Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Artificial Intelligence 145 (2003) 147–180

Artificial Intelligence

www.elsevier.com/locate/artint

Multiple agent-based autonomy for satellite constellations

Thomas Schetter [a,1], Mark Campbell [a,*], Derek Surka [b,2]

[a] *University of Washington, Box 352400, Seattle, WA 98195-2400, USA*
[b] *Princeton Satellite Systems, 33 Witherspoon St, Princeton, NJ 08542, USA*

Received 3 January 2002; received in revised form 8 September 2002

Abstract

Multiple, highly autonomous, satellite systems are envisioned in the near future because they are capable of higher performance, lower cost, better fault tolerance, reconfigurability and upgradability. This paper presents an architecture and multi-agent design and simulation environment that will enable agent-based multi-satellite systems to fulfill their complex mission objectives, termed ObjectAgent^{TM}. Its application is shown for a distributed aperture radar mission, although its applicability spans many types of missions. Required spacecraft functions, software agents, and multi-agent organisations are described for the radar mission, as well as their implementation. Agent-based simulations of mission case studies show the autonomous operation of the multi-agent architecture, which can then be used to build, evaluate and compare autonomous software architectures for multiple satellite systems.

Keywords: Multiple satellite autonomy; Multi-agent systems; Spacecraft autonomy; Software agents

1. Introduction

A new paradigm shift in spacecraft design is sweeping the space industry, including the science [17], defense [6], and commercial sectors [24]. The shift is from single large

* Corresponding author. Assistant Professor, Member IEEE, Mailing Address: 208 Upson Hall, Cornell University, Ithaca NY 14853, (607)255-4268, (607)255-1222 (fax).
E-mail addresses: thomas_schetter@hotmail.com (T. Schetter), mc288@cornell.edu (M. Campbell), surka@alum.mit.edu (D. Surka).
[1] Research Assistant.
[2] Technical Staff.

0004-3702/03/$ – see front matter
doi:10.1016/S0004-3702(02)00382-X

satellites that are expensive and have a lot of built in redundancy, to several smaller satellites working in tandem. The redundancy of the "cluster" is now in the large number of satellites rather than in the individual subsystems. Satellite clusters include several smaller satellites that collaboratively work together on a satellite mission, thus forming a "virtual" satellite. The reasons for this paradigm switch are many, including the increased usage of micro-electromechanical (MEMS) 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, control, and electric propulsion. The key technology that will enable multiple, distributed satellites to achieve their potential, however, is coordinated intelligent autonomy.

The cost of operating a *single* spacecraft after launch is a considerable portion of the overall mission cost. For commercial satellites, operations consist of monitoring the spacecraftŠs health and status, taking corrective measures when necessary, and performing maneuvers. Military and scientific satellites require additional ground personnel to process the tremendous amount of payload data gathered. Automating these activities through the use of agents will reduce the cost of missions and make spacecraft more robust, reliable, and efficient. In addition, the use of *multiple* satellites distributed over a small cluster will require much higher levels of autonomy than those that exist today.

There is very little "intelligence" on today's satellites. Current space flight software only measures sensors, acts on ground commands, and gracefully reboots when an anomaly occurs. In 1999, the first attempt to use agents for satellite autonomy was launched in NASA's Deep Space 1 (DS1) mission. The DS1 researchers developed Remote Agent [16], an autonomous agent architecture based on model based programming, on-board deduction and search, and goal-directed closed loop commanding. The complexity of automating activities in space systems to the level that owners/operators will use the autonomous software was shown when, because of technical difficulties, much of the Remote Agent software was stripped off the satellite prior to launch, although portions of the software were uplinked at a later date [8]. The DS1 work is slightly different than this work for several reasons. First, it was for one satellite, not a group of satellites. Second, DS1 was still based on traditional flight software rather than a hierarchy of intelligent agents.

Some of the most relevant work in autonomy for distributed systems has been in robotics and autonomous underwater vehicles. There has been recent work in emergent behavior [1], where robot colonies work together, even though no single robot knows the group objectives. Though this approach has had much success for robots and simple tasks, many useful tasks for multiple satellites will require the ability to plan. The MAUV/CoDA Project [25] focuses on controlling autonomous oceanographic networks, including autonomous underwater vehicles. The work uses two organizations: a task-level organization to control the system during the actual mission, and a meta-level organization to self-organize the system. Much of this work has been in simulation, although experiments will be used to evaluate the work.

Intelligent autonomy for multiple satellite systems, however, is different than many other systems being developed because of its many unique challenges, including:

- 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 [16].
- The dynamics and close proximity of multiple satellites create more challenging control and fault detection problems as compared to multiple rover systems [2,9].
- The cost and far proximity of space based systems require reliability to be much higher than ground based systems, such as distributed robotics for environmental clean-up operations [23].
- The complexity of the trajectory planning and resource allocation are not problems that many traditional AI technologies, such as the subsumption approach, usually address [2].

The use of agent-based software architectures represents a new technique in the area of space applications. The individual spacecraft and/or its sub-components are now seen as agents, that is, individual, independent autonomous entities. Agent based software differs from traditional space based approaches both in the modularity (i.e., the organizational structure) as well as in the intelligence (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 work is to create a software infrastructure for simulating and comparing architectures for multiple satellite systems; each architecture will create a different "virtual" satellite and therefore allow multi-satellite systems to fulfill their complex mission objectives at a lower cost. That is, the cluster is now seen as a single entity rather than several or many individual satellites. The software infrastructure must be adaptable in both its low level code (such as changing planning approaches) and high level code (such as adding or subtracting satellites to the cluster). This differs from the traditional approach both in the way they act together (i.e., the organisational structure) as well as in the "intelligence" they have (functional distribution).

This paper presents the agent based software infrastructure to enable autonomous, multiple satellite systems. More specifically, the paper shows how low level agents can be developed and combined into spacecraft level agents, which can then be combined into an agent based software architecture for clusters of satellites. Examples of agents at all levels are given, along with several simulations to show how the architecture(s) can be developed. It is noted that Ref. [22] details a full comparison of different types of agent based organizations. This paper details the software infrastructure, and is unique in three areas: (1) integration of (formal) control and AI technologies into a multi-agent framework, (2) a message passing software approach that easily allows comparisons of technologies and multi-agent organizisations, and (3) the application of the multi-agent approach to multiple satellite systems.

2. Distributed satellite systems

Many future space systems in Earth and space science, defense, and commercial industries will utilize multiple satellites systems. NASA's Origins program is interested in spaceborne interferometry to image far off planets for possible life forms [17]. The US

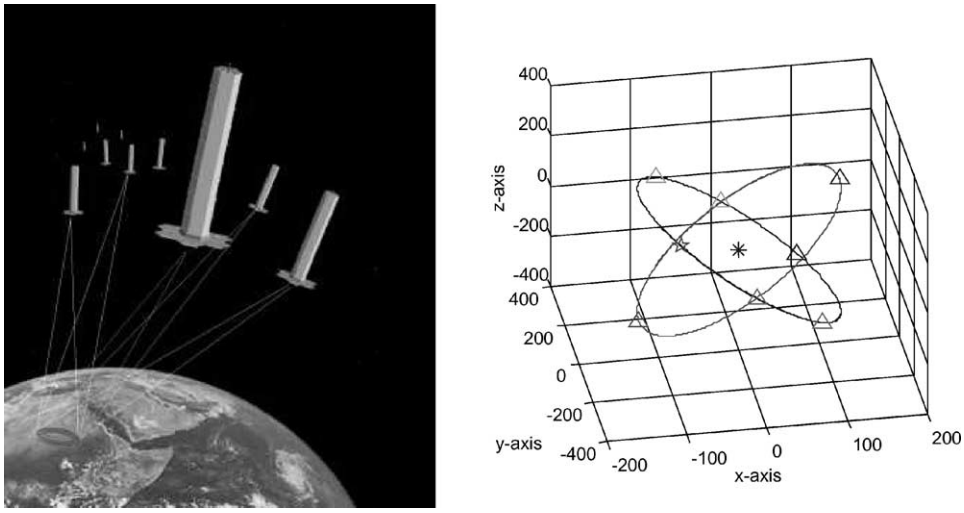


Fig. 1. Left: TechSat21—a revolutionary approach to space based sensing. Right: a 3D-animation of the TechSat21 mission, where '*' is the virtual center of the cluster, and the spacecraft 'Δ's are in two planes of four.

Air Force is interested in distributed space based radar because of increased performance and decreased cost [6]. Future commercial space based systems, such as Teledesic, will use many satellites for global telecommunications coverage [24].

The TechSat21 mission is an Air Force mission designed to explore the benefits of a distributed approach to satellite design. The initial demonstration is currently being designed for a space based distributed radar [6]. 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 this concept. TechSat21 takes advantage of the distributed satellites by using a sparse aperture array for radar imaging, which allows improved resolution because of the satellite spacing (Fig. 1).

The function of imaging (radar or others) with multiple aperture is done through the use of interferometry techniques, where different spatial frequencies that make up the image are recorded by pairwise interference of signals from selected satellites. In the case of an imaging radar such as TechSat21, the radar moves along a flight path and an area is illuminated by the radar, termed the *footprint*. Each spacecraft in the cluster illuminates the same footprint simultaneously. The antenna receives the return echoes from the target (area) and stores them. This return signal is the composite of each spacecraft in the cluster. The data is then processed to resolve a single pixel in the footprint.

In order to accomplish distributed aperture radar imaging, the satellites in the cluster must cover the Earth's surface area of particular interest with a more or less uniform distribution. Given the very strict constraints on the fuel available, the only conceivable approach for the cluster is to have the satellites be in *force free* orbits while in formation. The current configuration for TechSat21 has focused on clusters of spacecraft in two local ellipses, tilted at ± 30 degrees from the vertical z-axis. The center of the cluster, or also called the Hill frame with, is shown as a '*' in Fig. 1 and rotates about the Earth as any

single large satellite might. The nominal orbit for the cluster (or cluster center) is a circular, polar orbit. The Hill's equations [11] describe the relative motion of spacecraft by the use of linearized equations.

$$\begin{aligned} x &= -\frac{R_0}{2}\cos(w_n t + \varphi), \\ y &= R_0 \sin(w_n t + \varphi), \\ z &= R_0 \cos(w_n t + \varphi + \phi), \end{aligned} \tag{1}$$

where R_0 corresponds to the major axis of the elliptical motion along the y-axis and $R_0/2$ to the minor axis along the x direction. The phasing angle φ describes the phase of the spacecraft within each of the two ellipses and ϕ the rotation about the y-axis of the two ellipses. For ease and simplicity, a constellation of eight satellites was chosen, with four satellites placed in each ellipse. The central concept in planning for clusters is to change the size of the ellipse (R_0) and phasing of the ellipse (φ, ϕ) as the spacecraft maneuver from one target to the next.

3. Functional agent definitions

Agents are defined slightly differently in many texts, but one that is appropriate to this work is from Ref. [26]: "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".

In the following section, the ObjectAgent infrastructure is introduced to aid in the development of agents and their interconnects for multiple satellite systems. This is followed by a detailed description of the agents and their skills. The ObjectAgent framework enables the primary contributions of this work: (1) development of agents that can be easily integrated into the multi-agent system even after implementation, (2) development and comparison of different organizations of agents, such as using two different coordination schemes: centralized and distributed. The subsequent section then describes how they are integrated into spacecraft level agents and autonomous hierarchies.

3.1. ObjectAgent infrastructure

ObjectAgent is a MATLAB toolbox [19] for the design and simulation of multi-agent systems, especially spacecraft. In ObjectAgent, *agents* are software processes that represent software algorithms and remote terminals. *Remote terminals* (*RT*) are software objects that connect agents with hardware.

ObjectAgent is based on a *message passing architecture*, meaning that all agent-to-agent and agent-to-RT communication is done through messages that pass through message centers (MC) or post offices. The functions of the MC are:

- register and validate agents,
- process messages for itself,
- pass messages to registered agents, RT's, and other MC's for processing,

- allocate processor time for each agent.

Messages can be passed over several MC's; therefore it does not matter where the agent or RT is located. The MC functional process is shown in Fig. 12. Note that while the MC is explicitly used in the work presented here, an alternate real time implementation is to integrate the message passing into the operating system level.

Skills are the basic building blocks of agents and remote terminals within ObjectAgent. Agents and RT's are created in ObjectAgent by assigning to them a set of skills. These are special MATLAB files that represent agent functions. For example, a skill required by all agents is to register with the MC, represented by the `RegisterSkill.m` function. Generally, each skill corresponds to one basic function, has inputs and outputs, and triggers one or more actions. The primary action for each skill is an `update` action that the skill runs periodically based on a pre-defined update period. Each skill contains a data structure field that describes the assigned priority, the update period, the input and output interfaces and the communication method.

