# Autonomous and Autonomic Systems: A Paradigm for Future Space Exploration Missions

Walter F. Truszkowski, *Member, IEEE*, Michael G. Hinchey, *Senior Member, IEEE*, James L. Rash, *Member, IEEE*, and Christopher A. Rouff, *Member, IEEE*

***Abstract*—More and more, the National Aeronautics and Space Administration (NASA) will rely on concepts from autonomous systems not only in mission control centers on the ground, but also on spacecraft and on rovers and other space assets on extraterrestrial bodies. Autonomy facilitates not only reduced operations costs, but also adaptable goal-driven functionality of mission systems. Space missions lacking autonomy will be unable to achieve the full range of advanced mission objectives, given that human control under dynamic environmental conditions will not be feasible due, in part, to the unavoidably high signal propagation latency and constrained data rates of mission communications links.While autonomy supports cost-effective accomplishment of mission goals, autonomicity supports survivability of remote mission assets, especially when tending by humans is not feasible. In principle, the properties of autonomic systems may enable space missions of a higher order than any previously flown. Analysis of two NASA agent-based systems previously prototyped, and of a proposed future mission involving numerous cooperating spacecraft, illustrates how autonomous and autonomic system concepts may be brought to bear on future space missions.**

***Index Terms*—Autonomic systems, autonomous systems, intelligent systems, multiagent technology, spacecraft.**

## I. Introduction

With the National Aeronautics and Space Administration's (NASA) renewed commitment to space exploration, particularly missions to Mars and the return to the Moon, greater emphasis is being placed on both human and robotic exploration. In reality, even when humans are involved in the exploration, human tending of space assets becomes cost-prohibitive or is simply not feasible, and therefore, increasingly in future missions, remote mission assets will be required to work autonomously.

Moreover, much of the mission control work on the earth will be performed by fully computerized systems operating with little or no human intervention. In addition, certain exploration missions will require spacecraft that will be capable of venturing where humans simply cannot be sent. Spacecraft that, for cost or practical reasons to be described below, cannot be tended at all times by humans will be required to work autonomously.

Although autonomy will be critical for future missions, it will be essential that these missions exhibit autonomic properties. Autonomy alone, absent autonomicity, will leave the spacecraft vulnerable to the harsh environment in which they have to operate, and in which, most likely, performance will degrade or in which the spacecraft will be destroyed or will not be able to recover from faults. Ensuring that exploration spacecraft are endowed with autonomic properties will increase the survivability and therefore the likelihood of success of these missions.

This paper first discusses the need for autonomy and autonomicity in future NASA missions and then discusses the autonomic properties of three systems: two multiagent systems developed at the NASA Goddard Space Flight Center (GSFC) prior to the advent of the Autonomic Computing initiative and a concept mission that is currently planned to launch in the 2020–2030 timeframe. It is interesting to note that the previously prototyped systems (LOGOS and ACT) were found to require autonomic properties, although the initiative had not been formulated at that point. The concept mission (ANTS) is being defined with autonomicity in mind. We will describe the exhibition of autonomic properties of each of these systems, illustrating why future space exploration missions will necessarily be autonomic. We then conclude with some challenges in developing autonomic systems for future NASA missions.

## II. Autonomy and Autonomicity in NASA Missions

### A. Autonomy

Until the mid-1980s, all space missions were operated manually from ground control centers. The high costs of satellite operations prompted NASA, and others, to begin automating as many functions as possible. In this context, a system is autonomous if it can achieve its goals without human intervention. A number of more-or-less automated ground systems exist today, but work continues toward the goal of reducing operations costs to even lower levels. Cost reductions can be achieved in a number of areas; greater autonomy of satellite ground control and spacecraft operations are two such areas.

To develop greater autonomy for ground and space operations, NASA is putting more reliance on intelligent systems and

Manuscript received October 15, 2004; revised May 10, 2005. This work was supported in part by the NASA Office of Safety and Mission Assurance (OSMA), Software Assurance Research Program (SARP) project *Formal Approaches to Swarm Technologies (FAST)* and managed by the NASA Independent Verification and Validation (IV&V) Facility and by NASA Headquarters Code R. This work is based in part on the papers presented at the *Proceedings of the 11th IEEE International Conference and Workshop Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*. This paper was recommended by Guest Editors R. Sterritt and T. Bapty.

W. F. Truszkowski and J. L. Rash are with the Advanced Architectures and Automation Branch, NASA Goddard Space Flight Center, Greenbelt, MD 20771 USA (e-mail: Walter.F.Truszkowski@nasa.gov; James. L.Rash@nasa.gov).

M. G. Hinchey is with the NASA Software Engineering Laboratory, Goddard Space Flight Center, Greenbelt, MD 20771 USA (e-mail: Michael.G. Hinchey@nasa.gov).

C. A. Rouff is with the Advanced Concepts Business Unit, Science Applications International Corporation, McLean, VA 22102 USA (e-mail: rouffc@saic.com).

Digital Object Identifier 10.1109/TSMCC.2006.871600

less on human intervention. Intelligent systems will be able to make more of the operational and science decisions that are normally made by humans. This will allow the spacecraft to respond more quickly to opportunistic science as well as respond faster to the spacecraft anomalies, time sensitive problems, or even routine operational issues. In addition, as missions become more complex, the cost of personnel controlling missions has become a significant issue. Increasing the levels of autonomy and intelligence exhibited by missions will also reduce these costs.

The goals of greater autonomy have been further complicated by NASA's plans to use constellations and swarms of nanosatellites for future science-data gathering. These are significantly more complicated to operate compared to traditional single spacecraft missions. Indeed, it may be impossible for human operators to control such systems. Spacecraft in swarms and constellations must communicate in order to coordinate and cooperate with each other. Radio or laser communications between constellation elements, or with ground control, may suffer large propagation delays or complete outage (e.g., due to signal blockage) for extended periods of time. Therefore, because constellation/swarm elements will not always be able to rely on other elements or on ground systems, in addition to being autonomous, these systems will need to exhibit autonomic properties to ensure optimal performance and even survival.

### B. Autonomicity

NASA will require autonomicity to be exhibited in future missions, in order to ensure they can operate on their own to the maximum extent possible without human intervention or guidance. A case can be made that all of NASA's future systems should be autonomic and exhibit the four key objective properties of autonomic systems—self-configuring, self-optimizing, self-healing, and self-protecting—together with the attribute properties, viz., self-aware, environment-aware, self-monitoring, and self-adjusting [1]. We now discuss the need for each of these autonomic properties in NASA missions.

Self-configuration is needed in NASA missions because the nature of the mission may change as time goes on. New or different science may need to be analyzed based on data collected. Or, if one science instrument fails or deteriorates, another onboard instrument may need to be used instead or to help adjust the first's condition. Reconfiguring the spacecraft may be necessary when batteries or solar cells are deteriorating. In this case, unnecessary instruments or functions may need to be shut down to reduce the electrical load and the remaining systems reconfigured to take this into account.

Self-optimization is needed because the spacecraft, science instruments, and the science being collected may change as the mission proceeds, and the instruments may need to be adjusted or calibrated. Also, the spacecraft could optimize its operations over time by learning more about the phenomenon it is observing and how, or where, to best observe it. For constellations or swarms, vehicles will have to constantly adjust their mutually relative positions due to drift or optimize themselves when members of the constellation/swarm drop out due to malfunctions or other problems.

Self-healing is needed when a spacecraft is damaged, its software is corrupted, or a member of a swarm or constellation is lost. Examples of software self-healing would be when a spacecraft is hit by a large amount of radiation and the memory is damaged or altered. The spacecraft would have to recognize that the software has been modified or is not available and then request a new version from another spacecraft or from mission operations. Self-healing in a swarm or constellation would include moving another spacecraft into the place of the lost one or requesting a replacement.

Self-protection is needed to keep the spacecraft out of harm's way. An example of when self-protection is needed is when solar flares erupt. Solar flares release charged particles that can cause damage to electronics. In cases such as these, if a solar flare can be detected, the spacecraft can put itself into sleep mode until the flare passes. Another example would be a rover on Mars. Large dust storms can cause damage to many systems. When a dust storm is sensed, the rover would cover itself or go to a better-protected area, such as a rock outcropping or other sheltered area.

