Onboard Processing using the Adaptive Network Architecture

Dipa Suri and Adam Howell
Advanced Technology Center (ATC)
Lockheed Martin Space Systems Company
3251 Hanover Street
Palo Alto, CA 94304

Nishanth Shankaran, John Kinnebrew, Will Otte,
Doug Schmidt and Gautam Biswas
Institute for Software Integrated Systems (ISIS)
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203

Abstract— **Many future earth and space science missions will be composed of multiple spacecraft requiring autonomous capabilities for both opportunistic and coordinated science observations. The Adaptive Network Architecture (ANA) is a software framework composed of multiple, heterogeneous software agents designed for real-time operation of constellations or formations of spacecraft. The ANA is built upon mature terrestrial standards and best practices for software development, including CORBA Component middleware designed for distributed real-time embedded systems. In this paper we present the further development of the ANA's Science Agent to include a hierarchical computational architecture for reconfigurable onboard science processing. The architecture allows for runtime reconfiguration and/or redeployment of software components across a set of processors based on the available computational resources and changes in operating mode. Application of the science data processing framework to the upcoming Magnetospheric Multi-Scale (MMS) mission is also discussed.**

I. INTRODUCTION

Future space missions will rely on constellations of spacecraft with heterogeneous sensor/instrument suites to cooperatively meet their mission objectives. However, the traditional stovepipe operations model cannot sustain the increased complexity associated with these multi-spacecraft missions. This problem can be addressed by increasing the amount of onboard data processing and autonomy to reduce the ground operator's workload. Example tasks include sensor and computing resource management and the scheduling, execution, and monitoring of activities. Software agent technology provides a level of abstraction that is ideal for the distributed autonomy needed for spacecraft constellations.

The Adaptive Network Architecture (ANA) is a software framework composed of multiple, heterogeneous software agents designed to run integrated Guidance Navigation & Control (GNC), data collection, analysis, compression, and data streaming operations on constellations or formations of spacecraft. It provides a foundation for real-time autonomous responses to environmental events and ground user requests for managing

- The efficient allocation of computing and sensor resources
- Instrument reconfiguration as part of either current mission needs or fault management
- Distributed science processing and data aggregation.

The ANA is built upon mature terrestrial standards and best practices for software development, including the Component Integrated ACE ORB (CIAO) and the Deployment and Configuration Engine (DaNCE). CIAO and DaNCE are open source implementations of Object Management Group's (OMG) *Lightweight Common Object Request Broker Architecture (CORBA) Component Model (CCM)* [1] and *Deployment and Configuration (D&C)* [2] specifications. Component-based technologies are increasingly used in large-scale distributed real-time and embedded (DRE) systems, such as shipboard computing environments [3], avionics mission computing systems [4], and intelligence, surveillance and reconnaissance systems [5]. In these systems, applications can be viewed as workflow sequences of domain-related tasks. These workflow sequences are represented as *operational strings*, which are sequences of tasks (sequential and parallel with temporal constraints) that can be implemented by *software components*. Software components are defined units of implementation and composition that contain parameterized executable code with quality of service (QoS) requirements (such as maximum latency and minimum throughput values) and resource consumption profiles (such as expected CPU and memory usage).

In this paper, we describe our recent work in reconfigurable onboard science processing within the ANA framework using this concept of operational strings. Deployment and run-time management of the operational strings is achieved through the incorporation two key technologies:

Manuscript received May 19, 2006. This work was supported in part by the NASA Advanced Information Systems Technology (AIST) Program under Contract NNA04AA69C.

- A computationally efficient *Spreading Activation Partial Order Planner (SA-POP)* [6] for dynamic (re)planning under uncertainty into the ANA's Science Agent, and
- A *Resource Allocation Control Engine (RACE)* [7] for allocating computational resources and enforcing QoS requirements.

The remainder of the paper is organized as follows; Section II provides an overview of the ANA software framework and it's underlying component middleware infrastructure; Section III covers the Science Agent and the computational architecture used for onboard science processing in more detail; Section IV will describe the application of ANA in the context of managing and executing mission goals for a simplified representation of the upcoming Magnetospheric Multi-Scale (MMS) Mission; Section V compares our work with related research; and Section VI presents concluding remarks.

II. Adaptive Network Architecture overview

A. Overview

The ANA is constructed using agent technology to provide autonomous reconfigurability of on-board resources to ensure improved science data returns to users. The ANA agents are themselves a heterogeneous suite i.e. specific roles and responsibilities are distributed such that the various on-board functions of a science mission ranging from guidance, attitude control, communication, and health management to data collection, analysis, and streaming are properly addressed. The intent is that although each agent type has its own tasks to perform, more complex processes are achieved through interactions and collaborations of multiple agents [8].