There are several structures within ObjectAgent that allow communication between agents. *Messages* are exchanged between agents and *data* are blocks of information exchanged between agents within the messages. *Tasks* are the activities that an agent performs and each agent maintains a dynamic list of the tasks it is currently running. Messages and tasks each have the same structure in ObjectAgent; hence, a message can cause the receiving agent to take a particular action. The *verb* of the message or task dictates the particular action taken.

As described previously, each skill has at least one task associated with it—`update`. When skills are added to an agent, tasks associated with that skill are automatically generated. These tasks, when processed, can cause a message to be created and sent, and/or actions to be taken by agents that change its internal state.

Fig. 2 shows an example, where the task "`update CollAvoidSkill`" creates the message "`MoveCollAvoid sc_4|To: OrbitManAgent4(OrbitManSkill4)` (m1), because a possible collision involving spacecraft #4 was detected. The message m1, which tells "`OrbitManAgent4`" to command a maneuver, is then sent to MC4. When "`OrbitManAgent4`" receives the message, the verb function `MoveCollAvoid` is called which triggers the command generation for an avoidance maneuver within the `OrbitManeuverSkill`. Additionally, message m2 (the state of spacecraft #2) is transmitted back to the collision avoidance agent 1.

3.2. Lower level functional agents and skills

In order to demonstrate the usefulness of multi-agent systems applied to multiple satellite clusters in general, and TechSat21 specifically, four high-level tasks are defined:

- HT1: Performing science (Imaging),
- HT2: Formation maintaining and control,
- HT3: Cluster reconfiguration,
- HT4: Cluster upgrade.

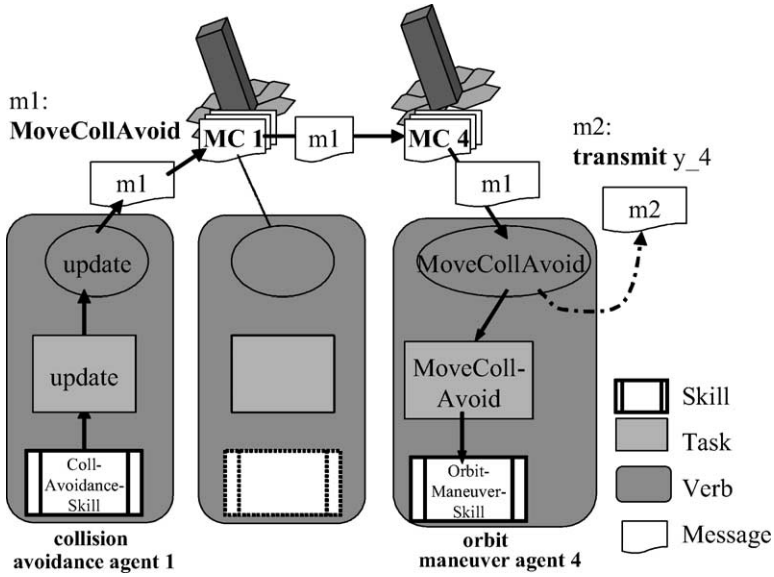


Fig. 2. Example of the relationship between skills, messages, and verbs.

These high-level tasks were then used to identify all necessary sub-level tasks, along with the elementary functional blocks required to implement these tasks. Fig. 3 shows the functional breakdown from high level tasks to lower level agents. The columns correspond to four high level tasks, and the rows to sub-tasks and functional blocks/agents. The particular agents are denoted with a two digit number. The first digit refers to the task category (i.e., 2-decision-making function) and the second to the sub-partition within the task category (i.e., 3-failure/loss).

Table 1 shows the implemented skills for the corresponding agents. Shown are also the priority assignments, the update period and tools which are used by the corresponding skills. These skills are also related to the spacecraft level agents, I_1–I_4, to be discussed subsequently. A variety of state of the art tools are used for these skills, including fuzzy logic for decision-making [18], the Cornwell Metric for cluster positioning based on radar imaging [12], contract net bidding and negotiation algorithms for planning [7,10], and linear programming for optimal trajectory generation [4]. Each of these agents and their associated skills are described next.

3.3. Interaction agents

`SensingAgent` (F11) continuously obtains and updates the state and health from the spacecraft, and makes it available for the entire cluster. This agent requires `SensingSkill.m`, which continuously reads the required spacecraft state $\underline{x}_i$ and health $\underline{h}_i$ for each i spacecraft. The state includes satellite position $\underline{\mathrm{x}}$ and velocity $\underline{\mathrm{v}}$, and the health $\underline{h}$ includes the status of the science h_s, power h_p, and thrust h_t subsystems, as well as the remaining fuel h_f. Other important parameters include the velocity increment for a

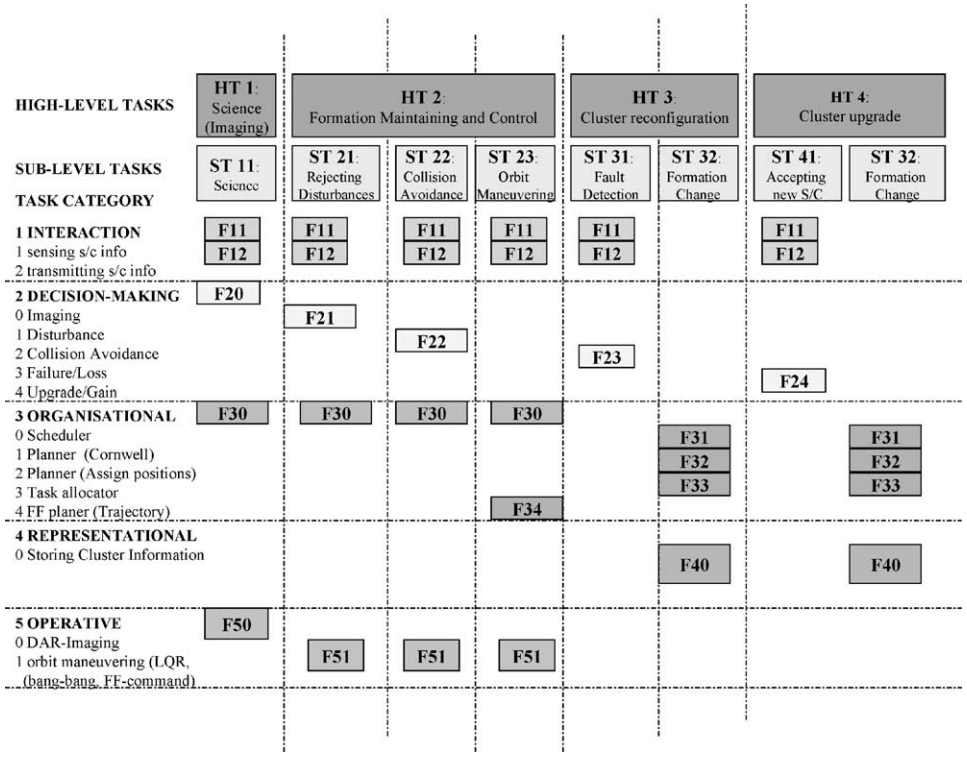


Fig. 3. Functional breakdown of the task structure specifically for TechSat21.

maneuver, ΔV_{ij}, error in the position and velocity state, $\underline{err}_i$, thrust sequence, $\underline{u}_{ff}$, and current formation summary, $\underline{ci}$. Some states are measured, while others can be estimated from measurements.

3.4. Decision making agents

Decision-making agents periodically make decisions or monitor specific system parameters for changes. Most of these agents use fuzzy logic as the base algorithm. `ScienceAgent` (F20) decides which satellites acquire which targets, and how many satellites are required to monitor changing points of interest. `StatKeepAgent` (F21) decides whether it is necessary to station keep (maintain position) or to perform an orbit correction maneuver due to external disturbances. This agent requires the `StatKeepSkill.m`, which is implemented using fuzzy logic. The agent uses one input: the relative error err_i for each spacecraft position from the nominal reference position $y_{\text{ref}\,i}$ minus the actual position y_i; and two outputs: dynamic allocation of the update period and of the priority of the corresponding task `RejectDist` (m21). Dynamic allocation of both the update period and the priority is based on the magnitude of the error err_i.

`CollAvoidAgent` (F22) detects when collisions may occur between the spacecraft in the cluster, and can be implemented in a centralized or distributed framework. This agent

Table 1
Examples of implemented software agents

#	"Agent"	Description	Priority	Tool	s/c agent			
					I_1	I_2	I_3	I_4
F11	`Sensing`	retrieving s/c state, health	fix	–	x	x	x	x
F12	`Xcommunicate`	exchange data with other s/c	fix	–	x	x	x	x
F13	`Communicate`	exchange data with ground	fix	–	x			
F20	`SciencePlan`	which s/c, which targets	fix	Fuzzy Logic	x	x		
F21	`DecStatKeep`	monitor for station keeping	var	Fuzzy Logic	x	x	x	
F22	`CollAvoid`*	collision monitoring (C)	var	Fuzzy Logic	x	x		
		collision monitoring (D)	var	Fuzzy Logic	x	x		
F23	`DecMakFail`	health monitoring	fix	Fuzzy Logic	x			
F24	`DecMakAdd`	cluster upgrade	fix	Fuzzy Logic	x			
F30	`Schedule`	Scheduler	fix	Prioritize using logic	x	x	x	
F31	`PlanReconfig`*	plan cluster reconfig (C)	fix	Symmetric placement	x			
		plan cluster reconfig (C)	fix	Cornwell Metric	x			
		plan local reconfig (D)	fix	Cornwell Metric	x		x	
F32	`PlanAssign`*	cluster assignment (C)	fix	Fuzzy Logic	x			
		local cost (D)	fix	Contract Net	x		x	
		local cost (D)	fix	Negotiation	x	x		
F33	`TaskAlloc`*	task allocation(C)	fix	Fuzzy Logic	x	x		
		local cost(D)	fix	Contract net	x		x	
F34	`PlanFF`	trajectory planning for s/c	fix	Linear Program	x	x	x	
F35	`Propulsion`	thruster logic	fix	Fuzzy Logic	x	x	x	x
F40	`ClusterHealth`	maintain cluster record	var	–	x	x		
F50	`Science`	perform radar	fix	–	x	x	x	x
F51	`OrbitMan`	orbit maneuvering	var	LQR control	x	x	x	x

∗—those agents that are implemented using two or more coordination schemes.
C—centralized.
D—distributed.

requires `CollAvoidSkill.m`, which implements a fuzzy logic controller for collision detection. Because the spacecraft fly in close formation (less than 250 m separation), some type of collision checking and avoidance functionality on board is required. Based on the current position $\underline{x}$ and velocity $\underline{v}$ of each spacecraft, the future relative positions of the spacecraft can be predicted in the near term for one update time step t_{update} of the collision avoidance task (Fig. 4). This is given as

$$\Delta x_{ij} = \min_{0 \leqslant t_s \leqslant t_{\text{update}}} \left(\underline{x}_j - \underline{x}_i + (\underline{v}_j - \underline{v}_i) \cdot t_s \right), \tag{2}$$

where Δx_{ij} is a scalar number used to evaluate how close to a collision two satellites may be. If this number is smaller than a predefined limit, action is taken. A fuzzy variable can be defined based on three cases: SMALL, MEDIUM, OR LARGE. These are shown in Fig. 4. The possible actions are: `DeltaX`(LARGE)—no action, just continue to monitor; `DeltaX`(MEDIUM)—simple action such as a small thruster firing or sending a warning to the other satellite; or `DeltaX`(SMALL)—drastic action such as an orbit maneuver. If the latter is the case, a corresponding collision avoidance task `MoveCollAvoid` (m22) is triggered consisting of a bang-bang control that "moves" the spacecraft apart.