Clearly, these objective properties, commonly referred to as the self-CHOP properties, necessitate self-awareness and environmental awareness and are achieved through being self-monitoring and self-adjusting.

### C. How Both Combine

The best possible situation for NASA would be to be able to launch a spacecraft and then simply receive science data from it with no in-flight directions or corrections. NASA is currently a long way from achieving this utopia. To reach such a state of operations, NASA needs its missions to be both autonomous and exhibit autonomic properties. Autonomous systems can operate independently and achieve self-direction. For NASA missions to be fully autonomous and achieve self-governance and survive in harsh environments, autonomicity is required [17].

Combining autonomy with autonomic properties will necessitate a new set of requirements and verification procedures above and beyond what is currently available. NASA currently has no truly autonomous or autonomic missions. New requirements will need to be developed for these types of missions. While autonomy may have similarities across missions, autonomic properties will vary depending on the type of the system and where it would operate. This is also true for verification of the autonomous and autonomic systems [2]. New verification procedures need to be developed, either through direct verification or through simulation if direct verification is not possible. Since these systems will be intelligent, their operation will vary over time and it is unlikely that the same patterns of behavior will be exhibited on a recurring basis. As a consequence, new specification and verification methods need to be developed in order to guarantee correct operation. This is a goal of the *Formal Approaches to Swarm Technologies* (*FAST*) project, described briefly in Section V.

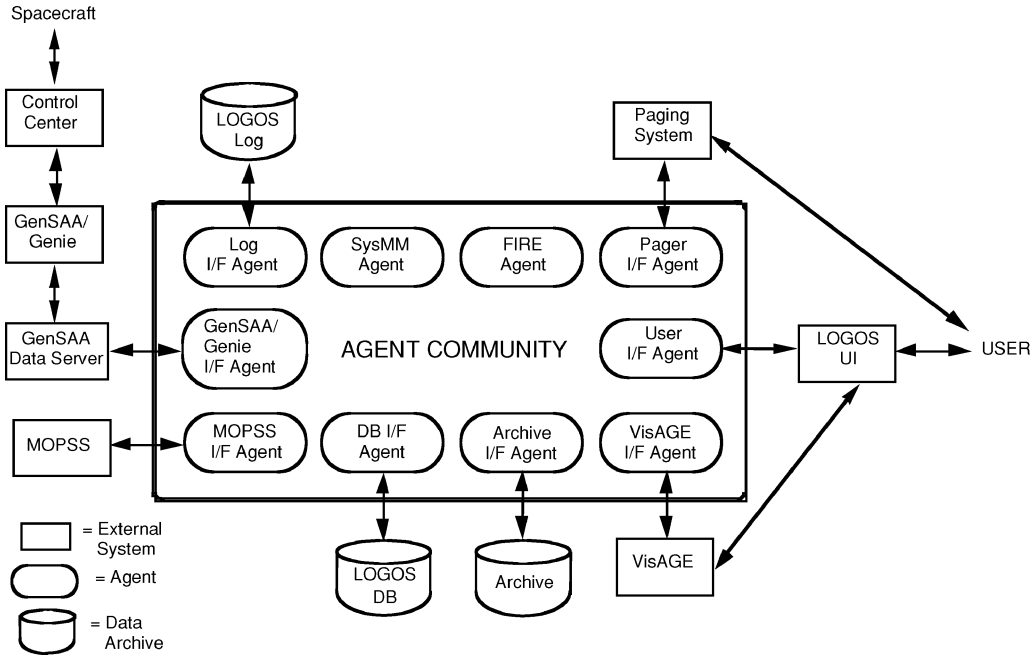


Fig. 1. LOGOS agent community and legacy software.

## III. Overview of Two Agent-Based Systems

### A. Background

The Advanced Architectures and Automation Branch at NASA GSFC has played a leading role in the development of agent-based approaches as a means of realizing NASA's autonomy goals. The aim of this work is to transition proven agent technology into operational NASA systems. Two major successes include the development of the prototype Lights-Out Ground Operations System (LOGOS) and the Agent Concept Testbed (ACT) [3], [4].

There have been many definitions of agents and agent-based systems [5], [6]. For the purpose of this paper, we define an agent to be a software system that is autonomous and has the ability to perceive and effect its environment and communicate with other agents (if present). A multiagent system, or community of agents, is simply a collection of such agents that collaborate and/or cooperate to accomplish a common goal.

LOGOS was the first multiagent system developed in the branch and provided an initial insight into the power of agent communities, autonomy, and autonomic properties of these systems. It should be emphasized that while the system predates the Autonomic Computing initiative, it was clear during development that the objective properties that we now associate with the initiative were essential to the success of the prototype.

Agents in the LOGOS system acted as surrogate human controllers and interfaced with the legacy software that controllers normally used, as well as with humans.

Based on the successful operation of this first prototype, development began on ACT—an environment in which richer agent and agent-community concepts were developed through detailed prototypes and operational ground-based and space-based scenarios. ACT has given GSFC more experience in architecting and developing communities of agents and the autonomous and autonomic systems, as well as giving an improved understanding of the tradeoffs and complications that accompany such systems.

The implementation of LOGOS and ACT provided an opportunity to exercise and evaluate the capabilities supported by the agent architectures and refine the architectures as required. It also provided an opportunity for space mission designers and developers to see the agent technology in action. This has enabled them to make a better determination of the role that the agent technology can play in their missions. We will now describe the LOGOS and ACT agent communities, give brief operational overviews of each, and highlight the autonomic objective properties of the two systems.

### B. LOGOS

LOGOS is a proof-of-concept system consisting of a community of autonomous software agents that cooperate in order to perform functions previously performed by human operators who used traditional software tools such as orbit generators and command sequence planners. The agents were developed in Java and used an in-house software backplane called Workplace for communication between the agents [16]. The following sections discuss the LOGOS architecture and give an example scenario of how LOGOS works.

*1) LOGOS Architecture:* The LOGOS community architecture is shown in Fig. 1; the architecture of an individual LOGOS agent is shown in Fig. 2. The LOGOS is composed of ten agents, some of which interface with legacy software, some which perform services for the other agents in the community, and others which interface with an analyst or operator. All agents in the community have the ability to communicate with all other agents in the community.

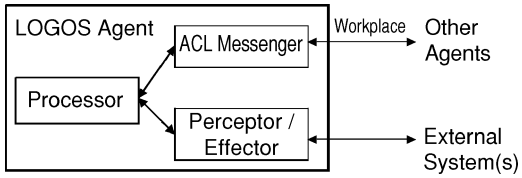


Fig. 2. Architecture of LOGOS agent.

The System Monitoring and Management Agent (SysMMA) maintains a list of all the agents and their addresses in the community and provides these addresses to other agents requesting services. When started, each agent must register its capabilities with SysMMA and request the addresses of other agents whose services it may need.

The Fault Isolation and Resolution Expert (FIRE) agent resolves satellite anomalies during satellite passes. FIRE contains a knowledge base of potential anomalies and a set of possible fixes for each. If it does not recognize an anomaly, or is unable to resolve it, it sends the anomaly to the user interface agent (UIFA) to be forwarded to an analyst for resolution.

The UIFA is the interface between the agent community and the user interface that the analyst or operator uses to interact with the LOGOS agent community. UIFA receives notification of anomalies from the FIRE agent, handles the logon of users to the system, keeps the user informed with reports, and routes commands to be sent to the satellite and other maintenance functions. If the attention of an analyst is needed but none is logged on, UIFA will send a request to the PAGER agent to page the required analyst.

The VisAGE Interface Agent (VIFA) interfaces with the VisAGE 2000 data visualization system. VisAGE is used to display spacecraft telemetry and agent log information. Real-time telemetry information is displayed by VisAGE as it is downloaded during a satellite pass. VIFA requests the data from the GIFA and AIFA agents (see below). An analyst may also use VisAGE to visualize historical information to help monitor spacecraft health or to determine solutions to anomalies or other potential spacecraft problems.

