Agent-Based Control of Multiple Satellite Formation Flying

Joseph B. Mueller and Derek M. Surka
Princeton Satellite Systems
150 S. Washington St., Suite 201, Falls Church, Virginia, USA 22046
{jmueller, dmsurka}@psatellite.com

Bogdan Udrea
Department of Aeronautics and Astronautics, University of Washington
Box 352400, Seattle WA 98195
udrea@aa.washington.edu

Keywords: software agents, autonomy, formation flying, satellite constellations, ObjectAgent

Abstract

Formation flying is an enabling technique for numerous proposed satellite missions. The TeamAgent software system, based on ObjectAgent technology, is designed to enable multiple satellites to cooperate autonomously with particular focus on formation flying. A generalized agent-based software architecture for formation flying has been devised and is being applied to the Air Force Research Laboratory's TechSat 21 technology demonstration mission.

1 Introduction

Multiple satellite constellations are envisioned for a wide variety of applications, including Earth and space science, military applications such as Synthetic Aperture Radar, and communications systems for everything from cellular phones to high-speed Internet access. Bauer et. al. [1] describe a number of these proposed missions and the technologies required for their success. Primary reasons for the use of multiple satellite constellations include higher performance and greater flexibility in terms of reconfigurability and upgradability options. By using a large number of platforms in a fleet, redundancy is increased and the chances of a point failure are reduced.

In each of the applications, cost is one of the most important factors. The small size, mass, and modularity of miniaturized devices can drastically decrease the development cost per platform, and thus open up the possibility of constellations composed of hundreds or even thousands of nanosatellites. With these large numbers, however, comes the daunting task of coordinating the multiple satellites. The development of a control architecture for fleets of satellites is now an enabling technology.

Two areas of research involved in the development of this control architecture are the development of the estimation and control algorithms for distributed satellite systems and the software implementation of these algorithms on a real-time system. Several researchers (e.g. [2]–[9]) are working on the first area while Princeton Satellite Systems (PSS) is developing the ObjectAgent and TeamAgent systems to address the second.

The ObjectAgent system is an agent-based real-time architecture for distributed, autonomous control and TeamAgent applies OA to the problem of controlling multiple cooperative satellites. These systems were originally developed under Air Force Research Laboratory (AFRL) SBIR funding in support of its TechSat 21 satellite mission, a technology demonstration program that will involve three satellites flying in formation and acting as a "virtual" satellite. The ObjectAgent cluster management software will enable the three Techsat-21 spacecraft to perform high precision formation flying to form a single virtual instrument [10], [11].

During the first phase of development, ObjectAgent was prototyped in Matlab. A complete, GUI-based environment was developed for the creation, simulation, and analysis of multi-agent, multi-satellite systems. Basic collision avoidance and reconfiguration simulations were performed for a cluster of four satellites [12],[13]. ObjectAgent has been ported to C++ at the V1.0 level and the present architecture runs on a PowerPC 750 running Enea's OSE operating system [14]. Collaborative efforts are underway to incorporate a number of the formation flying algorithms being developed by other researchers into the TeamAgent software. This paper describes the status of that effort and its implementation on TechSat 21.

The following section provides an introduction to agents for spacecraft autonomy and gives a general overview of the ObjectAgent software architecture. The next section describes an agent architecture for the control of multiple satellite formation flying. This is followed by a discussion of the application of this architecture to TechSat 21. The paper concludes with a description of the future work to be performed.

2 Software Agents in Space

The use of software agents is becoming increasingly popular as a method to improve the level of spacecraft autonomy. There is no consensus on the exact definition of a software agent, but a standard definition is given by Weiss [15]:

An agent is a computational entity that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends on its own experience.

Bradshaw [16] and Knapik and Johnson [17] provide good overviews of the many different definitions used by researchers in the fields of artificial intelligence and computer science.

The major benefit of agents is their autonomy. Intelligent agents can be given high level goals and then autonomously determine the appropriate actions to fulfill these goals. This can include interaction and collaboration with other agents. Agent based software is a form of distributed programming and as such, it maps naturally onto the requirements of distributed spacecraft [12].