The priority of the collision avoidance task, which is used by the `ScheduleAgent`, also increases as the satellites move closer. As a collision becomes more likely (`DeltaX` tends to 0), the priority of the collision avoidance agent increases, and the time update of its information, t_{update}, decreases (i.e., it receives information more frequently). The implemented fuzzy controller has one input (`DeltaX`), two outputs (`UpdateTime` and `TaskPriority`), and three rules, which are shown in Table 2. The collision avoidance

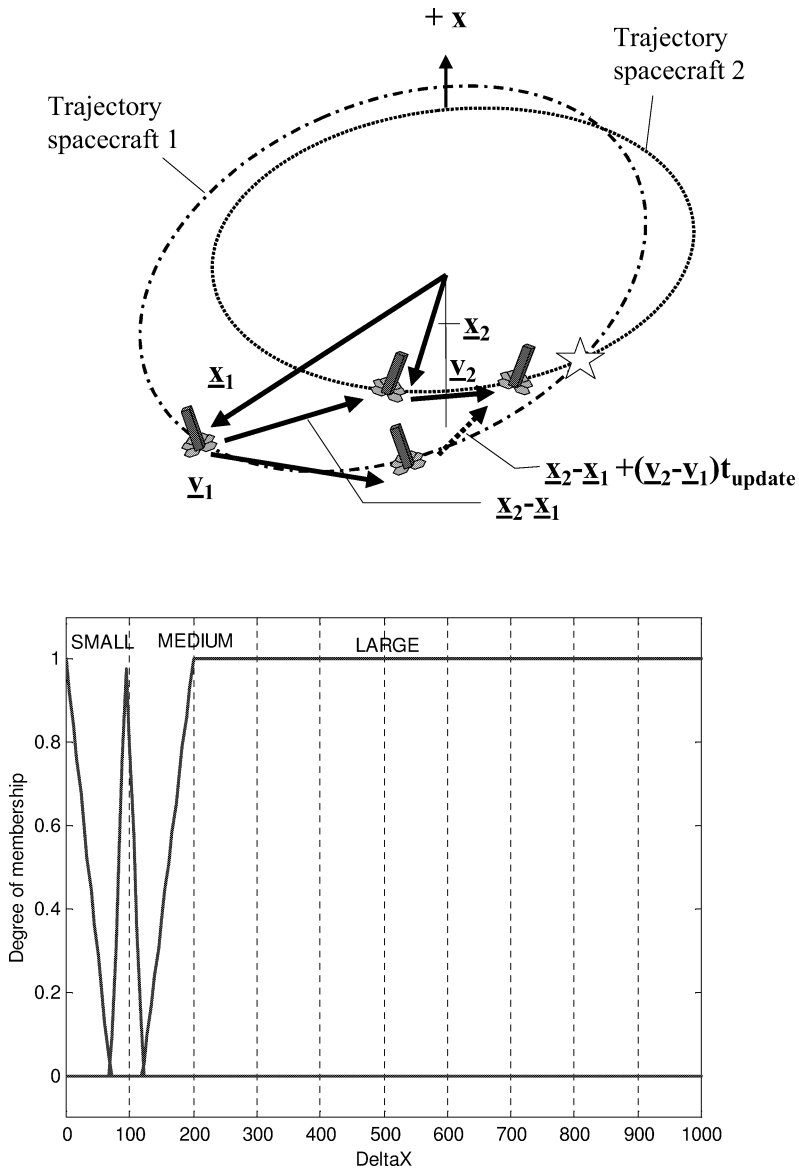


Fig. 4. Calculating of the minimum distance between spacecraft 1 and 2. The "star" symbol denotes a possible collision between the two spacecraft. Also shown is the fuzzy decision variable for collision monitoring.

task is only activated if the input variable `DeltaX` is SMALL. The fuzzy controller then regulates over the membership function HIGH of the output variable `TaskPriority` the priority of the collision avoidance task. Membership functions used for the input and output variables are triangular or Gaussian, as shown in Fig. 4. The membership functions for the

Table 2
Fuzzy inference rules for the collision avoidance agent

Input variable	Output variable	
	Update time?	Collision detected?
DeltaX(LARGE)	HIGH	NO, TaskPriority (LOW)
DeltaX(MEDIUM)	MEDIUM	NO, TaskPriority (MEDIUM)
DeltaX(SMALL)	SMALL	YES, TaskPriority (HIGH)

input variable DeltaX are plotted on the left and for the output variables UpdateTime and TaskPriority in the middle and on the right.

The centralized implementation of this agent is based on one satellite (the leader) using all monitored/sensed information to calculate all possible collisions. The distributed implementation requires all satellites to monitor collisions to itself using its own agent, and then broadcast warnings if there are potential problems. With each satellite monitoring its own potential collisions by monitoring the distances between each individual spacecraft, all potential collisions within the cluster are evaluated, but now using a distributed approach. Assuming that there are n spacecraft in the cluster, the total number of distances to check is then $(n-1)n/2$. If each spacecraft must check their own possible collisions, then $(n-1)n$ must be calculated; therefore, there is a penalty for distribution (more total computations), but benefits as well (no computational bottlenecks on the leader satellite, less information exchanged).

DecMakFailAgent (F23) monitors the health status $\underline{h}$ of the spacecraft to detect failures on-board, and, if required, starts a cluster reconfiguration. This agent requires the DecMakFailSkill.m, and is also implemented with a fuzzy logic controller. The fuzzy logic has four inputs: the health h_s, h_p, h_t of the science, power, thrust subsystems and h_f for the remaining amount of fuel; and three outputs: a variable indicating the status of the particular spacecraft (i.e., '0': spacecraft has failed and '1': spacecraft is working); a decision variable indicating the need for a cluster reconfiguration (i.e., $\leqslant 0.5$: 'NO' and > 0.5: 'YES'); and a decision variable indicating the need for a de-orbit maneuver of the particular spacecraft (i.e., $\leqslant 0.5$: 'NO' and > 0.5: 'YES'). Both update period and the priority of the corresponding tasks ReconfigureCluster (m23) and AllocateRole (m24) are fixed. The DecMakAddAgent (F24) is developed similarly.

3.5. Organizational agents

The SchedulerAgent (F30) uses a TaskPrioritize.m skill that orders items according to their assigned priority. This priority is set by other agents. Resolution of conflictual relationships between tasks is solved by having the task that is the most dominant inhibit the output of the less dominant tasks. A task with a higher priority value therefore suppresses a task with a lower priority. The science task and the cluster reconfiguration task have a fixed priority, whereas collision avoidance and disturbance rejection have a dynamic priority.

PlanReconfigAgent (F31) optimizes new spacecraft positions within the cluster based on maximizing usefulness for science (imaging). This agent, which is called in the

event of a failure or the addition of a new satellite, requires `PlanReconfigSkill.m` and calculates new optimal positions within the cluster for reconfigurations. There are several approaches to implementing this agent based on the science of radar and two current approaches are presented here: (1) symmetric placement, where the phasing between the satellites is the same, or (2) using the Cornwell metric [12], a numerical approach based on N radar targets, M satellites, and the type of target (moving or not). The latter skill can be implemented in a distributed or centralized fashion, based on the numerical algorithm. More details on the algorithm and its implementation can be found in Ref. [12]. The `PlanReconfigSkill` is run only on demand; i.e., is triggered by the message `ReconfigureCluster` (m23) from the decision-making agent cluster reconfiguration (F23).

`PlanAssignAgent` (F32) assigns new spacecraft positions within the cluster based on locations from the `PlanReconfigAgent` (F31), and can be implemented in a centralized or distributed fashion. The assignments are based on a cluster level cost function, which could be a function of fuel, time or other factors that may be important to the particular application. The agents developed here utilize fuel as the basis for the assignments as that is one of the most important factors. The normalized fuel required to maneuver the ith satellite to the jth location is given as

$$d_{ij} = \frac{\Delta V_{ij}}{h_{f,i}}, \tag{3}$$

where $h_{f,i}$ is the remaining fuel on each satellite. The velocity increment can be found by using either a linear program (to be described later in this paper), or classical optimal control and optimization [5]. The agent uses the skill `ffplanner.m` to calculate the fuel required for each maneuver.

The centralized implementation of this agent is based on the leader satellite having approximate knowledge of the remaining fuel for each satellite, using scheduled sharing of health information between satellites. The leader satellite then calculates the fuel required to maneuver to each location using a linear program. For instance, Fig. 5 shows the ΔV required for different orbital reconfiguration maneuvers over all phasing angles φ within the ellipse for a fixed duration time t_{final} of 1000 sec. The ΔV continuously increases from $\varphi = 0^\circ$ to 180° and decreases from $\varphi = 180^\circ$ down to 360°. The plot also shows an anomaly between $\varphi = 190^\circ$ and 250°, a result from the lack of convergence to the absolute minimum in the linear program. For the case of changing the phasing angle φ from $\varphi = 0^\circ$ to 180° (i.e., from one elliptical trajectory to another), the required ΔV is approximately bounded between 0.0075 m/s and 0.01 m/s over all phasing angles φ. After calculation of the d_{ij} cost factors, satellite assignments are made using a resource minimization algorithm, or Fuzzy logic if the variables are uncertain.

There are two approaches to implementing the `PlanAssignAgent` agent in a distributed fashion. The first is using a contract net protocol, or bids from each of the satellites. In this case, the agent (F32) acts as contractor and the remaining spacecraft (i.e., the trajectory planner agents F34) as bidders. The bids arrive at a leader satellite in the form of the normalized cost, d_{ij}. The contract net protocol attempts to assigns locations to satellites in order to best equalize the cost over the full cluster (or among all agents). The search space consists of the initial location of each satellite (initial station) and the possible

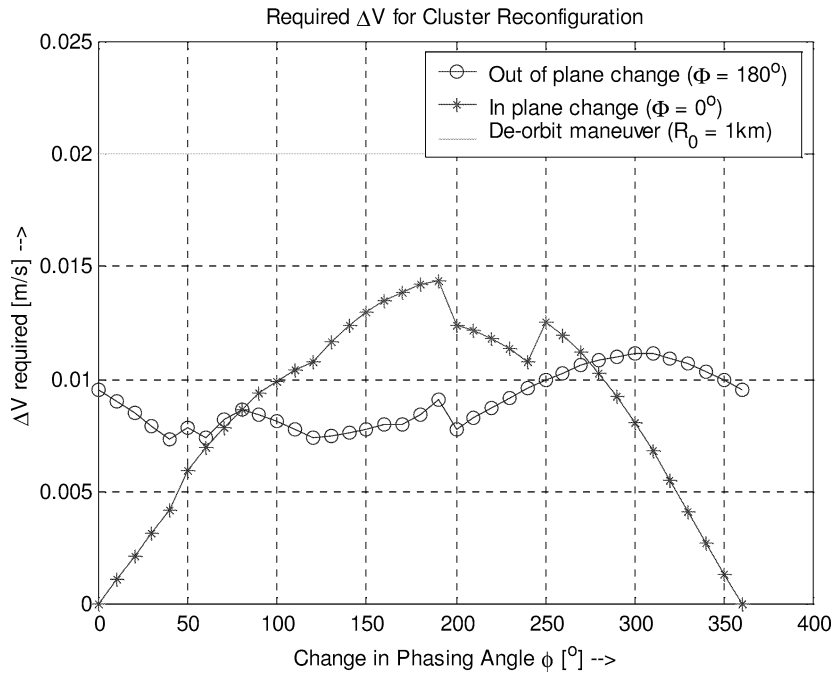


Fig. 5. ΔV requirements for different configuration changes. Each of these make up the bids for the contract net protocol for cluster reconfiguration.

goal locations (new stations) to which the satellite can move. For each satellite there are n possible stations to occupy. Thus, the search space D consists of a $n \times n$ array

$$D = \overset{\text{Goal State } j}{\longrightarrow} \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \cdots & d_{ij} & \vdots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{bmatrix} \quad \downarrow \text{Initial State } i. \tag{4}$$

The approach here is based on scarce resources [14,21], where the following five steps occur:

Step 1: Find the minimum bidder. For each of the goal states j, the bidder corresponding to $d_{\min}(j)$ is

$$d_{\min}(j) = \sum_{i=1}^{n} \min_{1 \leqslant i \leqslant n} (d_{ij}). \tag{5}$$

Step 2: *Find the maximum* (*of the minimum bidder*). Among the set of minimum bidders $d_{\min}(j)$, the maximum (of the minimum) bidder $d_{\max}$ is

$$d_{\max} = \max_{1 \leqslant j \leqslant n} \left(d_{\min}(j)\right) = \max_{1 \leqslant j \leqslant n} \left(\sum_{i=1}^{n} \min_{1 \leqslant i \leqslant n} (d_{ij}) \right). \tag{6}$$

Step 3: *Task Assignment*. For the bidder where $d_{ij} = d_{\max}$ is true, assign the goal state j (i.e., new cluster station of spacecraft j).

Step 4: *Reduce the search space*. After the bidder i and goal state j are assigned, the corresponding rows and columns in D (Eq. (4)) are deleted, and

$$n \to n - 1.$$

Step 5: *Check if all bidders are assigned*. If all bidders are assigned finish, else go to Step 1.

