Proceeding of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003, NARA, Japan, May 19-23, 2003

Integrated Synthesis, Analysis and Implementation of Model-based Distributed Spacecraft Control Systems

David P. Watson, Patrick A. Stadter, PhD, George R. Barrett, PhD
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road, Laurel, Maryland, USA 20723
{firstname.lastname}@jhuapl.edu

Keywords Discrete Event Systems, Model-based Programming, Formation Flying.

Abstract

Distributed spacecraft formations are becoming increasingly important to future NASA and DoD missions involving temporally correlated sensing requirements. As communications and crosslink navigation technology continues to evolve, the limiting factor in deploying large scale distributed spacecraft formations will become the availability of robust, validated autonomous controllers. It is clear that current approaches to spacecraft operation will not scale in complexity or cost as formation systems grow to tens or even hundreds of individual platforms. Instead, global autonomous coordination and local control will allow the formation to be managed as a single entity. The challenge to realizing this promise lies in managing the complexity of required spacecraft coordination and the ability to feasibly implement the component systems. The described research involves developments in autonomous command and control that leverage formal methods of discrete event systems and model-based control to generate, implement, test, and validate such systems.

Discrete Event Systems (DES)[1] provide a formal means to model and control highly complex systems at multiple levels of abstraction. DES methods can implement, for example, the coordination required to share health and safety information among multiple spacecraft and ground systems through the appropriate use of crosslink, uplink, and downlink communication resources. The formal structure of DES has enabled the development of control concepts analogous to results in conventional control theory, including formal definitions of stability, controllability, and observability and distributed control synthesis.

Model-based programming and execution[2] have emerged as important paradigms for developing robust autonomous controllers that can respond to both planned and anomalous mission events. Model-based programming languages provide constructs for explicit representation of preemption, parallel and sequential composition, and hierarchical construction. All system interaction is described in terms of state variables that map to a declarative plant model containing specifications of component behavior modes, logical constraints on those modes, and feasible transitions between modes.

This paper will describe model-based execution as an implementation framework for DES and illustrate the concept with an example in autonomous distributed spacecraft control that integrates high level controller synthesis, analysis, and implementation. In the example, an overarching coordination policy will be synthesized from a global specification of desired behavior for a group of spacecraft.[3] This coordination policy can be expressed in terms of executable control models that are used by the sequencing and deductive control components of a model-based executive. This formulation provides continuity across the full lifecycle of spacecraft systems engineering, from conceptual design through autonomy requirements development and into programming and runtime execution, using a consistent theme of model-based analysis and implementation.

1. Introduction

Advances in system miniaturization have resulted in sensor designs that improve the feasibility of deploying large-scale, distributed sensing assets. The need for this capability lies in the promising advantages that distributing sensing platforms have over monolithic system deployments. Broadly stated, for the cost of system complexity, distributed systems promise improved robustness, graceful degradation, and increased capability to accomplish complex sensing tasks relative to single-element systems. Such advantages can only be exploited through system designs that coordinate the operations and resources of a distributed system to achieve unified science or military goals.

Many terrestrial examples of coordinated distributed systems exist, including such commonplace applications as automated manufacturing plants and large-scale retailer shipping networks. The ability to apply this systems capability to disbursed space sensors, however, presents unique challenges because of the limited accessibility after deployment and the need for robust, autonomous operations in an environment naturally hostile to components. Both military and civilian space systems are poised to benefit from developing distributed spacecraft system technologies. Military applications include data fusion systems that use multiple, coordinated kill vehicles for interdicting ballistic missile threats. This system implementation would incorporate multiple, disparate sensors to provide distinct views of threat complex objects. Intelligently fusing the diverse sensor data from multiple kill vehicles requires a robust, real-time means to autonomously coordinate vehicle actions, collect and associate data, and fuse data into information for decision making and interdiction process control.[4]

The National Aeronautics and Space Administration (NASA) has also identified distributed spacecraft systems as a vital element in future space and Earth science missions. Whereas previous NASA missions have relied on extensive ground-based commanding for single, complex spacecraft, future mission concepts will take advantage of advances in autonomy that facilitate coordination needed for distributing capability among multiple platforms. Two essential applications that require autonomous distributed spacecraft systems are co-observation, in which multiple spacecraft observe a single target from many perspectives, and multipoint observation, in which multiple spacecraft sample a region of space simultaneously.

