Asynchronous Message Service for Deep Space Mission Operations

Scott C. Burleigh[*]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109

While the CCSDS (Consultative Committee for Space Data Systems) File Delivery Protocol (CFDP) provides internationally standardized file transfer functionality that can offer significant benefits for deep space mission operations, not all spacecraft communication requirements are necessarily best met by file transfer. In particular, continuous event-driven asynchronous message exchange may also be useful for communications with, among, and aboard spacecraft. CCSDS has therefore undertaken the development of a new Asynchronous Message Service (AMS) standard, designed to provide common functionality over a wide variety of underlying transport services, ranging from shared memory message queues to CCSDS telemetry systems. The present paper discusses the design concepts of AMS, their applicability to deep space mission operations problems, and the results of preliminary performance testing obtained from exercise of a prototype implementation.

I. Introduction

WHILE the CCSDS (Consultative Committee for Space Data Systems) File Delivery Protocol (CFDP[1]) provides internationally standardized file transfer functionality that can offer significant benefits for deep space mission operations, not all spacecraft communication requirements are necessarily best met by file transfer.

As the computers used to conduct space flight mission operations both in flight and on the ground increase in capability, the software operating on those computers tends to increase in functional scope and thus take on greater operational significance. With that increase in scope comes increasing size and complexity, which may be partially mitigated by decomposition into modules whose functionality can be readily defined and tested. However, this modularity in turn entails a growing reliance on effective communication among modules – the exchange of messages. Put another way, mere decomposition cannot diminish the functional complexity implied by complex requirements. It can only partition that complexity into manageable portions, while the resulting web of communication relationships among modules introduces new complexity of a different order: rather than a relatively simple system of a few increasingly large and complex modules, a modern mission is increasingly likely to require a large and complex system of relatively simple modules.

Increasing complexity tends to increase the likelihood of failure. The increasing complexity of mission systems based on communication among modules therefore tends to increase the chance of such systems failing even as the success of those systems become increasingly critical to the achievement on mission objectives. Measures that can minimize the chance of failure in complex systems – exhaustive regression testing and configuration management, flight rules constraining the exercise of unproven system capabilities and the introduction of improvements – increase cost if they are taken and increase risk if they are not.

These considerations argue for the desirability of a standard system of message exchange among mission software modules. Such a standard could reduce mission cost and risk by confining much of the complexity of modern mission systems to relatively static and proven reusable infrastructure.

Systems based on message exchange are typically less labor-intensive to configure, upgrade, and operate if message exchange can conform to the asynchronous peer-to-peer "publish/subscribe" (or "push") model rather than the synchronous "client/server" ("pull") model[2]. However, although standard APIs for publication of asynchronous messages are well established (e.g., Ref. 3-5), there is currently no broadly used international standard wire protocol for end-to-end asynchronous message exchange. Most existing asynchronous message exchange systems are

[*] Senior Software Engineer, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490, Pasadena, CA 91109.

proprietary, licensed products (e.g., SmartSockets, NDDS, TIBCO Rendezvous, IBM MQSeries). Moreover, no such system is designed for mission-critical operation on deep space robots.

For this reason, CCSDS has undertaken the development of a new Asynchronous Message Service standard. The present paper discusses the design concepts of AMS and their applicability to deep space mission operations problems. It also presents the results of preliminary performance testing obtained from exercise of a prototype implementation of the protocols.

II. Overview

The CCSDS Asynchronous Message Service (AMS) is a data system communications architecture under which the modules of mission systems may be designed as if they were to operate in isolation, each one producing and consuming mission information without explicit awareness of which other modules are currently operating. Communication relationships among such modules are self-configuring; this tends to minimize complexity in the development and operations of modular data systems.

A system built on this model is a "society" of generally autonomous inter-operating modules that may fluctuate freely over time in response to changing mission objectives, modules' functional upgrades, and recovery from individual module failure. The purpose of AMS is to reduce mission cost and risk by providing standard, reusable infrastructure for the exchange of information among these data system modules in a manner that is simple to use, highly automated, flexible, robust, scalable, and efficient.

AMS is "middleware" that is designed to provide common functionality over a wide variety of underlying transport services, ranging from shared memory message queues to CCSDS telemetry systems. This flexibility is intended to simplify software development and mission operations by enabling straightforward migration of functions between flight and ground systems. It also enables AMS to scale readily from high-speed on-board avionics applications to ground-based monitor and control of spacecraft constellations. Synchronous (client/server) operation is supported in addition to publish/subscribe. Mechanisms for access control, authentication, confidentiality, anomaly detection, failover, and recovery are also encompassed in the design.

A. Terminology

All AMS communication is conducted among three types of communicating entities: nodes, registrars, and configuration servers (all defined below). A *continuum* is defined as a closed set of entities that utilize the "meta-AMS" protocol, discussed below, for purposes of mutual discovery.

An AMS *application* is a data system implementation that relies on AMS protocols to accomplish its purposes.

A *role* is some part of the functionality of an application.

An *authority* is a unit of mission organization that may have responsibility for the configuration and operation of an application.