The second distributed approach is a negotiation technique [10], which is used in conjunction with one of the two coordination techniques above. The approach here is that once a nominal plan for cluster assignment is put forth, each of the individual satellites can negotiate based on parameters of each of the maneuvers. For instance, one satellite could optimize its own plan by delaying the start of the maneuver, in an effort to save fuel. This type of calculation is best suited in a distributed fashion because it will require more complex models and therefore more computation. Consider the case when an initial cluster reconfiguration plan has been developed. Fig. 6 shows an example of how the total amount of fuel for a maneuver (ΔV) varies as a function of the maneuver duration time t_{final} using the linear program and the `PlanTrajectAgent`. The amount of control force required increases as the duration time of the configuration maneuver decreases. Therefore, a large amount of fuel must be used in order to speed up the reconfiguration maneuver. Individual satellites may use this information to reduce their own fuel usage without inhibiting the cluster as a whole. As long as the overall cluster characteristics and requirements do not change, distributed negotiation can work quite well.

The `PlanAssignSkill` is run only on demand; i.e., it is triggered by the message `AssignCluster` (m31) from the cluster reconfiguration planner agent (F31).

`TaskAllocAgent` (F33) distributes tasks for the cluster when there is a potential failure with the leader satellite, based on a predefined cost, and can be implemented using a centralized or distributed approach. The centralized approach is based on a logic rule base. For instance, if there is one passive I_1 agent in each cluster, then the logic rule base could contain several priority levels for the nomination, such as:

```
IF (passive I1 agent is alive in cluster of failed I1 agent)
  THEN  (nominate new active I1 in own cluster)
ELSEIF (passive I1 agent is alive in other cluster)
  THEN  (nominate active I1 (old passive) agent in other
         cluster)
ELSEIF (active I1 agent from other cluster is alive)
  THEN  nominate active I1 (old passive) agent from other
        cluster
```

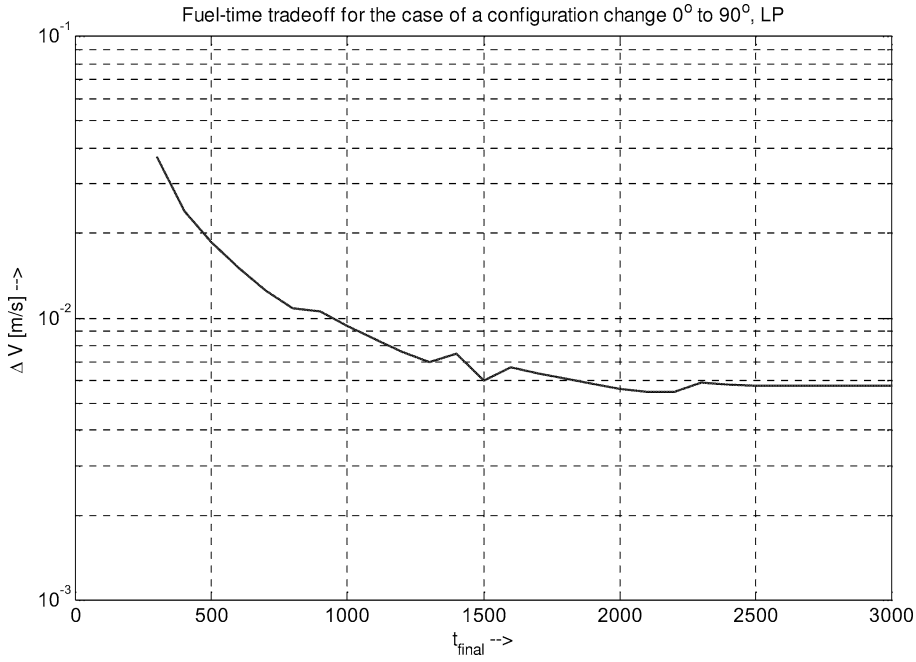


Fig. 6. Fuel-Time tradeoff using the LP-maneuver planer for the case of an in-plane configuration change from $\varphi = 0°$ to 90°. This can be used to negotiate an individual satellites plan.

The decentralized approach makes use of distributed task allocation techniques such as the contract net protocol [7], or negotiation [10] to nominate the "optimal" candidate spacecraft-level agent. The contract net protocol is described here to reconfigure the cluster. As an example, consider an eight satellite cluster, and there is a failure within spacecraft #1 (the leader). In this case, spacecraft #5 acts as contractor in nominating a new leader agent, and the other (passive) spacecraft-level agents I_1 act as bidders. The following steps detail the contract net protocol to this problem:

1. The task allocation planner agent (F33) spacecraft #5 is nominated as contractor for the contract net protocol using a logic based rule base.
2. The contractor sends out requests (`Allocate`) to the sensing agents (F11) on all passive spacecraft-level agents I_1 in the cluster, i.e., to spacecraft #2, #6, and #8, which act as bidders.
3. The bidders can either (`AcceptTask`) or deny (`DenyTask`) the request. The accept or denial is based on the status of the health of the power subsystem h_p. Spacecraft with a health value $h_p < 50\%$ deny the request and spacecraft with $h_p \geqslant 50\%$ accept it. In the case of an accept, the bidder transmits the bid in the form of their spacecraft health values $\underline{h}$, i.e., health values for science, power, thrust and the remaining fuel, to the contractor.
4. The contractor selects a new active spacecraft-level agent I_1 based on the smallest cost from the bidders. An example could be

$$C = \frac{c_1}{h_s} \cdot \frac{c_2}{h_p} \cdot \frac{c_3}{h_t} \cdot \frac{c_4}{h_f}, \tag{7}$$

where c_1–c_4 are weighting factors, chosen on the importance of the different subsystems and/or normalization, and the h's correspond to the health values of the different monitored spacecraft subsystems. In this case, spacecraft #8 is chosen to be the new leader.

5. The contractor transmits an `UpdateClusterInformation` message with a new value for the `ClusterInformation.ScActiveMaster` entry to all spacecraft (i.e., representational agents (F40)) in the cluster, which then update their internal cluster description.
6. The new active master spacecraft-level agent begins its operation.

The `TaskAllocSkill` is run only on demand; i.e., it is triggered by the message `AllocateRole` (m24) from the decision-making agent cluster reconfiguration (m23).

`PlanTrajectAgent` (F34) generates a fuel and/or time optimized control maneuver for a spacecraft. This agent requires `PlanFFSkill.m`, which is implemented using a linear program to calculate a thruster command sequences u_{ff} (d32) and velocity increments ΔV (d31) for time and fuel optimal trajectories. The linear program is implemented in the external function `ffcontroller.m`. The `ffcontroller` function has the following inputs: the start and end time of the orbital maneuver (t_0, t_{final}) and the orbital parameters for the start and final trajectories (initial and final phase angles φ and ϕ and major axis R_0 of the elliptical trajectories).

The linear program [15] is a very flexible approach to planning a trajectory move for a distributed satellite system. A cost function is used, such as minimizing the time or the fuel of the maneuver, with added constraints. The minimum fuel cost is given as:

$$J(x) = \sum_{j=1}^{n} |u_j| \tag{8}$$

where u_j is the thrust used at time j. Minimizing this cost is subject to an initial position $\underline{y}(t_0)$, desired final state $\underline{y}(t_{\text{final}}) = \underline{y}_{\text{ref}}$, and total number of time steps n. Additional constraints include the maximum thrust, such as that imposed by future precision on-off type thrusters [20], maximum position error at t_{final}, and minimum satellite separation to prevent collisions. These are added to the problem as linear inequality constraints, or

$$\begin{aligned} &\left|\underline{u}(t)\right| \leqslant \underline{u}_{\max} \quad \text{on-off thruster,} \\ &\left|\underline{y}_{\text{ref}} - \underline{y}(t)\right| \leqslant \underline{\varepsilon} \quad \text{minimum separation,} \\ &\left|\underline{y}_{\text{ref}} - \underline{y}(t_{\text{final}})\right| \leqslant \underline{\varepsilon_F} \quad \text{minimum final separation.} \end{aligned} \tag{9}$$

A final added benefit of the linear program is that the final position, $\underline{y}(t_{\text{final}})$, can be time varying, which it is in the TechSat21 case where each satellite is rotating about the virtual center (Fig. 1). 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* [15]. Details on this trajectory planner and its implementation in a closed loop formation flying control for multiple

satellites can be found in Ref. [4]. The `PlanFFSkill` is run only on demand; i.e., it is triggered by the messages `CalculateFFcontrol` (m50) from the orbit maneuver agent (F51) or `CalculateDeltaV` (m32) from the cluster allocation planner agent (F32).

`OrbitManAgent` (F40) keeps and continuously updates the internal cluster description. This agent requires the `OrbitManSkill.m`, which is also implemented in F51 (orbit maneuver agent). The cluster description contains the number of active spacecraft in the cluster; the particular tasks that each spacecraft are capable/allowed to carry out (i.e., passive, partial active and active); and the relative position for each spacecraft within the cluster. In addition, the health status of the satellite cluster is monitored, which is critical to many other agents. The health status of the satellite cluster can be formulated as a $n \times m$ array H, where n indicates the number of satellites in the cluster and m the number of health values to consider. In this work, subsystems to be tracked include science devices, power generating modules, propulsion modules including thrusters, and the remaining fuel of each spacecraft. Therefore, H can be written as

$$H = \begin{bmatrix} h_{1s} & h_{1p} & h_{1t} & h_{1f} \\ h_{2s} & h_{2p} & h_{2t} & h_{2f} \\ \vdots & \cdots & \cdots & \cdots \\ h_{ns} & h_{np} & h_{nt} & h_{nf} \end{bmatrix}, \tag{10}$$

where the subscripts s, p, t and f correspond to the science, power, thrust and fuel subsystems respectively. The health variables h_{ix} can take on values in the range from 0 to 1, where a '0' indicates 0% health and a '1' for 100% health.

3.6. Operative agents

`ScienceAgent` (F50) performs the radar imaging task, while `OrbitManAgent` (F51) performs the physical orbital maneuver commands (i.e., thrusting). For orbit maneuvering, the agent uses the thruster command sequence u_{ff} (d32), or a closed loop linear quadratic (LQR) controller [13]. The controller is triggered by setting the appropriate disable/enable flags. The `OrbitManSkill` uses an internal data structure array to store the current cluster description. The `OrbitManSkill` is run only on demand; i.e., it is triggered by the messages `RejectDist` (m21) from the decision-making agent station keeping (m21), `MoveNewPos` (m34), `DeOrbit` (m35) or `UpdateClusterInformation` from the cluster allocation planner agent (F32).

4. Agent based software architectures for multiple satellite systems

With the functional agents and their skills defined, they can be integrated into agent based hierarchies for autonomous control of the satellite clusters. Obviously, many different organizations exist with many levels of autonomy. A detailed comparison of several organizations is given in Ref. [22].

This section presents spacecraft level agents, which is the hierarchical integration of agents on each spacecraft. There are four *spacecraft level agents* presented based on the level of intelligence within the hierarchy of functional agents. This is done primarily to

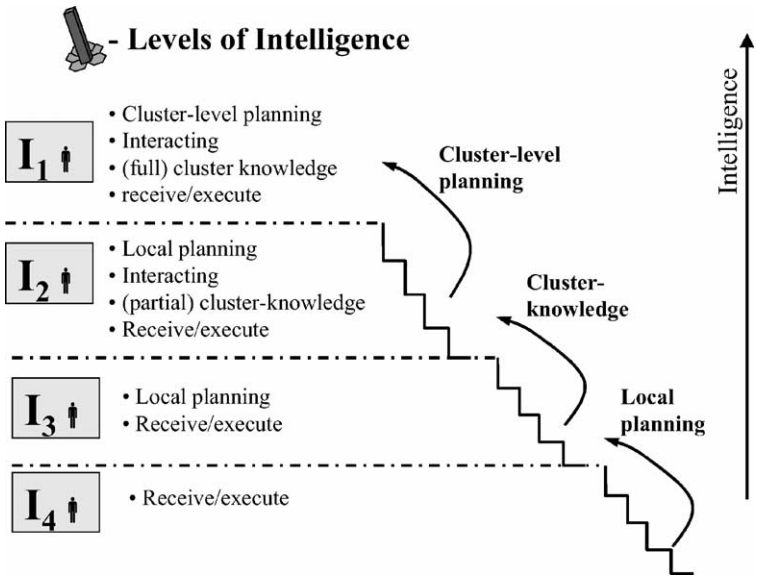


Fig. 7. Identification of spacecraft-level agents based on levels of capable intelligence.

narrow the scope of study. This is followed by a brief section on the agent organizations for clusters along with an example of an actual architecture and the information flow.

4.1. Spacecraft-level agents

In order to narrow the scope of study of agents for multiple spacecraft, spacecraft-level agents are defined as a function of their level of intelligence. Based on the sum of capable spacecraft functions, four levels of intelligence have been identified, where I_1 denotes the highest level of intelligence and I_4 the lowest level (Fig. 7).