There are many potential benefits to using agents onboard spacecraft [14]. These include:

- Greatly increasing the level of autonomy through automatic decomposition of high-level goals;
- Increasing the flexibility and adaptability of flight software through dynamic agent uploads;
- Improving the reliability of spacecraft and fleets of spacecraft by incorporating fault detection at both high and low levels; and
- Reducing the need for large ground support organizations.

2.1 NASA's Deep Space 1 Remote Agent Experiment

The first demonstration of agents used for control onboard spacecraft was NASA's Deep Space 1 (DS-1) Remote Agent Experiment [18]. During the experiment in May 1999, the spacecraft was sent a list of goals instead of the usual detailed sequence of commands to execute. The Remote Agent (RA) software generated a plan to accomplish these goals and then executed this plan, monitoring for hardware faults during execution. Despite some minor glitches, the Remote Agent Experiment was a complete success and achieved 100% of its validation goals [19].

The RA software is installed as a layer on top of the regular flight software, an approach that requires the agent to process a lot of information and to be very intelligent. The complexity of using this approach to space flight software was evident when many of its capabilities were stripped off prior to launch and replaced by the more conventional Mars Pathfinder software [20]. (Additional capability was uploaded after launch.)

2.2 ObjectAgent Software for Autonomous Spacecraft

The ObjectAgent Software Architecture goes beyond the DS-1 remote agent experiment by using agents as the basis of the system rather than as a top layer. This is a key feature that distinguishes ObjectAgent from other agent architectures. This architecture allows decision-making, including fault detection and recovery capabilities, to be built in at all levels of the software. This alleviates the need for extremely intelligent high-level agents and simplifies the software interfaces.

Each agent is multi-threaded and composed of skills. These skills are user-defined and determine the functionality of the agent. Generally, each skill corresponds to one basic function, has inputs and outputs, and triggers one or more actions. An agent is aware of its skills, inputs, and outputs and built-in skills enable it to hunt for inputs and automatically configure itself upon launch. In this sense, an ObjectAgent system is self-organizing.

A fundamental component of ObjectAgent is the flexible messaging architecture that provides a reliable method for agent-to-agent communication both on a single processor and across networks. Simplified natural language is used for all messages, which enables end-users to easily understand agents and command them. A message-based communication architecture was selected because it naturally supports distributed systems.

Figure 1 shows the C++ implementation of an OA agent running on Enea's real-time operating system, OSE. An agent consists of three primary OSE processes — the Main Agent process, the Dispatcher process, and the Collection Center process — and any number of additional skill processes. Each process can be thought of as a thread.

The Main Agent process is the first process to run when an agent is created. This process is responsible for initializing the agent, which consists of creating the common objects required by the agent and for starting the other processes. After initialization, the Main process responds to messages that do not involve the direct transmission of or request for data.

Each skill runs as a separate process and has its own inbox and outbox for receiving and sending data. The Collection Center receives all incoming messages and passes them on to the appropriate inbox for processing. Data and requests for data are placed in the inbox of the skill that either needs or can provide that information. All other messages are passed to the Main process. The Dispatcher sends the outgoing messages that have been created by the other processes.

More detailed information about the ObjectAgent software architecture can be found in [14].

2.3 TeamAgent for Multiple Satellite Constellations

The TeamAgent System applies ObjectAgent to the problem of controlling multiple cooperative satellites. The concept of using agents for formation flying is promising for several reasons. Agents provide a modular, reconfigurable way to implement algorithms in a distributed environment. Agents are also robust, flexible, and may be easily extended to larger numbers of spacecraft.

In Phase I, software development emphasized the robustness of agents. Preliminary comparisons were also made of several high-level agent organizations, along with varying degrees of satellite intelligence, to analytically assess their impact on communication, computation, performance, and reliability. The results indicate that an autonomous, agent-based design provides increased reliability and performance over traditional satellite operations for the control of these clusters [12], [13].