The ANA, along with the underlying CCM and D&C layers, provides additional flexibility by allowing different configurations of agents to be instantiated at system initialization or runtime, depending on the desired functionality. Figure 1 shows a schematic of the ANA architecture, which is composed of a set of heterogeneous agents that rely on several CORBA services for agent discovery and inter-agent communication. All agents contain a common basic level of functionality such as messaging, health reporting, and telemetry handling. These fundamental capabilities are provided for each specific agent through inheritance from a parent 'BaseAgent' class. The set of agents resident on a spacecraft include:

- **Executive (Exec) Agent**, which is responsible for overall health management
- **Communication (Comm) Agent**, which is responsible for collecting and formatting local telemetry streams and transmitting it to the Interface Agent

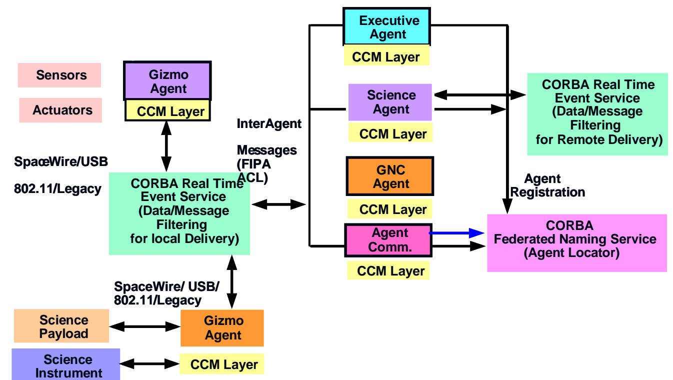

Fig. 1 The ANA is composed of a set of heterogeneous agents that rely on several CORBA services for agent discovery and interagent communication.

- **Gizmo Agent(s)**, which manage the operation and control of "negotiable" physical devices, such as the payload sensors.
- **Guidance Navigation & Control (GNC) Agent**, which is responsible for spacecraft guidance, navigation, and attitude control along with its set of specialized Gizmo agents.
- **Science Agent(s)**, which uses a planning and scheduling mechanism, discussed in section III, to generate the operational strings that define the sequence of tasks to be executed in order to meet the science goals of the mission. The agent also has a task map, which it then uses to allocate the tasks in the operational strings to the GNC, Comm, and other Gizmo agents.

The ground set is comprised of an **Interface Agent** that handles the telemetry processing and display and commanding of the space agents.

B. ANA's Middleware and Modeling Infrastructure

The ANA is developed in accordance with the OMG's Lightweight CCM [1]. This specification standardizes the development, configuration, and deployment of component-based applications that are not tied to any particular language, OS platform, or network. Components in Lightweight CCM are implemented by executors and collaborate with other components via the following types of *ports*:

- **Facets**, which define an interface that accepts point-to-point method invocations from other components.
- **Receptacles**, which indicate a dependency on point-to-point method interface provided by another component.
- **Event sources/sinks**, which indicate a willingness to exchange typed messages with one or more components.

The CCM implementation used for ANA is the *Component Integrated ACE ORB (CIAO)* and the *Deployment and Configuration Engine (DAnCE)*. CIAO and DAnCE are open-source (all open-source middleware and modeling tools described in this paper can be downloaded from

www.dre.vanderbilt.edu.) QoS-enabled component middleware built atop *The ACE ORB* (TAO). TAO is a highly configurable, open-source Real-time CORBA Object Request Broker (ORB) that implements key patterns to meet the demanding QoS requirements of DRE systems.

CIAO extends TAO by abstracting key QoS concerns (such as priority models, thread-to-connection bindings, and timing properties) into elements that can be configured declaratively via metadata (such as standards for specifying, implementing, packaging, assembling, and deploying components). Promoting these QoS concerns as metadata disentangles code for controlling these non-function concerns from code that implements the application logic, thus making space system development more flexible and productive. DAnCE extends TAO by allowing application deployers to specify how existing components should be packaged, assembled, and customized into reusable services.

In addition to QoS-enabled middleware, ANA also uses *Model-Driven Engineering (MDE)* technologies that combine

- *Domain-Specific Modeling Languages (DSMLs)* whose type systems formalize the application structure, behavior, and requirements within particular domains, such as software defined radios, avionics mission computing, satellite constellations, online financial services, warehouse management, or even the domain of middleware platforms. DSMLs are described using metamodels, which define the relationships among concepts in a domain and precisely specify the key semantics and constraints associated with these domain concepts. Developers use DSMLs to build applications using elements of the type system captured by metamodels and express design intent declaratively rather than imperatively.
- Transformation engines and generators that analyze certain aspects of models and then synthesize various types of artifacts, such as source code, simulation inputs, XML deployment descriptions, or alternative model representations. The ability to synthesize artifacts from models helps ensure the consistency between application implementations and analysis information associated with functional and QoS requirements captured by models. This automated transformation process is often referred to as "correct-by-construction," as opposed to conventional handcrafted "construct-by-correction" software development processes that are tedious and error-prone.