The spacecraft-level agent I_4 represents the most "unintelligent" agent. It can only receive commands and tasks from other spacecraft-level agents in the organisation or from the ground and execute them. An example includes receiving and execution of a control command sequence to move to a new position within the cluster. This type of intelligence is similar to what is being flown on most spacecraft today.

The next higher spacecraft-level agent is I_3, which has local planning functionalities on board. "Local" means the spacecraft-level agent is capable of generating and executing only plans related to its own tasks. An example includes trajectory planning for orbital maneuvers in case of a cluster reconfiguration. This type of intelligence is similar to DS1 [16].

Spacecraft level agent I_2 adds a capability to interact with other spacecraft-level agents in the organisation. This usually requires the agent to have at least partial knowledge of the full agent-based organisation, i.e., of other spacecraft-level agents. It must therefore continuously keep and update (or receive) an internal representation of the cluster (agent-based organisation). An example includes coordinating/negotiating with other spacecraft-level agents in case of conflicting requirements or enhancing performance.

The spacecraft-level agent I_1 represents the most "intelligent" agent. The primary difference between I_1 and other spacecraft-level agents is that it is capable monitoring all spacecraft-level agents in the organisation and planning for the organisation as a whole. This requires planning capabilities on the cluster level as well as having full knowledge of all other spacecraft-level agents in the organisation. An example includes calculation of a new cluster configuration and assigning new satellite positions within the cluster.

4.2. Organizations of spacecraft level agents

In order to develop a coherent working community within the cluster such that all of the necessary capabilities can be achieved, the organization must be designed very carefully. In addition, it must be adaptable to prevent faults, avoid bottlenecks, and allow reconfiguration. It must be efficient in terms of time, resources, information exchange and processing. And it must be distributed in terms of intelligence, capabilities and resources.

Again in order to narrow the scope of multi-agent systems on multiple spacecraft, several generic organizational levels are defined. The possible organizations for spacecraft-level agents include:

- Top-down coordination architecture,
- Centralized coordination architecture,
- Distributed coordination architecture,
- Fully Distributed coordination architecture.

Fig. 8 shows a summary of the four possible coordination options mentioned above for a spacecraft-level agent team, as a *function of individual, capable spacecraft-level agent intelligence*. The blocks represent the spacecraft-level agents labeled according to the capable level of intelligence required for the organization. As can be seen, the number and composition of the different spacecraft-level agents I_1–I_4 determines the organizational architecture. The top-down coordination architecture includes only one single spacecraft-level agent I_1 and the other spacecraft are 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 instead. The distributed coordination architecture consists of several parallel hierarchical decision-making structures, each of which is "commanded" by spacecraft-level agent I_1. Note that the different spacecraft-level agents I_1 can interact with each other as well as with 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.

Note also that the specific functional agents that make up the spacecraft level agents are shown in Table 1.

4.2.1. Top-down coordination architecture

In a simple top-down coordination architecture, similar to a Master-Slave Organization, agents are coordinated in a hierarchical fashion where the spacecraft-level agents I_1 at the top of the hierarchy make the majority of the intelligent group decisions. Then, decisions

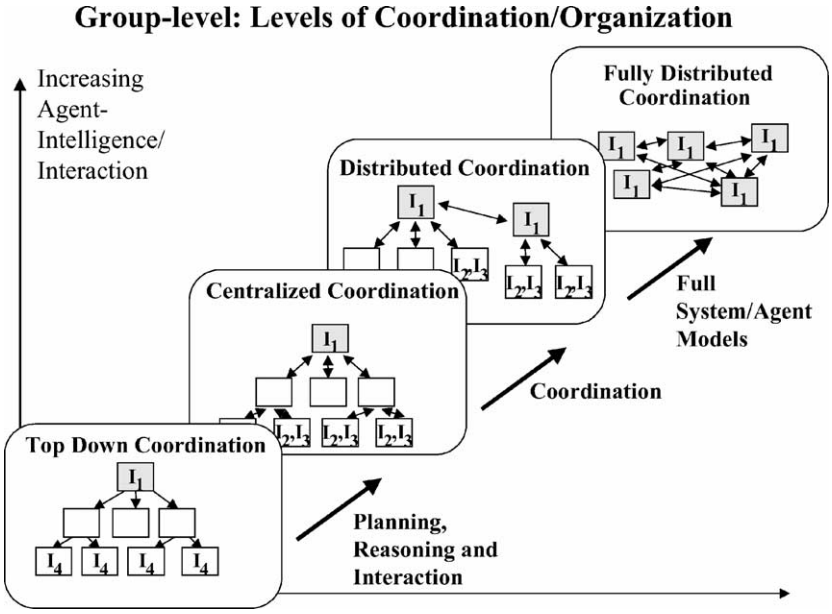


Fig. 8. Coordination architectures for coordination of multiple spacecraft-level agents.

are passed down to the rest of the spacecraft-level agents that are I_4 agents where the commands are executed. This organization is fairly rigid since it has one centralized intelligent spacecraft-level agent. But it is also the most straight-forward to implement as it requires almost no communication between the spacecraft-level agents I_4 (agents at the bottom of the hierarchy) because these agents exercise no group intelligence.

Applied to the TechSat21 example, one of the eight spacecraft is considered to have higher "intelligence" and therefore acts as a spacecraft-level agent I_1. (Even though they may be exactly the same for redundancy purposes, one is chosen as the leader.) Fig. 9 shows this scenario. Tasks of the I_1 spacecraft include high-level decision making, planning and scheduling for the cluster as well as performing all lower level tasks for the cluster.

In addition, the spacecraft-level agent I_1 serves as the communication center for information flow within the cluster. The remaining seven spacecraft form slaves and are "unintelligent" spacecraft-level agents I_4, and therefore only receive and execute commands. Note that each of the individual spacecraft-level agents I_4 transmits its state vector x_n and its health h_n to the spacecraft-level agent I_1 at a particular sampling rate. The master spacecraft then evaluates, plans and schedules for the cluster, and sends back the particular control output y_n to each spacecraft-level agent I_4.

4.2.2. Centralized coordination architecture

With new developments in on-board planning and reasoning, a new organization can be developed, termed "centralized coordination architecture", where a centralized hierarchy is still used, but now the underlying agents have increased intelligence and can interact with higher level agents for the betterment of the intelligent agent team. For instance, lower-

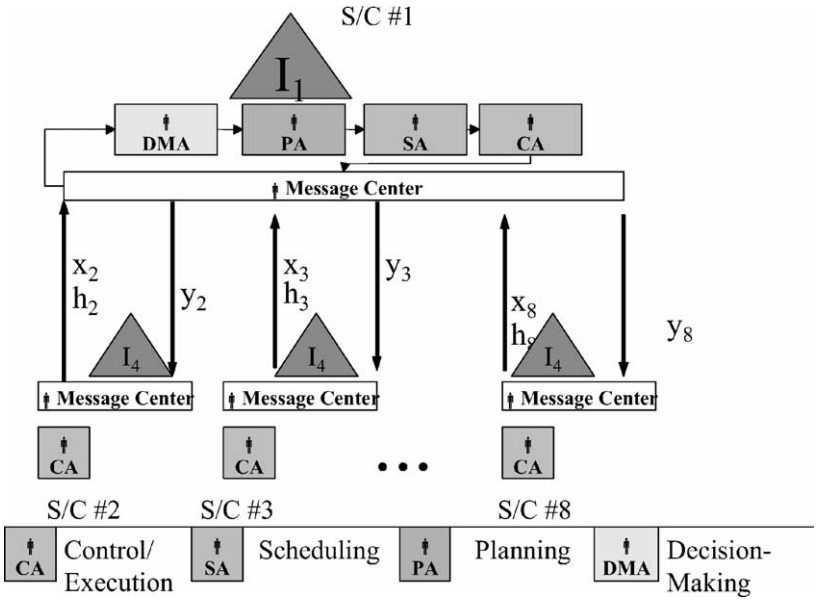


Fig. 9. Top-down coordination architecture with I_4 spacecraft-level agents with little intelligence and an I_1 spacecraft-level agent with high intelligence.

level agents may formulate plans for a part of or the entire organization to follow and send these up the chain of the organization. Then, a centralized spacecraft-level agent I_1 decides on the best plans. This organization is more complex and requires more communication between agents, but the intelligence is better distributed throughout the system. This makes for a more flexible, adaptive, and efficient organization.

Fig. 10 shows this organization as applied to the TechSat21 mission. In comparison to the previous organization, the lower level spacecraft agents are at the I_2 level. These spacecraft now have increased intelligence to allow them to perform low-level decision making and planning of basic tasks, as well as to interact with other agents. These tasks include rejecting disturbances or performing the science task (i.e., imaging). The I_1 level agent still performs the higher-level planning and decision making for the cluster as a whole, then sends particular tasks to the lower level spacecraft agents I_2. As an example of a high level task, the I_1 level spacecraft will plan a new reference position y_{ref} for each spacecraft in the case of a reconfiguration of the cluster or position vectors y_k in the case of collision avoidance. Each I_2 spacecraft-level agent is then responsible to perform all control tasks, low level decision making and planning.

Lower level spacecraft-level agents I_2 can now send results and other information pertaining to their tasks or reasoning processes back to the higher level I_1 agent. Consider the case when a higher level spacecraft-level agent I_1 has sent out a list of possible cluster positions for relative reconfiguration. The task of the lower level spacecraft-level agents I_2 is then to calculate the required velocity increment ΔV for each position and send it back to the I_1 level spacecraft agent, along with information about its remaining fuel h_f. The high-level planner of the spacecraft-level agent I_1 then attempt to equalize the fuel use

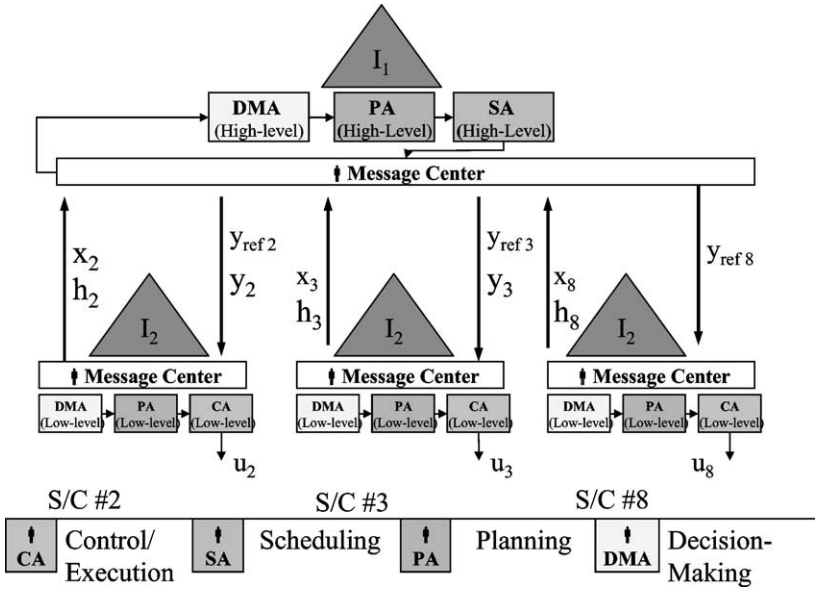


Fig. 10. Centralized coordination architecture with lower level agents having the capability to perform low-level tasks, including decision making, planning and control. They can also interact with the higher level agent.

across the cluster when assigning each spacecraft to a particular position within the cluster. In this way, the planner is now distributed.

4.2.3. Distributed coordination architecture

The next type of organization allows the agents to coordinate together in a distributed coordination architecture. This is a more ideal case for an intelligent team of agents, as they take full advantage of their capabilities in terms of adaptability, distribution and intelligence. The distributed organizations make full use of the distributed coordination algorithms presented in Table 1, such as using the contract net protocol or negotiation techniques for assignments and reconfiguration. The advantages of using these tools are that both performance and robustness can be improved, communications bottlenecks can be elimited, and computations can be distributed. The disadvantages are that the total computation increases.

The contract net protocol is an excellent example for utilizing the distributed coordination architecture. If an agent has a goal which it is trying to realize, it may attempt to enlist other agents with unused resources to help it accomplish its plan. Agents can contract other agents that are available, so that a small group works to fulfill the goal of a single agent. This approach requires large communication costs between agents, but it allows high-level planning of goals, flexibility in achieving the goals, and natural load balancing within the multi-agent system.

