JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 39, No. 2, March–April 2002

Comparison of Multiple Agent-Based Organizations for Satellite Constellations

Mark Campbell* and Thomas Schetter†
University of Washington, Seattle, Washington 98195-2400

A comparison is made of several autonomous software architectures for multiple satellite systems. Multiple, agent-based satellite systems are envisioned because they are capable of higher performance, lower cost, better fault tolerance, reconfigurability, and upgradability. Software agents are defined at the spacecraft and functional levels, using a wide variety of algorithms including multivariable control, linear programming, fuzzy logic, contract net protocol, negotiation, and radar placement. The agent definitions allow many types of agent-based software architectures to be developed. Several novel agent-based architectures with a more traditional system for a distributed space-based radar mission are compared. Agent-based simulations of several mission scenarios show the autonomous operation of different organizations of increasing autonomy and compare the benefits and drawbacks of each.

Nomenclature

d	=	nondimensional fuel required for maneuver, $\Delta V / h_f$
d_i	=	ith data set
G	=	ground station
h_f	=	remaining fuel, m/s
$\boldsymbol{h}$	=	vector of satellite health
(I_1, I_2, I_3, I_4)	=	spacecraft-level agents
m_i	=	ith message
t_{final}	=	final time of maneuver
t_s	=	sample time, s
$\boldsymbol{u}$	=	thrust, N
$\boldsymbol{v}$	=	spacecraft relative velocity, m/s
$\boldsymbol{x}$	=	spacecraft relative position, m
$\boldsymbol{y}$	=	relative position measurement, m
$\boldsymbol{y}_{\text{ref}}$	=	relative position reference trajectory, m
ΔV	=	velocity increment, m/s
ϵ	=	minimum satellite separation, m
ϵ_F	=	minimum final satellite separation, m

Introduction

MANY future space systems in Earth and space science, defense, and commercial industries will utilize multiple-satellite systems. These systems will include several smaller satellites flown in clusters that work collaboratively together on a space mission, thus forming a virtual satellite. NASA's Origins program is interested in spaceborne interferometry to image far-off planets for possible life forms.[1] The U.S. Air Force in interested in distributed space-based radar because of increased performance and decreased cost.[2] Future commercial space-based systems, such as Teledesic, will use many satellites for global telecommunications coverage.[3] The reasons for this paradigm switch are many, including the increased usage of microelectromechanical-based components to reduce mass, increased production rates to decrease unit cost, and better performance in terms of mission science, fault tolerance, reconfigurability, and upgradability. With these far-reaching benefits, however, comes a new set of challenges, including relative navigation between the satellites, distributed control, and electric propulsion. One key technology that will enable multiple, distributed satellites to achieve their potential, however, is coordinated intelligent autonomy.

Distributed autonomy for multiple satellites is no small task because most current space missions require significant input from the ground for even relatively simple decisions such as thruster burns. The current approach of several operators per satellite will not work for future distributed satellite systems. The individual satellites must be coordinated sufficiently to reduce operator costs while also maintaining or increasing the important science of the mission. For single satellites, the New Millennium Remote Agent architecture for autonomous spacecraft control systems was developed and flown on the Deep Space 1 (DS-1) mission.[4] This system integrates traditional real-time monitoring and control with heterogeneous components for constraint-based planning and scheduling, robust multithreaded execution, and model-based diagnosis and reconfiguration. The complexity of significantly increasing levels of autonomy in space flight software was evident when most of the capabilities of the remote agent were stripped off before launch[5] (although more capability was uplinked subsequently).

Multiple-satellite systems will benefit from technological advances in other areas as well. For instance, advanced global positioning system (GPS) techniques have been developed for ground-based systems that have allowed relative navigation to advance significantly,[6] even for those systems outside geosynchronous orbits. The artificial intelligence (AI) community has developed autonomous software for many distributed systems, including the Internet. Intelligent autonomy for multiple satellite systems, however, is different than many other systems being developed because of its many unique challenges, including the following:

1) There will typically be many vehicles that must coordinate to achieve the desired goals of the fleet, thus making it much more complex than architectures such as DS-1.[4]

2) The dynamics and close proximity of multiple satellites create more challenging control and fault detection problems as compared to multiple rover systems.[7,8]

3) The cost and far distance of space-based systems require reliability to be much higher than that of ground-based systems, such as distributed robotics for environmental cleanup operations.[9]

4) The complexity of the trajectory planning and resource allocation are not problems that many traditional AI technologies, such as the subsumption approach, usually address.[7]

The use of agent-based software architectures represents a new technique in the area of space applications. The individual spacecraft and/or its subcomponents are now seen as agents, that is, individual, independent autonomous entities. Agent-based software differs from traditional space-based approaches both in the modularity, that is the organizational structure, as well as in the intelligence

Received 27 November 2000; revision received 29 June 2001; accepted for publication 7 July 2001.

*Assistant Professor, Department of Aeronautics and Astronautics, Box 352400; mcamp@aa.washington.edu. Member AIAA.

†Research Assistant, Department of Aeronautics and Astronautics, Box 352400; thomas_schetter@hotmail.com. Student Member AIAA.

(functional distribution). Agent-based approaches also allow different agent organizations with varying levels of autonomy to be easily developed and tested.

The objective of this paper is to systematically develop and compare agent-based software architectures for multiple-satellite systems. In terms of functional and organization criteria, there are numerous possibilities for the design of multiagent- (MA-) based architecture. Parameters such as communication, performance, and reliability of an MA-based organization depend strongly on its design and the task to fulfill. This paper attempts to narrow the focus such that many but not all of the possible parameters are studied. The intent of this paper, therefore, is to give a broad, systems-level study to the software architecture of distributed satellite systems using agents and also to add pertinent information to the discussion of autonomy for these systems by addressing the difficult question of how the functions and autonomy of the system shall be distributed in multiple-satellite, space-based systems.

The approach to achieving the technical results of this study are as follows. First, a specific mission focus is defined. In this case, the mission is a distributed space-based radar mission. Second, four software architectures are defined based on four very broad categories: Traditional ground to satellite command and control; one master satellite with several slave satellites; a centralized architecture similar to the master slave, but with additional autonomy on the lower satellites; and a distributed architecture with at least two satellites with many autonomous functions. All but the traditional architecture are agent based. Third, for each of these organizations, lower-level functional agents are defined that include all of the major satellite subsystems and required autonomous functions. The organization simply defines how these are linked and whether they function internally or for the cluster as a whole. Whereas the depth in detail of these functional agents is not to the level of a realistic system, it is enough to achieve the comparison results of this paper. Fourth, a set of metrics are defined as to how to compare the software architectures: communications, reliability, computation, cost, and performance. Finally, four typical scenarios or simulations for the mission are defined. Each software architecture is simulated using these missions and the metrics are calculated and compared.

This paper is organized as follows. First, software agents and how they interact in the context of this work are defined. Next, the specific organizations and agent functionality are presented, followed by the definition of the metrics of comparison and simulation cases. Finally, the simulation comparison and results are presented, with an integrated discussion of their interpretation.

Agent-Based Software Architecture

Agents are a relatively new approach to software programming. There is no consensus on the exact definition of a software agent, but one that is appropriate to this work is taken from Ref. 10, "An agent is a computational entity that can viewed as perceiving and acting upon its environment, and that is autonomous in that its behavior at least partially depends on its own experience." References 11 and 12 provide good overviews of the many different definitions used by researchers in the fields of AI and computer science. One of the simplest agents that can be defined is that of a control system: The sensors help it perceive, the actuators help it act, and there are internal computations that allow it to define its own experience. The agent concept can then be taken further, for example, to the sensors. Each sensor could have an agent in charge of its function, taking the signal and passing it to the control agent. The sensor agent could also monitor for faults within the sensor.

