), and then assessment (independent system function tests under stress). For example, in the 2003 program year, engineering had 243 planned test cases, and conducted over 190 test runs over the course of 2 months.

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% of infrastructure. A key test is to kill various combinations of agents, at various points during system operation. Figure 6 shows the results from a set of these stress tests.

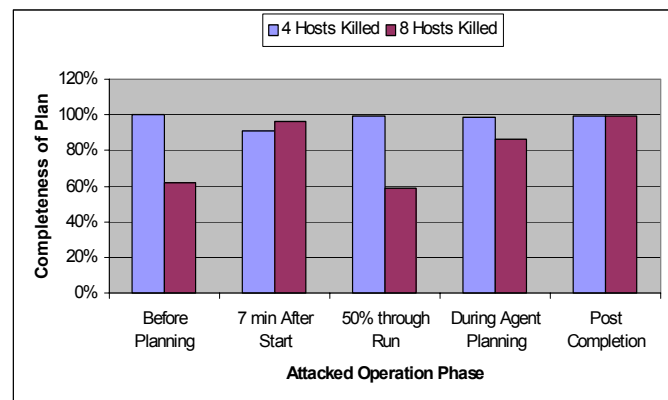


Figure 6. Sample 2002 UltraLog results

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 (MOP 1-1) scores with our approach, regardless of attack severity.

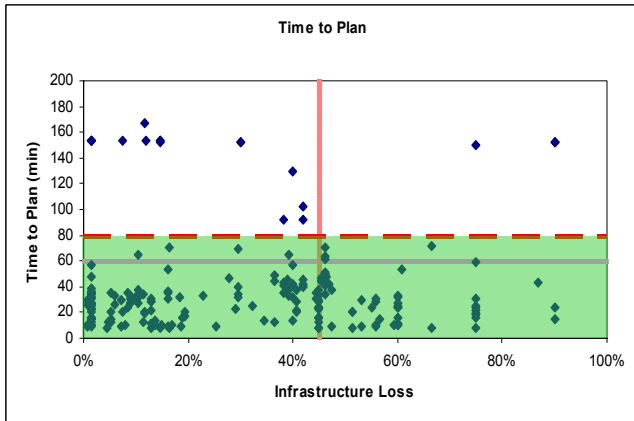
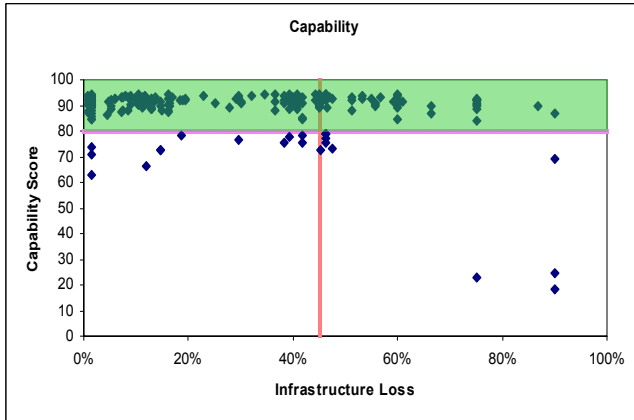


Figure 7. 2003 UltraLog survivability results

Learning from the 2002 results, in 2003 we fared even better. Figure 7 summarizes the 2003 assessment scores (the horizontal shaded area indicates acceptable performance, and the vertical line indicates 45% infrastructure loss). As the figure indicates, UltraLog has done quite well, although data points below 45% loss that are not shaded indicate additional defenses are required.

Over the years, the assessment results have been collected and compared. As Figure 8 indicates, the system now copes with most categories of stresses. Together these results indicate that UltraLog’s hierarchy of control loops managing layered defenses has produced emergent assessable survivability.

| | 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 | OK | 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 8. Annual survivability improvements

5. Interpretation of the approach

While the benefits of this approach to the survivability of the UltraLog application can be measured experimentally, an assessment of its potential benefits to the survivability of general DMAS requires different sorts of evidence. However, UltraLog is one example of a large class of potential applications. Applications for which one-dimensional six-sigma survivability is not appropriate, and those that can instead express overall success in terms of multiple dimensions, may potentially use this framework as a whole. More specifically, many of the elements of our approach are applicable elsewhere, assuming certain features of the primary application.

Metrics. While the specific UltraLog MOPs are unique to the application, the approach of separate MOPs rolled-up via scoring functions and weights to a single utility function is general in nature. The ability of the system to provide MOPs that are measurable in real-time is critical to the ability of the system to recognize the need to act and to respond in real time to changing loads and risks. The establishment of intermediate MOPs at component, agent, host and community levels allows for mapping progress at one layer to the next and assessing progress and reacting in a reasonable fashion.