The MDE tool suite used in ANA is called *Component Synthesis using Model Integrated Computing* (CoSMIC), which is an integrated set of DSMLs that support the development, deployment, configuration, and evaluation of enterprise DRE systems based on Real-time CCM. CoSMIC is implemented using the *Generic Modeling Environment (GME)*, which is an open-source MDE toolkit for creating and using DSMLs.

By combining CIAO, DAnCE, and CoSMIC as the infrastructure for ANA, we tackled many integration challenges associated with configuring and deploying space systems by leveraging MDE tools to enforce correct-by-construction design. For example, we used CoSMIC's model interpreters to generate Real-time CCM XML configuration files automatically and CIAO's DAnCE to deploy the resulting component assemblies on space system nodes, as shown in Fig. 2.

C. *Base Agent Implementation*

The adoption of CIAO, DaNCE, and RACE provides dynamic re-configurability, and the new Base Agent definition has to be cast as a CORBA component. While CORBA 2 (used in the previous version of the ANA) shields applications from dependencies that arise from the use of heterogeneous platforms, e.g. language, operating system, and network protocols, it does not handle the requirement that multiple interacting objects may be deployed on diverse platforms for DRE systems. The advantage this new scheme offers in the ANA context is the ability to assemble the agents and algorithms, into logical sets that can be dynamically (re)deployed by DaNCE across the network of spacecraft and ground nodes based upon the resource monitoring results from RACE. A more detailed discussion is presented in later sections of the paper.

Integration into the CCM framework required the agent description to be based on the Component Interface Definition Language (CIDL) shown in Table 1. The '*provides'* clause in the Agent_Base component illustrates the use of facets, the '*uses'* clause in the ExecAgent component illustrates the use of receptacles, and the '*publishes'* and '*consumes'* clauses in the Agent_Base component illustrates the use of event sources/sinks.

The Base Agent structure remains largely the same as presented in [6], but agent communication is conducted via Messages now defined as a CCM event type. The routing, transmission, and reception of these messages are handled by the CIAO middleware, thus shielding the developer from having to manage the requisite internal "plumbing". Further extension and specialization is provided by a distribution of func-

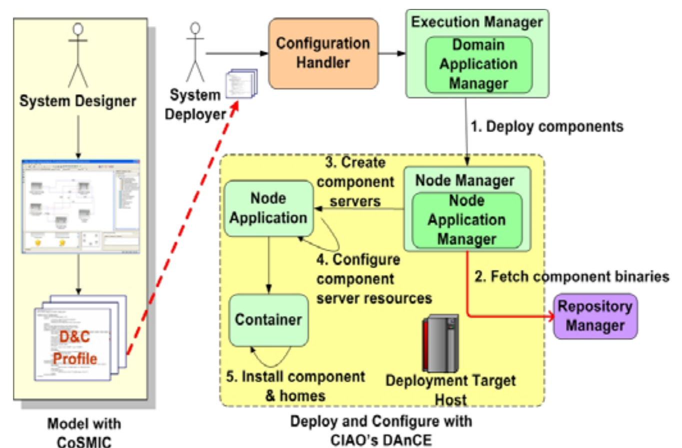

Fig. 2 Integrating CIAO, DAnCE, and CoSMIC.

tionality between the 'Agent' interface and the component 'Agent_Base'. All derived Agent Types in the ANA now inherit from the component Agent_Base. An example inheritance that illustrates this well is shown in Table 2. The Executive Agent encapsulated in the ExecAgent component provides run time connectivity to other agents local to a given host via the '*uses multiple*' clause for the receipt of Heartbeat messages. This connectivity is handled by the CIAO middleware at system initialization including the routing of messages via the TAO Real Time Event Service [9], much of which was previously handled directly by the Comm. Agent.

III. Onboard science processing via the Science agent

The Science Agent is responsible for performing the onboard data processing required to achieve pre-defined science mission goals for the spacecraft. These goals are typically chosen by the mission planners and scientists on the ground, or potentially other spacecraft when performing missions requiring distributed observations and measurements. The goals are communicated to the Science Agent using the Foundation for Intelligent Physical Agents (FIPA) standardized Messages [10] and Interaction Protocols [11] (e.g. Requests, Informs, or Publish/Subscribe) containing an Agent Communication Language predefined by the Science Agent developers.

The Science Agent employs the computational architecture shown in Fig. 3 to achieve its goals. The architecture is composed of two primary subsystems: (1) the *SA-POP* planner and scheduler that generate the operational strings directed to solving the specified goals, and (2) the *RACE* framework that monitors and manages runtime resource allocation to enforce QoS requirements.

