2004 Command and Control Research and Technology Symposium
The Power of Information Age Concepts and Technologies

An Environment for Comparing Command and Control Architectures

Dr. John James and MAJ Fernando Maymi

Corresponding Author: John James
Department of Electrical Engineering and Computer Science, USMA
Thayer Hall, Building 601, Room 1107
West Point, NY 10996
Phone: 845 938-5563
FAX: 845 938-5956
John-James@usma.edu

Abstract:

As the Services move to transform joint and coalition operations, a new capability being contemplated for the transformed forces is synchronizing manned-combat-vehicle and unmanned-combat-vehicle target engagements. However, we are just beginning to work out the details for implementing such a symphony of coordinated human and machine decisions and actions. One challenge in realizing an implementation is the selection of command and control architectural components and their relationships that will provide the precision and flexibility needed for joint and coalition warfare. We describe an experiment in building an environment for comparing command and control architectures. The experiment involves extending the One Semi-Automated Forces (OneSAF) simulation environment to support analysis of alternative architectures for integration of control of autonomous combat vehicles with control of manned combat vehicles. The autonomous combat vehicle being simulated is the Loitering Attack Missile (LAM) which is being considered as a supporting indirect fire weapon for the Future Combat System (FCS).

Background

As with current operations, future Joint Operations will require availability a range of direct-fire and indirect-fire (non-line-of-sight) weapons to engage enemy targets. One such weapon system being considered is the non-line-of-sight Launch System (NLOS-LS) with associated indirect-fire missiles: the Precision Attack Missile (PAM) and Loitering Attack Missile (LAM) [1]. The NLOS-LS set of systems is expected to be a key component of a suite of indirect fire weapons available to future commanders [2].

An active area of review, analysis and research has been in the consideration of the impact of the ongoing revolution in information systems to command and control systems [3, 4, 5, 6]. The notion that “information age” warfare is in some sense “network centric” implies a capability to share information across networks to dynamically “understand” the state of the battlespace better than your opponents (information dominance) and dynamically alter plans based on that understanding (i.e. “get inside the decision cycle” of opposing force commanders).

Fundamental to these considerations for future command and control systems is the issue of system architecture comparison and selection. This paper will provide an overview of an architecture comparison environment being implemented for an undergraduate project in control system architecture comparison. There have been many efforts in software architecture comparison over the past ten years [7, 8, 9]. The cadet work will focus on design and implementation of airspace deconfliction algorithms. For that area, they will compare three different missile control architectures: centralized control, semiautonomous control, and autonomous control. They will also "brainstorm" Information Assurance (IA) issues. In order for the cadet deconfliction work to be done, the missiles, targets, and obstacles have to be identified and the simulation dynamics provided.

*** This work was partially supported by an endowment establishing the Adam Chair in Information Technology. The views expressed herein are those of the authors and do not purport to reflect the position of the United States Military Academy, the Department of the Army, or the Department of Defense.**

Currently, we have the of the OneSAF Testbed Baseline (OTB) simulation environment as modified by the U. S. Army Communications Enectronics Command (CECOM) under a Defense Advanced Research Projects Agency (DARPA) project. The DARPA project has resulted in the capability to control simulated Future Combat System (FCS) unmanned ground vehicles. Faculty will modify the CECOM interface for ground vehicles to enable providing the data needed by the cadet project. However, the dynamics for the targets, obstacles, and missiles will also have to be added. The plan is to have faculty modify a solution available from the Mathworks that implements some prior work in controlling a swarm of missiles to engage a set of targets [10].

