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*—NASA increasingly will rely on autonomous systems concepts, not only in the mission control centers on the ground, but also on spacecraft and on rovers and other assets on extraterrestrial bodies. Autonomy enables 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 cost-effectively supports accomplishment of mission goals, autonomicity supports survivability of remote mission assets, especially when human tending is not feasible. Autonomic system properties (which ensure self-configuring, self-optimizing, self-healing, and self-protecting behavior) conceptually 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*— Autonomous Systems, Autonomic Systems, Multi-Agent Technology, Intelligent Systems, Spacecraft.**

I. Introduction

With NASA's renewed commitment to outer space exploration, particularly missions to Mars and the return to the Moon, greater emphasis is being placed on both human and robotic exploration. Indeed, NASA has a new (as of 2004) initiative with that title – Human & Robotic Exploration – operated out of NASA Headquarters, Code T. In reality, even when humans are involved in the exploration, human tending of assets becomes cost-prohibitive or is not feasible, and therefore increasingly in future missions, remote mission assets will work autonomously.

Moreover, much of the mission control work on 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 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.

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

The remainder of this paper first discusses the need for autonomy and autonomicity in future NASA missions and then discusses the autonomic properties of three systems: two multi-agent systems developed at NASA Goddard Space Flight Center (GSFC) and a concept mission that is currently planned to launch in the 2020 to 2030 time frame. We will emphasize the 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 our context, a system is autonomous if it can achieve its goals without

Manuscript received October 15, 2004. This work was supported in part by the NASA Office of Safety and Mission Assurance, Software Assurance Research Program.

Walter F. Truszkowski is with the Advanced Architectures and Automation Branch (Code 588), NASA Goddard Space Flight Center, Greenbelt, MD 20771 USA (email: Walter.F.Truszkowski@nasa.gov)

Michael G. Hinchey is with the NASA Software Engineering Laboratory at Goddard Space Flight Center, Greenbelt, MD 20771 (tel: 301 286 9057, fax: 301 286 5719, email: Michael.G.Hinchey@nasa.gov).

James L. Rash is with the Advanced Architectures and Automation Branch (Code 588), NASA Goddard Space Flight Center, Greenbelt, MD 20771 USA (email: James.L.Rash@nasa.gov).

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

human intervention. A number of more-or-less automated ground systems exist today, but work continues towards 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 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 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 cost. Increasing the autonomy and intelligence of 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, which are much more complicated, if not impossible, to operate compared to traditional single spacecraft missions. Spacecraft in swarms and constellations must communicate to coordinate and cooperate with each other. Radio or laser communications of constellation elements with each other 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, these systems in addition to being autonomous will need to have autonomic properties to ensure optimal performance and survival.

B. Autonomicity

NASA needs autonomicity in its missions 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 systems should be autonomic, and exhibit the four key properties of autonomic systems: self-configuring, self-optimizing, self-healing and self-protecting [1]. However, as described later, we find that these four properties of autonomicity are not mutually exclusive. The following discusses 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 of or to help adjust for 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 view 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 other spacecraft or mission operations. Self-healing in a swarm or constellation could include moving another spacecraft into the place of the lost one or requesting a replacement spacecraft from Earth.

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 a sleep mode until it 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 could cover itself or go to a better protected area, such as a rock outcropping or other sheltered area.

C. How both combine

The best possible situation for NASA would be to launch a spacecraft and then simply receive science data from it with no in-flight directions or corrections. Currently NASA is a long way from this utopia. To reach this state of operations, NASA needs its missions to be both autonomous with autonomic properties. Autonomy alone does not guarantee autonomic properties. Autonomous systems can operate independently but do not necessarily have self-configuring, optimizing, healing and protecting properties of autonomic systems. For NASA missions to be fully autonomous in harsh environments they will also need these properties.