A. SA-POP

To generate an operational string that achieves a given set of *goals*, e.g. study the physics of plasma reconnection and charged particle acceleration for the MMS mission, the SA-POP planner, shown in Fig. 4, first generates partial order *task sequences* that achieve specified goals using a spreading activation mechanism [12]. Individual tasks in the generated sequences are then mapped to available executable software components, e.g. the planner may pick a data compression task and then select an appropriate component implementation for a chosen compression algorithm. The planner uses a *task network,* which is a directed graph that represents both tasks and conditions (preconditions, data input, effects, and data output), to establish the preconditions required for a task component to execute successfully, the input data stream and the output that will be generated from this data stream, and other post condition effects resulting from their operation.

TABLE 1: ANA BASE AGENT CIDL

```
// Assumes all messages are passed through the event channel
// Modified FIPA ACL message structure
eventtype Message
{
  public PerformativeList performatives;

  // AgentName of sender
  public string sender;

  // AgentName of receiver
  public string receiver;

  // AgentName of agent to reply to
  public string reply_to;

  // Time stamp of message
public long time_stamp;

  // Content of the message.
  public any content;

  // Internal id to relate message -> conversation
  public string conversation_id;
};

// Heartbeat message is just the state and sender name
struct HeartBeat
{
      string sender;
     StateType CurrentState;
   };

// Standardized agent interface
interface Agent
  {
    readonly attribute string AgentName;
    readonly attribute AgentClasses AgentType;

    attribute float HeartBeatRate;

    //Request agent becomes dormant
    boolean Doze();

    //Request agent becomes active
    void Wakeup();
  };

component Agent_Base
  {
    // Name of the agent.
    attribute string AgentName;

    // Type of the agent.
    readonly attribute AgentClasses AgentType;

    // Provides a facet to control the state of the agent.
    provides Agent agent_interface;

    // Publishes messages.
    publishes Message outgoing_message;

    // Subscribes messages.
    consumes Message incoming_message;
  };

 home BaseAgentHome manages Agent_Base{ };
};
```

TABLE 2: ANA EXECUTIVE AGENT CIDL

```
module ANA
{
 module ExecModule
 {
  component ExecAgent : Agent_Base
  {
   uses multiple Agent local_agent;
  };

  home ExecAgentHome manages ExecAgent
  {
  };

 };

};
```

The output generated by a component is a function of the input and environmental conditions during the actual operation. Other computational properties of the component, e.g. the throughput and the quality of the output, depend on the available computational resources. As a result, there is uncertainty as to whether the component will produce the desired output. This uncertainty is captured by conditional probabilities associated with the component definitions. Together, the task-component relations and the conditional probability of success of components defines the *functional signature* of the task. Different parameterizations of a given component may produce different functional signatures. Conversely, different components that have the same functional signature may vary in time to completion, resource usage, and QoS parameters.

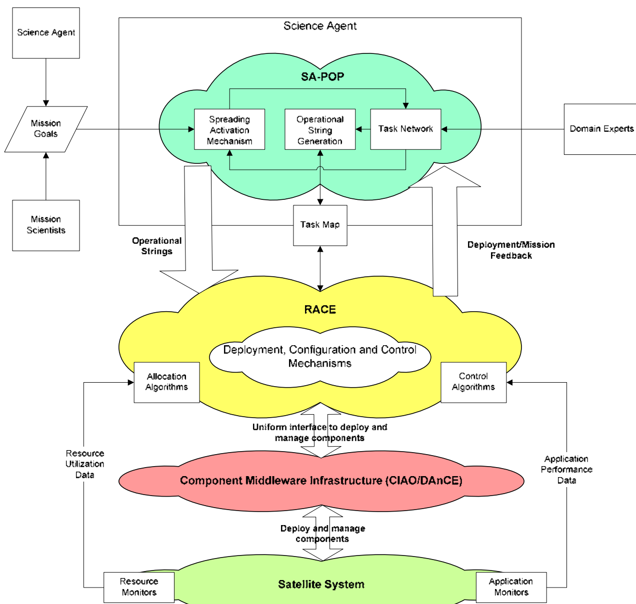

Fig. 3 The computational architecture for onboard science processing involves the SA-POP embedded in the Science Agent, RACE, and the CIAO/DaNCE middleware infrastructure