Whereas the first phase of development concentrated on the organization of spacecraft-level agents, the second phase is concentrating on the actual software implementation of formation flying

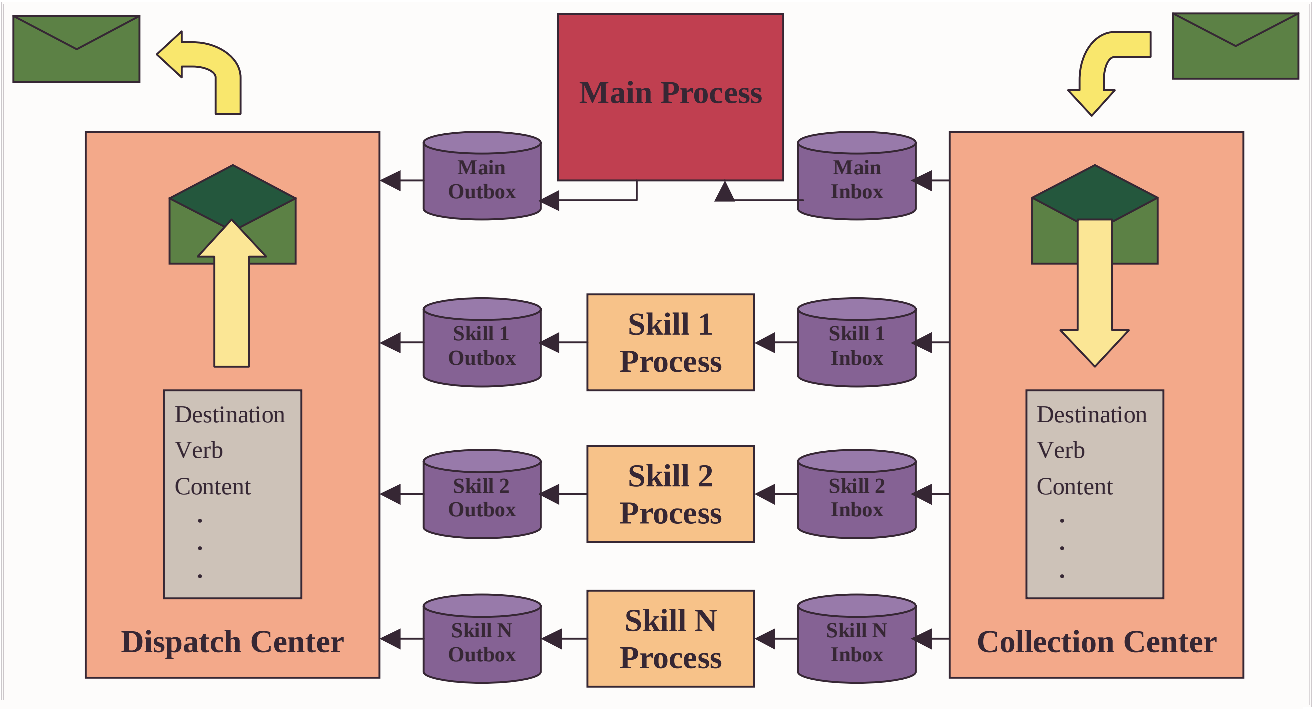

Figure 1. C++ Agent Architecture

algorithms. The following sections present an agent architecture that has been developed for formation flying and its application to TechSat 21.

3 Agent-based Formation Flying

As discussed earlier, formation flying is an enabling technology for many space-based applications. The control problem for multiple satellite formations is well known, and has been the recent subject of study for several researchers. At Texas A&M ([5], [6]), the effects of gravitational perturbations are investigated to identify J2-invariant orbits. At Stanford [3], inverse dynamic optimization is used to develop control laws for on-off thrust actuation. Collision avoidance and auctioneering algorithms are being developed at Honeywell [2] to improve cluster management.

This section presents a generalized agent architecture for the formation flying control of three satellites in a circular orbit. The agents were developed and simulated in Matlab with the ObjectAgent software package. The reference orbit has a semi-major axis of 7500 km and an inclination of 48 degrees. It is assumed that each satellite is equipped with GPS sensors and an inter-satellite link (ISL) that enables communication between all spacecraft. We first provide an overview of the control system, followed by a description of the agent block diagram. The results of a reconfiguration maneuver are presented at the end of the section.

3.1 Control System Overview

The objective of a formation flying control design is to command and control individual vehicles so that they function as a single system. The primary tasks of the control system should include:

- Formation Planning
- Formation Keeping
- Reconfiguration
- Collision Avoidance
- Fault Detection, Isolation, & Recovery

The design presented here incorporates only the first three tasks, which represent the basic operations of formation flying. The design could be easily extended, however, by simply adding the appropriate agents for collision avoidance and fault detection; the existing agents would require little or no modification. This is an example of the benefits of using agents.