The envisioned future military and civilian space applications that use coordinate distributed systems to achieve mission goals must weigh the consequences of traditional ground-based or relay-based coordination versus autonomous, on-board system control. While this trade is currently considered for single spacecraft systems (e.g., ground-based navigation versus autonomous, space-based Global Positioning System (GPS) orbit determination), its importance is magnified by the complexity of managing interactions among spacecraft in distributed systems. Geometric considerations for ground-based or space relay-based coordination typically lead to limitations in terms of line-of-sight contact to all elements in a constellation; this can impact science operations when commanding is delayed. The accuracy of essential measurements, such as relative navigation solutions, can also be impacted by the method employed for system coordination. Autonomous, on-board command and control techniques that incorporate navigation capabilities for distributed spacecraft systems provide real-time measurement updates to closed-loop orbit determination solutions.[5]

2. Discrete Event, Model-based Systems

A promising means of coordinating distributed spacecraft systems lies in the application of discrete event systems (DES), in conjunction with model based reasoning (MBR), to implement distributed, autonomous command and control functions. DES use formal means to model and control highly complex systems at multiple levels of abstraction.[6] DES methods can implement, for example, the coordination required to share health and safety information among

multiple spacecraft and ground stations through the appropriate use of crosslink, uplink, and downlink communication resources. More complex control operations, such as maintaining relative position and attitude among a cluster of spacecraft for co-observation can also take advantage of DES techniques. *Of fundamental importance is the fact that DES offer a formal means to synthesize the finite state implementations for spacecraft autonomy that are the developing trend in space system design.*

The remainder of this paper describes an end-to-end approach to autonomous command and control of distributed spacecraft systems. The approach spans from the theoretical formalization and synthesis of coordinated control concepts using DES, to an implementation framework based on model-based programming tools. The following sections provide an introduction to DES and MBR concepts, a description of our model-based programming and reactive execution framework, and an example of coordination policy synthesis and implementation for a simple distributed spacecraft application. The final section summarizes the work and describes on-going research efforts.

3. DES Command and Control

Conventional control techniques focus on meeting requirements in terms of typical parameters, such as response time, stability, and robustness, however the increasing complexity of systems has necessitated strategies that address augmented requirement sets. These augmented requirements include dynamic system behavior adaptation, coordination of autonomous system assets, goal planning and refinement, fault resolution, and learning.[6]

DES are dynamic systems that can be represented by a discrete state space and a state transition structure. The state space consists of all possible configurations that the system can assume, and the state transition structure defines the mechanisms by which a system evolves within its state space. The events of a DES capture physical occurrences that drive state transitions; these discrete events are typically assumed to be instantaneous, asynchronous, and nondeterministic.

Discrete Event System Fundamentals

Effective representation of DES has been achieved through the use of finite state automata (FSA) and associated formal language theory constructs.[7] While there is no theoretical limitation necessitating a finite state representation, most practical implementations do not require a state set of infinite cardinality. Using an FSA representation the DES is formally modeled as a 5-tuple:

$$R = (Q, \Sigma, \delta, q_0, Q_m) \text{ where} \quad (1)$$

Q is a set of states, possibly infinite,
Σ is a finite set of events, called the alphabet,
δ is a partial mapping from $Q \times \Sigma \to Q$, denoting state transitions,
q_0 is the starting state, and
$Q_m \subseteq Q$ is a set of marked states.