Once a group has determined a plan for all the agents of the group, another approach is to allow each agent to try to improve the overall group plan. Each individual agent attempts to modify the group plan so that it can achieve its goals more efficiently. This

plan negotiation is useful in that it distributes the intelligent process of optimizing a group plan among all agents within an organization, but it is complicated in that it requires high-levels of intelligence in each agent and substantial communication between all agents.

4.2.4. Fully distributed coordination architecture

One can extend these organizational ideas to the point where each agent in the system has "full group intelligence" where any agent has intelligence equal to any other agent. In this "fully distributed coordination" architecture, there is no hierarchy, meaning the organization is flat and fully distributed. But in order to achieve this, there must be extensive communication between all agents in the system. This has the advantage of being highly adaptable and very reliable, as any agent can exercise intelligence for the entire system as well as any other agent (so a decision never has to be passed to other agents). But the organization is complex and requires elaborate inter-agent communications. Fig. 11 shows this concept applied to the TechSat21 mission. Each spacecraft represents a spacecraft-level agent I_1 and there is no fixed structure that defines information flow and distribution of functionality. Instead, the agents now must coordinate between each other in order to achieve goals or perform tasks.

A final important issue with agent hierarchies for multiple spacecraft is redundancy. Spacecraft, unlike many other applications, requires very high reliability because of the far proximity of the system. Thus, most systems are usually very redundant. With the added software complexity of agent based systems, there must also be redundancy in the software architecture. Therefore, within the organizations, passive I_1 agents are used to increase the

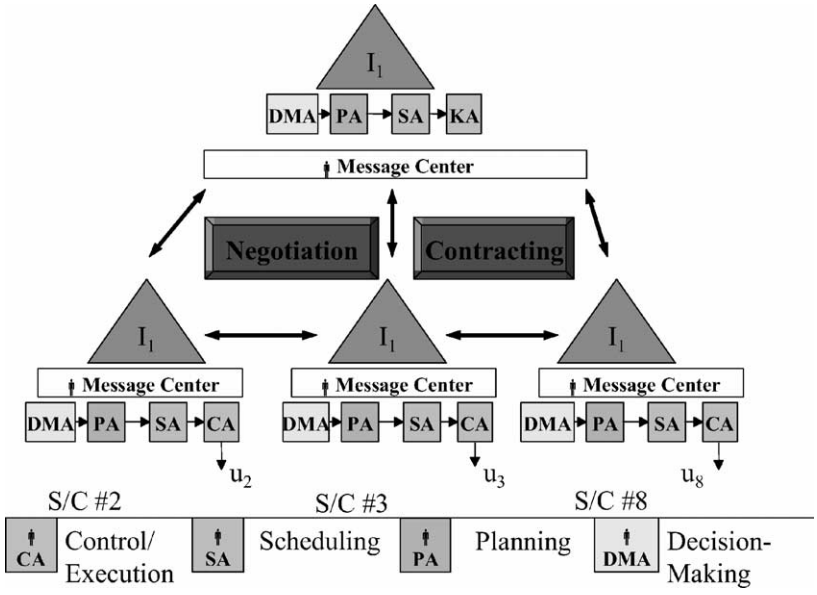
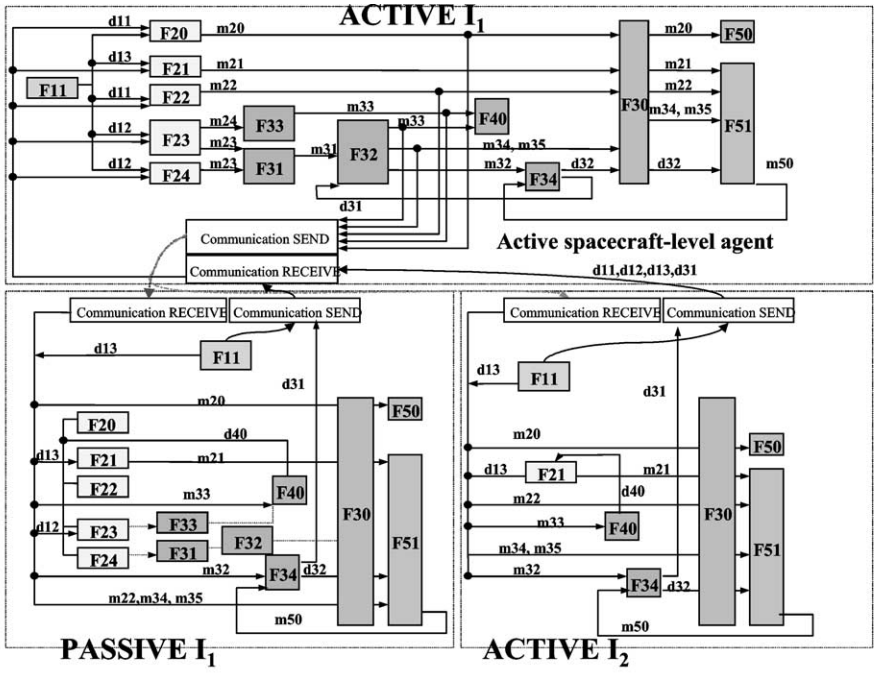


Fig. 11. Fully distributed coordination architecture. Each agent represents a spacecraft-level agent I_1 and has the same "intelligence" and capabilities.

reliability of the system as a whole. Their primary functions, however, are at the I_2–I_4 levels.

4.3. Architectures and information flow

Information flow between the agents are based on the *data* required to perform their actions and/or the *messages* that trigger corresponding tasks. Fig. 12 shows the information flow architecture for a distributed architecture, and the distribution of the functional agents onto spacecraft-level agents. A "passive" spacecraft level agent (such as passive I_1) indicates a redundant agent, or an agent with more intelligent capability (I_1), but acts with lower intelligence (I_3). Also, "m" refers to a message, and "d" refers to data.



Legend for functional agents:

F11 Sensing	F20 Dec.-mak. science	F21 Stat. keeping
F22 Coll. avoidance	F23 Dec.-mak. reconfig.	F24 Dec.-mak. upgrade
F30 Scheduler	F31 Cluster reconfig.	F32 Cluster assignment
F33 Task allocation	F34 Trajectory planner	F40 Representational
F50 Science	F51 Orbit maneuver	
m21 Station Keeping	m22 Collision avoidance	m23 Cluster reconfiguration
m24 Assigning of roles/tasks	m31 Cluster assignment	m32 Delta V calculation
m33 Update internal state	m34 Move to new position	m35 De-orbit s/c
m40 FF control generation		

Fig. 12. Information flow architecture for a distributed coordination architecture, with active and passive spacecraft-level agents I_1, and spacecraft-level agent I_2.

Table 3
Messages within ObjectAgent with corresponding verbs, sources and sinks

Identification	Verb required	Action	Source	Sink
m21	`RejectDist`	Station keeping	F21	F51
m22	`MoveCollAvoid`	Collision avoidance	F22	F51
m23	`ReconfigureCluster`	Cluster reconfiguration	F23	F31
m24	`AssignRole`	Assigning of roles/tasks	F23	F33
m31	`AssignCluster`	Cluster assignment	F31	F32
m32	`CalculateDeltaV`	ΔV calculation	F32	F34
m33	`UpdateClusterInformation`	Update internal state	F32, F33	F40
m34	`MoveNewPos`	Move to new position	F32	F51
m35	`DeOrbit`	De-orbit S/C	F32	F51
m40	`CalculateFFControl`	FF control generation	F51	F34

Table 4
Extract of defined data with corresponding content, sources and sinks

Ident.	Content	Description	Source	Sink
d11	$\underline{x}$	State vector	F11	F20, F22
d12	$\underline{h}$	Health	F11	F23, F24
d13	$\underline{err}$	Position error	F11	F21
d31	ΔV	Velocity increment	F34	F32
d32	$\underline{u}_{ff}$	Feed forward control sequence	F34	F51
d40	$\underline{ci}$	Internal cluster description	F40	F2X

Each lower level spacecraft agent performs local planning and decision-making, and interacts with the higher-level agent I_1 when a reconfiguration is required. Each spacecraft-level agent performs its own station keeping, F21, monitors the relative position error (d13) and produces, if required, a `RejectDist` message (m21) that triggers a station keeping task. Additionally, each spacecraft-level agent runs its own trajectory planner agent (F34) for the generation of the feed forward control sequence (d32). The primary difference lies in the case of a cluster reconfiguration, where each spacecraft-level agent interacts with the central spacecraft-level agent I_1. To assign new positions within the cluster, the spacecraft-level agent I_1 requests bids from each spacecraft by transmitting a `CalculateDeltaV` message (m32). Each spacecraft then submits a bid to the cluster allocation planner agent (F32) on the central spacecraft-level agent I_1 in form of the velocity increment (d31) required to move to these new positions. The latter then decides upon an optimal cluster assignment based on the received bids. If a failure within an intelligent I_1 level agent occurs, a dynamic reconfiguration mode is used to create a new organization of spacecraft-level agents using the task allocation planner agents (F33).

A summary of messages, data and agents is given in Tables 3 and 4.

5. Simulation results

A series of simulations have been developed to test and evaluate the contributions of this work, including the functional agent development, hierarchical agent integration,

and the message passing concept. This implementation is a very important first step in the realization of the agent based software for multiple satellites. As shown by the DS1 experience, however, a full real time implementation of the software is critical to its eventual success. The work here not only is an important initial step in the development of software for multiple satellite systems, but also allows easy comparisons of different software architectures.

A spacecraft simulator was developed in the MATLAB/Simulink environment. The agent based software was also developed in this environment, and there are specific interfaces between the two based on a real spacecraft system (thruster commands, communication cross-link, uplink, and downlink, sensor measurements, etc.). Three examples are shown to demonstrate specific technologies and compare centralized and distributed organizations: (1) Failure of a spacecraft and cluster reconfiguration, (2) Fuel Savings using Negotiation, and (3) Task priorities in collision avoidance.

5.1. Reconfiguration: Failure of a spacecraft component

The first case study is the reconfiguration of a system when a high level, intelligent spacecraft agent I_1 has failed. This is shown in Fig. 13 for both centralized and distributed organizations. In both cases, the active spacecraft-level agent I_1 (spacecraft #1) has failed, and the organization must be reconfigured.

The centralized case makes use of the centralized agents for collision avoidance, reconfiguration, assignments, and task allocation, as shown in Table 1 and described in the previous section. Most of these algorithms are rule based, with information flowing from the leader satellite (I_1) to the lower level satellites (I_3). The distributed organization

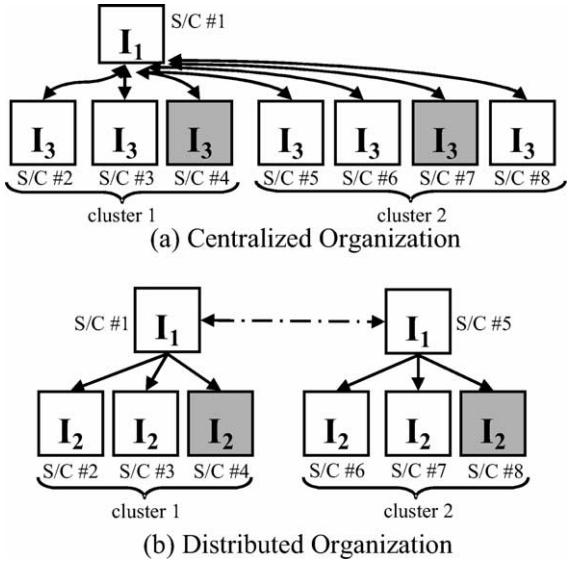


Fig. 13. Centralized (a) and distributed (b) coordination architecture for TechSat21. In both cases, spacecraft #1 has failed and the cluster must be reconfigured.