The primary benefit of an agent-based structure is that it allows the integration of higher levels of autonomy quite easily. Intelligent agents are able to receive high-level goals from users and then autonomously determine how to fulfill these goals and take the appropriate actions. Agent-based software is a form of distributed programming, and, as such, it maps naturally onto the requirements of distributed spacecraft. Although there are other approaches to software programming, the agent-based approach is flexible enough to develop and compare architectures with varying levels of autonomy, which is the subject of this paper.

One of the biggest challenges in MA systems and distributed programming is how the agents communicate. For this work, a message passing architecture is used. The architecture is based primarily on TeamAgent,[13] a simulation environment for MA systems especially tailored for the spacecraft domain. In TeamAgent, agents represent the software and remote terminals connect the agents with the hardware. Each of these function identically in the software architecture, with the capability of sending and receiving messages and data.

The agents/remote terminals are defined using a set of skills and tasks and can pass and receive information using messages and message centers. Message centers (MC) have two primary functions: 1) to register and validate agents and 2) to pass messages between agents and other message centers. Agents are created in TeamAgent by assigning them a set of skills, or basic software functions pertinent to how the agent performs. Generally, each skill corresponds to one basic function, has inputs and outputs, and triggers one or more actions. Messages are special data structures that describe information to be transmitted between agents. Tasks, on the other hand, contain the to do lists for agents. Tasks are special data structures in TeamAgent that describe 1) particular actions to be taken by the agent and/or 2) messages to be sent to another agent. Reference 13 contains more detail on how the agent infrastructure works. Note that in actual implementation, the message and MC concept can be implemented using an appropriate operating system.

MA Architectures for Distributed Satellite Systems

There are many approaches to organizing agents for autonomy within distributed satellite systems. For example, consider planning a cluster trajectory maneuver. The maneuver commands could be planned on the ground and sent to one or multiple satellites, commands could be planned on one satellite and distributed to the others, or commands could be planned in a distributed fashion on multiple satellites. The dynamics, potential collisions, communication, and onboard computation are each affected by the chosen architecture. Because of this and the complexity of how agents and defined and implemented, several choices are made to narrow the scope of study. First, four levels of spacecraft-level agents are defined and used to define four organizational architectures. Next, required functional agents (lower-level agents) are defined to implement the different autonomous functions. These are described next.

Spacecraft-Level Agents and Organizations

The agent-based software architectures described here are hierarchical in nature, and, as such, they are most easily understood by starting at the top. In this work, spacecraft-level agents are defined by their capability within the satellite cluster. This definition is primarily made to reduce the complexity of study because the spacecraft can then be combined into an agent-based organization. Spacecraft-level agents are defined based on the level of intelligence, or the sum of capable functions. Four levels of capable intelligence I_1–I_4 have been identified, where I_1 denotes the highest level of intelligence and I_4 the lowest level. Figure 1 (top) illustrates the identified spacecraft-level agents.

Spacecraft-level agent I_4 represents the most unintelligent agent. It can only receive commands and tasks from other spacecraft-level agents in the organization or from the ground and execute them. This type of intelligence is similar to what is being flown on most spacecraft today. Spacecraft-level agent I_3 has local planning functions onboard. This type of intelligence is similar to the DS-1.[4] Spacecraft-level agent I_2 adds a capability to interact with other spacecraft-level agents in the organization. This spacecraft-to-spacecraft interaction allows many higher level autonomous functions but also usually requires that the spacecraft-level agent has at least partial knowledge of the full agent-based organization, that is, other spacecraft-level agents. The I_2 spacecraft-level agent must, therefore, continuously keep and update an internal representation of the agent-based organization. Spacecraft-level agent I_1 represents the most intelligent agent because it is capable of monitoring and planning for all spacecraft-level agents in the organization. In other words, the I_1 level agent has full knowledge and can plan for the full cluster.

By the use of the spacecraft-level agents, four architectures are defined for the cluster. Figure 1 (bottom) shows these four coordination options as a function of individual, capable spacecraft-level agent intelligence. The top-down coordination architecture includes

Table 1 Organization types that are developed and compared

Organization	Spacecraft agents G	I_1	I_2	I_3	I_4	Figure
Traditional	1	——	——	——	8	
Top down	——	1	——	——	7	
Centralized	——	1 (1)[a]	——	7	——	
Distributed	——	2 (2)	6	——	——	

[a]The numbers in parentheses and shading refer to the passive I_1 agents for added redundancy.

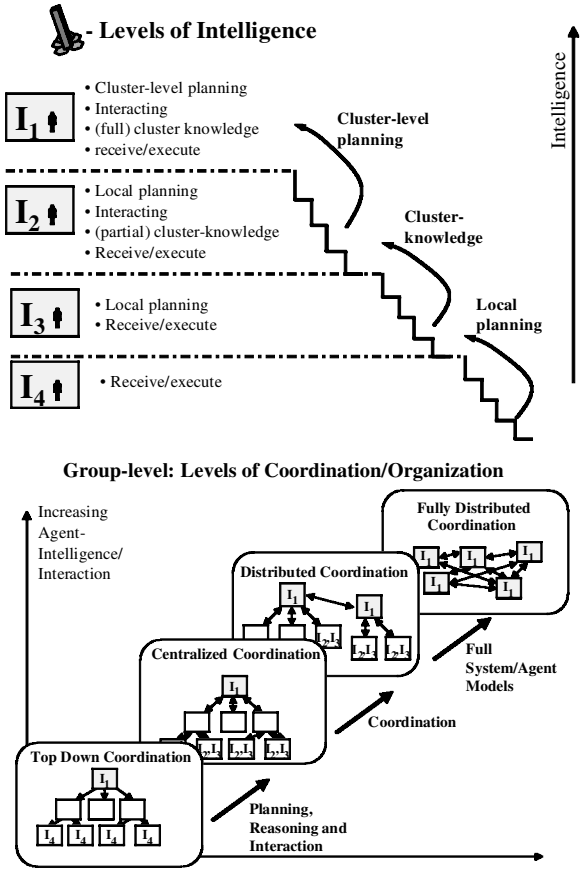


Fig. 1 Top shows identification of spacecraft-level agents based on levels of capable intelligence I_1–I_4; bottom shows coordination architectures for coordination of multiple spacecraft-level agents.

only one single (highly intelligent) spacecraft-level agent I_1, and all other spacecraft are (nonintelligent) I_4 agents. The centralized coordination architecture requires at least local planning and possibly interaction capabilities from each spacecraft. Thus, spacecraft-level agents I_3 or I_2 are required. The distributed coordination architecture consists of several parallel hierarchical decision-making structures, each led by an intelligent spacecraft-level agent I_1. Note that the different spacecraft-level agents I_1 can interact with each other as well as their lower-level I_2 or I_3 spacecraft-level agents. In the case of a fully distributed coordination architecture, each spacecraft in the organization represents a spacecraft-level agent I_1, resulting in a totally flat organization.

In this work, four specific architectures are compared, as shown in Table 1. The traditional architecture consists of eight I_4 satellites and a ground control center. The top-down architecture is similar, but most of the ground operations are automated and transfered to one I_1 level satellite. The centralized architecture is similar to top down, but there are now seven I_3 level satellites. Also, one of these I_3 level satellites is a passive I_1 satellite, implying that it has the capabilities of an I_1 satellite for redundancy, but functions as an I_3 satellite. Finally, there is a distributed architecture, with two I_1 level satellites, and six I_2 satellites, two of which are passive I_1 satellites. The fully autonomous case is not presented here because of its similarities to the distributed case. With these organizations, many comparisons can and are made: traditional vs top down, top down vs centralized, and centralized vs distributed.