Denote by Σ^* the set of all finite strings of elements from Σ, including the empty sting ε. Using the notation $\delta(q,s)!$ to indicate that $\delta(q,s)$ is defined, the transition function is inductively extended to strings as $\delta(\varepsilon, q) = q$ and $\delta(s\sigma, q) = \delta(\sigma, \delta(s,q))$, whenever $q' = \delta(s,q)!$ and $\delta(\sigma, q')!$. Any $K \subseteq \Sigma^*$ is a *language*, and $\overline{K} = \{ s \in \Sigma^* \mid \exists q \in \Sigma^* \ni sq \in K \}$ is the *prefix closure* of K. Therefore the language associated with R is

$$L(R) = \{ \sigma \mid \sigma \in \Sigma^* \text{ and } \delta(q_0, \sigma)! \} \quad (2)$$

And the language marked by R is

$$L_m(R) = \{ \sigma \mid \sigma \in L(R) \text{ and } \delta(q_0, \sigma) \in Q_m \} \quad (3)$$

This formal model of a DES operates by starting in the initial state and executing state transitions as defined by δ and given a finite sequence of events from the alphabet Σ. If the DES is in a marked state after executing a string of events, the DES *recognizes* the string; $L_m(R)$ therefore represents the set of all

finite length strings that are recognized by R. Qualitatively, marked states distinguish those strings that have significance to the operation of the DES, such as task completion or goal achievement.[1]

Control of a DES R can be realized by the developing a supervisor $T = (T_a, \gamma)$ that consists of an FSA, T_a, augmented by an appropriate state feedback map γ. This requires an alternative view of the DES R in Eq. (1). Specifically, in addition to acting as a recognizer of strings $s \in L_m(R)$, R can be viewed as a *generator*, which begins in the start state and nondeterministically selects an event that is generated as an output symbol as the DES transitions to the appropriate state based on the state transition function. The operation of R under the supervision of T is denoted by the DES *T/R* and is shown in Figure 1.

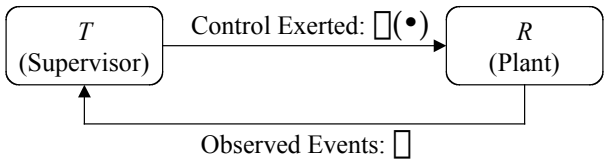

Figure 1. Closed Loop Control of a Discrete Event System.

Control is realized by modeling supervisor $T = (T_a, \gamma)$ as a recognizer with state feedback map γ, and DES R as a generator. Thus, referring to Eq. (1), $T_a = (X, \Sigma, \delta, x_0, X_m)$ and $\gamma : X \to 2^{\Sigma}$. where 2^{Σ} represents the power set of the alphabet Σ. The set Σ can be partitioned into subsets of controllable events (Σ_c) and uncontrollable events (Σ_u) such that $\Sigma = \Sigma_c \cup \Sigma_u$. A controllable event is an action or occurrence that can be disabled, while an uncontrollable event cannot be affected by supervisory actions and is therefore considered to be enabled at all times. Conceptually, as R generates symbols in response to the occurrence of events, supervisor T performs state transitions, which cause the state feedback map γ to produce a set of events that are enabled. The supervisor effectively enables and disables controllable events in the recognizer R, thereby restricting or allowing state transitions by the subordinate DES.

Formally, a supervisor, T, or control policy is a function

$$T : L(R) \to 2^{\Sigma_c} \tag{4}$$

This supervisor functional requirement basically states that a supervisor does not attempt to disable an uncontrollable event: the control policy of the supervisor contains the planned response to all disruptive contingencies that can possibly occur. The language generated by the supervisor acting on the plant is denoted $L(T/R)$, and ideally, one would like to select T such that $L(T/R) = K$, where K is a subset of $L(R)$ referred to as the specification (or "legal", allowable, or desired part of the behavior of R). In practice, however, such T may not exist, but there is always an "optimal" T. To formalize this statement, the following definition is needed:

Definition: $K \subseteq \Sigma^*$ is controllable with respect to $L(R)$ and Σ_{uc} if $\overline{K}\Sigma_{uc} \cap L(R) \subseteq \overline{K}$.

