

Multiple Agent-Based Autonomy for Satellite Constellations

Thomas Schetter¹, Mark Campbell¹, and Derek Surka²

¹ University of Washington, Box 352400, Seattle WA 98195-2400, USA,
mcamp@aa.washington.edu,

² Princeton Satellite Systems, 33 Witherspoon St, Princeton NJ, 08542, USA,
dmsurka@psatellite.com

Abstract. There is an increasing desire to use constellations of autonomous spacecraft working together to accomplish complex mission objectives. Multiple, highly autonomous, satellite systems are envisioned 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 TeamAgentTM. Its application is shown for TechSat21, a U.S. Air Force mission designed to explore the benefits of distributed satellite systems. Required spacecraft functions, software agents, and multi-agent organisations are described for the TechSat21 mission, as well as their implementation. Agent-based simulations of TechSat21 case studies show the autonomous operation and how TeamAgent can be used to evaluate and compare multi agent-based organisations.

1 Introduction

Interest in dividing the functions of a single large satellite among several smaller and simpler units working in tandem is gaining more momentum for an increasing number of space applications. Satellite clusters that include several smaller satellites that collaboratively work together on a satellite mission, thus forming a “virtual” satellite, are commonly referred to as *Distributed Satellite Systems (DSS)*. Multiple, distributed agent-based satellite systems are envisioned because they are capable of higher performance, lower cost, better fault tolerance, reconfigurability and upgradability.

The cost of operating spacecraft after launch is a considerable portion of the overall mission cost of the mission. For commercial satellites, operations consist of monitoring the spacecrafts 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.

Most intelligence on current satellites is non-existent. 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 [1], an autonomous agent architecture based on model based programming, on-board deduction and search, and goal-directed closed loop commanding. 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. In addition, because of technical difficulties, much of the Remote Agent software was stripped off the satellite prior to launch, although it was planned to uplink part of its functionality at a later date [2].

The most relevant work in autonomy for distributed systems has been in robotics and autonomous underwater vehicles. There has been a recent interest in emergent behavior [3], 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 [4] 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. The work to date is missing many pieces to the full architecture has only been implemented in simulation.

The purpose of this paper is to present an multiple agent based software (MAS) architecture that will enable agent-based multi-satellite systems to fulfill their complex mission objectives. Termed TeamAgent, this system is applied to TechSat21, a space based radar demonstration. First, a short outline of the TechSat21 mission is given. Then, required functional agents are identified and possible agent-based organisations are presented. Next, implementation within the TeamAgent framework is described. Finally, agent-based simulations of several TechSat21 case studies show the autonomous operation and how multi agent-based organisations can be evaluated and compared.

2 TechSat21

The TechSat21 mission is an Air Force mission designed to explore the benefits of a distributed approach to satellites. The initial demonstration is currently being designed to be a space based distributed radar [5]. 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 DSS by using a sparse aperture array for radar imaging, which allows improved resolution because of the satellite spacing, Figure 1.

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* or "natural" orbits while in formation. The current configuration for the TechSat21 mission has focused on clusters of spacecraft in two local ellipses,

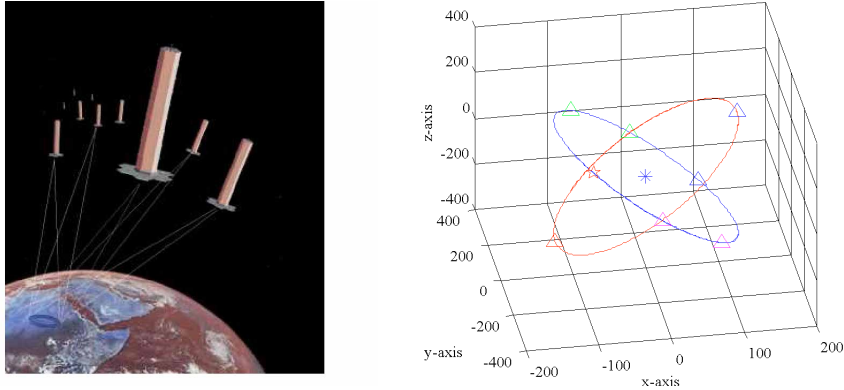


Fig. 1. Left: TechSat21 - a revolutionary approach to space based sensing. Right: a 3D-animation of the TechSat21 mission, where ‘*’ is the virtual center of the cluster, and the spacecraft ‘ \triangle ’s are in two planes of four.

tilted at ± 30 degrees from the z -axis in the Hill-frame. The Hill’s equation [6] describe the relative motion of spacecraft by the use of linearized equations. The nominal orbit for the cluster (or cluster center) is a circular, polar orbit.