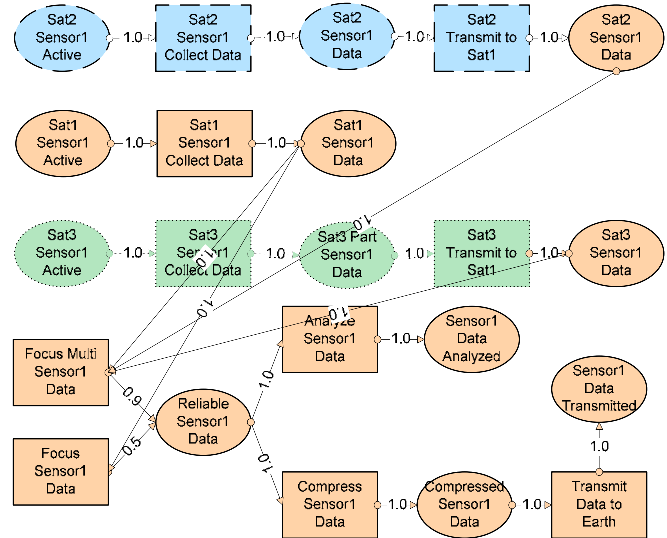

Fig. 4 SA-POP Planner.

We define a *task* as one or more parameterized components with a single functional signature. The functional signature of each task is also captured in the *task network*. With the task network whose links encode the requisite probability of success information, and a given set of utility values for goal conditions and/or data, the planner computes expected utility values for each task using the spreading activation mechanism.[12]

To ensure applications do not violate resource constraints, the planner also requires knowledge of each task's resource consumption and execution time, i.e. its *resource signature*. A given task may be associated with multiple parameterized components, each with different resource signatures. SA-POP and RACE therefore use a shared *task map* that maps each task to a set of parameterized components and their associated resource signatures. The combination of functional and resource signatures in a task sequence defines an operational string, which specifies the tasks, a suggested implementation for each task, the control (ordering) dependencies, the data (producer and/or consumer) dependencies, and required start and end times for tasks, if any. These operational strings are given as input to RACE for deployment and runtime monitoring.

B. RACE

The Resource Allocation and Control Engine (RACE) is a reusable framework that separates resource allocation and control algorithms from the underlying middleware deployment, configuration, and control mechanisms so that different algorithms can reuse these common middleware mechanisms to (re)deploy components onto nodes and manage the node's resources among competing applications. RACE provides a range of resource allocation and control algorithms that use the middleware deployment and configuration mechanisms of

the OMG D&C specification to allocate resources to operational strings and control system performance after operational strings have been deployed.

RACE's algorithms determine how to deploy and redeploy operational strings of application components at system initialization and during runtime. Its allocation algorithms determine the initial component deployment using a bin packing algorithm that maps these components to the appropriate target nodes based on available system resources. For example, an allocation algorithm could apportion CPU resources to components in such a way that avoids saturating these resources.

Likewise, RACE's control algorithms adapt the execution of an operational string's components at runtime in response to changing environmental conditions and variations in resource availability and/or demand. For example, a control algorithm could (1) modify an application's current operating mode, (2) dynamically update component implementations, and/or (3) redeploy all or part of an operational string's components to other target nodes to meet end-to-end QoS requirements.

The RACE architecture consists of the entities shown in Figure 5. These entities are implemented as CCM components using CIAO and are deployed via DAnCE. The key entities in RACE are described below:

- **Application QoS Monitors** are CCM components that track the performance of application components by observing QoS properties, such as throughput and latency. One or more Application QoS Monitors are associated with each type of application component.
- The **Target Manager** is a CCM component defined in the D&C specification [2] that receives periodic resource utilization updates from resource monitors within a domain. It uses these updates to track resource usage of all resources within the domain. The Target Manager provides a standard interface for retrieving information pertaining to resource consumption of each component and an assembly in the domain, as well as the domain's overall resource utilization. The Target Manager provides information on resource utilization component ports in operational strings.
- The **Deployment Manager** is an assembly of CCM components that encapsulates and coordinates one or more allocation and control algorithms. This manager deploys assemblies by allocating resources to individual components in an assembly. After assemblies are deployed, the Deployment Manager manages the performance of (1) operational strings and (2) domain resource utilization. This manager ensures desired performance of the operational strings by performing the following actions to the components that make up the operational strings: (1) (re)allocating resources to the component, (2) modifying component parameters such as execution mode, and/or (3) dynamic replacing the component implementations.

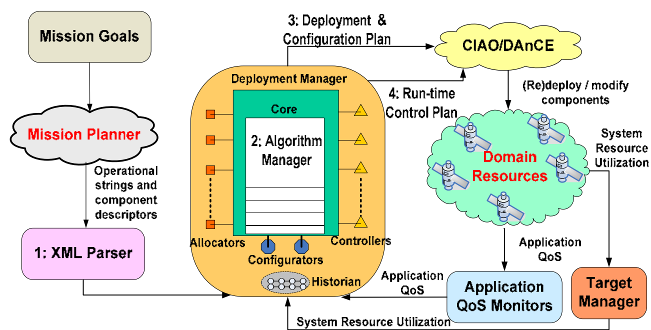

Fig. 5 The RACE Architecture

IV. Application to the MMS Mission

The upcoming NASA MMS mission was chosen as an exemplar application to assess the effectiveness and performance of both the onboard science processing framework and the ANA as a whole. Although the mission does not currently require distributed processing or high levels of autonomy, the mission does have many characteristics (e.g. multiple spacecraft and heterogeneous sensors, multiple operating modes, etc.) that the ANA was designed to address. First, an overview of the MMS mission will be presented, followed by a description of how the onboard processing can be conducted using the ANA.