Partitioning of system components

The cadet project will compare different command and control architectures so a high-level task is to decide upon an approach for architecture comparison. A fundamental issue in comparing command and control architectures is the relative effectiveness of the architecture in supporting development and execution of a commander's *concept of the operation*. For the cadet project we will not explore the semantics of commander's *intent* in the framework of a *concept of the operation* but constrain the problem at hand to issues surrounding implementing alternatives for centralized, semi-autonomous and autonomous engagement of targets by loitering missiles. Such a problem requires close attention to the details of communicating time-sensitive information among architecture components. Such time-sensitive information includes: updated target lists, updated target prioritizations, changes in the defense condition, and changes in the rules of engagement. The cadet project Software Requirements Specification (SRS) states that:

> There are three architectures that must be examined for use in the system: Centrally controlled, man-in-the-loop semi-autonomous, and autonomous. The centrally controlled architecture has a ground-based controller giving commands to the missiles for everything they do. The semi-autonomous architecture allows a controller to input commands to missiles, however missiles will operate on their own without additional commands. Fully autonomous operation occurs when the ground-based controller selects the autonomous mode or the missile has lost communication with the controller. The pros and cons of each architecture must be determined and weighed in the implementation of the system.

An architecture comparison approach that has been used in the past [7] has been provided to the cadet team. Also, iterative architecture development through a spiral process of "build a little, test a little" requires an architecture comparison methodology to indicate directions for improvement. The remainder of this section discuses an initial methodology proposed by researchers at the Software Engineering Institute and suggests changes which make the approach appropriate for comparing time-sensitive architectures.

The Software Architecture Analysis Method (SAAM) [11] has been proposed as a methodology for comparing alternative software architectures. The steps proposed in SAAM are:

1. Characterize a canonical functional partitioning for the domain.
2. Map the functional partitioning onto the architecture's structural decomposition.

3. Choose a set of quality attributes with which to assess the architecture.
4. Choose a set of concrete tasks that test the desired quality attributes.
5. Evaluate the degree to which each architecture provides support for each task.

However, while SAAM provides a methodology for architecture comparison, it must be modified for use in evaluating distributed, real-time architectures. Specifically, SAAM is incomplete for comparing alternative distributed, real-time architectures. The incompleteness occurs in two areas: (1) explicit consideration of communication between architectural components is not discussed and is fundamental to distributed, real-time architectures since communications links in an application architecture may vary over time between zero bandwidth and essentially infinite bandwidth, and (2) distributed, real-time processes contain many feedback loops which result in: (a) a need to analyze a set of components to determine the next state of the set of components (i.e. it is not correct to analyze a component in isolation) and (b) the notion of letting a set of components "settle out" over a period of time before the next set of input values are processed (i.e. the idea of a time constant associated with a process).

Concerning the first SAAM incompleteness issue, communication can often be assumed to *not* be an issue, especially whenever the architecture under consideration will be implemented such that communication between modules is almost instantaneous. Even in this case, communication between modules probably should be accounted for at the reference architecture level. However, for architectures involving large distributed systems, analyzing communications processes between modules is necessary and will normally involve at least a fixed delay (latency) of messages at the simplest level and, for complex systems, may require use of specialized tools to record or simulate actual message preparation, transmission, propagation, receiving, and processing activities. Certainly for our domain of interest, distributed real-time systems, communication is an integral member of the problem space and must be explicitly considered. Establishing communication between modules should be a step in the architecture development process, equal with partitioning the problem space and assigning functional modules to a structure.

Concerning the second SAAM incompleteness issue, the canonical functional partitioning will normally result in components whose internal state depends only on the previous state and current inputs. The component independence assumption is true most of the time for those components supporting higher-level decisions leading to engagement events, especially force operations decisions which set the environment for use of deadly force. However, the component independence assumption is almost never true for modeling lower-level physical processes, such as aircraft and missile guidance control, sensor control, and control of engagement processes, all of which are integral processes of the distributed, real-time problem space. Stated another way, for military applications, the failure of the independence assumption for distributed, real-time components arises from the fact that the distributed nature of motion in the battlespace (e.g. ships, missiles, aircraft, tanks, helicopters, troops, …) means that very high-level decisions can result in producing constraints which dramatically change the operational environment for low-level components. The low-level components then quickly produce different outputs which change the state of the higher-level components inside their decision cycle (i.e. the

component independence assumption is invalid because we have a mixed-signal, or hybrid, problem space).