For ease and simplicity of the simulation, a constellation of eight satellites was chosen, with four satellites placed in each ellipse. Figure 1 shows a 3D-Animation of the simulation, where spacecraft movements are shown with respect to the Hill-coordinate frame. The ‘*’-symbol locates the center of the Hill-frame, i.e. the trajectory of the circular, polar reference orbit.

3 Agent Definitions for Multiple Satellites

In the following section, agent descriptions are presented at both the spacecraft and functional levels. The former is referred to as a *spacecraft-level agent*. The term “skill” refer to the software functions that describe each agent.

3.1 Spacecraft-Level Agents

In order to narrow the scope of study of agents for multiple spacecraft, spacecraft-level agents 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 (Figure 2).

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.

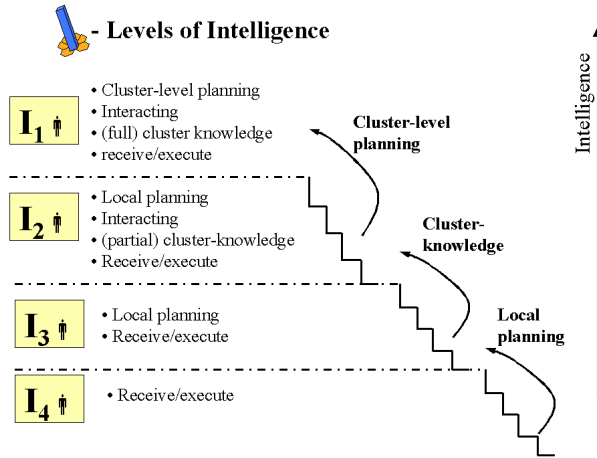


Fig. 2. Identification of spacecraft-level agents based on levels of capable intelligence.

The next higher spacecraft-level agent is I_3 , with 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 [1].

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 agent-based organisation. An example includes coordinating/negotiating with other spacecraft-level agents in case of conflicting requirements.

The spacecraft-level agent I_1 represents the most “intelligent” agent. The primary difference between I_1 and the spacecraft-level agents outlined thus far 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.

3.2 Lower Level Functional Agents

In order to demonstrate the usefulness of MAS 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.

HIGH-LEVEL TASKS	HT 1 Science (Imaging)	HT 2 Formation Maintaining and Control			HT 3 Cluster reconfiguration		HT 4 Cluster upgrade	
SUB-LEVEL TASKS	ST 11 Science	ST 21 Rejecting Disturbances	ST 22 Collision Avoidance	ST 23 Orbit Maneuvering	ST 31 Fault Detection	ST 32 Formation Change	ST 41 Accepting new S/C	ST 32 Formation Change
TASK CATEGORY								
1 INTERACTION	F11	F11	F11	F11	F11		F11	
1 sensing s/c info	F12	F12	F12	F12	F12		F12	
2 transmitting s/c info								
2 DECISION-MAKING	F20							
0 Imaging		F21						
1 Disturbance			F22					
2 Collision Avoidance					F23			
3 Failure/Loss							F24	
4 Upgrade/Gain								
3 ORGANISATIONAL	F30	F30	F30	F30				
0 Scheduler						F31		F31
1 Planner (Cornwell)						F32		F32
2 Planner (Assign positions)						F33		F33
3 Task allocator								
4 FF planner (Trajectory)				F34				
4 REPRESENTATIONAL								
0 Storing Cluster Information						F40		F40
5 OPERATIVE	F50							
0 DAR-Imaging		F51	F51	F51				
1 orbit maneuvering (LQR, (bang-bang, FF-command))								

Fig. 3. Functional breakdown of the task structure specifically for Techsat21.