Simulation and Functional Agents

The focus mission for this work is based partially on TechSat21, a U.S. Air Force mission being designed to explore the benefits of a distributed approach to satellites.[2] An initial demonstration is on track for launch in 2004. The ability to perform a space-based radar mission, which historically has required very large, high-power satellites, is seen as an extreme test of the distributed satellite concept. TechSat21 takes advantage of distributed satellites

Table 2 Examples of implemented software agents

Number	Agent	Description	Tool	S/C agent I_1	I_2	I_3	I_4
F11	Sensing	Retrieving S/C state, health	——	x	x	x	x
F12	Xcommunicate	Exchange data with other S/C	——	x	x	x	x
F13	Communicate	Exchange data with ground	——	x	——	——	——
F20	SciencePlan	Which S/C, which targets	Fuzzy logic	x	x	——	——
F21	DecStatKeep	Monitor for station keeping	Fuzzy logic	x	x	x	——
F22	CollAvoid	Collision monitoring	Fuzzy logic	x	x	——	——
F23	DecMakFail	Health monitoring	Fuzzy logic	x	——	——	——
F24	DecMakAdd	Cluster upgrade	Fuzzy logic	x	——	——	——
F30	Schedule	Scheduler	Prioritize using logic	x	x	x	——
F31	PlanReconfig	Plan cluster reconfiguration	Symmetric placement	x	——	——	——
		Plan local reconfiguration	Symmetric placement	——	x	x	——
F32	PlanAssign	Cluster assignment	Contract net	x	——	——	——
		Local cost	Contract net	——	x	x	——
F33	TaskAlloc	Task allocation	Negotiation	x	x	——	——
		Local cost	Negotiation	——	x	x	——
F34	PlanFF	Trajectory planning for S/C	Linear program	x	x	x	——
F35	Propulsion	Thruster logic	Fuzzy logic	x	x	x	x
F40	ClusterHealth	Maintain cluster record	——	x	x	——	——
F50	Science	Perform radar	——	x	x	x	x
F51	OrbitMan	Orbit maneuvering	Linear quadratic regulator control	x	x	x	x

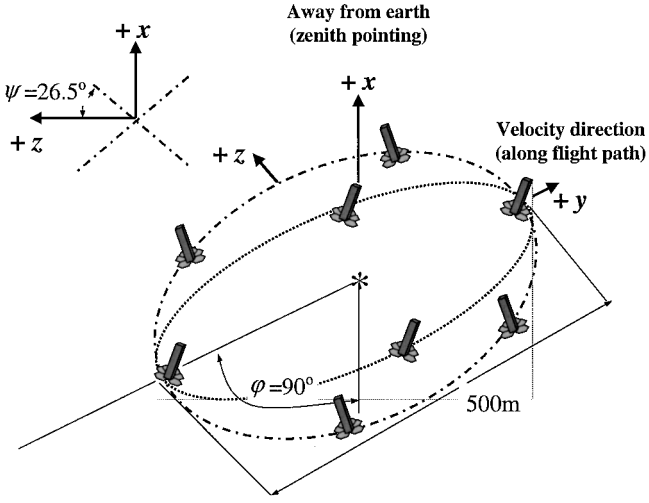


Fig. 2 Focus mission is an eight-satellite cluster, with four satellites in two local planar ellipses tilted at 30 deg to the vertical.

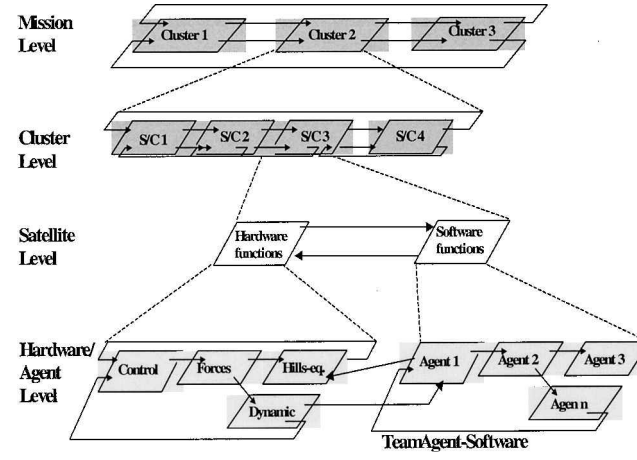


Fig. 3 Combined agent software/dynamics simulation environment for multiple-satellite systems.

by using a sparse aperture array for radar imaging, which allows improved resolution. The focus mission for this work is a constellation of eight satellites, with four satellites placed in one of two ellipses, each with a 250-m major axis and tilted at a ±30-deg angle from the vertical axis. The orbit is a circular polar orbit, at an altitude of 700 km, and spacing between the satellites is 10–250 m. Figure 2 shows this constellation, where the asterisk locates the virtual center of the cluster, and orbits the Earth as any single satellite might.

A simulation environment was developed using a combination of Simulink[14] for the spacecraft dynamics, and TeamAgent/MATLAB® for the software agent architecture. Figure 3 shows the combined dynamic/agent simulation environment using Simulink's capabilites to create hierarchical block structures. From the top mission level, one can move down through different levels to see more detail. The spacecraft (S/C) level contains blocks for both hardware components and software agents, represented by special Simulink *s*-functions. These *s*-functions build an interface between the agent-based software and the dynamic simulation.

Lower-level functional agents are defined based on current state of the art algorithms and are used to build up each spacecraft level agent. Table 2 shows the agents and implemented skills based on different engineering tools. Also shown is the applicable satellite-level agent (I_1–I_4) that the functional agent/skill is a characteristic.

This paper focuses on comparing different agent-based organizations with varying levels of autonomy, rather than the agent definition and implementations. Therefore, only several of the agents are described here, specifically focusing on those that clarify the differences in the organizations shown in Table 1. A detailed description of these functional agents and skills can be found in Refs. 13 and 15.

All decision-making skills are implemented using fuzzy logic and/or control,[16] which is a standard representation in the AI community. For instance, the agent CollAvoid monitors for possible collisions. This agent uses a reactive algorithm and fuzzy logic to monitor for collisions. Specifically, the agent uses the relative position and velocity of all satellites to define a scalar quantity for each satellite pair, denoted as

$$\Delta x_{ij} = \min_{0 \le t_s \le t_{\text{update}}} [\boldsymbol{x}_j - \boldsymbol{x}_i + (\boldsymbol{v}_j - \boldsymbol{v}_i) \cdot t_s] \qquad (1)$$

where Δx_{ij} is a scalar collision metric. This variable gives a measure of satellites drifting too close and/or too fast. A fuzzy variable is then defined based on three cases: 1) no action, just continue to monitor; 2) simple action such as a small thruster fire or sending a warning to the other satellite; or 3) drastic action such as a deorbit. As the output of the fuzzy logic changes, the priority of the collision avoidance task increases or decreases. The priority is used in the Schedule agent. A higher priority case requires the collision avoidance agent to be queried more often and the time update of its information t_{update} to

be faster. For the centralized case, CollAvoid is implemented on one agent on one satellite, based on satellite state information distributed over the communication crosslink system (XCommunicate). In the distributed case, all satellites monitor their own health and collisions.