An *instance* of an application is a functioning projection of the application – for which some authority is responsible – onto a set of one or more running computers.

An AMS *message* is an octet array of known size which, when copied from the memory of one module of an application instance to that of another (exchanged), conveys information that can further the purposes of that application instance.

A *node* (i.e., a mission data system module) is a communicating entity that implements some part of the functionality of some AMS application instance – that is, performs some specific application role – by, among other activities, exchanging messages with other nodes.

A *message space* is the set of all of the nodes of one AMS application instance that are members of the same AMS continuum.

A *zone* is an administrative subset of a message space – that is, a set of nodes – declared during message space configuration as specified by the controlling authority of the AMS application instance of which the message space is a subset. Zones may contain other zones; the zone that is coterminous with the message space itself (i.e., contains all of its nodes) is termed the *root zone*. By registering as a member of some zone, a node acquires the ability to exchange messages with all other nodes in the application instance.

A *registrar* is a communicating entity that catalogues information regarding the registered membership of a single zone of a message space.

A *configuration server* is a communicating entity that catalogues information regarding the message spaces established within some AMS continuum, specifically the locations of the registrars of all zones of all message spaces.

The *subject* of a message is an integer embedded in the message that indicates the general nature of the information the message conveys, in the context of the AMS application instance within which the message is exchanged.

To *send* a message is to cause it to be copied to the memory of a specified node. To *publish* a message on a specified subject is to cause it to be sent to one or more implicitly specified nodes, namely, all those that have requested copies of all messages on the specified subject. To *announce* a message is to cause it to be sent to one or more implicitly specified nodes, namely, all those nodes that are members of a specified zone (possibly the root zone) and that perform a specified role in the application (possibly “any role”).

The *domain* of an AMS service request is the set of nodes to which the request pertains. Domains may be limited to the nodes in specified continua, specified zones, and/or specified roles.

A *subscription* is a statement requesting that one copy of every message published on some specified subject by any node in the subscription’s domain be sent to the subscribing node.

An *invitation* is a statement of the manner in which messages on some specified subject may be sent to the inviting node by nodes in the domain of the invitation.

B. Architectural Elements

AMS comprises three protocols: a “meta-AMS” (MAMS) protocol for automated continuum configuration and discovery of communicating entities; the base AMS protocol for asynchronous message exchange among nodes; and a “remote AMS” protocol for inter-connecting multiple AMS continua that may be separated by troublesome barriers to message publication such as firewalls, links operating at low data rates, and/or interplanetary distances.

The architectural elements involved in operating the asynchronous message service protocol within a single continuum are depicted in Figure 1.

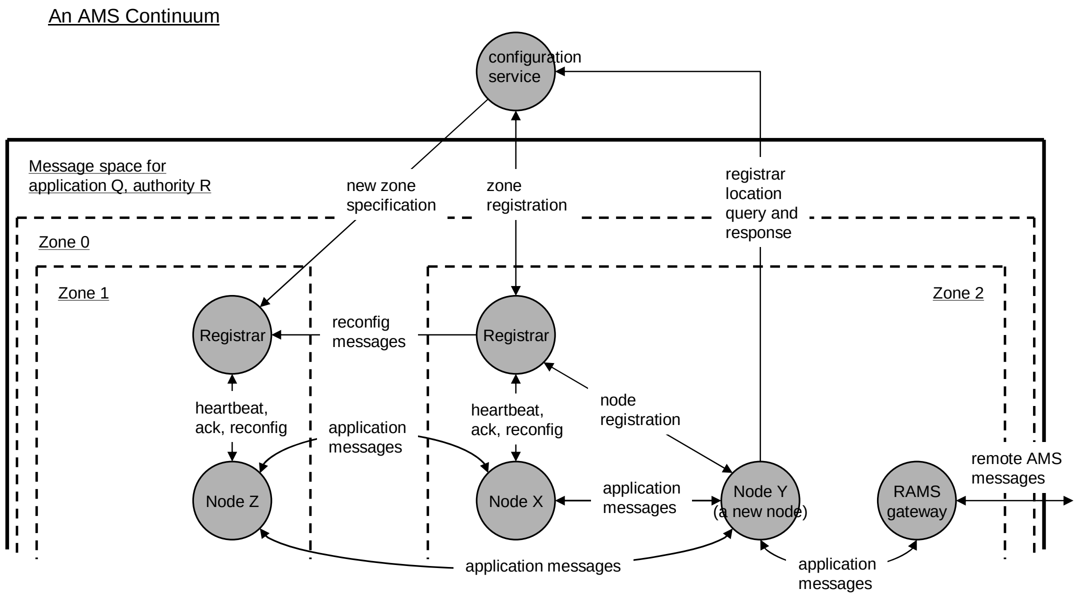

Figure 1. Architectural Elements of AMS

Because issuance of an asynchronous message need not suspend the operation of the issuing node while a response is pending, AMS’s message exchange model enables a high degree of concurrency in the operations of data system modules. It also largely insulates applications from variations in signal propagation time between points in the AMS continuum.