Only controllable subsets (sublanguages) of K can be achieved by means of the control method outlined above. In Ramadge and Wonham[8] it is shown that the supremal controllable sublanguage of K, the optimal solution, denoted $K^{\uparrow}$, always exists. Moreover, there are many algorithms in the literature for computing T's that achieve $K^{\uparrow}$ when $L(R)$ and K are given, respectively, by the finite-state machines R and H, where $L(H) = K$.

The previous results assume *full observation*, that is, the supervisor has access to sufficient information-retrieval devices to ensure that every event that occurs in the model is actually observed. In the case of partial observation, where some events are not observed, there is not a unique supremal control policy but numerous maximal control policies. The existence of many polices is due to the interaction of control decisions with the information available to make decisions. This link between partial observation of events and control policies is also the primary reason that *state estimation* only makes sense when the control or decision policy

is taken into account in the state-estimate update rules.

A fundamental advantage to modeling and controlling complex systems with DES techniques is the formal structure that exists. This has resulted in DES control concepts analogous to results in conventional control theory, including formal definitions of stability, controllability, and observability as well as methods of supervisor synthesis.[9,7,1]

4. Model-based Programming and Execution

Model-based programming and execution provides an implementation framework for DES analysis and policy synthesis results. Our formation of Model-based execution systems is based on the work of Williams, Ingham, et.al.[2], where plant behavior is described as a partially observable Markov decision process (POMD). Programs for control of the plant are described as deterministic automata that are implemented with a "model-based executive" that generates control actions through an iterative process of state estimation and configuration goal sequence synthesis. The basic concept of iterative state estimation was first demonstrated for fault identification on an individual spacecraft in flight as part of the NASA New Millennium Deep Space 1 Remote Agent Experiment.[10] More recent work in this area has extended the scope of model-based reasoning to include response planning and execution within a single representational framework. Our work seeks further extensions to address the problem of coordination of multiple independent control systems in order to accomplish a consistent set of top level goals consistent with formally verifiable DES policy specifications.

The foundation of our implementation framework is a prototype of the *Titan* model-based execution (MBE) kernel and Reactive Model-based Programming Language (RMPL).[2] The combination of these tools enables high level reasoning about system contingencies, scheduling of commanded operations, inference of a system's hidden state, and control of that state. Figure 2 shows an overview of

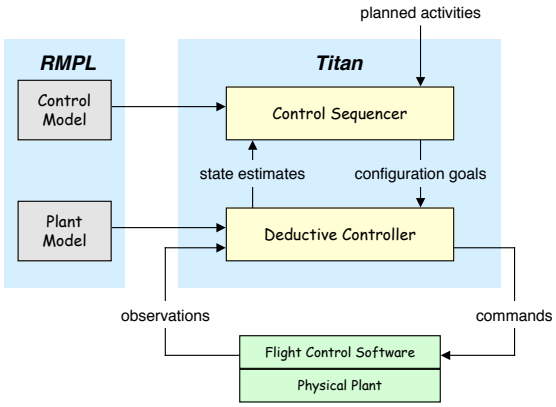

Figure 2. Model-base Execution Kernel Architecture

the Titan execution kernel and its layered relationship with a real-time flight control system (FCS). Low level command loops, sensor filtering, vehicle safing logic, and hardware interface are all implemented in the FCS layer. The Titan/FCS interface is specified in terms of filtered telemetry data and subsystem commands. Current practice in spacecraft commanding would be to build, test, and upload scripts or sequences of FCS commands that trigger directly on telemetry values. In contrast, Titan's Deductive Controller uses filtered telemetry data ("observations") to infer the state of spacecraft subsystems, then generates a set of control actions to achieve a particular goal state configuration. Goal configurations are provided to the Deductive Controller from a Sequencer component in the executive that executes a high level control model. Behavior specification to Titan is in the form of activity goals that are provided by a mission level planner. The process of reasoning through complex subsystem interactions to create subsystem commands, and specifying responses to anomalous mission events is performed by the executive using online propositional inference and reactive planning algorithms. As a result, the executive operates at longer time scales than the FCS, providing a supervisory level of runtime control.