The Pager Interface Agent (PAGER) is the agent community interface to the analyst's pager system. If an anomaly occurs, or another situation arises that necessitates an analyst's attention, a request is sent to the PAGER agent, which, in turn, causes the analyst to be paged.

The Database Interface Agent (DBIFA) and the Archive Interface Agent (AIFA) store short-term and long-term data, respectively, and the Log agent (LOG) stores agent logging data for debugging, illustration, and monitoring purposes. The DBIFA stores such information as the list of valid users and their passwords; the AIFA stores telemetry data.

The GenSAA/Genie Interface Agent (GIFA) interfaces with the GenSAA/Genie ground station software [7] that handles communications with the spacecraft. GIFA has the capability to download telemetry data, maintain scheduling information, and upload commands to the spacecraft. Upon downloading anomalies and other data from the spacecraft, GIFA routes the data to other agents based on their requests for information.

The MOPSS (Mission Operations Planning and Scheduling System) Interface Agent (MIFA) interfaces with the MOPSS ground station planning and scheduling software. MOPSS keeps track of the satellite's orbit, the time of the next pass, and how long it will last. It also sends out updates to subscribing agents when the schedule changes.

The agent architecture for LOGOS was rather simplistic (which was some of the motivation behind development of the ACT architecture) and consisted of three components: Processor, ACL Messenger, and Perceptor/Effector. It also used an in-house developed software backplane for interagent communication, called Workplace, that serialized agent messages, routed them over the Internet to the destination agent, and then deserialized them. The main component of the LOGOS agents is the Processor that contained all of the noncommunication functionality. The ACL Messenger component performed the message sending and receiving via Workplace, and the Perceptor/Effector interfaced with the external legacy systems, databases, or user interfaces.

*2) Example Scenario:* An example scenario of how agents in LOGOS communicate and cooperate starts with MIFA receiving data from the MOPSS scheduling software, which indicates that the spacecraft will be in a contact position in 2 min. MIFA then sends a message to the other agents, informing them of the upcoming event, in case they need to perform some preprocessing before the contact. When GIFA receives the message from MIFA, it sends a message to the GenSAA Data Server to start receiving transmissions from the control center.

After receiving data, the GenSAA Data Server sends the satellite data to GIFA, which has rules indicating what data to send to which agents. Besides sending data to other agents, GIFA also sends all engineering data to the archive agent (AIFA) for storage, as well as trend information to the visualization agent (VIFA). GIFA sends updated schedule information to the scheduling agent (MIFA) and sends a report to the UIFA to be sent on to an analyst for monitoring purposes. If there are any anomalies, GIFA sends them to the FIRE agent for resolution.

If there is an anomaly, the FIRE agent tries to resolve it automatically via a knowledge base containing anomalies and possible resolutions for each. To fix an anomaly, FIRE would send a spacecraft command to GIFA to be forwarded on to the spacecraft. After exhausting its knowledge base, if FIRE is not able to fix the anomaly, it forwards the anomaly to the UIFA which then pages an analyst and displays it on the analyst's computer for action. The analyst would then formulate a set of commands to send to the spacecraft to resolve the situation. The FIRE agent, upon receiving the commands, would add the new resolution to its knowledge base for future reference and send the commands to the GIFA agent, which would then send them to the GenSAA/Genie system for forwarding on to the spacecraft.

There are many other interactions going on between the agents and the legacy software that are not covered above. Examples include the DBIFA requesting user logon information from the database, the AIFA requesting archived telemetry information from the archive database to be sent to the visualization agent, and the pager agent sending paging information

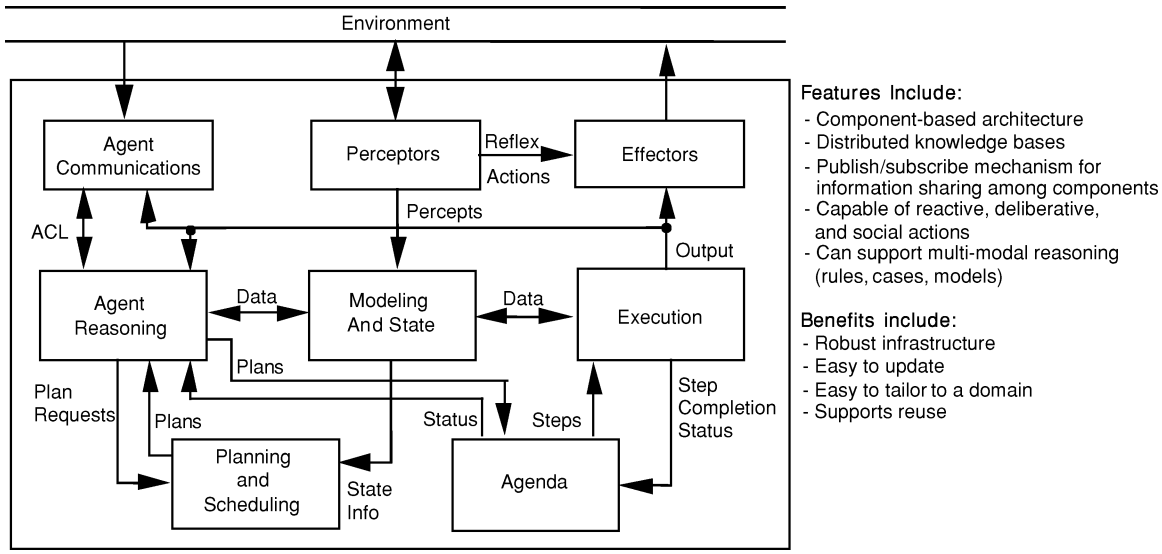


Fig. 3. ACT agent architecture.

to the paging system to alert an analyst of an anomaly requiring attention.

### C. ACT

The motivation behind ACT was to develop a more flexible architecture than LOGOS for implementation of a wide range of intelligent or reactive agents. After developing the architecture, sample agents were built to simulate ground control of a satellite constellation mission as a proof of concept. The following discusses the ACT agent architecture and gives an operational scenario using the satellite constellation proof of concept.

*1) ACT Architecture:* Agents in ACT are built using a component architecture, where a component can be easily swapped out and replaced by another more advanced component. This allows for easy removal of unneeded components for reactive agents and the inclusion of the necessary components to implement intelligent agents. It also allows for additional unforeseen components implemented with new AI technologies to be added as they become available, without affecting previously implemented components. A simple (reactive) agent can be designed by using a minimum number of components that receive percepts (inputs) from the environment and react according to those percepts. A robust agent may be designed using more complex components that allow the agent to reason in a deliberative, reflexive, and/or social fashion. This robust agent would maintain models of itself, other agents, objects in the environment, and external resources. Fig. 3 depicts the components involved in a robust ACT agent.

The following sections describe the components listed in Fig. 3 and the framework in which the components are implemented.

*a) Modeler:* The modeling component is responsible for maintaining the domain model of an agent, which includes models of the environment, other agents, and the agent itself. The Modeler receives data from the Perceptors and agent communication components. This data is used to update state information in its model. If the data causes a change in a state variable, it publishes this information to other components that have subscribed to updates regarding that variable. It is also responsible for reasoning with the models to act both proactively and reactively both with the environment and events that affect the model's state. In the future, the Modeler will also dynamically modify its model based on experience.

The Modeler can also handle "what-if questions." These questions would primarily come from the planning and scheduling component, but may also come from other agents or from a person who wants to know what the agent would do in a given situation or how a change in the agent's environment would affect the values in its model.

*b) Reasoner:* The Reasoner component works with information in its local knowledge base, as well as with the model and state information from the Modeler, in order to make decisions and formulate goals for the agent. This component reasons with state and model data to determine whether any actions need to be performed to affect the agent's environment, change its state, perform housekeeping tasks, or influence other general activities. The Reasoner will also interpret and reason with agent-to-agent messages. When action is necessary for the agent, the Reasoner will produce goals for the agent to try to achieve. Currently, the Reasoner works more in a reactive manner. Either an input coming in or a trigger from the clock sets it in motion. Work is also being undertaken to make the Reasoner more proactive.