Distributed, Real-time Architecture Comparison Requirements:

While functional segmentation is a natural approach to follow in construction of software modules (since implemented functionality of software process models and data schema can be directly related to user functional requirements), the functional partitioning of components may not be the best approach for architecture development. An architectural comparison approach is thus required. The relative ability of alternative software, hardware and communications architectures to react to expected failure modes will be determined by the detailed partitioning of required operations into functional modules, the mapping of resulting distributed software processes onto the distributed computation and communication resources, and the execution of combined system functionality across components which may be widely distributed in space and time.

A Real Time Software Architecture Analysis Method (RT - SAAM):

An approach for comparing alternative distributed, real-time software architectures is:

1. Build a set of software architectures for the distributed, real-time problem space by repeatedly:
   a.1 Identify a level above which system behavior is to be determined by modifying logical parameters only and partition the problem space (tasks) into appropriate higher-level functional modules using event-based models,
   a.2. Below the level identified in step a1, partition the problem space (tasks) into functional modules, some strictly event-based models, some a mixture of event-based models and differential-algebraic-equation-based models.
   b. Assign modules to a computational structure (usually pipe and filter computational style), and
   c. Establish communication between modules.
2. Choose a set of quality attributes with which to assess the architectures (pick success criteria),
3. Choose a set of concrete tasks which test the desired quality attributes, and
4. Evaluate the degree to which each architecture provides support for each task.

For the cadet problem, the simulation environment, the One Semi-Automated Forces (OneSAF) Test Bed (OTB), handles the details of the physics-based modeling. Thus, the cadet team implementation must explore the details of step 1a, 2, 3, and 4 of RT-SAAM.

Simulation system and interface between components

We are using the eXtensible Markup Language (XML) as an interface definition language for the interface between the OneSAF simulation environment and the student simulation components. An example of messages to the missiles is given in Figure 1. The syntax of the messages is given in the data type definition of Figure 2

```
<?xml version="1.0" ?>
<!DOCTYPE messages (View Source for full doctype...)>
essages>
   <!-- Message from missile network to simulation environment.
        Missile m1: vioalte physical laws and reposition as indicated
        Missile m2: set a new temporary waypoint to avoid a collision
        Missile m3: set a new (permanent) loiter pattern  -->
essage command="PUT" messageId="100">
ssile command="godHand" missileId="m1">
ocation lat="22311" lon="56478" alt="350" />
elocity north="112" east="0" down="0" />
</missile>
ssile command="setWaypoint" missileId="m2">
vaypoint waypointId="0" lat="12345" lon="12345" alt="250" />
</missile>
ssile command="setWaypoint" missileId="m3">
vaypoint waypointId="1" lat="13345" lon="13345" alt="250" />
vaypoint waypointId="2" lat="14345" lon="13345" alt="250" />
vaypoint waypointId="3" lat="14345" lon="14345" alt="250" />
vaypoint waypointId="4" lat="13345" lon="14345" alt="250" />
</missile>
</message>
   <!-- Message from missile network to simulation environment.
   Missile m3: launch  -->
essage command="PUT" messageId="101">
nissile command="launch" missileId="m3" />
</message>
   <!-- Message from missile network to simulation environment.
   Get all update messages for all missiles.    -->
nessage command="GET" messageId="101" />
   <!-- Message from missile network to simulation environment.
   Get all update messages for missile m2.     -->
essage command="GET" messageId="102">
nissile missileId="m2" />
</message>
   <!-- Message from missile network to simulation environment.
   Abort (detonate) missile m2.  -->
essage command="PUT" messageId="103">
nissile command="abort" missileId="m2" />
</message>
   <!-- Message from simulation environment to missile network.
   Update message for missile m3.      -->
iessage messageId="644" command="get">
issile missileId="abc">
ocation lat="13345" lon="13345" alt="250" />
elocity north="112" east="0" down="0" />
</missile>
</message>
  </messages>
```