Combining autonomy with autonomic properties will require a new set of requirements and verification procedures above and beyond what is currently available. NASA currently has no truly autonomous or autonomic missions. Requirements will have to be developed that reflect these types of missions. While autonomy may have similarities across missions, the autonomic properties would vary depending on the type of the system and where it would operate. This would also be true for verification of autonomous and autonomic systems [2]. New verification procedures need to be developed, either through direct verification, or through simulation if direct verification would damage the system. Since these systems will be intelligent, new methods will have to be developed that can guarantee correct operation.

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 to realize NASA's autonomy goals. The goal of this work is to transition proven agent technology into operational NASA systems. Two major successes of the Branch were the development of the 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 purposes of this paper we will 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 multi-agent 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 multi-agent system developed and provided an initial insight into the power of agent communities, autonomy, and autonomic properties of these systems. The agents in LOGOS acted as surrogate human controllers and interfaced with legacy software that controllers normally used, and with humans.

Based on the success 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 autonomous and autonomic systems, as well as giving an improved understanding of the trade-offs 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" agent technology in action. This has enabled them to make a better determination of the role that agent technology can play in their missions. The remainder of this section describes the LOGOS and ACT agent communities, gives brief operational overviews of each, and highlights the autonomic 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 discusses the LOGOS architecture and gives an example scenario of how LOGOS works.

1) LOGOS Architecture:

The LOGOS community architecture is shown in Figure 1 and the LOGOS agent architecture is shown in Figure 2. LOGOS is made up of ten agents, some that interface with legacy software, some that perform services for the other agents in the community, and others that interface with an analyst or operator. All agents have the ability to

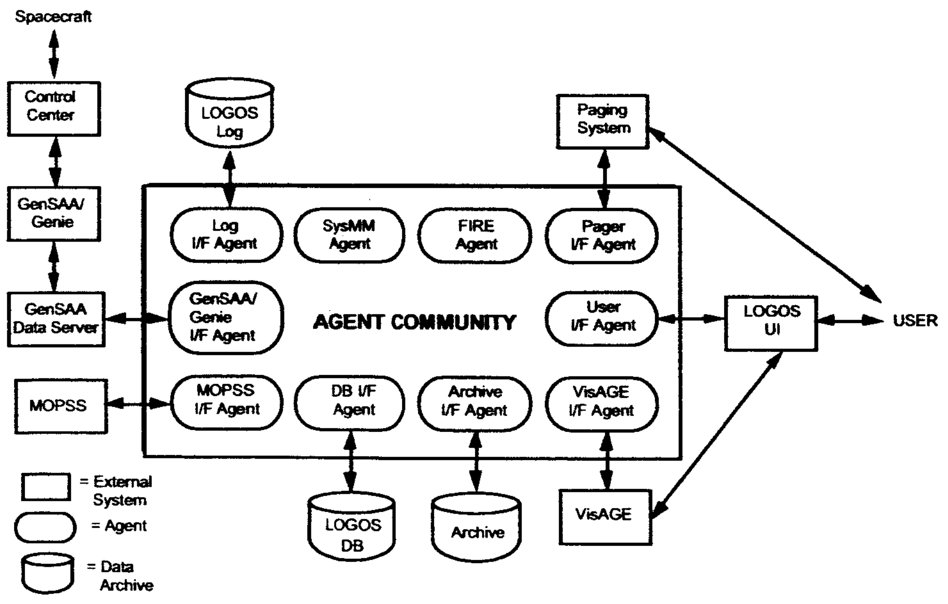


Fig. 1 LOGOS agent community and legacy software.

communicate with all other agents in the community.

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

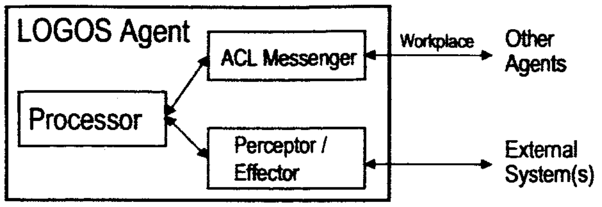


Fig. 2. Architecture of LOGOS Agent

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 to be forwarded to an analyst for resolution.