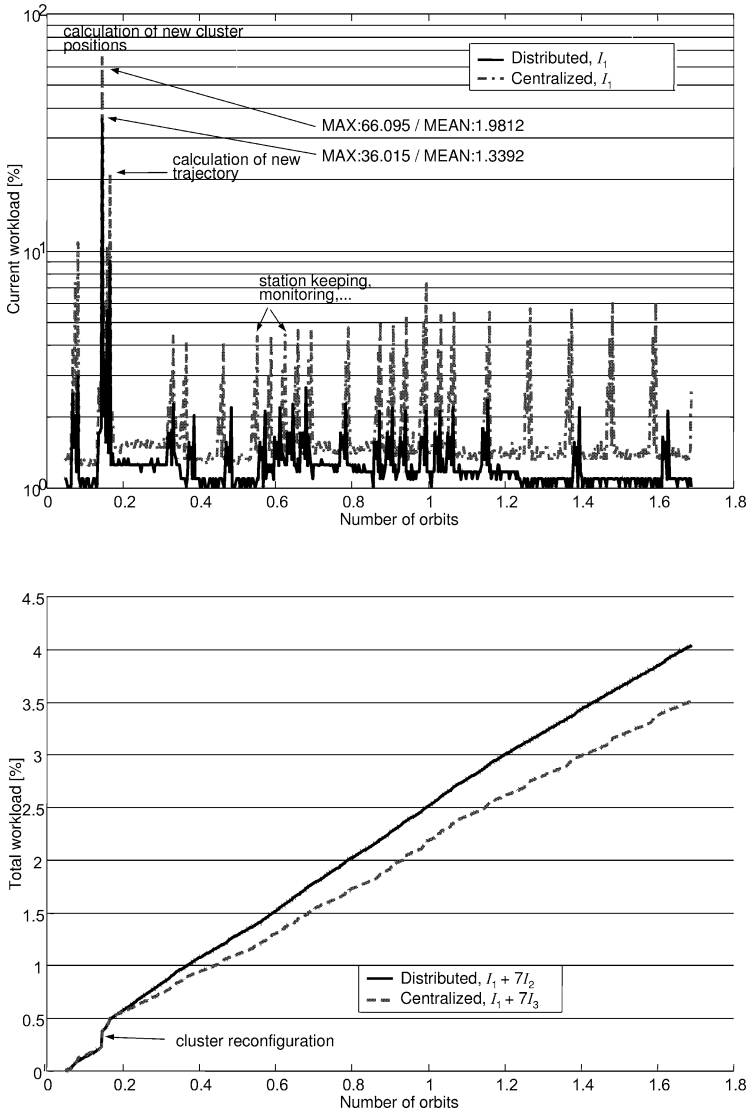


Fig. 14. Leader satellite CPU workload and cluster total CPU time comparing the centralized and distributed organizations.

makes use of the distributed approaches (such as the contract net protocol) to these agents, as shown in Table 1 and described in the previous section.

As part of the simulation and comparison, the workload (i.e., CPU time) and communication effort (i.e., transmitted bytes) are compared for both cases. Fig. 14 shows the average and total CPU time, while Fig. 15 shows the average and total communications load. The centralized case has a higher peak computational usage, but a lower average

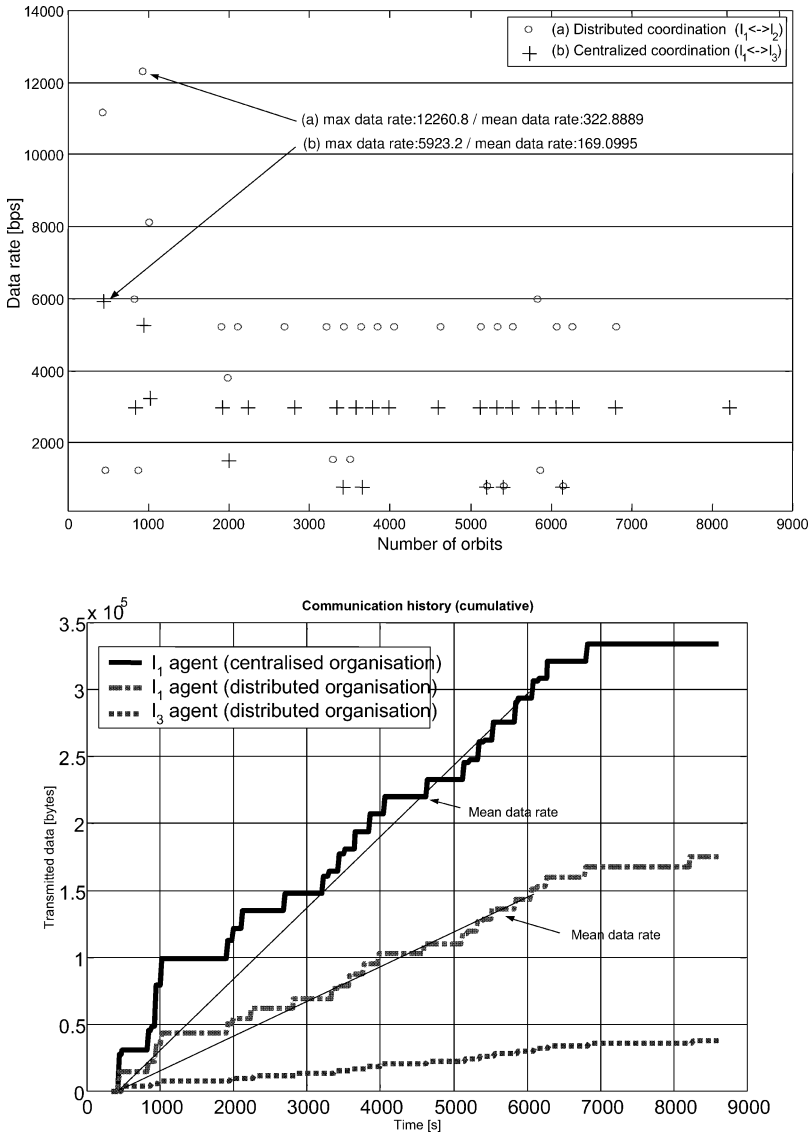


Fig. 15. Communication data rates and total data sent for the centralized and distributed organizations.

because the lower level spacecraft do not perform intense computational functions. The communications, however, is larger in the case of the centralized coordination because the higher level spaceraft must send specific commands as a function of time to each of the lower level spacecraft. Note that if the bidding mechanism of the distributed case required more bids, or negotiation was allowed to run more often, the communication load of the distributed case would be higher.

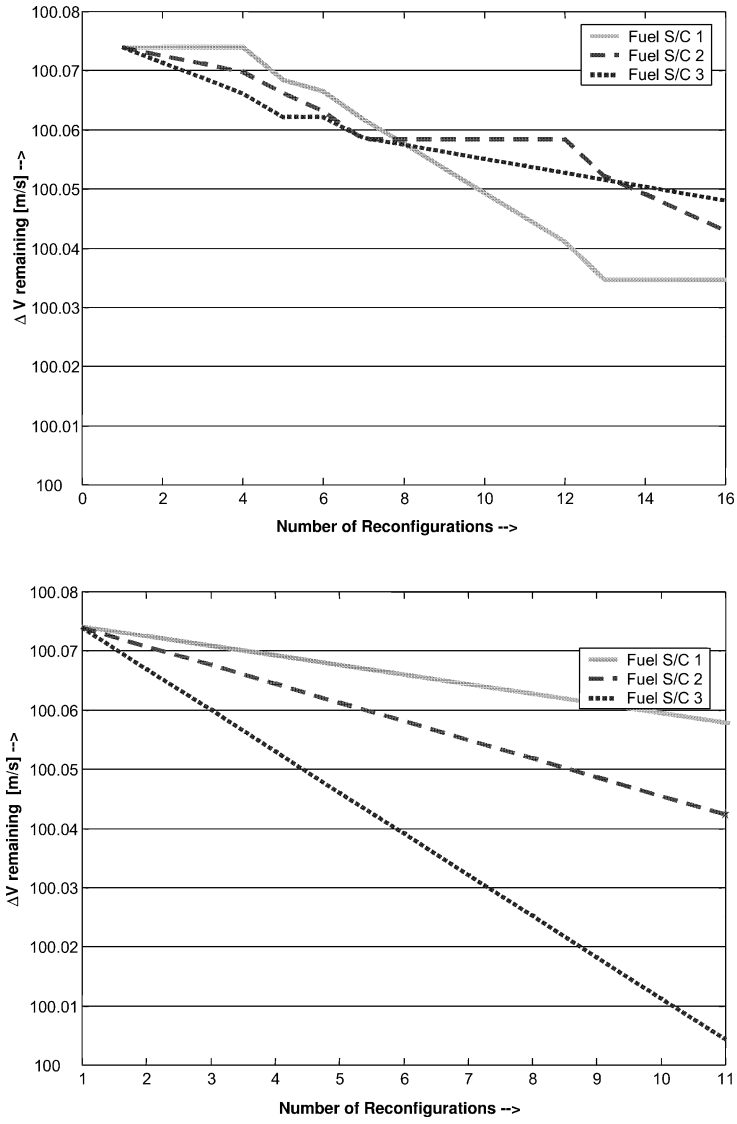


Fig. 16. Fuel usage for cluster reconfiguration for the distributed organization using contract net protocol (top), and a centralized case using simple assignments (bottom).

5.2. *Reconfigurations: Fuel savings*

The simulation for the reconfiguration case shows that the agent architecture works well. Although the benefit of the distributed case is not explicitly clear in the previous example, it is in the case of nominal operations, where reconfigurations of the cluster into different formations for the science take place frequently. Fig. 16 shows the fuel usage

for three satellites as a function of the number of science reconfigurations for both the centralized and distributed cases. Notice that the contract net protocol described in the previous section works well to (1) equalize resources across the cluster, and (2) minimize the total fuel usage across the cluster. The CPU and communication histories are similar to those in Figs. 14 and 15. Therefore, the two examples show that advantages in fuel savings and similar metrics are traded for computational and communication effort. In general, it appears that satellite clusters ruled by distributed organisations are more flexible and adaptive to changes. It does, however, increase the complexity of the software, and this must be traded against the higher lifetime that results from equalizing the fuel across the satellites. The interaction between the satellites in an intelligent manner also opens up other possibilities, such as exchanging and using overall satellite health information. As an example, consider the case when multiple new target are to be imaged. The clusters can dynamically group themselves for the targets based on health, fuel, and image quality, and then move on to the next set of targets.

5.3. Conflict resolution: Collision avoidance

The final simulation shows the autonomous operation of the multi agent system for the case of a collision avoidance maneuver between two spacecraft, followed by a cluster reconfiguration. This case studies a conflict resolution between more than one agent for a distributed organization. Conflictual relationships between tasks and agents arise when they can be run in parallel. The sub-level tasks ST11 (science), ST21 (rejecting disturbances), ST22 (collision avoidance) and ST23 (orbit maneuvering) occasionally require execution at the same time. A conflict resolution is therefore required. In this case, the monitoring for a collision and subsequent orbit maneuver must override the priority of the science (radar processing).

The simulation considers one cluster with one active master spacecraft (spacecraft #1) and three slave spacecraft (spacecraft #2–4). Fig. 17 shows the steps of this simulation, including the trajectories for the spacecraft #1 and #2. The following steps are simulated:

Step A A failure occurs within spacecraft #2, and it begins to drift towards spacecraft #1 in the cluster.

Step B The collision checking agent detects the possible collision between spacecraft #1 and #2 and initiates a collision avoidance maneuver between the two spacecraft.

Step C A deorbit of the failed spacecraft #2 is performed (meaning that it is placed on an elliptical trajectory with a larger major axis).

This is a difficult task for traditional approaches where spacecraft commands are received only from the ground. The close proximity of the satellites requires more autonomy with collision avoidance.

The resolution of conflictual relationships between tasks is implemented using an approach similar to the subsumption architecture [2,3]. When a conflict occurs, the task with a higher priority suppresses a task with a lower priority.

Fig. 18 shows the priority for the tasks `PerformScience` (m20), `RejectDist` (m21), `MoveCollAvoidance` (m22), and `ReconfigureCluster` (m23) as a func-

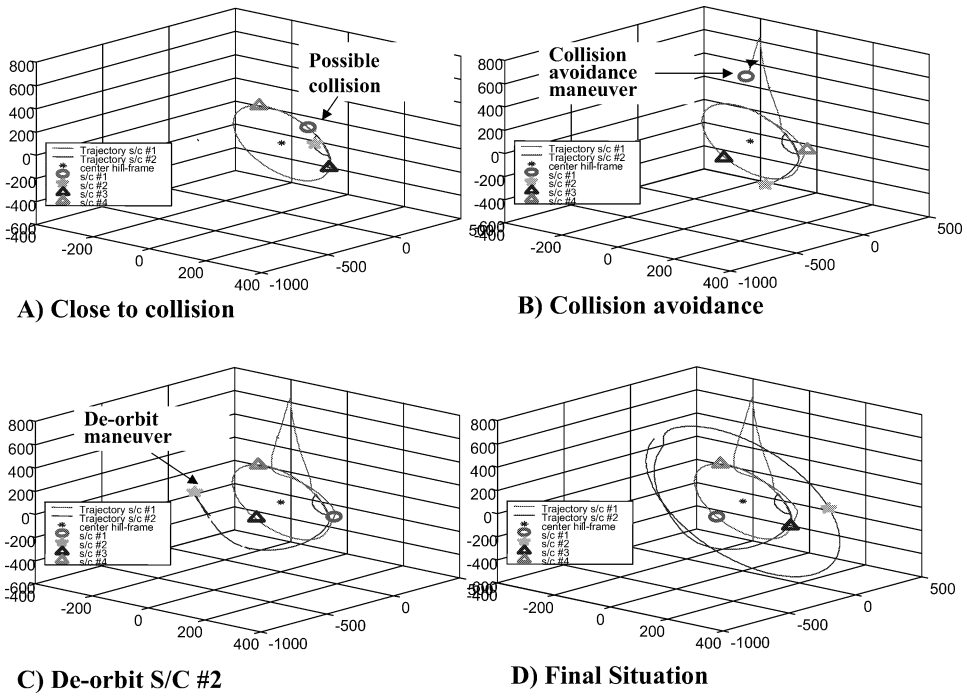


Fig. 17. A 3D animation of the collision avoidance simulation with a cluster reconfiguration. Shown are four snapshots of the scenario for the agent based organisation: (A) Close to collision, (B) Collision avoidance, (C) De-orbit of S/C #2, and (D) Final situation.

tion of the degree of membership of a fuzzy output variable. This variable is the prime factor within the decision-making skill. The science task and the cluster reconfiguration task have a fixed priority because the science is always performed in a healthy situation, and the cluster is always reconfigured as new targets arise. The collision avoidance and disturbance rejection tasks, however, have a dynamic priority, depending on whether a collision is imminent or if a disturbance has been measured and requires action. Collision avoidance and cluster reconfiguration can have a higher priority than disturbance rejection or science because they must be accomplished prior to all other tasks.