Figure1. Example messages to missiles

```
<?xml version="1.0" ?>
<!ELEMENT messages (message+)>
<!ELEMENT message (missile*)>
        <!ATTLIST message command (put|get) "get" >
        <!ATTLIST message messageId CDATA #REQUIRED >
        <!ATTLIST message timestamp CDATA #IMPLIED>
<!ELEMENT missile (location?, velocity?, waypoint*)>
        <!ATTLIST missile command (abort|godHand|launch|setWaypoint)
#IMPLIED>
        <!ATTLIST missile missileId CDATA #REQUIRED >
        <!ATTLIST missile timestamp CDATA #REQUIRED>
<!ELEMENT location EMPTY>
        <!ATTLIST location lat CDATA #REQUIRED>
        <!ATTLIST location lon CDATA #REQUIRED>
        <!ATTLIST location alt CDATA #REQUIRED>
<!ELEMENT velocity EMPTY>
        <!ATTLIST velocity north CDATA #REQUIRED>
        <!ATTLIST velocity east CDATA #REQUIRED>
        <!ATTLIST velocity down CDATA #REQUIRED>
<!ELEMENT waypoint EMPTY>
        <!ATTLIST waypoint waypointId CDATA #REQUIRED>
        <!ATTLIST waypoint lat CDATA #REQUIRED>
        <!ATTLIST waypoint lon CDATA #REQUIRED>
        <!ATTLIST waypoint alt CDATA #REQUIRED>
```

Figure 2. Document Type Definition for the missile interface

Summary

As discussed above, we are building a simulation environment to experiment with the comparison of command and control architectures. In particular, a cadet team is investigating the relative efficacy of autonomous, semi-autonomous, and centralized control of a next-generation autonomous combat vehicle. However, we expect that the simulation framework we are creating will also be useful for experimenting with a wide range of issues surrounding the interface of autonomous and man-in-the-loop decision support systems.

In addition, we expect the simulation environment to support faculty research into other areas of interest. Concerning possible faculty/other research issues, it should be noted that the OneSAF software is expected to provide event-based simulations of operational scenarios and the Matlab/Simulink software is expected to provide continuous time and space simulations as well as the link between event-based and continuous simulations. A rich environment of the system state will thus be available. In this environment, the only

invariant is expected to be the *commander's intent* with all other system parameters being subject to change during the duration of the simulation. The set of issues associated with NLOS-LS networked communications (such as QoS, bandwidth allocation, trustworthiness of system state parameters, ...) is an area of research that is essentially open-ended. Another area that has been investigated for many years without resolution is the fusion of network sensor data (such as target identification, target update, obstacle identification, obstacle update, ...).

REFERENCES:

1. An overview of the NLOS-LS is found in the Army RDT&E Budget Item Justification, http://www.dtic.mil/descriptivesum/Y2004/Army/0604645A.pdf .
2. A visualization of the NLOS-LS operational concept http://www.gordon.army.mil/symposium/2002/2002pri/briefings/DCD-TSM/JTRS/FA%20Briefing.pdf
3. Network-Centric Warfare: Its Origin and Future.,Vice Admiral Arthur K. Cebrowski, U.S. Navy, and John J. Garstka, JCS J-6, January 1998.
4. Network Centric Warfare Report to Congress., July 2001
5. Understanding Information Age Warfare, Davis S. Alberts, John J. Gartska, Richard E. Hayes, and David A. Signori, ISBN: 1-893723-04-06, 2001
6. Network Centric Warfare, Davis S. Alberts, John J. Gartska, and Frederick P. Stein, 2nd edition, ISBN: 1-57906-019-6, 2000.
7. James, J. R. and R. McClain, "Tools and Techniques for Evaluating Control Architectures", proceedings of the 1999 IEEE Conference on Computer-Aided Control System Design, Kohala, Hawaii, August 1999.
8. Medvidovic, Nenad and Richard N. Taylor "A Classification and Comparison Framework for Software Architecture Description Languages"
9. Mary Shaw and David Garlan "Tutorial Slides on Software Architecture" August 1997.
10. http://www.mathworks.com/company/digest/may03/modeling.shtml
11. Kazman, R, L. Bassm G. Aboud, and M. Webb "SAAM: A Method for Analyzing the Properties of Software Architectures", 1995.