Two formation types are considered:

1) *Leader/Follower*: The spacecraft are equally spaced in the along-track direction with no relative motion.
2) *Hill's*: The spacecraft are equally phased around an ellipse with relative motion that follows the first order Hill's equations.

For the cluster to maintain either of these formations, it performs *Formation Keeping*, which incorporates closed loop control. In order to transition from one formation to the other, or to resize the current formation, it carries out a *Reconfiguration* maneuver. The *Formation Planning* task includes generating the desired trajectories for the spacecraft to follow, whether it is for a static formation or a reconfiguration.

Long-term formation keeping is perhaps the most challenging goal in a formation flying mission. There are three steps necessary to achieve this goal: 1) obtaining accurate estimates of relative position and velocity, 2) rejecting those disturbances which cause the satellites to drift apart, and 3) conserving fuel for long duration missions. Clearly, these issues are closely related.

Closed loop control techniques can be easily designed to combat the gravitational, atmospheric and solar perturbations that would disturb a satellite's desired orbit. Traditional station-keeping methods used in industry apply periodic orbit correction maneuvers to control the satellite's orbital elements. These techniques seek to minimize the required lifetime ΔV, while maximizing the elapsed time between maneuvers [21]. Formation keeping of multiple satellites, however, requires precise control of each vehicle's *relative* position and velocity—absolute control in the typical station-keeping sense will not suffice.

In formation flying, it is important to ensure that the chosen control law is fuel-optimal for the entire cluster. Given the significant noise content inherent in relative state estimation, and the oscillatory motion caused by gravitational perturbations, the challenge is to design a control system that does not waste fuel by constantly fighting acceptable disturbances.

Work in [5] presents methods for establishing J2-invariant orbits, where the secular drift between satellites is eliminated. Here, the relative orbits of two satellites are defined such that the J2 gravitational perturbation (earth oblateness) causes them to drift at the same rate, greatly reducing the need to perform repeated formation keeping maneuvers. This approach enables long-term formation flying with reduced fuel usage, but places restrictions on the achievable formation types.

In addition, [6] presents a time-varying state transition matrix for relative motion that includes the effects of both the orbit eccentricity and the J2 perturbation. With continuous updates of the reference orbital elements, this technique promises to provide a more accurate model of the relative motion. Desired state trajectories could be generated using this transition matrix, preventing the controller from fighting J2.

Another interesting approach is presented in [3], where a two vehicle weighted fuel-time optimal control law is defined for vehicles using "on-off" thrusters. Combining this type of control approach in a J2 invariant orbit might be a practical design for long-term formation keeping.

The agent-based design that follows employs feedback control to enable formation keeping, and uses a linear program solver to compute the fuel optimal control history for reconfiguration maneuvers.

3.2 Agent Architecture

The agent-based architecture is designed to enable autonomous formation flying activities. High-level commands are received from either the ground or an onboard planner, and the agents work together to control the cluster appropriately. The three satellites are organized in a hierarchy, with one satellite acting as the *Cluster Manager*. It receives all high-level commands and performs the cluster-level planning.

A block diagram of the agents is shown in Figure 2. The two agents in bold blocks are unique to the Cluster Manager satellite, while the remaining seven agents are active on all satellites. Note that the block entitled *CASPER* is meant to emulate the types of commands that JPL's onboard planner/scheduler "CASPER" might provide [11]. Also note that while the Collision Avoidance agent is not yet implemented, its location in the block diagram has already been identified.

Each of the agents in the diagram consists of one or more skills that define its functionality. The following paragraphs provide a brief description of each agent's role in the system. A discussion of the skills is omitted for brevity.

Formation Planner

This agent is unique to the Cluster Manager, and its primary function is to command each satellite to the proper formation. It receives high level commands from CASPER in the input *command*, which includes the desired formation type and size, the start and stop times, and a command ID. The types of commands it may receive from CASPER include:

- COMPUTE_FORMATION — The agent will compute the fuel optimal trajectory for each satellite to follow in order to achieve the new formation (see below), along with the required ΔV. It will store the target states for each satellite along with the command ID, and return the result to CASPER in the output, *response*.
- ACHIEVE_FORMATION — Given the command ID, the agent will find the previously computed target states and send them out in *FF_Config* to the Trajectory Generator agent on board each satellite.
- KEEP_FORMATION — The agent will take the information from the *command* input and route it to the Trajectory Generating agents in *FF_Config*.