A. *Mission Overview*

The goal of the MMS mission is to study the microphysics of three fundamental plasma processes in the Earth's magnetosphere; magnetic reconnection, particle acceleration and turbulence [13]. MMS consists of a constellation of four identical spacecraft that maintain a tetrahedral formation in specific regions of scientific interest (ROI) within the constellation's orbit, as shown in Fig. 6. Furthermore, each spacecraft has a suite of four primary payload sensor packages, i.e. FPI, FIELDS, HPCA, and EPD, which have varying data rate, data size, and compression requirements [14].

Since the plasma processes are inherently transient (especially magnetic reconnection), MMS requires *reactive* onboard autonomy to enable the spacecraft to transition between three modes of operation; slow survey, fast survey, and burst. Slow survey mode is entered outside the ROI's and enables only a minimal set of data acquisition (primarily for health monitoring). The fast survey mode is entered when the spacecraft are within a ROI, which enables data acquisition for all payload sensors at a moderate rate. While in fast survey mode, the data from a subset of the payload sensors is analyzed onboard to compute a *quality* value indicating the likelihood of a transient plasma event as determined by

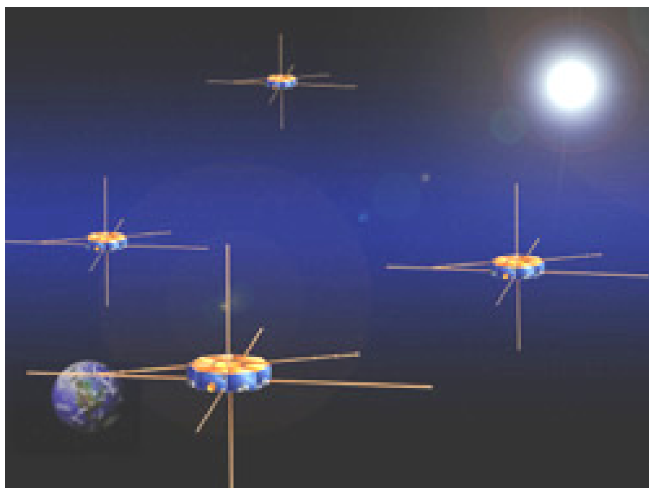

Fig. 6 An artist's rendering of the MMS spacecraft in the tetrahedral formation. Courtesy of the SWRI [15].

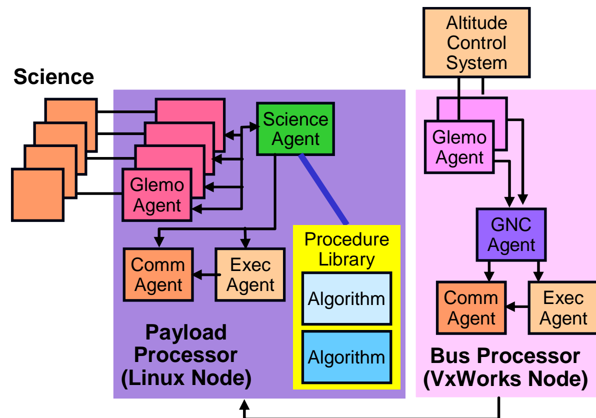

Fig. 7 ANA Agents packaged as logical CCM assemblies divided between the Payload and Bus processors. Simulators for the Payload sensors are executed on a separate processor.

changes in particle, ion, and field measurements. This quality value is communicated to the other spacecraft in the constellation via crosslinks. A set of rules on each spacecraft determines when burst mode should be entered based upon a weighted combination of the local and remote quality values. Once entered, burst mode enables all payload instruments to acquire data at high rates (up to ~1.6 Mb/s), however this mode can be entered for at most 17.5 minutes per day because of onboard storage limitations [14].