*c) Planner/Scheduler:* The Planner/Scheduler component is responsible for any agent-level planning and scheduling. The planning component receives a goal or set of goals to fulfill in the form of a plan request. This typically comes from the Reasoner component, but may be generated by any component in the system. At the time a plan request is received, the planning and scheduling component acquires a state of the agent and system, usually the current state, as well as actions that can be performed by this agent (typically from the modeling and state component). The planning and scheduling component then generates a plan as a directed graph of steps, which is

composed of preconditions, an action to perform, and expected results (postcondition). Each step is also passed to any Domain Expert components/objects for verification of correctness. If a step is deemed incorrect or dangerous, the Domain Expert may provide an alternative step or solution to be considered. Upon completion, the Planner/Scheduler sends the plan back to the component that requested it (usually the Reasoner). The requesting component then either passes it on to the Agenda to be executed or uses it for planning/what-if purposes.

*d) Agenda/Executive:* The Agenda and the Executive work together to execute the plans developed by the Planner/Scheduler. The Agenda typically receives a plan from the Reasoner, though it can receive a plan from another component that is acting in a reactive mode. It interacts with the Executive component to send the plan's steps, in order, for execution and keeps track of which steps are being executed, finished executing, idle, or waiting for execution and updates the status of each step as it moves through the execution cycle. The Agenda reports the plan's final completion status to the Planner and Reasoner when the plan is complete.

The Executive executes the steps it receives from the Agenda. If the preconditions are met, the action is executed. When execution finishes, the Executive evaluates the postconditions and generates a completion status for that step. The completion status is then returned to the Agenda.

A watch, attached to the Executive, monitors given conditions during execution of a set of steps. Watches allow the Planner to flag things that have to be "looked out for" during real-time execution, which can be used to provide "interrupt" capabilities within the plan. An example would be to monitor drift from a guide star while performing an observation. If the drift exceeds a threshold, then the observation is halted. In such a case, the watch would notify the Executive, which in turn would notify the Agenda. The Agenda would then inform the Reasoner that the plan failed and the goal was not achieved. The Reasoner would then formulate another goal (e.g., recalibrate the star tracker).

*e) Agent communications:* The agent communication component is responsible for sending and receiving messages to/from other agents. The component takes an agent data object that needs to be transmitted to another agent and converts it to a message format understandable by the receiving agent. The message format being used is based on Foundation for Intelligent Physical Agents (FIPA) [8] standards, and messages are sent to the appropriate agent using the Workplace messaging backbone.

The reverse process occurs for an incoming message. The communications component takes the message and converts it to an internal object and sends it to the other components that are subscribing to incoming messages. The communications component can also have reactive behavior, where, for a limited number of circumstances, it produces an immediate response to a message.

*f) Perceptors/Effectors:* The Perceptors are responsible for monitoring the environment on behalf of the agent. An example of an environment is a spacecraft subsystem. Any data received by the agent from the environment, other than agent-to-agent messages, enters through the Perceptors. An agent may have zero or more Perceptors, where each Perceptor receives information from specific parts of the agent's environment. A Perceptor may just receive data and pass it on to another component in the agent or it may perform some simple filtering/conversion before passing it on. A Perceptor may also be designed to act intelligently through the use of reasoning systems. If an agent is not monitoring the environment, then it would not have any Perceptors (an example of this would be an agent that only provides expertise, such as fault resolution, to other agents).

The Effector is responsible for effecting or sending output to the agent's environment. Any agent output data, other than agent-to-agent messages, leaves through Effectors. Typically the data leaving the Effectors will be sent from the Executive, which has just executed a command to send data to the environment. There may be zero or more Effectors, where each Effector sends data to specific parts of the agent's environment. An Effector may perform data conversions and act intelligently and in a proactive manner when necessary. As with the Perceptors, an agent may not have an Effector if it is not required to interact with the environment.

*g) Agent framework:* A framework is used to provide base functionality for the components as well as the intercomponent communication facility. The framework allows components to be easily added and removed from the agent while providing a standard communications interface and functionality across all components. This makes developing and adding new components easier and makes new additions transparent to existing components in the agent. Each component in the architecture can communicate information to/from all other components as needed.

The primary communications for components is based on a publish-and-subscribe model with direct links between components if large amounts of data need to be transferred. Components communicate to each other the types of data that they produce when queried. When one component needs to be informed of new or changed data in another component, it subscribes to that data in the other component. Data can be subscribed to whenever it is changed, or on an as needed basis. With this mechanism, a component can be added or removed with no need to modify other components in the agent.

*h) Data flow between components:* Consider an example of how data flows between components of the ACT architecture when a spacecraft's battery is discharging. The scenario reads as follows.

1) The agent detects a low voltage when reading data from the battery via a Perceptor. The Perceptor then passes the voltage value to the Modeler that has subscribed to the Perceptor to receive all percepts.
2) When the Modeler receives the voltage from the Perceptor, it updates the value in its model. In this case, the new value puts it below the normal threshold and changes the voltage state to "low," which causes a state change event and causes the Modeler to publish the new value to all subscribing components. Since the Reasoner is a subscriber, the low voltage value is sent to the Reasoner.
3) In the Reasoner, the low voltage value fires a rule in the expert system. This rule calls a method that sends the

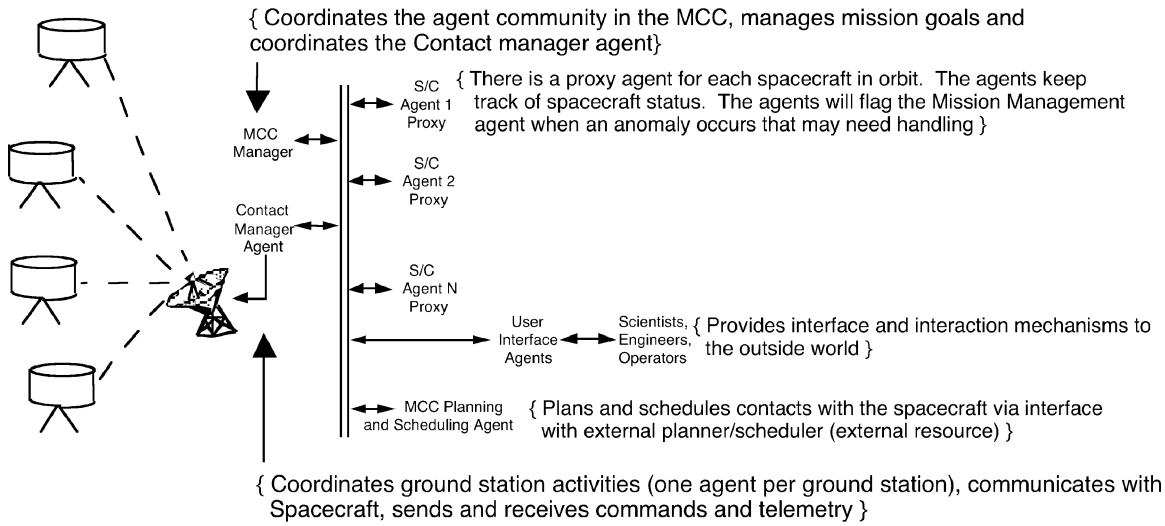


Fig. 4. Agent community being developed in ACT to test out the new agent architecture and some community concepts.

Planner/Scheduler a goal to achieve a battery voltage level that corresponds to a full charge.

4) When the Planner/Scheduler receives the goal from the Reasoner, it queries the Modeler for the current state of the satellite and a set of actions that can be performed.
5) After receiving the current state of the satellite and the set of available actions from the Modeler, the Planner/Scheduler formulates a list of actions that need to take place to charge the battery. It then sends the plan back to the Reasoner for validation.
6) The Reasoner examines the set of actions received from the Planner/Scheduler and decides that it is reasonable. The plans are then sent to the Agenda.
7) The Agenda puts the action steps from the plan into a queue for the Executive.
8) When the Executive is ready to execute a new step, the Agenda passes it along for execution, in the normal one-step-at-a-time fashion.
9) The Executive executes each action until the plan is completed and then notifies the Agenda when it is done.
10) The Agenda marks the plan as finished and notifies the Reasoner that the plan finished successfully.
11) After the plan is executed, the voltage rises and triggers a state change in the Modeler when the voltage returns to a fully charged state. At that point the Reasoner is again notified that a change in a state variable has occurred.
12) The Reasoner then notes the voltage has been restored to a fully charged level and marks the goal as accomplished.