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. Figure 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. The functional blocks, or 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).

F11 (sensing agent) continuously obtains the state and health from the spacecraft, and makes it available for the entire cluster. The state \underline{x} includes satellite position and velocity, 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 .

Decision-making agents periodically make decisions or monitor specific system parameters for changes. F20 (decision-making agent science) decides which satellites acquire which targets, and how many satellites are required to monitor changing points of interest. F21 (decision-making agent station keeping) decides whether it is necessary to station keep or to perform an orbit correction maneuver due to external disturbances. F22 (decision-making agent collision avoidance) detects when collisions may occur between the spacecraft in the cluster. F23 (decision-making agent cluster downgrade) monitors the health status \underline{h} of the spacecraft to detect failures on-board, and, if required, start a cluster

reconfiguration. F24 (decision-making agent cluster upgrade) decides if there is a new spacecraft to be added to the cluster.

In the case of a failure or adding of a new spacecraft to the cluster, F31 (cluster reconfiguration planner agent) is called, which optimizes a new spacecraft position within the cluster based on maximizing its usefulness for science (imaging). Once a new cluster configuration is known, F32 (cluster allocation planner agent) assigns new spacecraft positions within the cluster. This can be done in many ways, but the approach here is to equalize fuel use across the cluster. F33 (task allocation planner agent) distributes tasks for the cluster based on a predefined cost. F34 (trajectory planner agent) generates a fuel and/or time optimized control maneuver for each spacecraft.

F40 (representational agent) keeps and continuously updates the internal cluster description. This description contains the number of active spacecraft in the cluster; a description of the particular tasks that the spacecraft are capable/allowed to carry out (i.e. passive, partial active and active); and a description of the relative position for each spacecraft within the cluster.

F50 (science agent) performs the radar imaging task. F51 (orbit maneuvering agent) performs the physical orbital maneuver.

3.3 Agent Skills

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 can be variable.

All decision-making skills are implemented using Fuzzy logic [16]. The ideal `PlanReconfigSkill` uses the *Cornwell Metric* to calculate new optimal cluster positions based on radar imaging, Ref. [13]. A contract net bidding mechanism is used to implement the cluster assignment (`PlanAssignSkill`) and task allo-

Table 1. Implemented software agents in TeamAgent.

Skill	Identification	Priority	Update Period	Tool
<code>SensingSkill</code>	F11	fix	fix	-
<code>ScienceSkill</code>	F20	fix	fix	-
<code>StatKeepSkill</code>	F21	variable	variable	Fuzzy Control
<code>CollAvoidSkill</code>	F22	variable	variable	Fuzzy Control
<code>DecMakFailSkill</code>	F23	fix	variable	Fuzzy Control
<code>DecMakAddSkill</code>	F24	fix	fix	Fuzzy Control
<code>Scheduler</code>	F30	-	fix	-
<code>PlanReconfigSkill</code>	F31	fix	on demand	Cornwell Metric
<code>PlanAssignSkill</code>	F32	fix	on demand	Contract net
<code>TaskAllocSkill</code>	F33	fix	on demand	Contract net
<code>PlanFFSkill</code>	F34	fix	on demand	Linear Program
<code>OrbitManSkill</code>	F40,F51	variable	on demand	LQR

cation (**TaskAllocSkill**). The trajectory planner **PlanFFSkill** is implemented with a Linear Program (LP) [14]. The **OrbitManSkill** uses a dynamic simulation of the relative spacecraft motion (Hill's equations) with an LQR controller.

4 Multi-Agent Based Organisations for Satellites

The type of multi-agent organization is a complex design process. The organization must be adaptable to prevent of faults, avoid bottlenecks, and allow re-configuration. It must be efficient in terms of time, resources, information exchange and processing. And it must be distributed in terms of intelligence, capabilities, resources. Figure 4 shows a summary of options as a function of individual, capable spacecraft-level agent intelligence. Note that lower level functional agents are implied. Ref. [15] presents a comparison of these organizations based on the TechSat21 mission.