The most interesting aspect of the Formation Planner agent is how it computes the fuel-optimal trajectories of each spacecraft for a reconfiguration maneuver. First, a family of potential target states is identified for each spacecraft. For example, in the Leader/Follower formation there are six possible configurations (each spacecraft may occupy one of the points in the line). For each set of potential target states, the simplex linear programming algorithm is used with the discretized linear plant to calculate the optimal control history for achieving that state. The

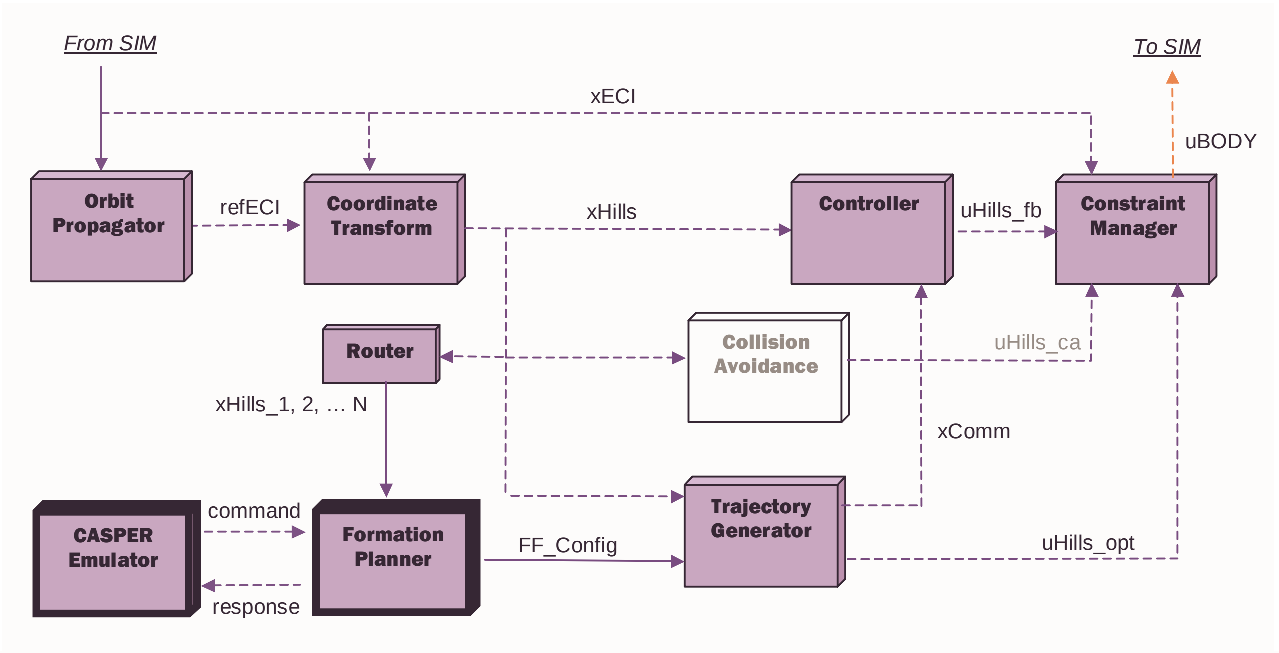

Figure 2: Cluster Manager Block Diagram

case that uses the least amount of fuel for the cluster is chosen, resulting in a globally fuel-optimal solution.

The cost function minimized in this algorithm is the total fuel usage for that maneuver. Another practical option is to equalize the fuel usage across the cluster. In this case, the weighted fuel-time optimal solution is found, where the weight is defined, in general, as the ratio of the current satellite's remaining fuel to the total remaining fuel of the cluster.