*2) ACT Operational Scenario:* Fig. 4 illustrates an operational scenario involving a possible ACT agent community for a nanosatellite constellation. It is based on the idea of a ground-based community of proxy agents—each representing a spacecraft in the nanosatellite constellation—which provide for autonomous operations of the constellation. Other scenarios for the migration of this community of proxy agents to the spacecraft are discussed in terms of space-based autonomy concepts [9].

In the current scenario, there are several nanosatellites in orbit collecting magnetosphere data. The Mission Control Center (MCC) makes contact with selected spacecraft when they come into view according to the schedule. The agents that make up the MCC are as follows.

1) Mission Manager Agent (MMA): Coordinates the agent community in the MCC, manages mission goals, and coordinates with the Contact Manager Agent.
2) Contact Manager Agent (CMA): Coordinates ground station activities, communicates with the spacecraft, and sends and receives data, commands, and telemetry.
3) UIFA: Sends data to users for display and gets commands for the spacecraft.
4) MCC Planning/Scheduling Agent: Plans and schedules contacts with spacecraft via external Planner/Scheduler.
5) Spacecraft Proxy Agents: Keep track of spacecraft status, health and safety, etc. The proxies notify the MMA when anomalies occur that need handling.

Each of the above agents registers with the GCC manager agent. The GCC manager agent notifies them when a contact is approaching for their spacecraft, whether another agent is going to be added to the community, and how to contact another agent.

The following is a spacecraft contact scenario that illustrates how the agents work with the GCC manager agent.

1) Agents register with the GCC Manager Agent at startup.
2) GCC Planner/Scheduler Agent communicates with the proxy agents to get spacecraft pass time data. It then creates a contact schedule for all orbiting spacecraft.
3) GCC Manager Agent receives the schedule from the GCC Planner/Scheduler Agent and gives details of the next contact to the CMA.
4) The CMA contacts the spacecraft at the appropriate time and downloads the telemetry and sends it to the appropriate spacecraft proxy agent for processing.
5) The spacecraft proxy agents process the telemetry data, update the spacecraft's status, evaluate any problems, and send any commands to the Contact Manager to upload.

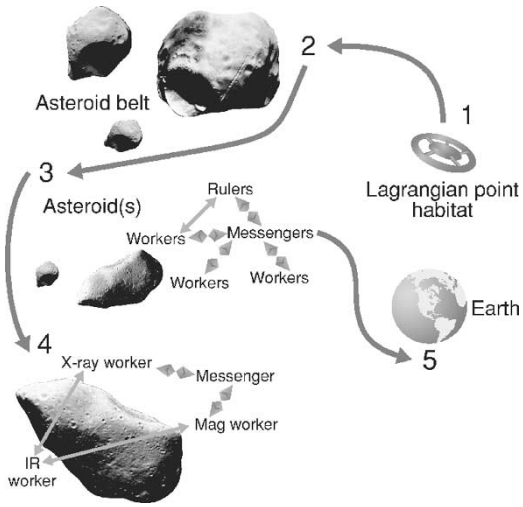


Fig. 5. ANTS concept mission.

6) If a proxy agent determines that a problem exists and an extended or extra contact is needed, it sends a message to the GCC Planner/Scheduler Agent, which replans the contact schedule and redistributes it to the GCC Manager.
7) The Contact Manager downloads data from, and uploads any commands to, the spacecraft as instructed by the spacecraft proxy agent. The CMA ends the contact when scheduled.

An example of a typical contact with a satellite would be as follows.

1) The CMA receives an acquisition of signal (AOS) from a spacecraft. The MCC is now in contact with the spacecraft.
2) The CMA requests the spacecraft to start downloading its telemetry data and sends the data to its proxy agent.
3) The proxy agent updates the state of its spacecraft model from the telemetry received. If a problem exists, the MMA is contacted and appropriate action (if any) is planned by the system.
4) The CMA analyzes the downloaded telemetry data. If there is a problem, the CMA may alter the current contact schedule to deal with the problem.
5) The CMA executes the contact schedule to download data, delete data, or save data for a future contact.
6) The MMA ends contact.

## IV. Concept Autonomous and Autonomic Mission

The NASA Autonomous Nano-Technology Swarm (ANTS) mission [10]–[13] will be made up of swarms of autonomous pico-class (approximately 1 kg) spacecraft that will explore the asteroid belt. There will be approximately 1000 spacecraft involved in the mission consisting of several types (Fig. 5). Approximately 80% of the spacecraft will be workers (or specialists) that will have a single specialized instrument onboard (e.g., a magnetometer, X-ray, gamma-ray, visible/IR, neutral mass spectrometer) and will obtain specific types of data. Some will be coordinators (called *leaders*), and will have rules that determine the types of asteroids and data the mission is interested in, and will coordinate the efforts of the *workers*. The third type of spacecraft are *messengers*, which will coordinate communications between the workers, leaders, and the earth. Each worker spacecraft will examine asteroids it encounters and send messages back to a leader that will evaluate the data and send other spacecraft with specialized instruments to the asteroid to gather further information.

This mission will involve a high degree of autonomy for reasons to be discussed, and autonomic properties will enhance its survivability. To implement this mission, a heuristic approach is being considered that uses an insect analogy of a hierarchical social structure based on the above spacecraft hierarchy. A transport ship will assemble the spacecraft during the journey to the asteroid belt and then release them upon arrival. Artificial intelligence technologies such as genetic algorithms, neural nets, fuzzy logic, and onboard planners are being investigated to assist the mission to maintain a high level of autonomy. Subswarms will be formed which will act as teams that explore a particular asteroid based on the asteroid's characteristics. To examine an asteroid, spacecraft will need to cooperate since they each have only a single instrument onboard. Crucial to the mission will be the ability to modify operations autonomously to reflect the changing goals of the mission and the distance and low-bandwidth communications to the earth.

A scenario for the ANTS mission is based on the ANTS targeting an asteroid on which to perform an experiment and then forming a team to carry out that experiment. Team leaders contain models of the types of science they wish to perform. Parts of this model are communicated to the messenger spacecraft which relay it on to the worker spacecraft. The worker spacecraft will then take measurements of asteroids, using their specialized instrument until data matches the goal that was sent by the leader. The data will then be sent to a messenger to be relayed back to the leader. If the data matches the profile of the type of the asteroid that is being searched for, an imaging spacecraft will be sent to the asteroid to ascertain the exact location and to create a rough model prior to the arrival of other spacecraft, which they will use to maneuver around the asteroid.

Other spacecraft would then work together to finish the model and mapping of the asteroid as well as form virtual instruments that would include the following:

1) an asteroid detector/stereo mapper team that would consist of two spacecraft with field imaging spectrometers and a dynamic Modeler with an enhanced radio science instrument for measuring dynamic properties (such as spin, density, and mass distribution);
2) a petrologist team that would consist of X-ray, near infrared, Gamma-ray, thermal IR, and a wide field imager to determine the distribution of elements, minerals, and rocks present;
3) a photogeologist team that would consist of narrow field and wide field Imagers and Altimeter to determine the nature and distribution of geological units based on texture, albedo, color, and apparent stratigraphy;
4) a prospector team consisting of an altimeter, magnetometer, near infrared, infrared, and X-ray spectrometers to determine the distribution of resources.

## V. Analysis

This section discusses the autonomic properties of the LOGOS and ACT multiagent systems and the ANTS asteroid mission.

### A. Autonomic Properties of LOGOS

The operational scenarios of LOGOS exhibit the four objective autonomic properties, or self-CHOP properties, as a community.