As can be seen, the number and composition of the different spacecraft-level agents I_1 - I_4 determines the organisational architecture. The top-down coordination architecture includes only one single (highly intelligent) spacecraft-level agent I_1 and the other spacecraft are (non-intelligent) I_4 agents. The centralized coordination architecture requires at least local planning and possibly interaction capabilities from each spacecraft, requiring I_3 or I_2 agents. The distributed coordination architecture consists of several parallel hierarchical decision-making structures, each of which is “commanded” by intelligent spacecraft-level agent I_1 . In the case of a fully distributed coordination architecture, each spacecraft in the organisation represents a spacecraft-level agent I_1 , resulting in a totally “flat” organisation.

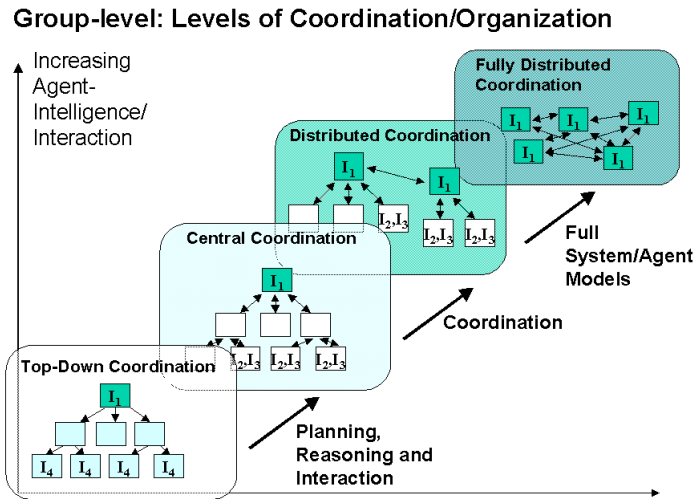


Fig. 4. Coordination architectures for coordination of multiple spacecraft-level agents.

5 TeamAgent Software Architecture for MAS

5.1 TeamAgent

TeamAgent is a MATLAB toolbox [11] for design and simulation of multi-agent systems, especially spacecraft. In TeamAgent, *agents* represent software and remote terminals and *remote terminals (RT)* connect the agents with hardware.

TeamAgent MATLAB 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). The function of the MC is to:

- register and validate agents,
- process messages for itself,
- pass messages to registered agents, RT's, and other MC's for processing,
- allocates 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 Figure 6.

The basic building blocks of agents and remote terminals within TeamAgent are *skills*. Agents and RT's are created in TeamAgent by assigning a set of skills, or software functions. 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 a **update** action, or that the skill is run 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.

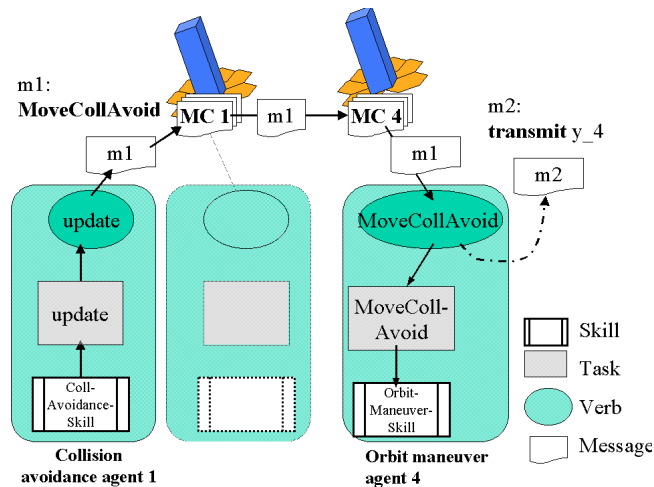


Fig. 5. Example of the relationship between skills, messages, and verbs.

Messages and tasks are identical in TeamAgent, and therefore have the same data structure. The `message.content` data structure is used by special *verb* functions to perform appropriate actions. The first element of `message.content` determines which verb function is called. Other entries describe the object of the verb, sender and receiver agent and associated skills. 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 the internal state. Figure 5 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 is then sent to message center 4. The verb function MoveCollAvoid is called which triggers the same named action of the OrbitManeuverSkill. Additionally, message m2 is transmitted back to the collision avoidance agent 1.