B. *Science Processing using the ANA*

With the above concept of operations, we have a mission configuration that is well-suited to demonstrate the utility of the ANA. Each MMS spacecraft has multiple payload sensors interfaced to a payload processor, while the spacecraft bus functions are handled by a bus processor. The agent components are logically packaged into two separate CCM assemblies as shown in Fig. 7. The assemblies are initially deployed by DaNCE on the two target processors based on the division of functionality between the payload and bus, however the deployment can change at runtime based on user needs or resource constraints. The payload processor assembly contains the Science Agent supported by Gizmo Agents to provide a direct interface to the payload sensors. In addition, the task library contains multiple data compression algorithms that RACE can employ as directed by the SA-POP. Executive and Communication Agents are resident on both nodes. The bus processor assembly contains the GNC Agent that provides orbital information to the Science Agent on the payload processor to determine the entry/exit from ROI's. The Science Agent(s) on all spacecraft have mission goals that represent user or other Science Agent requests for the times and types of data to be acquired.

Using this MMS Mission configuration, several scenarios have been developed to exercise the ANA. While the following scenarios have not been fully tested to date, individual subsets of the technologies have been demonstrated. Furthermore, we are progressing towards an end-to-end demonstration in our Distributed Systems Laboratory on multiple robots - developed in house - as a representative 2D simulation of a spacecraft constellation. Each robot hosts two different processor and operating system families representing the payload and bus processor. We also provide a simulated ground control station with a GUI and an interface agent for command/telemetry processing.

A nominal day-in-the-life scenario is to be exercised starting from system initialization through autonomous exit/entry into Fast Survey Mode, followed by the detection of an event that causes a transition to Burst Mode. RACE will continually monitor resource use and provide feedback to SA-POP. The SA-POP can then change the active operational strings, e.g. swap in a different compression algorithm, if user-specified resource constraints are violated. The SA-POP can similarly alter the data acquisition and processing parameters to ensure acceptable data quality is maintained. This interaction is shown in Fig. 8.

Potential off-nominal scenarios range from (among many) fault conditions such as (1) a lack of local storage capacity leading to compressed data being transmitted for storage to another spacecraft in the constellation, to (2) a catastrophic payload processor failure leading to a redeployment of the payload agent assembly on the spacecraft processor with minimal degradation in science data returns to the users.

V. Related Work

As component middleware becomes more pervasive, there has been an increase in focus on technologies, platforms, and tools for deploying components effectively within distributed systems. We compare our work on ANA, SA-POP, and RACE with related efforts.

The Autonomic Deployment and Management Engine (ADME) [16] provides a framework for deploying and

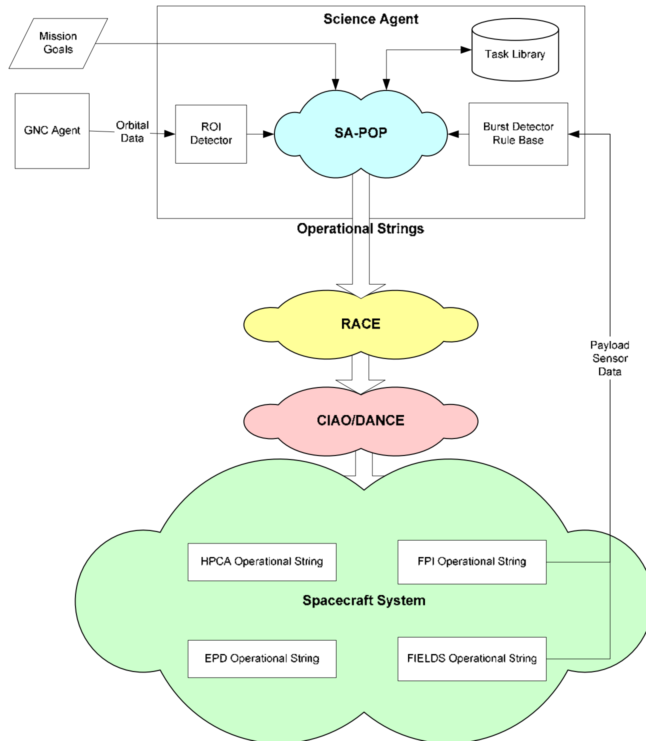

Fig. 8 Onboard science processing for the MMS mission using the computational architecture described in Section III.

autonomically managing application components in distributed systems. Allocating resources to application components in ADME is framed as a constraint solving problem, where domain resources are allocated to application components, subject to specified constraints. ADME uses a domain-specific constraint language called ``DEclarative Language for Describing Autonomic Systems'' (DELDAS) to specify desired system performance as goals at design time. At run-time, the ADME infrastructure deploys and manages application components to satisfy these goals. RACE has similar motivations as ADME, though RACE provides a pluggable framework where multiple resource allocation and control algorithms can be (re)configured at runtime. RACE also focuses more on the (re)deployment and (re)configuration of QoS-enabled applications executing in DRE systems.