The User Interface Agent (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], which 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.

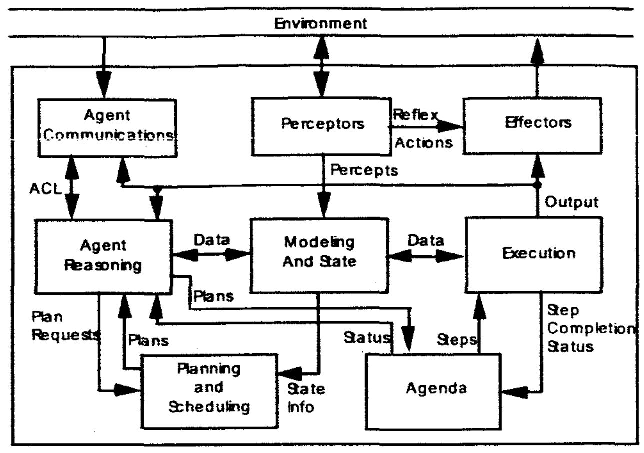
The agent architecture for LOGOS was rather simplistic (which was some of the motivation behind development of the ACT architecture) and consisted of three components: Processor, ACL Messenger, and Perceptor/Effector. It also used an in-house developed software backplane for inter-agent communication,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

2) An Example Scenario

An example scenario of how agents in LOGOS communicate and cooperate starts with MIFA receiving data from the MOPSS scheduling software that indicates the spacecraft will be in contact position in two minutes. MIFA then sends a message to the other agents, informing them of the upcoming event in case they need to do 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. As well as 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 user interface agent (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, then FIRE forwards the anomaly to the user interface agent, 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 would send the commands to the GIFA agent, which would send them to the GenSAA/Genie system for forwarding on to the spacecraft.



Features Include:
- Component-based architecture
- Distributed knowledge bases
- Publish/subscribe mechanism for information sharing among components
- Capable of reactive, deliberative, and social actions
- Can support multi-modal reasoning (rules, cases, models)

Benefits include:
- Robust infrastructure
- Easy to update
- Easy to tailor to a domain
- Supports reuse

Fig. 3. ACT Agent Architecture.

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 to the paging system to alert an analyst of an anomaly needing 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. Figure 3 depicts the components involved in a robust ACT agent.

The following sections describe the components listed in Figure 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, the Modeler publishes this information to other components that have subscribed to updates regarding that variable. The Modeler is also responsible for reasoning with the models to act proactively and reactively 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 model and state information from the Modeler 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 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 (post condition). 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. The Agenda interacts with the Executive component to send the plan's steps, in order, for execution. The Agenda 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 post-conditions, 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 software.

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 for 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 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 affecting 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 inter-component 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 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