The Distributed Model-based Controller (DMC) extends the application of model-based programming

to include the management and coordination of large scale formation systems. The top level architecture, as shown in Figure 3, replicates the layered model-based controller and provides a runtime communications infrastructure and unified high level programming environment.

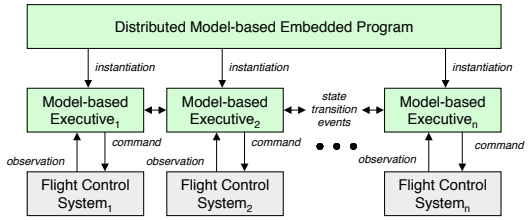

Figure 3. Distributed Model-based Control Architecture

The autonomy developer writes a single control model in RMPL that is then instantiated across multiple independent spacecraft controllers. In this way, a variety of formation control schemes can be implemented, including centralized control, hierarchical distributed control, and fully distributed control.

Although the FCS is responsible for implementing crosslink communications, the semantics of crosslink coordination messages is defined at the MBE level, in terms of modeled state transitions. In this way, we intend to map RMPL constructs into a DES framework in order to provide an analysis infrastructure for distributed model-based control, including formal methods for controller validation.

5. Preliminary Results

We have prototyped two canonical spacecraft coordination scenarios in order to demonstrate the synthesis of optimal distributed control policies and to experiment with their implementation within the model-based controller framework.[11] One of these scenarios is centered on the problem of formation deployment. Two spacecraft must be deployed from a stack that has just separated from a post-boost vehicle. This could represent a sensor deployment task for science observations or the autonomous launch of multiple interceptors in a missile defense scenario. Spacecraft 1 (S/C1) is on the top of the stack and S/C2 is located underneath it on the stack. Either spacecraft can be in one of four states: docked, released, stable, or disturbed. The specification of the desired behavior is that S/C1 should be released first, and S/C2 should be released when S/C1 has been released and is stable. These two behaviors (global unsupervised and legal languages) are depicted in Figure 4 (unsupervised behavior) and Figure 5 (legal behavior). The states of each spacecraft are {0 = docked, 1 = released, 2 = stable, 3 = disturbed}. The Cartesian product of the individual state spaces forms the global statespace. The events for S/C1 are denoted by □, and events for S/C2 are denoted by □. The local events are {1 = deploy spacecraft, 2 = spacecraft stabilized, 3 = spacecraft experiencing transients}. The initial state of the two spacecraft system is denoted by a solid black arrow in the figures.