Plaint [7] is a tool that uses a temporal planner to manage and reconfigure a software system. A plan is defined as a sequence of execution steps that ensures desired system perf

ormance. Plaint generates to types of plans: (1) deployment plans that allocate resources to application components, and (2) reconfiguration plans that dynamically reconfigure systems in response to changes in their operation that may be attributed to factors such as external attacks that result in loss of critical application components. The output from various planning techniques can be viewed as deployment plans and control plans that RACE can execute to ensure desired system performance. RACE also augments this planning approach to system reconfiguration by providing the capability to link and unlink various planning mechanisms at runtime to handle system reconfiguration more transparently.

VI. Conclusion

It is recognized that autonomy is an important feature of future science missions that will involve networked space, airborne, terrestrial, and oceanic resources. Any real-life system that provides autonomy involves multiple entities that require collaborative interactions and intelligent behavior in order to meet their own specific as well as overall mission goals. Although designing systems of such complexity is hard, agent technology and multi-agent systems show promise in helping to alleviate development issues. The ANA provides many key elements needed for autonomous operations of NASA missions.

This paper describes the design and application of the *Spreading Activation Partial Order Planner (SA-POP)* and the *Resource Allocation and Control Engine (RACE)*. RACE manages system resource utilization and ensures QoS requirements of operational strings are met even under varying operational contexts and/or varying resource requirement/availability.

References

[1] Object Management Group. *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 edition, May 2003.

[2] Object Management Group. *Deployment and Configuration Adopted Submission*, OMG Document ptc/03-07-08 edition, July 2003.

[3] D. C. Schmidt, R. Schantz, M. Masters, J. Cross, D. Sharp, and L. DiPalma. Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems. *CrossTalk – The Journal of Defense Software Engineering*, Nov. 2001.

[4] D. C. Sharp and W. C. Roll. Model-Based Integration of Reusable Component-Base Avionics System. In *Proc. of the Workshop on Model-Driven Embedded Systems in RTAS 2003*. May 2003.

[5] P. Sharma, J. Loyall, G. Heineman, R. Schantz, R. Shapiro and G. Duzan. Component-Based Dynamic QoS Adaptations in Distributed Real-time and Embedded Systems. *In Proc. of the Intl. Symp. On Dist. Objects and Applications (DOA '04)*, Agia Napa, Cyprus, Oct. 2004.

[6] N. Shankaran, J. Balasubramanian D. Schmidt, G. Biswas, P. Lardieri, E. Mulholland, and T.Damiano, A Framework for (Re)Deploying Components in Distributed Realtime and Embedded Systems, poster paper at the Dependable and Adaptive Distributed Systems, Track of the 21st ACM Symposium on Applied Computing, April 23 -27, 2006, Bourgogne University, Dijon, France.

[7] N. Arshad, D. Heimbigner, and A. L. Wolf. Deployment and Dynamic Reconfiguration Planning For Distributed Software Systems. In Proc. of the 15th IEEE International Conference on Tools With Artificial Intelligence (ICTAI 2003), Sacramento, CA, USA, Nov. 2003.

[8] D. Suri, A. Howell. The Adaptive Network Architecture for formations of heterogeneous spacecraft. In *Proc. of the Earth-Sun System Technology Conference (ESTC2005)*, 2005.

[9] T. H. Harrison, D. L. Levine, D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service, Proceedings of ACM OOPSLA '97 conference, Atlanta, GA, October 1997.

[10] Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, 2002. Available: http://www.fipa.org/specs/fipa00061/index.html.

[11] Foundation for Intelligent Physical Agents. *FIPA Interaction Protocol Specifications*. Available: http://www.fipa.org/repository/ips.php3.

[12] S. Bagchi, G. Biswas and K. Kawamura. Task Planning under Uncertainty using a Spreading Activation Network. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(6):639-650, Nov. 2000.

[13] *The Magnetospheric Multiscale Mission - Resolving Fundamental Processes in Space Plasmas*. Report of the NASA Science and Technology Definition Team for the Magnetospheric Multiscale (MMS) Mission, December 1999. Available: http://stp.gsfc.nasa.gov/missions/mms/mms_documents.htm.

[14] Southwest Research Institute. *SMART Proposal and Concept Report*, 2003. Available: http://mms.space.swri.edu/proposal+CSR.html

[15] Southwest Research Institute. *MMS-SMART Homepage*. Available: http://mms.space.swri.edu

[16] A. Dearle, G. N. C. Kirby, and A. J. McCarthy. A Framework for Constraint-Based Deployment and Autonomic Management of Distributed Applications. In ICAC, pages 300-301. IEEE Computer Society, 2004.

[17] J. Kinnebrew, N. Shankaran, G. Biswas, and D. Schmidt, A Decision-Theoretic Planner with Dynamic Component Reconguration for Distributed Real-Time Applications,. In *Proceedings of Twenty-First National Conference on Artificial Intelligence*, July 16 20, 2006, Boston, Massachusetts.