Trajectory Generator

This agent receives the *FF_Config* command from the Formation Planner agent and computes a corresponding trajectory for the satellite to follow (*xHills_com*). In the case of a reconfiguration maneuver, the agent is commanded to achieve a target state at a specified time. It first uses simplex to compute the fuel-optimal control history (*uHills_opt*), then computes the associated state trajectory (*xHills_com*) that it should follow during that maneuver. This approach combines the open loop optimal solution with feedback control to combat disturbances during the maneuver.

Orbit Propagator

This agent propagates the reference orbit, which is used by each spacecraft to define its relative position and velocity. First, it uses the absolute state of each spacecraft in the ECI frame (*xECI*) to calculate the centroid of the cluster. This state is then propagated forward in time to produce a time-tagged set of future reference states, and the result (*refECI*) is sent to both the Coordinate Transform and the Constraint Manager agents.

Coordinate Transform

This agent is used to supply several other agents with the individual satellite's relative position and velocity in the Hill's frame (*xHills*). Given the reference state from the Orbit Propagator, and the estimated absolute state of the satellite in the ECI frame, the relative state is computed.

Controller

This agent is used to carry out low-level feedback control. Given the desired state (*xHills_com*) from the Trajectory Generator agent, and the measured state (*xHills*) from the Coordinate Transform agent, it computes the 3-axis thrust (*uHills_fb*) using the linear controller presented in [7].

This constant gain controller is chosen solely for its simplicity, as the focus of this paper is on the agent architecture. It should therefore be noted that the current approach could easily be replaced with a more complex control law by simply designing a new skill for the *Controller* agent.

Constraint Manager

This agent receives thrust commands in the Hill's frame from both the Controller agent (*uHills_fb*) and the Trajectory Generator agent (*uHills_opt*), and attempts to resolve any conflicts before commanding the thrusters. If the Collision Avoidance agent were active, its thrust commands would supercede those of the other two agents.

In order to practically implement any true form of feedback control, it is necessary to have positive and negative thrusting available in all three axes. If this is not the case, it becomes necessary to slew the spacecraft to the proper orientation before carrying out a maneuver. This approach can become increasingly complex when multiple fuel tanks are used on a single spacecraft, for an orientation must be chosen which provides the best balancing of fuel.

3.3 Preliminary Results

Here we present the results of a reconfiguration maneuver from a Hill's formation to a Leader/Follower formation. A plot of the cluster's relative position in the Hill's frame during this maneuver is shown in Figure 3. The hollow circles indicate the Hill's orientation at time 0, while the solid circles show the final leader/follower formation.

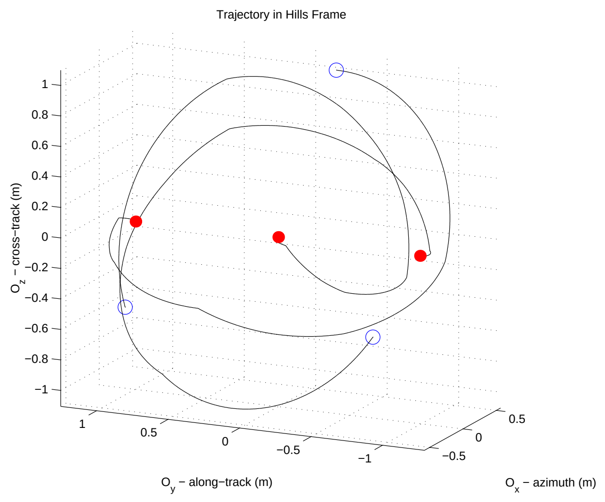

Figure 3: Reconfiguration Maneuver

At time t = 0 seconds, the cluster is initialized to a Hill's formation, with each satellite rotating around the cluster center at a 1 km radius. At time t = 120 seconds, the CASPER emulator sends a command to the Formation Planner agent, instructing it to compute the ΔV required to achieve a Leader/Follower orientation. The maneuver will begin at time 1000 and complete at time 7000, taking just under one full orbit to complete.

The Formation Planner agent takes the latest update of *xHills* from each satellite, and uses Hill's equations to solve for their state at time 1000. (For much longer duration planning, a high-fidelity propagator would become necessary.) The planner then uses simplex to compute the fuel-optimal control for maneuvering each satellite to the three possible new positions. The case that results in the minimum ΔV is chosen. The total ΔV for this maneuver is 6.06 m/s, with individual spacecraft ΔV's ranging from 1.94 m/s to 2.10 m/s.