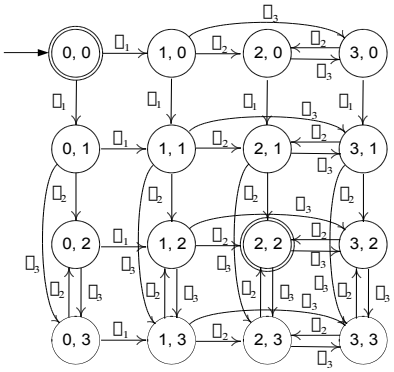

Figure 4. Shuffle of Local Spacecraft Behaviors

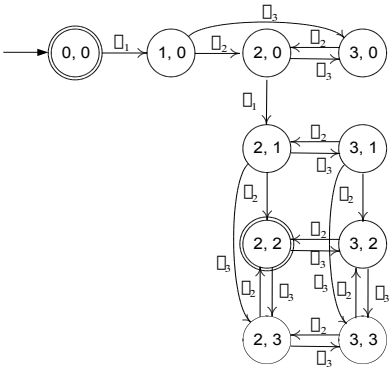

Figure 5. Legal Global Behavior

It is assumed that each spacecraft has control of its own local events; however, it cannot directly observe the events of the other spacecraft. Coordination must be performed among the spacecraft. This coordination will be realized through information sharing and joint synthesis of each spacecraft's control mechanisms; however, for the purposes of efficiency, it is desirable to minimize the communication required to "factor" the globally desired behavior into local spacecraft control objectives.

The synthesis procedure needed to minimize communication and produce an optimal coordination policy begins with constructing the information structure associated with the two spacecraft prior to the introduction of any communication. This preliminary information structure is shown in Figure 6. The figure shows states composed of three boxes. The upper box represents the set of states that S/C1 infers, given its event observations, that the global system could be in. The middle box represents the set of states that S/C2 infers the global system could be in. The lower box represents the actual global state. The contents of these boxes change depending upon what event occurs and which spacecraft controller can observe that event.

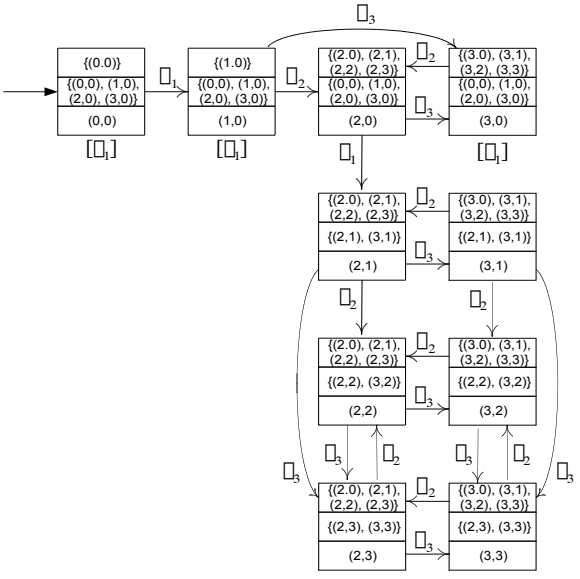

Figure 6. Information Structure for Deployment that is Inconsistent with Control Actions

Upon inspection of Figure 6, note that the information available to S/C2 is insufficiently refined to enable the spacecraft to make the proper decision regarding its "release" event. In some of the inferred possible global states, S/C2 should disable the release event (represented by brackets). In other inferred possible global states S/C2 should enable the release event.

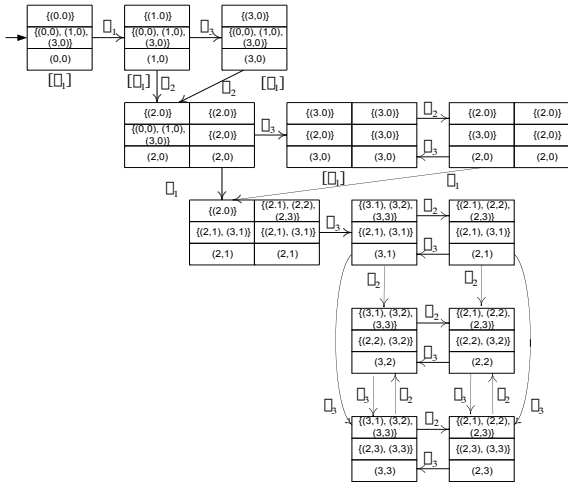

Figure 7. Optimal Coordination Policy for Two-vehicle Deployment

Coordination policy synthesis proceeds by introducing information sharing in the form of global state estimates being exchanged among the spacecraft. Space constraints prevent a self-contained discussion of the entire synthesis procedure; however this example concludes with the solution shown Figure 7. Communication *between* the spacecraft is shown in the information structure diagrams as double boxes.