The agent PlanReconfig is called in the case of a failure or adding a new spacecraft to the cluster. The goal is to reconfigure the cluster to maximize its continued usefulness for the radar imaging task. Note that the agent SciencePlan performs a similar function by retargeting the cluster based on a new set of targets. For each of these agents, a new reference position and velocity are calculated for spacecraft within the cluster based on an optimality criterion. For the PlanReconfig agent, a symmetric placement within the ellipse is used. For the SciencePlan agent, a new set of science targets is used to assign placement.

Once a new cluster configuration (or reconfiguration) is known, the PlanAssign agent assigns the individual spacecraft to the chosen positions. Assignments are made in this case by attempting to equalize fuel use across the cluster and, thus, maximize total lifetime. This can be done by calculating the ratio of the required velocity increment ΔV and the status of the remaining fuel h_f:

$$d = \Delta V / h_f \tag{2}$$

The PlanAssign agent is implemented using one agent to calculate the fuel usage for all satellites moving to all positions. This approach works well in the case presented here because of the small size of the cluster (eight satellites); however, for larger clusters, heuristic search methods may have to be used. In the centralized and distributed cases, each individual satellite calculates its velocity increment for each move and sends this information to the I_1 agent, thus employing a parallel programming approach. Additional intelligence in the I_2 satellite allows the agent to attempt to improve the plan through a decentralized negotiation.[17] For instance, the lower-level I_2 agent examines a proposed maneuver plan and evaluates that it could save fuel by waiting and using orbital dynamics do a portion of the work. If the added time does not violate other higher-level mission constraints (time, collisions, etc.), then the plan is approved by the I_1 level agent.

The PlanFF (trajectory planner) agent calculates the velocity increment required to move to new S/C positions within the cluster. This agent uses a linear programming optimization technique because of its flexibility as applied to the cluster planning problem. The linear program minimizes a cost function, such as the time or fuel of the maneuver, and allows the addition of simple constraints. As an example, the minimum fuel cost is given as

$$J(x) = \sum_{j=1}^{n} |u_j| \tag{3}$$

Minimizing this cost is subject to an initial position $\boldsymbol{y}(t_0)$, desired final state $\boldsymbol{y}(t_{\text{final}}) = \boldsymbol{y}_{\text{ref}}$, and total number of time steps n. Additional constraints include the maximum thrust, such as that imposed by future precision on–off-type thrusters,[18] maximum position error at t_{final}, and minimum satellite separation to prevent collisions. These are added to the problem as linear inequality constraints, or

$$\begin{aligned} &|\boldsymbol{u}(t)| \le \boldsymbol{u}_{\max}, && \text{on–off thruster} \\ &|\boldsymbol{y}_{\text{ref}} - \boldsymbol{y}(t)| \le \boldsymbol{\epsilon}, && \text{minimum separation} \\ &|\boldsymbol{y}_{\text{ref}} - \boldsymbol{y}(t_{\text{final}})| \le \boldsymbol{\epsilon}_F, && \text{minimum final separation} \end{aligned} \tag{4}$$

A final added benefit of the linear program is that the final position $\boldsymbol{y}(t_{\text{final}})$ can be time varying, which it is in the TechSat21 case where each satellite is rotating about the virtual center (Fig. 2). Because no classical method in calculus or linear algebra offers a closed-form solution to this problem, numerical techniques for solving linear programming problems have been developed such as the Simplex Method.[19] Details on this trajectory planner and its implementation in a closed-loop formation flying control for multiple satellites can be found in Ref. 20.

The TaskAlloc agent distributes task assignments to each S/C level agent within the cluster. An example using this agent is when a S/C failure occurs. If an I_1 S/C fails, the TaskAlloc agent assigns cluster-level decision-making tasks to a new spacecraft. In the top-down organization, a static method is used where the hierarchy of S/C level agents is decided a priori. In the centralized and distributed cases, a contract net protocol is used.[21]

Consider the contract net protocol applied to a reconfiguration based on a failure within an active I_1 S/C level agent. A S/C-level agent I_1 acts as contractor. The other (passive) S/C-level agents I_1 are then the bidders. The task of the contractor is to minimize a cost function on the bids. In other words, the contractor selects the most appropriate S/C-level agent for the given situation based on the bids received. The bids can consist of many functions, such as the satellite health status $\boldsymbol{h}$ of the S/C. An example of a possible cost function could be

$$C = (c_1/h_s) \cdot (c_2/h_p) \cdot (c_3/h_t) \cdot (c_4/h_f) \tag{5}$$

where c_1–c_4 are weighting factors, chosen based on the importance of the different subsystems, and the h correspond to the health values of the different monitored spacecraft subsystems, such as science, power, thrust, and fuel.

The agent organizations (S/C level down through the functional level) can now be developed and integrated. As an example, consider Fig. 4, which summarizes the agents and information flow for a centralized organization. Three agents are shown: an active (top) and passive (left) S/C-level agent I_1 and a S/C-level agent I_2 (right). This architecture has been implemented in the MATLAB/Simulink/TeamAgent software environment shown in Fig. 3. The messages m_i and data d_i are shown flowing between the agents.

Evaluation Metrics and Simulations

Comparing software agent organizations, even at a high level, in an attempt to narrow the scope of further study for autonomous satellite clusters is a difficult task. This work attempts to do this by first defining a set of metrics that can be used to compare the organizations, simulate several missions, and interpret the results. Whereas this is not an all-inclusive study (more metrics and more simulations would undoubtedly add to the discussion), the work here is an important first step in answering questions such as the following: 1) How can we compare different software architectures of autonomous agents for space-based clusters? 2) What are the strengths and weaknesses of each?

Evaluation Metrics

The evaluation metrics for the comparisons include communications, computation, fuel usage, cost, and reliability. Each of these are discussed next.

Communication C is the act of transferring data between satellites and between the ground and the satellites. This obviously is different for each organization and is an important parameter used to evaluate information bottlenecks, that is, waiting for information, or power usage, that is, using more power when required. The parameters measured for each organization/simulation are the average and peak data rate for the downlink and crosslink. This is calculated by measuring the number of messages that are being passed between the satellites (crosslink) and between the satellites and ground using only the defined agents. Thus, the quoted numbers do not include full telemetry, radar, or other parameters. The data rate, or number of messages per second, is then converted to a more familiar bits per second by assuming a message length of 16 bits. These several assumptions obviously preclude this metric from accurately reflecting the actual rates for the links, but they will still allow relative comparisons. Table 3 shows the specific parameters that were chosen for the two communication links based on the TechSat21-like mission.

Computation W is the average and peak CPU workload for the system. The current workload can be thought of as the percentage of the maximum computational rate (computations per second) or percentage of CPU time dedicated to tasks. In the simulation, the current workload of an individual spacecraft-level agent I_i for the time interval t_s is calculated by

$$\frac{\text{CPU}(I_i)}{\text{maxCPU}} \cdot 100\% \tag{6}$$

where CPU(I_i) denotes the CPU usage (floating point operations per second or FLOP) over the time interval t_s. The workload of the full cluster takes into account the series or parallel CPU usage. Note also that the required computation for the ground station is not factored into this metric.

Performance P can be measured in many ways for the radar mission, such as the number of targets per time period, the onboard fuel consumption, the required time to reconfigure due to a failure or retargeting, etc. An added complexity is that many of these measures are coupled, such as time and computations during parallel processing of a cluster planner. Because of this, the radar processing, targeting, and fuel usage performance parameters are all assumed to be constant across each organization, that is, these are required, mission-critical functions. Performance as presented here is the time required to execute the primary portion of the mission scenario. For example, if during a 60-min simulation a 5-min maneuver is planned and executed, then the performance is 5 min. The time performance metric is based on a fixed CPU speed and communication rate to reduce bias.