The target states for each satellite and the target time of 7000 are stored in the agent memory, and the resulting ΔV and projected control history are sent back to the CASPER emulator in *response*. Upon receipt of this input, CASPER immediately sends the Formation Planner another *command*, this time to achieve the formation.

The Formation Planner agent sends the target states, start time and stop time in the output *FF_Config*, which goes to the Trajectory Generator agents on board each satellite. This agent first uses its latest estimate of *xHills* to solve for its future state at time 1000. Given its beginning state and the target state at time 7000, it calls simplex to find the fuel-optimal control solution. The result is sent to the Constraint Manager agent in *uHills_opt*.

The Trajectory Generator also uses the discretized plant to compute the desired trajectory of the satellite during the reconfiguration. This reference command is then sent to the Controller agent, which will calculate feedback control thrusts to stay on the trajectory. Previously, the Controller agent had been following a reference trajectory corresponding to unforced motion in the Hill's frame. It will continue to control to this reference until time 1000, and will then begin controlling to the reconfiguration trajectory.

The Constraint Manager receives the *uHills_opt* input, which is a set of 3-axis forces in the Hill's frame at specified times. When it is time to command a thrust, the agent transforms the force into the body frame, and sends it to the simulation in *uBODY*.

The current simulation assumes 3-axis thrusting capability, with no constraint on the thruster's achievable force. In a real system, however, constraints on the size and direction of the thruster will certainly be an issue. With limited directional capability, this agent would have to compute the appropriate ECI-to-Body quaternion for the spacecraft to align the thruster correctly. If the magnitude is not achievable by the thruster, then thrust mapping techniques must be used to ensure the desired change in momentum is achieved at the appropriate time.

4 Application to TechSat 21

This generalized architecture is being adapted for use on TechSat 21. TechSat 21 (TS21) is an Air Force Research Laboratory technology demonstration mission to be launched in 2004 that will involve three satellites flying in formation and acting as a "virtual" satellite. During the one-year mission, the satellites will fly in various configurations with relative separation distances ranging from 100 m to 5 km and one of the objectives is to do so autonomously.

One of the major advantages of this architecture for the TechSat 21 mission is its modularity. The architecture is expressly designed to be able to receive commands from multiple sources. This will enable the TechSat 21 cluster to be commanded to new formations by a ground operator, an onboard planner (such as JPL's CASPER), or another spacecraft. The modularity also enables different formation flying algorithms to be simulated on the ground prior to orbit, as well fully tested and researched on-orbit.

The Constraint Manager will be modified so that TS21-specific constraints are not violated. The TechSat 21 spacecraft have only one thruster for orbital maneuvering so the spacecraft must first be pointed in the appropriate direction before thrusting. The Constraint Manager will first command the appropriate ECI-to-Body quaternion to align the thruster in the desired direction, and then command the required force. Repeated attitude maneuvers are impractical, for each maneuver takes a significant amount of time and tends to reduce the power available to the spacecraft. This constraint prohibits the use of the Controller agent for closed-loop control. However, formation keeping may still be accomplished by using the Formation Planner to conduct periodic reconfiguration maneuvers.

A major concern with any flight program is reliability. In addition to the robustness built into the formation flying agents, a number of supporting agents will be included for fault management. Some of these agents will monitor the state of spacecraft hardware while others will be used for the detection of agent-level software faults. An Agent Monitor will monitor all other agents to detect run-away processes and deadlock. The Agent Monitor will then have the ability to shut down any such problem agent. A plan for rolling over the functionality of the Cluster Manager to another spacecraft is also being devised.

5 Conclusion and Future Work

A generalized agent-based architecture for multiple satellite formation flying has been developed and tested in Matlab. This architecture is presently being ported to C++ for demonstration on a real-time testbed and eventual use on TechSat 21.

Work has recently begun on applying and extending the TeamAgent architecture to larger formations of satellites. This research is being performed in collaboration with Prof. Jonathon How at MIT and Prof. Mark Campbell at Cornell and the goal is to create a robust architecture for the formation flying of eight or more satellites.