*1) LOGOS Self-Configuration:* LOGOS self-configures when the GIFA agent receives signals from the GenSAA/Genie ground station software that a spacecraft pass is about to happen. When this occurs, the GIFA configures the system by waking up the necessary agents for the pass. For example, if there are no anomalies, then the FIRE agent is not needed and is not woken up. If it is needed, then LOGOS is configured for that pass with the FIRE agent up and ready to receive the anomaly. The same is true for the visualization agent and the UIFA. If there is no user logged on, then those agents do not need to be woken up for the spacecraft pass.

*2) LOGOS Self-Optimization:* LOGOS self-optimizes itself through learning. One example of this is through the learning that the FIRE agent does when it does not know how to fix an anomaly and notifies an analyst that it needs help. After the analyst provides a set of commands to fix the anomaly, the FIRE agent stores those commands and the parameters to that anomaly in its knowledge base for future reference. In this way, it will be able to fix this problem when it occurs again; this may also be viewed as being *self-healing* and illustrates the interdependency of autonomic properties.

A second way that LOGOS self-optimizes is through the user interface and visualization agents. These agents keep track of which analyst looks at what data so that the information would be prefetched and available to the analyst when he or she logs on to the system. This saves time for the analyst, especially in a critical situation. A third example is the pager agent which notifies analysts when an anomaly is present. This agent also keeps track of information that specifies which analysts are available at what times and modifies who it calls first, based on their usual availability.

*3) LOGOS Self-Healing:* LOGOS self-heals primarily through the actions of the FIRE agent. The FIRE agent examines anomalies that occur and then issues commands to fix/heal the anomalies based on its knowledge base. It also self-heals through the intervention of the human in the loop, who can fill in information when the FIRE agent does not have the requisite knowledge to solve a problem. The human is viewed as part of the overall system architecture. The FIRE agent also learns how to fix future anomalies based on inputs from the analyst when the FIRE agent needs help. The self-healing aspect of LOGOS was its primary function, and is what made the system "lights-out" and enabled lower costs of future operations through reduced man-power requirements.

*4) LOGOS Self-Protecting:* The self-protecting aspects of LOGOS are limited. The self-protection is primarily achieved by the FIRE and the UIFA. UIFA performs self-protection when it authenticates a user logging on to the system to ensure the user has proper credentials. For the FIRE agent, self-protection is accomplished when checking commands entered by the analyst to ensure they do not harm the spacecraft, although it can be overridden by the analyst.

### B. Autonomic Properties of ACT

The various operational scenarios of ACT exhibit at least three types of autonomic functionality and some of a fourth. The autonomic functionalities exhibited are self-configuring (adaptation to changing environment), self-optimizing (steps to maximize utilization), self-healing (ability to recover from anomalies), and self-protecting (protect against failures). The following further discusses ACT's autonomic properties.

*1) ACT Self-Configuration:* As an example of this property, when ACT detects, from analysis of downloaded telemetry, that there is a problem, the Contact Manager alters the current satellite contact schedule to enable the problem to be addressed. What is being reconfigured, in this case, is the spacecraft functionality for managing communications contacts with ground systems and controllers.

*2) ACT Self-Optimization:* As an example of this property, consider what happens when a proxy agent determines that a problem exists with its associated spacecraft. When this situation arises, a replanning/rescheduling activity is performed to optimize the behavior of the entire ACT system.

*3) ACT Self-Healing:* As an example of this, again consider what happens when a proxy agent detects a problem with its associated spacecraft. Following a diagnosis of the problem (which may involve access to the human component of the ACT), corrective actions, in the form of commands, are generated and made ready for transmission to the affected spacecraft. This problem-diagnosis/corrective-action cycle is a major part of ACT's self-healing capability.

It should be noted that the three autonomic responses discussed above stem from ACT's determination that a problem has occurred. In attending to the problem, ACT reconfigures, tries to optimize its operations, and proceeds to diagnose and solve the identified problem.

*4) ACT Self-Protection:* ACT is self-protecting in the sense that it constantly monitors the spacecraft systems and modifies its operations if a parameter ranges outside its normal bounds. An example of self-protection is given in the above explanation of the dataflow between components of the architecture. In this example, the battery is discharging and if nothing is done the spacecraft will lose power and become inoperable. ACT then takes the necessary actions to recharge the battery (e.g., turning towards the sun). In addition, it also has self-protection through validation of system commands to ensure that command sequences executed will not harm the spacecraft or put it in a position where it could be harmed.

### C. Autonomic Properties of ANTS

*1) ANTS Self-Configuration:* ANTS has an overall requirement to prospect thousands of asteroids per year with large but limited resources. To accomplish this, it is anticipated that there

will be approximately one month of optimal science operations at each asteroid prospected. A full suite of scientific instruments will be deployed at each asteroid. The ANTS mission resources will be configured and reconfigured to support concurrent operations at hundreds of asteroids over a period of time.

The overall ANTS mission architecture calls for specialized spacecrafts that support division of labor (rulers, messengers) and optimal operations by specialists (workers). A major feature of the architecture is support for cooperation among the spacecraft to achieve mission goals. The architecture supports swarm-level mission-directed behaviors, sub-swarm levels for regional coverage and resource-sharing, team/worker groups for coordinated science operations, and individual autonomous behaviors. These organizational levels are not static but evolve and self-configure as the need arises. As asteroids of interest are identified, appropriate teams of spacecraft are configured to realize optimal science operations at the asteroids. When the science operations are completed, the team disperses for possible reconfiguration at another asteroid site. This process of configuring and reconfiguring continues throughout the life of the ANTS mission.

Reconfiguring may also be required as the result of a failure or anomaly of some sort; the following are some examples: A worker may be lost due to collision with an asteroid, failure of its communication devices, or hardware failure. The loss of a given worker may result in the role of that worker being performed by another, which will be allocated the tasks and resources of the original. Loss of communication with a worker may mean that the system has to assume loss of the worker and the role may be allocated to another spacecraft. Loss of use of an instrument by a worker may require the worker to take the role of a communication device.

*2) ANTS Self-Optimization:* Optimization of the ANTS is undertaken at the individual level, as well as at the system level. These optimizations are as follows:

1) rulers learning about asteroids;
2) messengers adjusting their position;
3) workers learning about asteroids.

Optimization at the ruler level is primarily through learning. Over time, rulers will be collecting data on different types of asteroids and will evolve to be able to better determine the characteristics of the types of the asteroids that are of interest, and perhaps the types of asteroids that are difficult to orbit or extract data from (e.g., an asteroid with a fast rotation that is difficult to focus on). From this information, the system as a whole is being optimized since time is not being wasted on asteroids that are not of interest or too difficult to map.

Optimization for messengers is achieved through positioning. Messengers need to provide communications between the rulers and workers as well as back to the earth. This means that a messenger will have to be constantly adjusting its position to balance the communications between the rulers and workers and perhaps adjusting its position to send data to the earth while also maintaining communications between rulers and workers.

Optimization at the worker level is primarily through its experience gained with asteroids. As a worker observes asteroids and builds up a knowledge base of the different characteristics of asteroids, a worker may be able to automatically skip over asteroids that are not of interest, thus saving time and optimizing the exploration of the mission as a whole.

*3) ANTS Self-Healing:* The view of self-healing here is slightly different from that given in [1]. ANTS is self-healing not only in that it can recover from mistakes, but self-healing in that it can recover from failure, including damage from outside force. In the case of ANTS, these are nonmalicious sources: events such as collision with an asteroid, or another spacecraft, loss of connection, etc., will require ANTS to heal itself by replacing one spacecraft with another.

ANTS mission self-healing scenarios span the range from negligible to severe. An example entailing negligible self-healing would be an instance where one member of a redundant set of gamma ray sensors fails before a general gamma ray survey is planned. In such a scenario, the self-healing behavior would be the simple action of deleting the sensor from the list of functioning sensors. At the severe end of the scale, an example scenario would arise when the team loses so many workers that it can no longer conduct science operations. In this case, the self-healing behavior might be to advise the mission control center and, when a replacement worker arrives, to incorporate the replacement into the team, performing, additionally, any necessary self-configuration and self-optimization. In some possible ANTS mission concepts, instead of "calling home" for help, an ANTS team may only need to request a replacement from another team or from a fielded repository of spares orbiting in the vicinity.