Control Actions. The extent to which an application provides multiple responses to achieve a given goal enhances the framework’s availability to adapt efficiency to different circumstances. The presence of a policy framework in which constraints can be propagated

throughout the infrastructure and policed helps assure that the bounds of a reasonable performance envelope is maintained and/or restored as much as possible.

Mechanisms. The hierarchical approach to resource allocation and control is well suited to granular distributed systems. Small, granular processing units (agents, components) allow for a variety of agile control responses that larger, monolithic units do not. The agent Adaptivity Engine mechanism can modify operating modes for intra-agent processing to the extent that these modes are provided by the application. The defense coordination engine provides a generic mechanism for managing high-level diagnoses and alternative responses to external potential and perceived situations.

6. Future work and conclusions

Current implementation efforts in UltraLog seek to complete a defense coordinator to reason about current MOP scores, and then intelligently deploy defenses across communities to maximize survivability. We will also work on models of component behavior and resource consumption. By defining the essential, steady-state behavior of components in terms of their constraints, inputs/outputs, and performance specifications, we expect to model aggregate system behavior by means of Monte-Carlo simulation. In this way, we expect to be able to develop constructive algorithms for building configurations to maintain a given level of survivability in a given threat environment. This will enable us to continue formalizing the argument that the approach described here produces survivable DMAS for particular classes of applications and in particular circumstances.

We have laid out a general approach for dynamically improving overall system survivability for the large class of applications that have finite resources, multiple dimensions of success, and a premium on overall success rather than maximizing success in any one dimension. Assessment of a prototype implementation indicates reliable controlled emergent survivability. A hierarchical control architecture that reasons about an MAU algebra of survivability allows DMAS to achieve cost effective survivability for complex systems by controlling the emergent survivability of those systems.

7. Acknowledgements

The work described here was performed under the DARPA UltraLog contract #MDA972-01-C-0025. These ideas represent contributions by the many individuals who participated in the DARPA ALP and UltraLog programs.

8. References

- [1] Aggarwal, V., Gautam, N. "Fluid models for evaluating threshold-based control policies for survivability of a distributed network." http://www.ie.psu.edu/faculty/gautam/papers/vineet_ierc.pdf
- [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] Cassandra, A., "The POMDP Page", <http://www.cassandra.org/pomdp/>
- [4] Chen, S., Gorton, I., Liu, A., Liu, Y. "Performance Prediction of COTS Component-based Enterprise Applications", AAMAS, '03.
- [5] Cougaar Web Site (<http://www.cougaar.org>)
- [6] Davidsson, P. and Wernstedt, F., "A multi-agent system architecture for coordination of just-in-time production and distribution." Proceedings of SAC 2002, Spain. pp. 294-299.
- [7] DARPA UltraLog Web Site (<http://www.ultralog.net>)
- [8] 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.
- [9] Klein, M. and Bar-Yam, Y., "Handling Emergent Dysfunctions in Open Peer-to-Peer Systems," ROMA Working Paper ROMA-WP-2001-03, Massachusetts Institute of Technology, 2001. <http://necsi.org>
- [10] 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.
- [11] Kokar, M., Baclawski, K., Eracar, Y., "Control Theory-Based Foundations of Self-Controlling Software," IEEE Intelligent Systems, May/June, 1999.
- [12] Lerman, K. and Galstyan, A., "A General Methodology for Mathematical Analysis of Multi-Agent Systems", USC Information Sciences Technical Report ISI-TR-529, 2001.
- [13] Odrey, N. G. and Mejía, G., "A re-configurable multiagent system architecture for error recovery in production systems." Robotics and Computer Integrated Manufacturing, Vol. 19, 2003, pp. 35-43.
- [14] Tianfield, H, Tian, J, Yao, X, "On the Architectures of Complex Multi-Agent Systems", Knowledge Grid and Grid Intelligence Workshop at 2003 IEEE/WIC Conference on Web Intelligence/Intelligent Agent Technology, Canada, pp. 195-206.
- [15] Ulvila, J. Gaffney, J. and Boone, J. A preliminary draft multiattribute utility analysis of UltraLog metrics. Vienna, VA: Decision Science Associates, 2001.
- [16] Yurcik, W. "Adaptive Multi-Layer Network Survivability", Workshop on Countering Cyber-Terrorism, 1999.
- [17] Yurcik, W., Doss, D., Kruse, H. "Survivability-Over-Security: Providing Whole System Assurance". Information Survivability Workshop 2000, Boston, MA. <http://www.cert.org/research/isw/isw2000/papers/40.pdf>