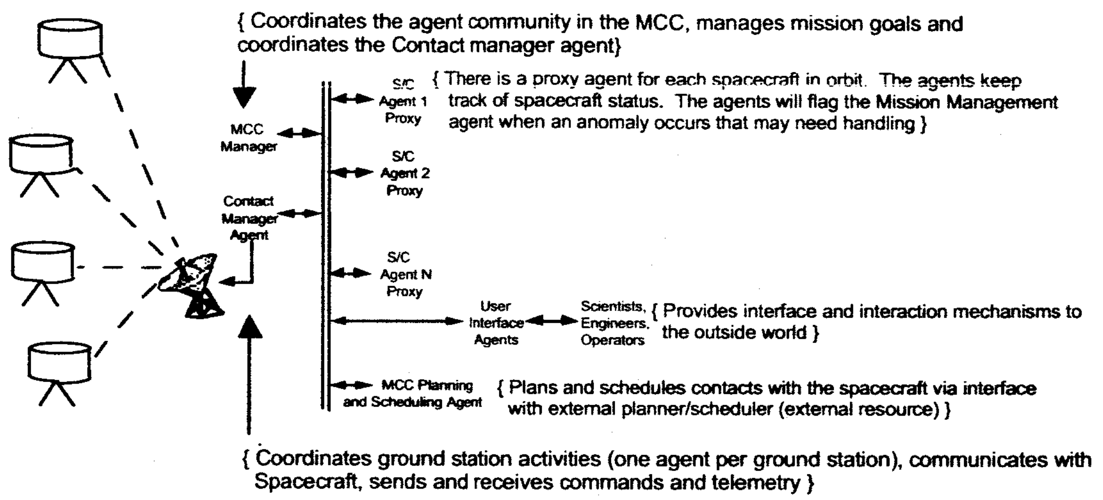

{ Coordinates the agent community in the MCC, manages mission goals and coordinates the Contact manager agent}
MCC Manager
Contact Manager Agent
S/C Agent 1 Proxy
{ There is a proxy agent for each spacecraft in orbit. The agents keep track of spacecraft status. The agents will flag the Mission Management agent when an anomaly occurs that may need handling }
S/C Agent 2 Proxy
S/C Agent N Proxy
User Interface Agents
Scientists, Engineers, Operators
{ Provides interface and interaction mechanisms to the outside world }
MCC Planning and Scheduling Agent
{ Plans and schedules contacts with the spacecraft via interface with external planner/scheduler (external resource) }
{ Coordinates ground station activities (one agent per ground station), communicates with Spacecraft, sends and receives commands and telemetry }

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

voltage value to the Modeler, which 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 the Modeler to publish the new value to all subscribing components. Since the Reasoner is a subscribers, 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 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 then 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 finished and then notifies the Agenda 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*

Figure 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 present 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:

- Mission Manager Agent (MMA): coordinates the agent community in the MCC, manages mission goals, and coordinates with the Contact Manager Agent.
- Contact Manager Agent (CMA): coordinates ground station activities, communicates with the spacecraft, and sends and receives data, commands, and telemetry.
- User Interface Agent: sends data to users for display and gets commands for the spacecraft.
- MCC Planning/Scheduling Agent: plans and schedules contacts with spacecraft via external Planner/Scheduler.
- Spacecraft Proxy Agents: keeps track of spacecraft status, health and safety, etc. The proxies notify the Mission Manager Agent 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:

- Agents register with the GCC Manager Agent at startup.
- 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.
- GCC Manager Agent receives the schedule from the GCC Planner/Scheduler Agent and gives details of the next contact to the Contact Manager Agent.
- The Contact Manager Agent contacts the spacecraft at the appropriate time and downloads the telemetry and sends it to the appropriate spacecraft Proxy Agent for processing.
- 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.
- If a Proxy Agent determines a problem exists and an extended or extra contact is needed, it sends a message to the GCC Planner/Scheduler Agent, which re-plans the contact schedule and redistributes it to the GCC Manager.
- The Contact Manager downloads data from, and uploads any commands to, the spacecraft as instructed by the spacecraft Proxy Agent. The Contact Manager agent ends the contact when scheduled.

An example of a typical contact with a satellite would be:

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

IV. A 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 1kg) satellites that will explore the asteroid belt. There will be approximately 1,000 spacecraft involved in the mission consisting of several types (Figure 5). Approximately 80 percent of the spacecraft will be workers (or specialist) which 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) will have rules that decided 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 and will coordinate communications between the workers, leaders and Earth. Each worker spacecraft will examine asteroids they