Reliability R is the reliability of the satellites and software architecture at the mission level. To do this, a typical set of reliability numbers have been assigned to each of the lower-level functional agents based on average satellite data.[22] Then, using functional diagrams such as those presented in Fig. 4, the reliability of the S/C-level agents are calculated. These have been rounded, based on assumed information, and are given in Table 4. The secondary ground station is a ground station placed in another part of the world with the task of gathering and relaying data to the primary ground station. Second ground stations allow the traditional system to be in full contact, yet minimize costs. Note that this metric is a function of how the organization was developed and does not change with the different mission simulations presented in the next section.

Cost (dollars) is the approximate cost to develop, test, implement, and maintain the flight software and ground operations. Costs for other components are not factored in because they would be similar across each organization. The software developed for each S/C level agent is used to calculate the number of S/C lines of code. The cost to develop this software for each S/C-level agent, shown in Table 4, was calculated using \$726 per line of code.[22,23] The cost for the ground-based system was developed using an around-the-clock team to monitor the cluster. Assuming 50 people at the primary ground station, the total cost to develop and maintain the ground station is \$6 million/year. Secondary ground stations, which might be located around the world to relay information back to the central ground station, can have a skeleton crew that primarily maintains the facilities. Therefore, these costs are \$2 million/year. As with the

Table 3 Selected communication link parameters for the TechSat21 mission

Parameter	Downlink d/l	Crosslink c/l
Carrier frequency f, GHz	2	60
Transmission path loss L_a	0.3	1
Transmitter line loss L_l	0.1	0.1
System noise T_s, K	550	1800
Efficiency η	1	1

Table 4 Reliability and cost numbers for the spacecraft-level agents

Component/agent	Reliability	Operations and software costs (FY2000\$M)
Ground station	0.99999	6/yr
Secondary ground station	0.9999	2/yr
I_1	0.99	10 total
I_2	0.995	5 total
I_3	0.999	2 total
I_4	0.9999	1 total

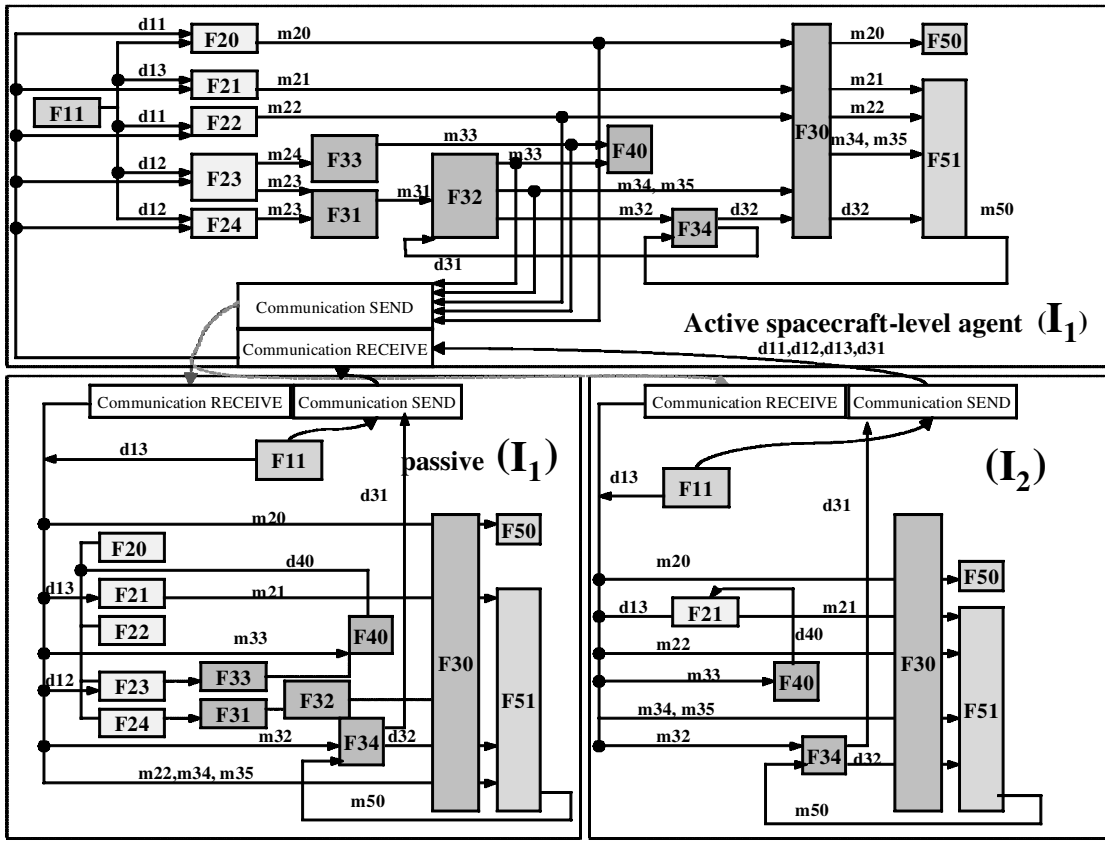


Fig. 4 Information flow architecture for centralized coordination architecture. Functional agents: F11, Censing; F20, SciencePlan; F21, DecStatKeep; F22, CollAvoid; F23, DecMakFail; F24, DecMakAdd; F30, Schedule; F31, PlanReconfig; F32, PlanAssign; F33, TaskAlloc; F34, PlanFF; F40, ClusterHealth; F50, Science; and F51, OrbitMan.

Table 5 Mission simulations that were used to compare each of the software architectures

Mission	Steps for task success
M1: nominal operations	1) S/C perform radar processing while measuring relative spacing and thrusting to maintain accuracy. 2) New target is selected. 3) New set of positions and orientations are selected based on the new target. 4) Command sequence is generated to reconfigure the cluster. 5) Reconfiguration occurs, and radar processing starts again.
M2: collision avoidance	1) S/C begins to drift toward another S/C during the radar processing. 2) Possible collisions must be monitored using propagated orbits and/or GPS data. 3) After evaluating that the collision is a distinct possibility, a command sequence is generated to successfully avoid the collision and reconfigure the cluster. 4) Maneuver and reconfiguration occur.
M3: deorbit/reconfigure	1) One S/C is older and running out of fuel. 2) Deorbit command sequence is generated and implemented. 3) New satellite is added to the cluster by generating and implementing an insertion trajectory and reconfiguring the satellite/software hierarchy. 4) New satellite insertion and reconfiguration occur.
M4: failure of I_1	1) Failure of the primary I_1 S/C occurs. 2) Deorbit command sequence is generated and implemented. 3) New satellite/software hierarchy is generated and implemented.

Table 6 Simulation results for four missions and four organizations

		Computation, %		Communication, bps: Downlink		Communication, bps: Crosslink		Performance, time, %	Reliability	Cost	
Simulation	Organization	Peak	Average	Peak	Average	Peak	Average			Operations, $M/yr	Software, $M
M1	Traditional	3.6	1.01	24,000	24,000	48	48	+1,220	0.99999	24	1.7
	Top down	66	1.65	3,330	8,720	3,330	12,500	+400	0.98999	6.0	10.7
	Centralized	33	1.69	3,330	8,720	860	12,400	+318	0.99989	6.0	12.2
	Distributed	19	1.90	3,200	8,700	1,240	12,400	——	0.99999	6.0	14.5
M2	Traditional	3.6	1.01	24,000	24,000	48	48	+1,330	0.99999	24	1.7
	Top down	78	1.69	5,210	9,890	3,330	18,200	+400	0.98999	6.0	10.7
	Centralized	66	1.99	3,330	8,720	1,720	18,100	+318	0.99989	6.0	12.2
	Distributed	36	2.59	3,200	8,500	3,330	18,100	——	0.99999	6.0	14.5
M3	Traditional	3.6	1.01	24,000	24,000	48	48	+1,220	0.99999	24	1.7
	Top down	76	1.69	5,210	9,890	3,330	12,500	+400	0.98999	6.0	10.7
	Centralized	66	1.98	3,330	8,720	1,690	12,400	+318	0.99989	6.0	12.2
	Distributed	36	2.56	3,200	8,500	3,220	12,400	——	0.99999	6.0	14.5
M4	Centralized	36	1.30	3,330	8,720	990	4,210	+318	0.99989	6.0	12.2
	Distributed	19	1.92	3,200	8,500	1,500	5,460	——	0.99999	6.0	14.5

reliability metric, cost is a function of how the organization was developed and does not change with the different mission simulations presented in the next section.