5.2 Information Flow Architecture

Figure 6 shows the information flow architecture for a central coordination ar-

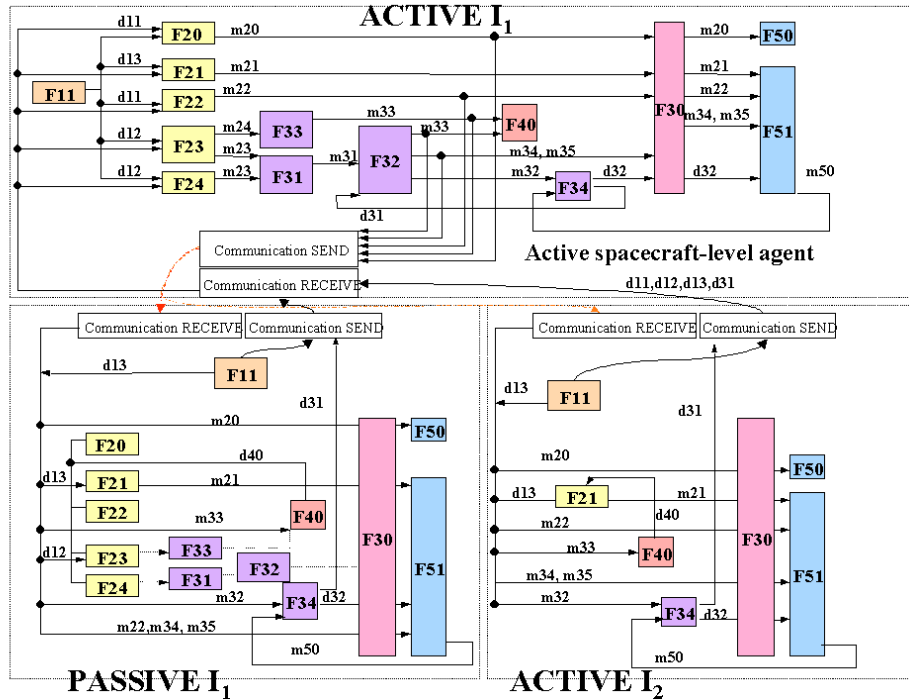


Fig. 6. Information flow architecture for a distributed coordination architecture, with active and passive spacecraft-level agents I₁, and spacecraft-level agent I₂.

chitecture, 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.

Shown are an active spacecraft-level agent I_1 , a passive spacecraft-level agent I_1 , and an active spacecraft-level agent I_2 . Each lower level spacecraft agent performs local planning and decision-making, and interacts with the higher-level agent I_1 in the case of a reconfiguration. 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 and agents is given in Table 2.

Table 2. Messages within TeamAgent with corresponding verbs, sources and sinks.

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

6 Simulation Results

The spacecraft and functional agents for TechSat21 described previously were then integrated and implemented in the TeamAgent environment. The following two sections show two case studies for the simulation.

6.1 Reconfiguration of an Agent-Based Organisation

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 Figure 7 for a distributed coordinated organization, each with I_1 spacecraft level agents. The top of the figure depicts the nominal operation for the organisation, while the bottom figure shows the case where the active spacecraft-level agent I_1 in cluster 2 (spacecraft #5) has failed, and the organization must be reconfigured.

Two primary approaches exist for organization reconfiguration: static and dynamic reconfiguration. *Static reconfiguration* is based on a logic rule base, while *dynamic reconfiguration* makes use of distributed task allocation techniques such as the contract net protocol [9], or negotiation techniques [10] to nominate the “optimal” candidate spacecraft-level agent.