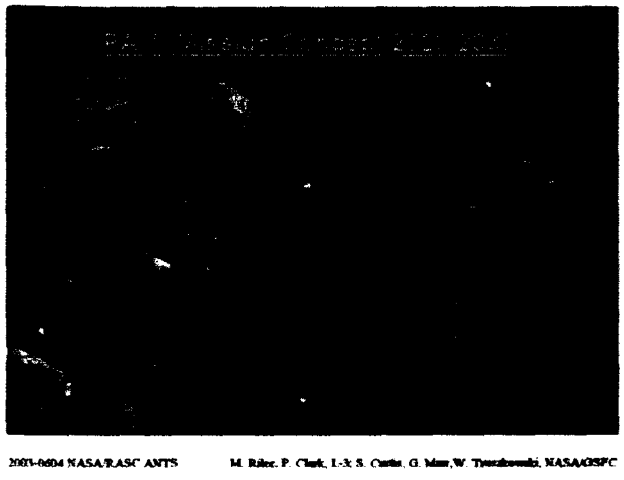

2003-0604 NASA/RASC ANTS
M. Rilee, P. Clark, L-X. S. Curtis, G. Marr, W. Truszkowski, NASA/GSFC

Fig. 5. ANTS Mission Concept.

encounter and send messages back to a coordinator 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 hierarchical social structure based on the above spacecraft hierarchy. A transport ship will manufacture the spacecraft during the journey to the asteroid belt and then release them upon arrival. Replacement spacecraft will be sent from Earth on an as-needed basis. Artificial intelligence technologies such as genetic algorithms, neural nets, fuzzy logic, and on-board planners are being investigated to assist the mission to maintain a high level of autonomy. Sub-swarms will exist that will act as teams that explore a particular asteroid based on the asteroid's characteristics. To examine an asteroid spacecraft will have to cooperate since they only have a single instrument on board. Crucial to the mission will be the ability to modify operations autonomously to reflect the changing nature of the mission and the distance and low bandwidth communications to Earth.

A scenario for the ANTS mission is based on the ANTS targeting an asteroid on which to do an experiment and then forming a team to carry out that experiment. Team leaders contain models of the types of science they want to perform. Parts of this model are communicated to the messenger spacecraft that then relay it on to the worker spacecraft. The worker spacecraft 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 sent back to the leader. If the data matches the profile of the type of 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 so they have a model to use for maneuvering around the asteroid.

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

- 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)
- 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
- 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
- a prospector team consisting of an altimeter, magnetometer, near infrared, infrared, and X-ray spectrometers to determine the distribution of resources

V. Analysis

The following discusses the autonomic properties of the LOGOS and ACT multi-agent systems and the ANTS asteroid mission. It also discusses the interdependencies of the autonomic properties and why having one property will necessitate the need for others.

A. Autonomic Properties of LOGOS

The operational scenarios of LOGOS exhibit the four autonomic properties: self-configuring, self-optimizing, self-healing and self-protection as a community. The following discusses each of these properties of LOGOS in more detail.

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 needed 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 user interface agent. If there is no user logged on, then those agents do not have to be woken up for the spacecraft pass.

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.

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 that information would be pre-fetched 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, so anomalies or just their normal job in general can be done faster. A third example is the pager agent, which notifies analysts when an anomaly is present. This agent also kept track of information that specified which analysts were available at what times and modified who it called first based on their usual availability.

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 would also learn how to fix future anomalies based on inputs from the analyst when the FIRE agent needed 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.

LOGOS self-protecting. The self-protecting aspects of LOGOS are limited. The self-protection is primarily done by the FIRE and the user interface agent. UIFA does 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, though it could 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 forth. 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.

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.

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 occurs done to optimize the behavior of the entire ACT system.

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 all 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.

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 example of 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 insure 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

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' resources will be configured and re-configured to support concurrent operations at hundreds of asteroids over a period of time.

The overall ANTS mission architecture calls for specialized spacecraft 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. Some examples are the following. 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.

ANTS self-optimization. Optimization of the ANTS is done at the individual level as well as at the system level. These optimizations are:

- Rulers learning about asteroids
- Messengers adjusting their position
- Workers learning about asteroids

Optimization at the ruler level is primarily done through learning. Rulers over time will be collecting data on different types of asteroids and will over time 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 get 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.

Optimization for messengers is done through positioning. Messengers need to provide communications between the rulers and workers as well as back to 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 Earth while also maintaining communications between rulers and workers.