6 Acknowledgements

This work was originally supported by two United States Air Force SBIR Phase II contracts from the Surveillance and Control Division of the Air Force Research Laboratory's Space Vehicles Directorate. The contract numbers are F29601-99-C-0029 and F29601-00-C-0025 and the program manager is Paul Zetocha. Additional support for further development is being provided by subcontracts with JPL and MIT.

7 References

[1] F.H. Bauer, K. Hartman, J. P. How, J. Bristow, D. Weidow, and F. Busse, “Enabling Spacecraft Formation Flying through Spaceborne GPS and Enhanced Automation Technologies,” *Proc. of the Institute of Navigation GPS-99 Conference*, Nashville, TN, September 1999.

[2] B. Morton, N. Weininger, and J.E. Tierno, “Collective Management of Satellite Clusters,” AIAA-99-4267.

[3] J. Robertson, G. Inalhan, and J.P. How, “Spacecraft Formation Flying Control Design for the Orion Mission,” AIAA-99-4266.

[4] R. Carpenter, “System Description and User’s Guide for Matlab Version of *Pluribus* Formation Flying Controller.”

[5] K.T. Alfriend and H. Schaub, “Dynamics and Control of Spacecraft Formations: Challenges and Some Solutions,” AAS 00-259.

[6] D. Gim and K.T. Alfriend, “The State Transition Matrix of Relative Motion for the Perturbed Non-Circular Reference Orbit,” *AAS/AIAA Space Flight Mechanics Meeting*, Santa Barbara, CA, February 2001.

[7] H. Schaub and K.T. Alfriend, “Hybrid Cartesian and Orbit Element Feedback Law for Formation Flying Spacecraft,” *2000 AIAA Guidance, Navigation and Control Conference*, Denver, CO, August 2000.

[8] M. Meshabi and F.Y. Hadaegh, “Formation Flying Control of Multiple Spacecraft: Graph Theoretic Properties and Switching Schemes,” *1999 AIAA Guidance, Navigation and Control Conference*, Boston, MA, August 1999.

[9] P.K.C. Wang and F.Y. Hadeagh, “Coordination and Control of Multiple Microspacecraft Moving in Formation,” *Journal of Astronautical Sciences*, Vol. 44, No. 3, 1996, pp. 315-355.

[10] P. Zetocha and M. C. Brito, “Development of a Testbed for Distributed Satellite Command and Control,” *2001 IEEE Aerospace Conference Proceedings*, Big Sky, Montana, March 2001.

[11] R. Sherwood, et. al., “The TechSat 21 Autonomous Sciencecraft Constellation Demonstration,” *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Montreal, Canada, June 2001.

[12] T.P. Schetter, M.E. Campbell, and D.M. Surka, “Comparison of Multiple Agent-based Organizations for Satellite Constellations,” *2000 FLAIRS AI Conference*, Orlando, Florida, May 2000.

[13] T.P. Schetter, M.E. Campbell, and D.M. Surka, “Multiple Agent-Based Autonomy for Satellite Constellations,” *Second International Symposium on Agent Systems and Applications*, Zurich, Switzerland, September 2000.

[14] D.M. Surka, M.C. Brito, and C.G. Harvey, “The Real-Time ObjectAgent Flight Software Architecture for Distributed Satellite Systems,” *2001 IEEE Aerospace Conference Proceedings*, Big Sky, Montana, March 2001.

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

[16] J.M. Bradshaw (ed.), *Software Agents*, Cambridge, MA: AAAI Press/MIT Press, 1997.

[17] M. Knapik and J. Johnson, *Developing Intelligent Agents for Distributed Systems*, New York: McGraw-Hill, 1998.

[18] D.E. Bernard et al., “Design of the Remote Agent Experiment for Spacecraft Autonomy,” *1998 IEEE Aerospace Conference Proceedings*, Snowmass/Aspen, CO, 1998.

[19] D.E. Bernard et al., “Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment,” *1999 AIAA Space Technology Conference & Exposition*, AIAA Paper 99-4512, Albuquerque, NM, September, 1999.

[20] M.A. Dornheim, “Deep Space 1 Launch Slips Three Months.” *Aviation Week and Space Technology*, April 27, 1998, p. 39.

[21] B.N. Agrawal, *Design of Geosynchronous Spacecraft*, Prentice-Hall, Englewood Cliffs, NJ, 1986.