Using different values for membership functions, the intersection points for the task priorities (points “A” and “B” in the figure) can be regulated. For example, it is natural that the relative distance between two spacecraft can become smaller during a reconfiguration maneuver. However, the collision avoidance task is activated only when the relative distance between the spacecraft reaches a certain limit (A). Similarly, the science task must be canceled if the error between actual and reference position of the spacecraft reaches a point at which the radar imaging task is not possible (B).

Fig. 19 shows the CPU and communication workload. The CPU workload obviously increases near the collision avoidance detection and maneuver generation. The communication increases only slightly near the maneuver time.

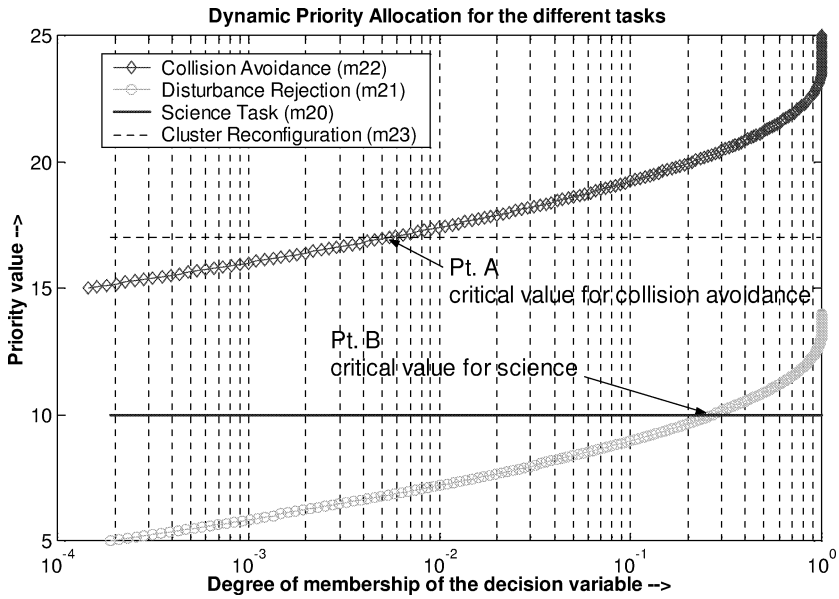


Fig. 18. Resolution of the conflictual relationships between different tasks using dynamic allocation of the priority of the corresponding tasks.

As a final note, Ref. [22] gives a very detailed comparison of these four types organizations using this multi-agent approach, along with a more traditional system that is highly dependent on specific commands from the ground. Ref. [22] includes a description of metrics for comparison (such as communication, computation, reliability) and a variety of scenarios (nominal operations, reconfiguration, collision avoidance). This paper addresses the underlying technologies of the agents, the software infrastructure, and the details of each of the organizations. Based on the results in Ref. [22], the trends indicate that the distributed organization is the best for multiple satellite systems. Please see Ref. [22] for more details.

6. Conclusions

A software architecture for multiple satellite autonomy using a message passing simulation environment (ObjectAgent) for multi-agent systems has been presented. The required functional software agents have been developed and integrated into a complex, yet enabling software architecture for multiple spacecraft systems. Tools such as fuzzy control, linear programming and the contract net have been used for the implementation of the functional agents and agent-based organisations. Conflict resolution between the agents is accomplished by using of a dynamic priority allocation for the tasks. ObjectAgent is well suited for the simulation of multi-agent based systems applied to the space domain. Thus, quick comparisons and design evaluations are critical, all of which can be accomplished in the environment. Initial results show that the multi-agent approach is promising for

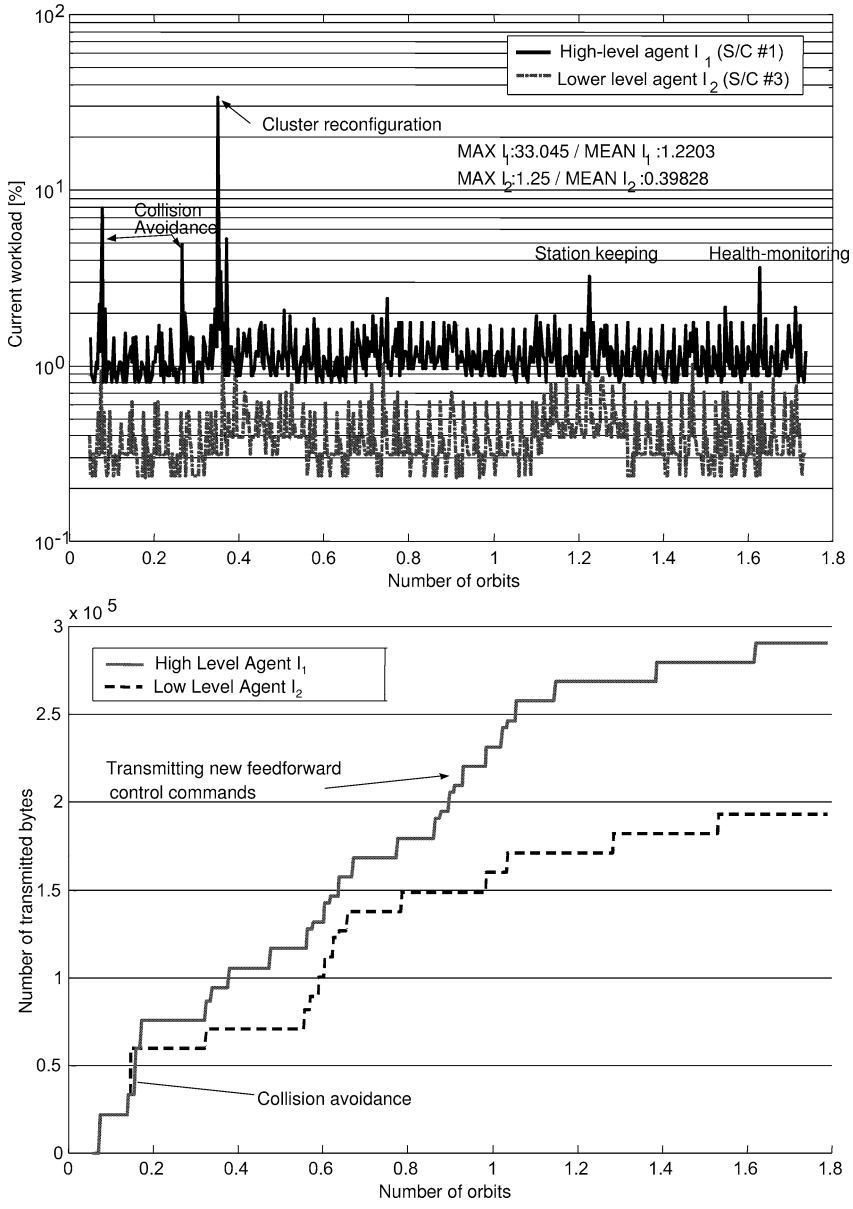


Fig. 19. CPU workload and total communication data for the collision avoidance maneuver using a distributed agent based organization.

these systems because it can prolong lifetime and enhance performance. Further study into software reliability, especially for multiple spacecraft systems in closed proximity, is an important future issue to address.

Acknowledgements

This work is supported under an United States Air Force SBIR contract with Princeton Satellite Systems, Contract Number F29601-99-C-0098.

References

[1] R.A. Brooks, A robust layered control system for a mobile robot, IEEE J. Robotics and Automation RA-2v1 (1986) 14–23.
[2] R.A. Brooks, Achieving Artificial Intelligence through building robots, MIT AI Lab Memo 899, 1986.
[3] R.A. Brooks, Elephants don't play chess, Robotics and Autonomous Systems 6 (1990) 3–15.
[4] M.E. Campbell, T. Schetter, Formation flying mission for UW Dawgstar Satellite, in: IEEE Aerospace Conference (Big Sky, Montana), 2000.
[5] M.E. Campbell, Planning algorithm for large satellite clusters, in: Proceedings, AIAA Guidance, Navigation and Control Conference (Monterey, CA), 2002.
[6] A. Das, R. Cobb, M. Stallard, A revolutionary concept in distributed space based sensing, in: AIAA Defense and Civil Space Programs Conference & Exhibit (Huntsville, AL), 1998, pp. 1–6.
[7] R. Davis, R. Smith, Negotiation as a Metaphor for distributed problem solving, Artificial Intelligence (1983) 63–109.
[8] M.A. Dornheim, Deep space 1 launch slips three months, Aviation Week and Space Technology (April 27 1998) 39.
[9] K.S. Evans, C. Unsal, J.S. Bay, A reactive coordination scheme for a many-robot system, IEEE Trans. Systems Man Cybernet. 27 (4) (1997) 598–610.
[10] S. Green, L. Hurst, B. Nangle, D.P. Cunningham, F. Somers, D.R. Evans, Software Agents: A Review, version 1.0 ed., Trinity College Dublin, Broadcom Eireann Research Ltd., May 1997.
[11] G. Hill, Researches in the Lunar theory, Amer. J. Math. 1 (1) (1878) 5–26.
[12] E.M. Kong, M.V. Tollefson, J.M. Skinner, J.C. Rosenstock, Techsat 21 cluster design using AI approaches and the Cornwell metric, AIAA Paper AIAA-99-4635, 1999.
[13] H. Kwakernaak, R. Sivan, Linear Optimal Control Systems, Wiley-Interscience, 1972.
[14] D.B. Morton, N. Weininger, D.J.E. Tierno, Collective management of satellite clusters, in: Proceedings of the AIAA Conference on Guidance, Navigation and Control (Portland, OR), 1999, pp. 1576–1584.
[15] K.G. Murty, LINEAR Programming, Wiley, New York, 1983.
[16] N. Muscettola, P.P. Nayak, B. Pell, B.C. Williams, Remote agent: To boldly go where no AI system has gone before, Artificial Intelligence 103 (1–2) (1998) 5–47.
[17] NASA Origins Program, http://origins.jpl.nasa.gov/.
[18] T.H. Nguyen, M. Sugeno, R. Tong, R.R. Yager, Theoretical Aspects of Fuzzy Control, Wiley, New York, 1995.
[19] Princeton-Satellite Systems Web-Page, http://www.psatellite.com.
[20] C. Rayburn, M. Campbell, A. Hoskins, J. Cassady, Development of a micro-PPT for the Dawgstar Nanosatellite, in: AIAA Joint Propulsion Conference, 2000.
[21] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, 1995.
[22] T. Schetter, M. Campbell, Comparison of agent organizations of multiple satellite autonomy, AIAA J. Spacecraft and Rockets Nov–Dec (2001).
[23] M. Schneider-Fontan, M. Mataric, Territorial multi-robot task division, IEEE Trans. Robotics and Automation 14 (5) (1998) 815–822.
[24] Teledesic's "Internet-in-the-Sky", http://www.teledesic.com/.
[25] R. Turner, E. Turner, D. Blidberg, Organization and reorganization of autonomous oceano-graphic sampling networks, in: IEEE Robotics and Automation (ICRA'98) (Leuven, Belgium), 1998.
[26] G. Weiss (Ed.), Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, Cambridge, MA, 1999.