Optimization at the worker level is primarily done 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 skip over asteroids that are not of interest automatically, thus saving time and optimizing the exploration of the mission as a whole.

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 non-malicious sources: events such as collision with an asteroid, or another satellite, 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 range, an example scenario would arise when the team loses so many workers 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 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.

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 Earth. An alternative mechanism would be to provide rulers with a solar storm sensing capability through on-board, 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. *Interdependency of Autonomic Properties*

The LOGOS and ACT agent architectures and implementation were developed to demonstrate autonomous ground and space operations to lower costs and to provide technology for future missions requiring lengthy times between contacts with the ground. To accomplish these goals, autonomic properties of the systems were developed. Without these properties there would constantly have to be a human in the loop, which would increase costs and limit the scope of future missions. This not only is true of NASA missions, but also would be true of other real-time mission critical systems in the commercial world.

The self-healing, self-configuring, self-optimizing and self-protecting properties of both LOGOS and ACT qualify them as autonomic systems. It is also noted that, at least in the spacecraft domain, all of these properties are closely tied together. The self-healing actions can be the result of self-protecting actions. The self-healing actions then may cause self-configuration, which in turn may trigger self-optimization.

From the analysis of ANTS in terms of the four properties of autonomic systems, we see significant overlap in the scenarios. In particular, self-healing is often likely to require self-configuration. Clearly this will not always be the case – a system where one component is replaced by a homogeneous component will likely not need to be re-configured. In another example, where a worker loses so many of its sensors that it can no longer make science observations, the ruler may give it the goal to take the role of a communications node (messenger agent), and this would entail a degree of self-reconfiguration (and possibly self-re-optimization) by the ANTS team.

Similarly, self-protection may require the addition of components (whether or not they are identical to other components in the system) or replacement of components with others that have better protection mechanisms. This will likely require some degree of re-configuration (in the case of an autonomic system, this will be performed autonomously by the system itself), and possibly some degree of optimization to take advantage of the new components and to ensure that all resources are being used effectively. In the class of systems we are discussing, these actions would represent self-configuration and self-optimization.

From our experience and analysis, we view the complementary dimensions of [1] as interrelated.

VI. Conclusions

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 unneeded 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 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 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. In addition, the implication that the properties of an autonomic system can be viewed as disjoint properties becomes more and more naïve. Illustrations with relevance to ANTS make it clear that these properties are, in general, interrelated and overlapping.

ACKNOWLEDGMENT

This work was supported in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) and managed by the NASA Independent Verification and Validation (IV&V) Facility, and by NASA Headquarters Code R.

This paper is based in part on [14] and [15].

REFERENCES

[1] J. Joseph and C. Fellenstein, *Grid Computing*. IBM Press, 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. 2nd IEEE International Conference on Software Engineering and Formal Methods. Beijing, China, 26-30 Sept., 2004.

[3] W. Truszkowski, and C. Rouff, "An overview of the NASA LOGOS and ACT agent communities," 5th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2001), Orlando, Florida, July 22-25, 2001.

[4] W. Truszkowski and H. Hallock, "Agent technology from a NASA perspective." *CIA-99, Third Int. Workshop on Cooperative Information Agents*, Springer-Verlag, Uppsala, Sweden, 31 July - 2 August 1999.

[5] J. Ferber, *Multi-agent systems, An introduction to distributed artificial intelligence*. Addison-Wesley, 1999.

[6] M. Wooldridge, "Intelligent Agents", in *Multiagent Systems*, Gerhard Weiss, Ed. MIT Press, 1999.

[7] P. Hughes, G. Shirah, and E. Luczak, "Advancing Satellite Operations with Intelligent Graphical Monitoring Systems, in:*AIAA Computing in Aerospace Conference*, San Diego, CA, Oct. 19-21, 1993.

[8] "FIPA Specification Part 2: Agent Communication Language", Foundation for Intelligent Physical Agents (FIPA), Geneva, Switzerland, November 28, 1997.