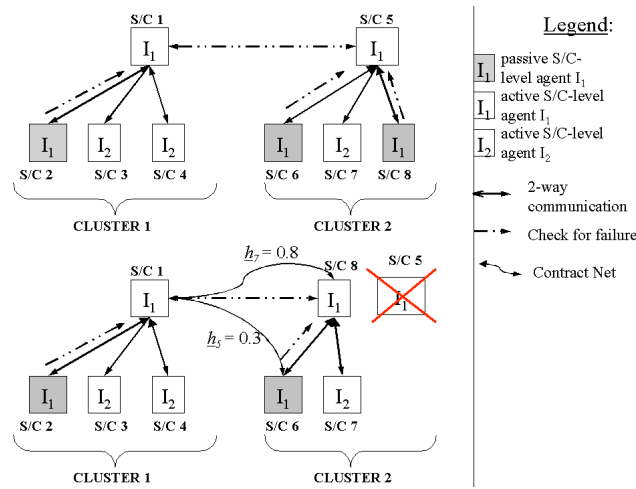


Fig. 7. Distributed coordination architecture for TechSat21. Top: nominal organisation, bottom: reconfiguration example where spacecraft #5 has failed.

For the case of a static reconfiguration, a logic rule base is used. 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  $I_1$  agent is alive in cluster of failed  $I_1$  agent)
  THEN (nominate new active  $I_1$  in own cluster)
ELSEIF (passive  $I_1$  agent is alive in other cluster)
  THEN (nominate active  $I_1$  (old passive) agent in other cluster)
ELSEIF (active  $I_1$  agent from other cluster is alive)
  THEN nominate active  $I_1$  (old passive) agent from other cluster
  
```

Note that if there were more I_1 agents, such as two passive I_1 level agents shown in cluster 1 of Figure 7, then a more complicated logic rule base would

be required to select between the two. This selection could be made based on spacecraft health or other factors.

For the case of a dynamic reconfiguration, the contract net protocol is applied to a reconfiguration based on a failure within spacecraft #5. In this case, spacecraft #1, which is a spacecraft-level agent I_1 , 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. Intelligent spacecraft-level agent I_1 on spacecraft #1 is nominated as contractor for the contract net protocol using a logic based rule base.
2. The contractor sends out requests to 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 accept or deny the request. 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

$$C = \frac{c_1}{h_s} \cdot \frac{c_2}{h_p} \cdot \frac{c_3}{h_t} \cdot \frac{c_4}{h_f} \quad (1)$$

where $c_1 - c_4$ are weighting factors, chosen on the importance of the different subsystems and the h 's correspond to the health values of the different monitored spacecraft subsystems. Spacecraft #8 is chosen, because it has the smallest cost ($C = 1/0.8$).

5. The contractor updates the internal cluster description of the organisation.
6. The new active master spacecraft-level agent begins its operation.

6.2 Conflict Resolution

The second case study is that of conflict resolution between more than one agent. 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.

The resolution of conflictual relationships between tasks is implemented using an approach similar to the subsumption architecture ([7], [8]). When a conflict occurs, the most dominant task inhibits the output of the less dominant tasks. A task with a higher priority value therefore suppresses a task with a lower priority.

Figure 8 shows the priority for the tasks `PerformScience` (m20), `RejectDist` (m21), `MoveCollAvoidance` (m22), and `ReconfigureCluster` (m23) as a function 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.

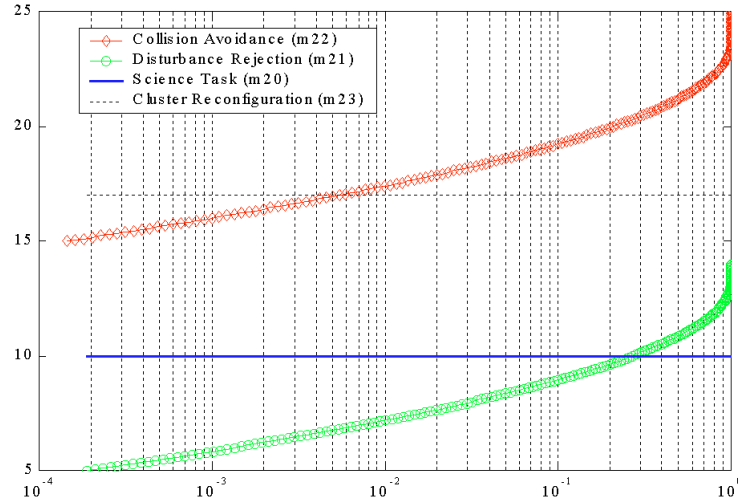


Fig. 8. Resolution of the conflictual relationships between different tasks using dynamic allocation of the priority of the corresponding tasks.

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).

7 Conclusions and Future Work