Not only the ANTS team, but also the ANTS individuals may have self-healing behaviors. For example, an individual may have the capability of detecting corrupted code (software). In such a case, self-healing behavior would result in the individual requesting a copy of the affected software from another individual in the team, which would enable it to restore itself to a known operational state.

*4) ANTS Self-Protection:* The self-protecting behavior of the team will be interrelated with the self-protecting behavior of the individual members. The anticipated sources of threats to ANTS individuals (and consequently to the team itself) will be collisions and solar storms.

Collision avoidance through maneuvering will be limited, because ANTS individuals will have limited ability to adjust their orbits and trajectories, since thrust for maneuvering is obtained from solar sails. Individuals will have the capability of coordinating their orbits and trajectories with other individuals to avoid collisions with them. Given the chaotic environment of the asteroid belt and the highly dynamic trajectories of the objects in it, occasional near approaches of interloping asteroidal bodies (even small ones) to the ANTS team may present threats of collisions. Collision-avoidance maneuvering for this type of spacecraft presents a large challenge and is currently under consideration. The main self-protection mechanism for collision avoidance is achieved through the process of planning. The ruler's plans involve constraints that will result in acceptable risks of collisions between individuals when they carry out the observational goals given by the ruler. In this way, ANTS exhibits a kind of self-protection behavior against collisions.

Another possible ANTS self-protection mechanism could protect against effects of solar storms. Charged particles from solar storms could subject individuals to degradation of sensors and electronic components. The increased solar wind from solar storms could also affect the orbits and trajectories of the ANTS individuals and thereby jeopardize the mission. ANTS mechanisms that are protective against effects of solar storms have not been determined or included in the mission design. One possible mechanism would involve a capability of the ruler to receive a warning message from the mission control center on the earth. An alternative mechanism would be to provide rulers with a solar storm sensing capability through onboard direct observation of the solar disk. When the ruler recognizes a solar storm threat exists (either upon receipt of a solar storm warning from the control center or upon reaching its own conclusion from direct observations), the ruler would invoke its goal to protect the mission from harm from the effects of the solar storm. In addition to its own action to protect itself, part of the ruler's response would be to give workers the goal to protect themselves. Part of an individual's protective response might be to orient solar panels and sails to minimize impact of the solar wind. An additional response might be to power down subsystems to minimize disruptions and damage from charged particles.

Thus, with such capabilities, an ANTS mission will exhibit self-protecting behavior. As noted in the section on self-configuring behavior, after-effects of protective action will, in general, necessitate ANTS self-reconfiguration. For example, after solar sails had been trimmed for the storm blast of solar wind, individuals will have unplanned trajectories, which will necessitate trajectory adjustments and replanning and perhaps new goals. Further, in case of the loss of individuals due to damage by charged particles, the ANTS self-healing behavior and the self-optimizing behavior may also be triggered. Thus, there is an interrelatedness of the self-protecting behaviors of the ANTS team and the ANTS individuals.

### D. Lessons Learned

Whereas LOGOS demonstrated that typical control center activities could be emulated by a multiagent system, the major objective of the ACT project was to demonstrate that ground-based surrogate agents, each representing a spacecraft in a group of spacecraft, could control the overall dynamic behaviors of the group of spacecraft in the realization of some global objective. The ultimate objective of the ACT was to help in the understanding of the idea of progressive autonomy which would, as a final goal, allow the surrogate agents to migrate to their respective spacecraft and then allow the group of autonomous spacecraft to have control of their dynamic behaviors independent of relying on ground control.

ACT correctly emulated the correct interaction between surrogate agents and their respective spacecraft. This "correctness" was determined by comparison between what the surrogate did versus what a human controller on the ground would have done, in conjunction with what the controllers associated with the other surrogates would have done, to achieve a global objective. This analysis was undertaken more at a heuristic level than at a formal level. The design of the surrogates was realized in a modular fashion. This design was performed in this fashion in order to support the concept of incremental placement of the functional capabilities of the surrogate agent in the respective spacecraft, until the spacecraft was truly agent-based and autonomous. This particular aspect of the ACT project was heuristically realized but not rigorously (formally) tested out.

### E. Verification and Correctness

The fact that the correctness of ACT (which was after all a prototype system) was heuristically realized rather than formally, or even systematically, verified raises an issue: Given that NASA is moving towards more complex systems based on the idea of swarm technologies and moving from self-direction to self-governance, how can we be assured of the correctness of these systems?

The use of *formal methods* has been identified as a means of dealing with this complex problem. Formal approaches were previously used in the specification and verification of the LOGOS system. A formal specification in CSP highlighted a number of errors and omissions in the system. These, and other, errors were also found by an automated tool [22] which implements a NASA patent-pending approach to requirements-based programming [23].

Mission such as ANTS, however, pose even greater problems: large numbers of interacting components, emergent behavior, evolving behavior (due to learning) with the same behavior patterns unlikely to be repeated, and extremely complex functionality. While ANTS is currently still a concept mission, it is likely that it and other swarm-based approaches will form the basis of many future NASA exploration missions, so the need to address these issues is likely to become more significant in the future [21].

*Formal Approaches to Swarm Technologies (FAST)* is a current NASA project to examine the issues in formally specifying and verifying swarm-based systems. The project has looked at several potential approaches for specifying swarm, using ANTS as its testbed, and is developing a hybrid formal method that will be most appropriate for use in the development of such systems [19], [20].

## VI. Conclusion

NASA missions represent some of the most extreme examples of the need for survivable systems that cannot rely on support and direction from humans while accomplishing complex objectives under dynamic and difficult environmental conditions. Future missions will embody greater needs for longevity in the face of significant constraints, in terms of cost and the safety of human life. Future missions also will have increasing needs for autonomous behavior not only to reduce operations costs and overcome practical communications limitations (signal propagation delays and low data rates), but also to overcome the inability of humans to perform long-term missions in space. There is an increasing realization that future missions must be not only autonomous, but also exhibit the properties of autonomic systems for the survivability of both individuals and systems.

As described, the LOGOS and ACT architectures provide for a flexible implementation of a wide range of intelligent and autonomic agents. The ACT architecture allows for easy removal of components not needed for reactive agents and the inclusion of the necessary components to implement intelligent and autonomic agents. It is also flexible so that additional unforeseen needs can be satisfied by new components that can be added without affecting previous components.

The ultimate goal of our work is to transition the proven agent and autonomic technology into operational NASA systems. The implementation of the scenarios discussed above (and others under development) will provide an opportunity to exercise, evaluate, and refine the capabilities supported by the agent architectures. It will also provide an opportunity for space mission designers and developers to see agent and autonomic technology in action and their resulting benefits. This will enable them to make a better determination of the role that this technology can play in their missions.

We have illustrated the interrelationship of autonomous and autonomic systems with reference to two existing NASA prototype systems, namely, ACT and LOGOS and have examined the relationship for a future mission, ANTS. It is clear that the separation of autonomy and autonomicity as mission characteristics will decrease in the future and eventually will become negligible, with autonomicity being essential to move from self-direction to self-governance.

## REFERENCES


[1] R. Murch, *Autonomic Computing*. Upper Saddle River, NJ: IBM Press and Prentice-Hall, 2004.

[2] C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash, "Properties of a formal method for prediction of emergent behaviors in swarm-based systems," in *Proc. 2nd IEEE Int. Conf. Software Engineering and Formal Methods*, Beijing, China, Sep. 26–30, 2004, pp. 24–33.

[3] W. Truszkowski and C. Rouff, "An overview of the NASA LOGOS and ACT agent communities," presented at the *5th World Multiconf. Systemics, Cybern., and Informatics (SCI 2001)*, Orlando, FL, Jul. 2001.