From Theorem 3.2 in Barrett[3] it can be observed by inspecting the structure shown in Figure 7 that the coordination required for S/C1 and S/C2 cannot optimally be performed by communication and control policies designed independently from each other (there are cases where some separation of design is possible). This type of coordination policy, called *myopic,* does exist for this example; communication from S/C1 to S/C2 following the first observed occurrence of $\square_2$ and S/C2 communicating to S/C1 following $\square_1$ induces an

informationally consistent coordination policy. The resulting estimator structure is shown in Figure 7, and due to the formal synthesis procedure, the associated communication policy satisfies correctness requirements and is provably optimal.

6. Conclusions and Future Work

The growth of distributed spacecraft systems applications has highlighted the need to provide coordination methods that are capable of controlling complex behaviors in a robust, but tractable, manner. Discrete event systems and model based reasoning techniques have demonstrated the ability to achieve this goal in terrestrial systems, and the on-going research in this area has demonstrated applicability of these powerful theoretical and practical methods to space systems.

The next step in our integration of MBR and DES will be to develop a mapping from POMD to DES models, with the goal of formulating RMPL programs with a supervisory control theoretic framework[9], thus providing the potential for automated synthesis of distributed model-based coordination programs.

The next step in our implementation research will be to model additional spacecraft components to a higher level of fidelity, with an emphasis on the cross-link communications and navigation and to scale up to larger formation systems. Additionally, we plan to now address high level fault management at the formation level. As we have discussed, fault management is a particular strength of the model-based programming approach. As formation size increases, the detection and response planning function in anomalous conditions will be a significant risk area in manually operated systems. Model-based controllers offer the potential to manage this risk in a verifiable, automated manner.

Reference

[1] Cassandras, C. and S. Lafortune, “Introduction to Discrete Even Systems,” Kluwer Academic Publishers, 1999.

[2] Williams, B.C, M.D. Ingham, S. Chung, and P. Elliott, Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers, To appear: IEEE Proceedings Special Issue on Embedded Software, IEEE Press, 2002.

[3] Barrett, G. and S. Lafortune, “Decentralized Supervisory Control with Communicating Controllers.” IEEE Trans. on Auto. Cntrl. 45(9), 2000.

[4] Stadter, P. A., "Decision Level Identity Fusion in Exo-atmospheric Target Discrimination for Ballistic Missile Defense," Internal memo SEA-00-0048, JHU/APL, September 2000.

[5] Stadter, P. A., et. al. "Confluence of navigation, communications, and control in distributed spacecraft systems," 2001 IEEE Aerospace Conf., Big Sky, MT, March 2001.

[6] Stadter, P. A., “Discrete Event Command and Control for Formation Flying of Distributed Small Spacecraft Systems,” 13th AIAA/USU Conf. on Small Satellites, August 1999.

[7] Peluso, E. M., “A Hierarchical Structure of Interacting Automata for Modeling Battlefield Dynamics,” Ph.D. Thesis, Dept. of Computer Sci., The Pennsylvania State University 1996.

[8] Ramadge, P. J., and W. M. Wonham, “Supervisory Control of a Class of Discrete Event Processes,” SIAM J. of Control and Optimization, 25(1) January 1987.

[9] Barrett, G and S. Lafortune, "Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines" J. Discrete Event Dynamic Systems: Theory and Applications, 8(4), December 1998.

[10] Muscettola, N., Nayak, P., Pell, B., Williams, B.C., The New Millennium Remote Agent: To boldly go where no AI system has gone before. Artificial Intelligence. 103(1-2). 5-48. 1998.

[11] Harris, A.J., Marshall, S.J., Pekala, P.A., Stadter, P.A., Watson, D.P., Model-based Autonomous Coordination of Distributed Spacecraft Formations. Proceedings of Nanotech 2002.