Mission Simulations

The four organizations are compared by simulating each over a series of typical scenarios that may/will occur over the lifetime of the mission. For each organization and simulation, each of the five metrics are evaluated and compared. The work presented here comprises the more interesting results that help to draw conclusions in comparing the organizations.

Table 5 presents four mission simulations that are evaluated, along with the steps required to complete this task. The four missions include 1) nominal operations, where the S/C formation flies and then must retarget when new commands are sent from the ground; 2) a possible collision between two S/C during the radar processing, when no thrusting occurs; 3) deorbit and reconfigure, where one satellite is nearing the end of its lifetime, it deorbits, and the cluster must add a new member; and 4) failure of I_1, where the primary communication link to the ground has failed on the I_1 level satellite itself, and the cluster must reorganize into a new hierarchy.

M1: Nominal Operations

The first mission scenario simulates a standard case of nominal operations. The chosen mission is sparse aperture radar, which requires precise relative positioning. However, during radar processing, there can be no thrusting, and the S/C can and will drift. Throughout the simulation, potential collisions are monitored, as are the subsystems, for faults.

The steps for this case are shown in Table 5. The cluster performs radar processing on a target, while also monitoring for collisions. Each S/C thrusts to offset disturbances and anomalies between periods when radar processing occurs. Once a new target is selected on the ground, a communication message is sent to the cluster. For the traditional case, the new positions within the cluster for sparse aperture radar are translated into a series of thruster burns and sent to each of the eight satellites. The other architectures require only the target position and time for processing, and all planning is done onboard within the agent hierarchy. In the top-down case, one satellite plans all maneuvers and sends thruster commands to each satellite. In addition, the I_1 S/C leader performs all monitoring for collisions and faults. In the centralized case, each satellite plans and monitors locally. In the distributed case, each satellite plans and monitors locally, and each satellite optimizes the plan with respect to fuel using a negotiation technique.

The first four rows of Table 6 show the evaluation metrics for simulation M1 for each organization. There are several items to note. First, the computational effort for the traditional case is much lower than all others (3.6% vs 66, 33, and 19% peak and 1.01% vs 1.65, 1.69, and 1.90% average). This is because the eight I_4 satellites simply receive and execute commands and require no higher-level computation. The top-down case requires the largest peak in computational effort because it executes all high-level functions for the cluster. The centralized and distributed cases reduce this peak, but also increase the average. Time simulations for the top-down and centralized cases are shown in Fig. 5 (top). Notice that the peak computational workload comprises the reconfiguration and station-keeping tasks.

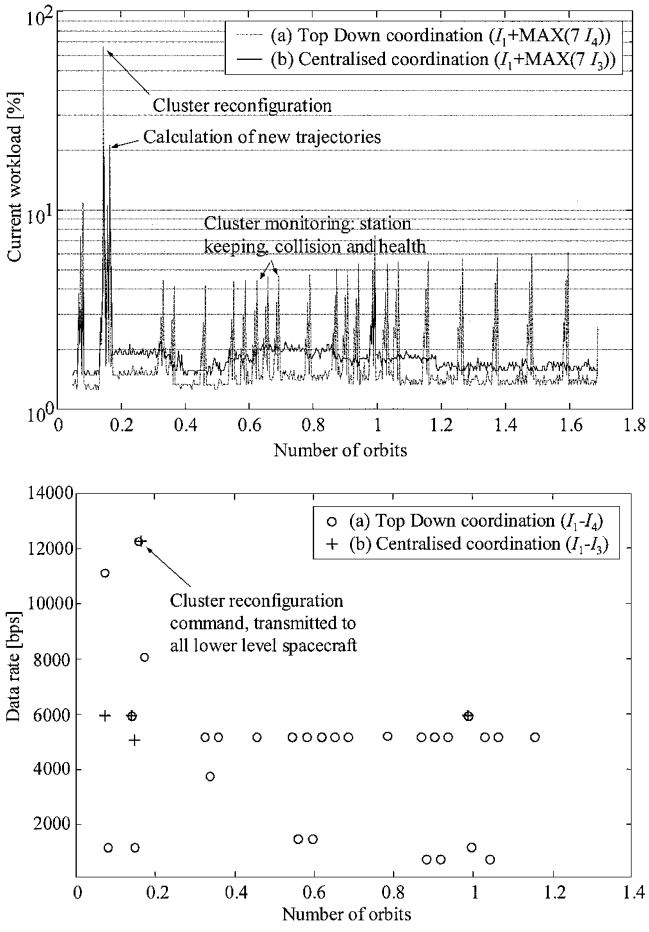


Fig. 5 Current CPU workload for the top-down and centralized coordination architectures is shown in top panel; bottom panel shows crosslink data rate during mission simulation M1 between the active spacecraft-level agent I_1 and lower-level spacecraft agents.

The traditional case requires the largest downlink communication data rate because each satellite must downlink all health telemetry data so that ground personnel can evaluate anomalies and plan maneuvers. In each of the agent-based cases, these data are either kept within the satellite or transferred to other satellites within the cluster over the crosslink. Obviously, the traditional case requires much more power for communications and, therefore, will require more power budgeting. Time simulations of the communication results are shown in Fig. 5 (bottom). The peak again occurs during the cluster reconfiguration. The crosslink communications vary from 48 bps for the traditional case, where only position and velocity data are shared between satellites in the lower-level control loops, to 12,500 bps in the top down case, where the I_1 level satellite must send all thruster commands to each satellite.

Based on the total time it takes to plan and execute the cluster reconfiguration, the distributed architecture executes 318% faster than the centralized architecture because of the parallel processing, and 1220% faster than the traditional case, where all data must be downlinked and evaluated on the ground. Note that the time invested for human decision making on the ground for the traditional case was not factored into this calculation.

The reliability for the traditional case is approximately the same as the distributed case and slightly higher than the other two. However, the cost to attain this level of reliability is quite high. The traditional case requires at least 10 ground stations spread across the globe to monitor the cluster constantly. As shown in Table 6, this requires an operations costs level of $24 million/year for the traditional architecture, which is much higher than the other cases. Over several years, these operations costs will accumulate dramatically. Note that there could be other traditional architectures, such as using geosynchronous Earth orbit satellites for communication relay. However, the general trend of the results will be the same.