A software architecture for multiple satellite autonomy using a message passing simulation environment (TeamAgent) for MAS has been presented. The required software agents along with possible agent based organisations for TechSat21, which is a distributed multi-satellite mission, have been identified. Tools such as fuzzy control, linear programming and the contract net have been used for the implementation of the spacecraft-level agents and agent-based organisations

in TeamAgent. Conflict resolution between the agent is accomplished by using of a dynamic priority allocation for the tasks. TeamAgent is well suited for the simulation of multi-agent based systems applied to the space domain. The multi-agent approach is complicated, yet promising. Thus, quick comparisons and design evaluations are critical, all of which can be accomplished in the TeamAgent environment.

Future steps include the evaluation of the “optimal” agent based organisation for the TechSat21 mission, as well as design and implementation for the TechSat21 mission. The message center concept will be evaluated further because it will need improvement/streamlining for eventual implementation.

References

1. Nicola Muscettola; P. Pandurang Nayak; Barney Pell; Brian C. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before, 1998 Artificial Intelligence. Invited Talk, IJCAI-97, Nayoga, Japan.
2. Dornheim, M. A., Deep Space 1 Launch Slips Three Months. Aviation Week and Space Technology, April 27, 1998, p. 39.
3. Brooks, R. A., A Robust Layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, RA-2v1 (1986), 14-23.
4. Turner, R., Turner, E., Blidberg, D. Organization and reorganization of autonomous oceanographic sampling networks, IEEE Robotics and Automation (ICRA'98), Leuven, Belgium, 1998.
5. Alok Das; Richard Cobb; Michael Stallard. A Revolutionary Concept in Distributed Space Based Sensing. In *AIAA Defense and Civil Space Programs Conference & Exhibit*, pages p. 1–6, Huntsville, AL, 1998.
6. Hill, G. W. 1978. Researches in the Lunar Theory. *American Journal of Mathematics*, Vol 1, No. 1, pages p. 5–26.
7. Rodney A. Brooks. Achieving Artificial Intelligence through Building Robots, 1986.
8. Rodney A. Brooks. Elephants Don't Play Chess. In *Robotics and Autonomous Systems 6*, pages p. 3–15, 1999.
9. R. Davis; and R. Smith. Negotiation as a Metaphor for Distributed Problem Solving. In *Artificial Intelligence*, pages p. 63–109, 1983.
10. Shaw Green; Leon Hurst; Brenda Nangle; Dr. Pdraig Cunningham; Fergal Somers; Dr. Richard Evans. *Software Agents: A Review*. Trinity College Dublin, Broadcom Eireann Research Ltd., version 1.0 edition, May 1997.
11. Princeton-satellite systems web-page. <http://www.psatellite.com>.
12. Techsat21: Advanced Research and Technology Enabling Distributed Satellite Systems. <http://www.vs.afrl.af.mil/VSD/TechSat21/>.
13. Edmund M. Kong; Mark V. Tollefson; James M. Skinner; Jeremy C. Rosenstock. TechSat21 Cluster Design Using AI Approaches and the Cornwell Metric. In *AIAA Paper AIAA-99-4635*, 1999.
14. M. E. Campbell; T. Schetter. Formation Flying Mission for UW Dawgstar Satellite. IEEE Aerospace Conference, March 2000.
15. T. Schetter; M. E. Campbell. *Comparison of Agent Organizations of Multiple Satellite Autonomy*. Flairs AI Conference, Orlando, FL, May 2000.
16. Hung T.Nguyen; Michio Sugeno; Richard Tong; and Ronald R. Yager. *Theoretical Aspects of Fuzzy Control*. John Wiley Sons, Inc., 1995.