[9] W. Truszkowski and C. Rouff. "A Process for Introducing Agent Technology into Space Missions." In *Proc. IEEE Aerospace Conference*, March 11-16, 2001.

[10] P. E. Clark, S. A. Curtis, and M. L. Rilee, "ANTS: Applying a New Paradigm to Lunar and Planetary Exploration", in *Proc. Solar System Remote Sensing Symposium*, Pittsburg, 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", in *Proc. International Astronautical Federation, 51st Congress*, October 2000.

[12] S. Curtis, W. Truszkowski, M. Rilee, and P. Clark, "ANTS for the Human Exploration and Development of Space", in *Proc. IEEE Aerospace Conference*, 2003.

[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 Conference*, Big Sky, Montana, 9-16 March 2002.

[14] W. Truszkowski, J. Rash, C. Rouff and M. Hinchey, "Asteroid Exploration with Autonomic Systems", in *Proc. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*, Brno, Czech Republic, IEEE Computer Society Press, 24-27 May 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 International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS), Workshop on Engineering of Autonomic Systems (EASe)*, Brno, Czech Republic, IEEE Computer Society Press, 24-27 May 2004, pp 490-498.

[16] T. Ames and S. Henderson, "The Workplace Distributed Processing Environment", in *Proc. 1993 Goddard Conference on Space Applications of Artificial Intelligence*, NASA Goddard Space Flight Center, Greenbelt, Maryland, USA. NASA Conference Publication 3200. May 10-13, 1993, pp. 181-188.

Walter F Truszkowski (M'00) received the M.S. degree in Computer Science from the University of Maryland, USA, and the B.A. degree in Mathematics from Loyola College, USA.

He is currently the Senior Technologist in the Advanced Architectures and Automation Branch located at NASA - Goddard Space Flight Center. He is the author of move than 30 technical papers and book chapters, and has edited 4 books. His current research interests are in the areas of formal methods, agent/multi-agent systems, swarm technologies, evolutionary robotics, autonomic computing and the semantic web.

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

Michael G. Hinchey (M'91–SM'01) received the Ph.D. in Computer Science from University of Cambridge, UK, the M.Sc. degree in Computation from University of Oxford, UK, and the B.Sc. degree in Computer Science from University of Limerick, Ireland.

He is currently Director of the NASA Software Engineering Laboratory, located at Goddard Space Flight Center. Prior to joining the US Government, he held academic positions at the level of Full Professor in the USA, UK, Ireland, Sweden and Australia. He is the author of move than 70 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 Senior Member of the IEEE, a Fellow of the IEE and the British Computer Society. He is a Chartered Engineer, Chartered Professional Engineer, Chartered Information Technology Professional and Chartered Mathematician. He is currently Chair of the IEEE Technical Committee on Complexity in Computing, and is the IEEE Computer Society's voting representative to IFIP TC1.

James L. Rash (M'87) received the M.A. in Mathematics from the University of Texas at Austin, USA, and the B.A. degree in Mathematics and Physics from the University of Texas at Austin, USA.

He currently leads formal methods research and development in the Advanced Architectures and Automation Branch at the NASA Goddard Space Flight Center, where his other major responsibilities include managing the Operating Missions as Nodes on the Internet (OMNI) Project. He has authored/co-authored move than 25 technical papers and articles, co-edited 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.

Mr. Rash is a Member of the IEEE.

Christopher Rouff received the Ph.D. in Computer Science from the University of Southern California, a M.S. in Computer Science from University of California, Davis and a B.A. in Mathematics/Computer Science from California State University, Fresno.

He is currently a senior scientist in the Advanced Concepts Business Unit at Science Applications International Corporation. He is currently doing research and development on multi-agent systems, verification of intelligent systems and collaborative robotics for NASA and DARPA. Previously he was with NASA Goddard for nine years where he researched and prototyped cooperative multi-agent systems for ground and spaceflight applications and led a number of software research and development projects.

Dr. Rouff has over forty publications and twenty years of experience in software engineering and intelligent systems.