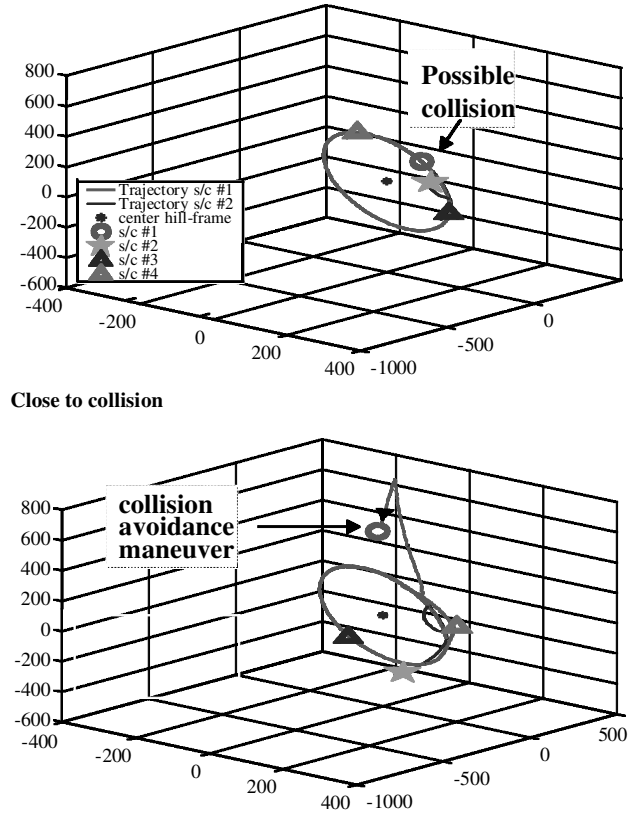


Fig. 6 Three-dimensional animation of mission simulation M2: collision avoidance of an I_4 spacecraft and a cluster reconfiguration.

M2: Collision Avoidance Maneuver

The second mission simulation, M2, addresses a possible collision due to the very close nature of the satellites within the cluster. Although this simulation uses a 200–400-m relative spacing as its nominal configuration, there are cases that require 10-m spacing, for 5-m-tall satellites. In this case, a thruster fails while radar processing occurs, and a lower-level I_4 satellite begins to move toward a second satellite. If no maneuvers occur, the satellites will collide within 0.2 orbits, or 20 min. The necessary steps that the cluster must perform include detecting the collision within a threshold of 50 m using propagated orbit calculations and GPS data, followed by a maneuver to avoid the collision. Figure 6 shows the scenario and required maneuver. As can be seen, the bang–bang control for the collision avoidance maneuver B results in a collision avoidance trajectory change for S/C 1 (represented by the circle symbol).

Table 6 shows the results of the five metrics for this simulation. For the traditional architecture, the decision making and initialization of the collision avoidance maneuver and cluster reconfiguration must be performed based on ground commands. There must be full ground contact with the cluster, that is, at least 10 ground stations, to ensure there are no observation/communication blocks. Obviously, it is quite a challenge (perhaps even impossible) to evaluate the possible collision, plan the new trajectory, uplink the information, and maneuver within the 20-min window.

Figure 7 shows a plot of the relative distance over time between the two S/C for the agent-based simulations. The time from when the collision avoidance monitoring becomes higher priority (based on a change in velocity from an inadvertent thruster firing) to when the S/C are within 50 m is less than one-fifth of an orbit, or 20 min. This 50-m threshold is the critical distance for activating the collision avoidance maneuver. The maneuver lasts approximately 18 min until a safe parking orbit has been reached.

As shown in Table 6, the increased communication distance to the Earth for the traditional approach results in a significantly higher performance time. It is clear for satellite clusters that if relative spacing is within meters, building autonomous collision avoidance monitoring is critical to the full reliability of the system. As with

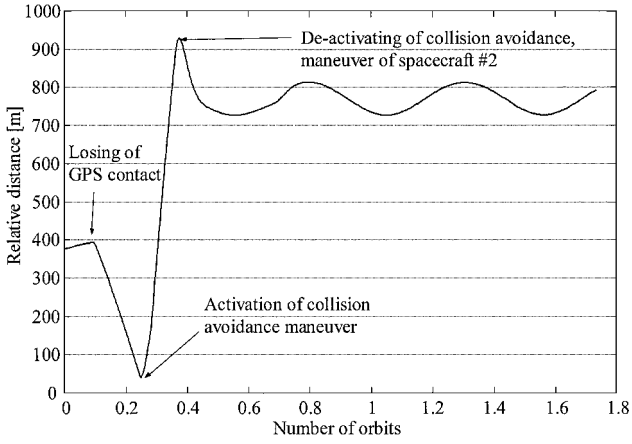
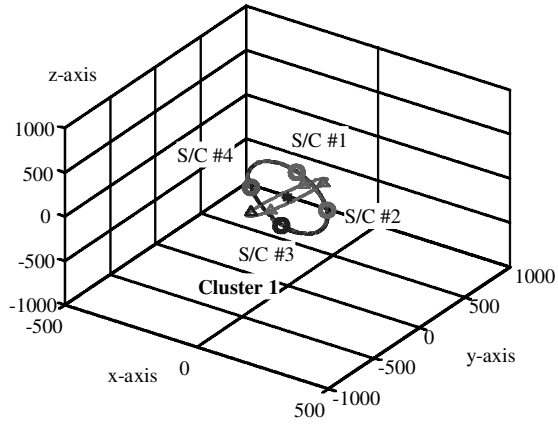
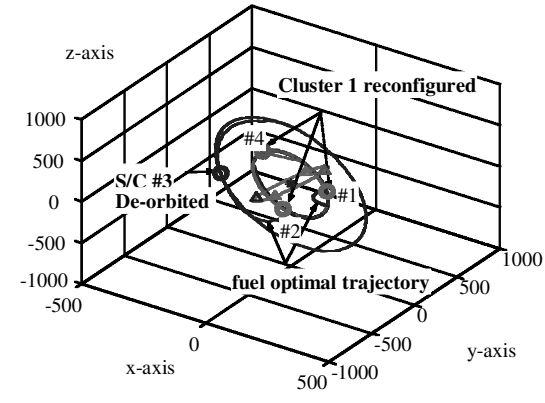


Fig. 7 Relative distance between spacecraft 1 and 2 for mission simulation M2 for the agent-based cases; minimum distance, 41 m.



Situation before cluster reconfiguration



Deorbit S/C 3 and cluster reconfiguration

Fig. 8 Three-dimensional animation of mission simulation M3: deorbit of a satellite (top) and reconfiguration (bottom).

M1, M2 shows that the agent-based organizations require much more computational workload as compared to the traditional case because many of the automated tasks are performed onboard. Both the computational workload and communication data rate for M2 have increased compared to M1 because of the unexpected collision avoidance monitoring and maneuver.

M3: Deorbit/Reconfiguration

Mission simulation M3 is the case of a cluster reconfiguration where S/C 3 in cluster 1 and S/C 7 in cluster 2 execute a planned deorbit, two clusters are reconfigured, and two new S/C are added. Figure 8 shows the simulation for cluster 1. (Cluster 2 is similar.) Many of the trends in this simulation are similar to the M1 and M2, including 1) the traditional case requires more communication bandwidth, 2) the agent-based approaches require much more computational effort because of the many autonomous functions, and 3) the agent-based approaches perform the planning much faster than the traditional approach.

Several additional items are noted. First, the computational workload has slightly decreased from M2 to M3. This is because the maneuver for M3 is planned (and end of life deorbit), whereas the maneuver for M2 is unplanned initially (collision avoidance). Second, the traditional case, whereas it is still slower than the agent-based cases, is slightly better in M3 as compared to M2. This is again because the maneuver is preplanned and can be more easily accomplished with a ground station.

M4: Failure of an I_1 S/C

The final simulation compares centralized and distributed coordination architectures when a failure occurs in an active I_1 satellite. In this case, after the failure has been detected, an existing partially active S/C-level agent I_1 must take over the role of the failed active S/C-level agent I_1 in the cluster. The passive I_1 S/C monitor the active I_1 S/C for failures. With a voting scheme, failures can be reliably identified. One of the passive S/C then is voted forward to perform the reconfiguration.