[4] W. Truszkowski and H. Hallock, "Agent technology from a NASA perspective," in *CIA-99, 3rd Int. Workshop Cooperative Information Agents*, Uppsala, Sweden, Jul. 31–Aug. 2 1999.

[5] J. Ferber, *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*. Reading, MA: Addison-Wesley, 1999.

[6] M. Wooldridge, "Intelligent agents," in *Multiagent Systems*, G. Weiss, Ed. Cambridge, MA: MIT Press, 1999.

[7] P. Hughes, G. Shirah, and E. Luczak, "Advancing satellite operations with intelligent graphical monitoring systems," presented at the *AIAA Computing in Aerospace Conf.*, San Diego, CA, Oct. 1993.

[8] "FIPA Specification Part 2: Agent Communication Language," Foundation for Intelligent Physical Agents (FIPA), Geneva, Switzerland, Nov. 1997.

[9] W. Truszkowski and C. Rouff, "A process for introducing agent technology into space missions," in *Proc. IEEE Aerospace Conf.*, Mar. 11–16, 2001, pp. 2743–2750.

[10] P. E. Clark, S. A. Curtis, and M. L. Rilee, "ANTS: Applying a new paradigm to lunar and planetary exploration," presented at the *Solar System Remote Sensing Symp.*, Pittsburg, PA, 2002.

[11] S. A. Curtis, J. Mica, J. Nuth, G. Marr, M. Rilee, and M. Bhat, "ANTS (Autonomous nano-technology swarm): An artificial intelligence approach to asteroid belt resource exploration," presented at the *Int. Astronautical Federation, 51st Congr.*, Oct. 2000.

[12] S. Curtis, W. Truszkowski, M. Rilee, and P. Clark, "ANTS for the human exploration and development of space," in *Proc. IEEE Aerospace Conf.*, Mar. 8–15, 2003, pp. 1–261.

[13] M. L. Rilee, S. A. Boardsen, M. K. Bhat, and S. A. Curtis, "Onboard science software enabling future space science and space weather missions," in *Proc. 2002 IEEE Aerospace Conf.*, Big Sky, MT, vol. 5, Mar. 9–16, 2002, pp. 2071–2084.

[14] W. Truszkowski, J. Rash, C. Rouff, and M. Hinchey, "Asteroid exploration with autonomic systems," in *Proc. 11th IEEE Int. Conf. Workshop Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*, Brno, Czech Republic, May 24–27, 2004, pp. 484–489.

[15] W. Truszkowski, J. Rash, C. Rouff, and M. Hinchey, "Some autonomic properties of two legacy multi-agent systems—Logos and ACT," in *Proc. 11th IEEE Int. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*, Brno, Czech Republic, Los Alamitos, CA: IEEE Comput. Soc. Press, May 24–27, 2004, pp. 490–498.

[16] T. Ames and S. Henderson, "The workplace distributed processing environment," in *Proc. 1993 Goddard Conf. Space Applications of Artificial Intelligence*, NASA Goddard Space Flight Center, Greenbelt, MD: NASA Conf. Pub. 3200, May 10–13, 1993, pp. 181–188.

[17] R. Sterritt, C. A. Rouff, J. Rash, W. Truszkowski, and M. Hinchey, "Self-* properties in NASA missions," presented at the *Software Engineering Research and Practice (SERP 2005)*, Las Vegas, NV, Jun. 2005.

[18] C. Rouff, J. Rash, and M. Hinchey, "Experience using formal methods for specifying a multi-agent system," in *Proc. 6th IEEE Int. Conf. Engineering of Complex Computer Systems (ICECCS 2000)*, Tokyo, Japan, 2000, pp. 72–80.

[19] C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash, "Properties of a formal method for prediction of emergent behaviors in swarm-based systems," in *Proc. 2nd IEEE Int. Conf. Software Engineering and Formal Methods*, Beijing, China, Sep. 26–30, 2004, pp. 24–33.

[20] C. Rouff, M. Hinchey, J. Rash, and W. Truszkowski, "A survey of formal methods for intelligent swarms," NASA Goddard Space Flight Center, Greenbelt, MD, Tech. Rep., 2005.

[21] M. G. Hinchey, J. L. Rash, W. F. Truszkowski, C. A. Rouff, and R. Sterritt, "Autonomous and autonomic swarms," presented at the *Software Engineering Research and Practice (SERP 2005)*, Las Vegas, NV, Jun. 2005.

[22] J. Rash, M. Hinchey, C. Rouff, and D. Gracanin, "Experiences with a requirements-based programming approach to the development of a NASA autonomous ground control system," in *Proc. IEEE Workshop on the Engineering of Autonomic Systems (EASe 2005), 12th Annu. IEEE Int. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, Greenbelt, MD, Apr. 4–7, 2005, pp. 490–497.

[23] M. G. Hinchey, J. L. Rash, and C. A. Rouff, "A formal approach to requirements-based programming," in *Proc. 12th Int. Conf. Engineering of Computer-Based Systems (ECBS 2005)*, Greenbelt, MD, Apr. 4–7, 2005, pp. 339–345.


**Walter F. Truszkowski** (M'01) received the B.A. degree in mathematics from Loyola College, Baltimore, MD, and the M.S. degree in computer science from the University of Maryland, Baltimore.

He is currently the Senior Technologist in the Advanced Architectures and Automation Branch, NASA Goddard Space Flight Center, Greenbelt, MD. He is the author of more than 30 technical papers and book chapters and has edited four books. His current research interests are in the areas of formal methods, agent/multiagent systems, swarm technologies, evolutionary robotics, autonomic computing, and the semantic web.

Mr. Truszkowski is a member of the ACM, AAAI, and the AIAA, where he serves on the Technical Committee for Autonomous Systems.

**Michael G. Hinchey** (M'91–SM'02) received the B.S. degree in computer science from the University of Limerick, Limerick, Ireland, the M.S. degree in computation from the University of Oxford, Oxford, U.K., and the Ph.D. degree in computer science from the University of Cambridge, Cambridge, U.K., in 1991, 1992, and 1995, respectively.

Prior to joining the U.S. Government, he held academic positions at the level of Full Professor in the U.S., U.K., Ireland, Sweden, and Australia. He is currently the Director of the NASA Software Engineering Laboratory, Goddard Space Flight Center, Greenbelt, MD. He is the

author of more than 100 technical papers and 15 books. His current research interests are in the areas of formal methods, system correctness, and agent-based technologies.

Dr. Hinchey is a Fellow of the Institute of Electrical Engineering and the British Computer Society. Currently, he is the Chair of the IEEE Technical Committee on Complexity in Computing and is the IEEE Computer Society's voting representative to IFIP TC1. He is a Chartered Engineer, Chartered Professional Engineer, Chartered Information Technology Professional, and Chartered Mathematician.

**James L. Rash** (M'87) received the B.A. degree in mathematics and physics and M.A. degree in mathematics from the University of Texas, Austin, in 1963 and 1965, respectively.

Currently, he leads formal methods research and development in the Advanced Architectures and Automation Branch, NASA Goddard Space Flight Center, Greenbelt, MD, where his other major responsibilities include managing the Operating Missions as Nodes on the Internet (OMNI) Project. He has authored/coauthored more than 25 technical papers and articles, coedited three books, and edited eight journal special issues, and has been an organizer of more than 15 conferences and workshops on artificial intelligence, formal methods, and Internet technologies for space missions. His current research-interest areas are formal methods and agent-based technologies.

**Christopher A. Rouff** (S'90–M'90) received the B.A. degree in mathematics/computer science from the California State University, Fresno, the M.S. degree in computer science from the University of California, Davis, and the Ph.D. degree in computer science from the University of Southern California, Los Angeles.

Currently, he is a senior Scientist in the Advanced Concepts Business Unit, Science Applications International Corporation, McLean, VA. He is currently doing research and development on multiagent systems, verification of intelligent systems, and collaborative robotics for NASA and DARPA. Previously, he was with the NASA Goddard for nine years, where he researched and prototyped cooperative multiagent systems for ground and spaceflight applications and led a number of software research and development projects. He has over 40 publications and 20 years of experience in software engineering and intelligent systems.