In the centralized case, where all lower-level S/C are I_3 level, a new active S/C-level agent I_1 is nominated using a static task allocation algorithm, that is, a logic rule base. For the distributed case, a new active S/C-level agent I_1 is selected using a dynamic task allocation mode, that is, a contract net-based bidding mechanism. The peak computational workload for the distributed case is 47% higher than the centralized case, primarily because only one I_1 level satellite monitors the cluster in the centralized case. The average workload, however, is 95% larger in the distributed case because the distributed architecture requires more computation and crosslink communication to perform the contract net-based bidding mechanism. After the cluster reconfiguration, the distributed case also has a higher computational workload because the I_2 level agents must periodically receive and update an internal cluster description.

Summary of M1–M4 Simulations

Table 6 shows a summary of each of the metrics for each of the four simulations. The following is a summary of the trends:

1) The traditional case requires more downlink communication bandwidth because all data must be downlinked to the ground for processing.

2)The agent-based approaches require much more computational effort because of the many autonomous functions onboard the satellites.

3) The agent-based approaches perform most functions much more quickly than the traditional approach.

4) Autonomous collision avoidance (using agents or other programming techniques) is required for clusters with satellites in close proximity.

5) To achieve similar reliability, a larger operations team is required for the traditional case, which creates much higher yearly costs.

6) Distributed architectures decrease the peak computational workload by removing bottlenecks in one satellite; however, the average computational workload and communications both increase slightly.

Conclusions

The use of MA systems for autonomous satellite clusters has been explored. Using distinct functions for satellite clusters such as linear programming for minimum fuel maneuvers and fuzzy logic for decision making, four types of organizations were developed. These four organizations each increase in intelligence and autonomy: traditional (with a ground station and no intelligence), top down (with one intelligent master satellite and several unintelligent slave satellites), centralized (with one intelligent master satellite and several satellites that can plan locally), and distributed (with two or more intelligent satellites). Each organization was simulated over four typical scenarios and compared in terms of computation, communication, time

performance, reliability, and cost. Table 6 shows the summary of the numerical results.

The following conclusions can be drawn from the results. The agent-based approaches are always faster than the traditional approach, albeit at a lower reliability. Several important autonomous functions, such as collision avoidance, however, are enabling for low-Earth-orbit missions with satellites in close proximity because the functions cannot be completed in the time window using a ground communication link. Traditional approaches cannot achieve the same reliability at a good cost because of the large ground operations. Each level of autonomy adds much more computational workload; thus, the primary tradeoff in designing autonomous software architectures (such as MA systems) is trading reliability and performance for increased computational effort. It could be concluded, however, that of the five metrics, computational effort is the least influential because of the increasing performance of onboard flight computers over time. Distributed architectures decrease the peak computational workload by removing bottlenecks in one satellite. However, the average computational workload and communications both increase slightly for distributed systems. Reliability in the agent-based systems can closely match that of the traditional case by adding passive I_1 level agents. Although more simulations could be run, the best software architecture based on this study is a centralized or partially distributed architecture with a small percentage of highly intelligent satellites for redundancy.

Acknowledgment

This work is supported under a U.S. Air Force Small Business Innovation Research Program contract with Princeton Satellite Systems, Contract F29601-99-C-0098.

References

[1]Capps, R. W. (ed.), *Recommendations for Technology Development and Validation Activities in Support of the Origins Program*, NASA CR-203426, JPL-PUBL-96-21, June 1996.

[2]Das, A., Cobb, R., and Stallard, M., "TechSat21, A Revolutionary Concept in Distributed Space-Based Sensing," AIAA Paper 98-5255, Sept. 1998.

[3]Matossian, M. G., "A Teledesic Space Infrastructure Overview," *Proceedings, Mission Design and Implementation of Satellite Constellations*, Kluwer, Dordrecht, The Netherlands, 1997, pp. 153–156.

[4]Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C., "Remote Agent: To Boldly Go Where No AI System has Gone Before," *Artificial Intelligence*, Vol. 103, Nos. 1–2, 1998, pp. 5–47.

[5]Dornheim, M. A., "Deep Space 1 Launch Slips Three Months," *Proceedings, Aviation Week and Space Technology*, Vol. 39, 27 April 1998, p. 39.

[6]Olsen, E., Park, C.-W., and How, J., "3D Formation Flight Using Differential Carrier-phase GPS Sensors," *Journal of the Institute of Navigation*, Vol. 146, No. 1, 1999, pp. 35–48.

[7]Brooks, R. A., "Achieving Artificial Intelligence Through Building Robots," Massachusetts Inst. of Technology Artificial Intelligence Lab., AD-A174364, Cambridge, MA, 1986.

[8]Evans, K. S., Unsal, C., and Bay, J. S., "A Reactive Coordination Scheme for a Many-Robot System," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 27, No. 4, 1997, pp. 598–610.

[9]Schneider-Fontan, M., and Mataric, M., "Territorial Multi-Robot Task Division," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 5, 1998, pp. 815–822.

[10]Weiss, G. (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, 1st ed., MIT Press, Cambridge, MA, 1999, p. 27.

[11]Bradshaw, J. (ed.), *Software Agents*, 1st ed., MIT Press, Cambridge, MA, 1997.

[12]Knapik, M., and Johnson, J., *Developing Intelligent Agents for Distributed Systems*, 1st ed., McGraw–Hill, New York, 1996.

[13]Schetter, T., Campbell, M., and Surka, D., "Multiple Agent-Based Autonomy for Satellite Constellations," *Proceedings, Lecture Notes in Computer Science*, Vol. 1882, Springer-Verlag, Berlin, 2000, pp. 151–165.

[14]The Mathworks, Inc., *SIMULINK User's Guide*, Natick, MA, 1997.

[15]Schetter, T., "Autonomous Control for Multiple Satellite Clusters," M.S. Thesis, Dept. of Aeronautics and Astronautics, Univ. of Washington, Seattle, WA, Dec. 1999.

[16]Nguyen, H. T., Sugeno, M., Tong, R., and Yager, R. R. (eds.), *Theoretical Aspects of Fuzzy Control*, 1st ed., Wiley, New York, 1995.

[17]Wangermann, J. P., and Stengel, R. F., "Optimization and Coordination of Multiagent Systems Using Principled Negotiation," AIAA Paper 96-3853, July 1996.

[18]Rayburn, C., Campbell, M., Hoskins, A., and Cassady, J., "Development of a Micro Pulsed Plasma Thruster for the Dawgstar Nanosatellite," AIAA Paper 2000-3256, July 2000.

[19]Murty, K. G., *LINEAR Programming*, 1st ed., Wiley, New York, 1983, pp. 1–34.

[20]Campbell, M. E., and Schetter, T., "Formation Flying Mission for UW Dawgstar Satellite," *Proceedings, IEEE Aerospace Conference*, Vol. 7, IEEE Publications, Piscataway, NJ, 2000, pp. 117–125.

[21]Davis, R., and Smith, R., "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, Vol. 20, No. 1, 1983, pp. 63–109.

[22]Wiley, J., and Larson, J. R. W., *Space Mission Analysis and Design*, 3rd ed., Kluwer Academic, Norwell, MA, 1992, pp. 783–820.

[23]*Unmanned Space Vehicle Cost Model*, 6th ed., Space and Missile System Center, Directorate of Cost, Los Angeles AFB, CA, 1994.

A. C. Tribble
Associate Editor