However, some critical MAMS communication is unavoidably synchronous in nature: in particular, a newly registering node must wait for responses from the configuration server and the registrar before proceeding with application activity. For this reason, the core AMS protocol is most suitable for use in operational contexts characterized by generally uninterrupted network connectivity and relatively small and predictable signal propagation times, such as a local area network. It is usually advantageous for all entities of any single AMS continuum to be running within one such “low-latency” network.

AMS application messages may readily be exchanged between nodes in different AMS continua, however, by means of the Remote AMS (RAMS) procedures.

RAMS procedures are executed by special-purpose application nodes called *RAMS gateways*. Each RAMS gateway has interfaces to two communication environments: the AMS message space it serves and the *RAMS grid* – the application instance's network of mutually aware RAMS gateways – that enables AMS messages produced in one continuum to be forwarded to others.

The use of RAMS results in extension of the basic AMS architecture pictured in Figure 1 to a more general structure as shown in Figure 2 below.

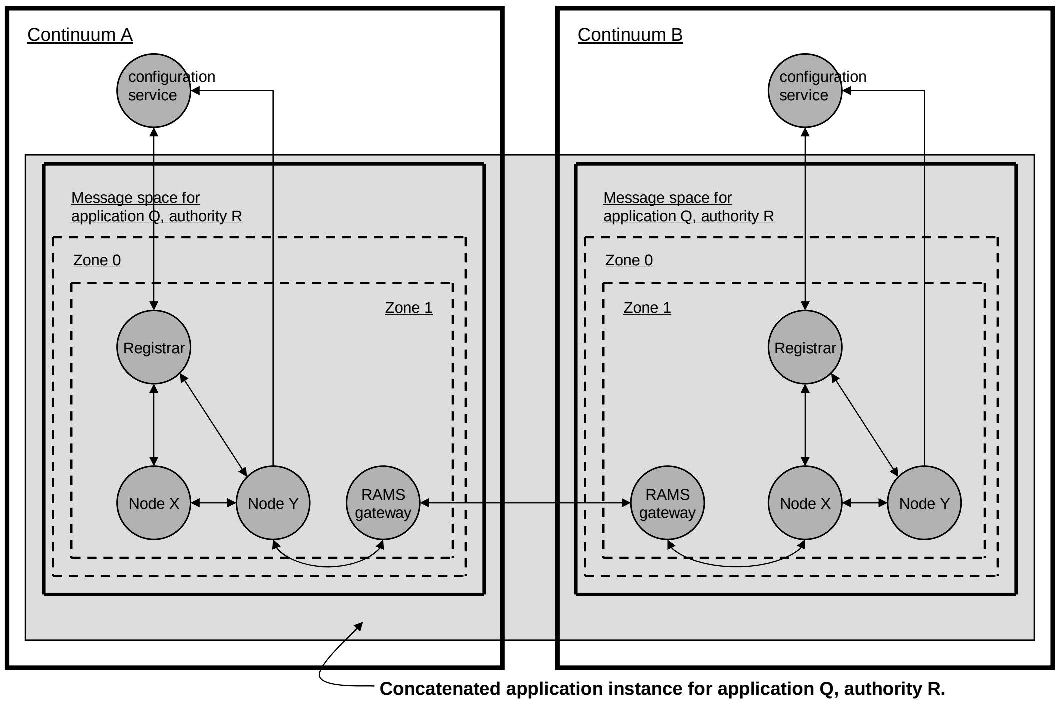

Figure 2. General AMS Application Structure

Again, this extension of the publish/subscribe model to inter-continuum communications is invisible to application nodes. Application functionality is unaffected by these details of network configuration, and the only effects on behavior are those that are intrinsic to variability in message propagation latency.

C. Protocol Functionality

The base AMS message exchange protocol itself is simple:

- Nodes request that messages be transmitted. (All AMS messages are exchanged directly between nodes rather than through any central message dispatching nexus.)
- Message transmission is constrained by the receiving nodes' subscriptions and invitations for messages on specific subjects. These constraints may confine message transmission to specified domains, i.e., sets of senders that (a) are located in specific continua and/or zones, and/or (b) are characterized by specific application roles. The sending node may also constrain message transmission, limiting the set of receivers to a single node or to a set of nodes that (a) are located in specific continua and/or zones, and/or (b) are characterized by specific application roles.
- Additionally, messages may be transmitted in reply to received messages.

The MAMS protocol for automatic continuum configuration is considerably more complex, as it is required to perform a broader range of functions:

- Direct message exchange among nodes is made possible by real-time universal propagation of information regarding changes in the population of zones (node registration activity) and in the message reception preferences of registered nodes (subscriptions and invitations). This is a key MAMS function.
- Each continuum responds to anomalies, such as the unplanned termination of nodes, registrars, and configuration servers, by recovering as fully as possible and continuing nominal operations among whichever communicating entities remain. For this purpose it is necessary for all entities to be able to detect such anomalies; to this end, heartbeat MAMS messages are exchanged between nodes and their registrars – and between registrars and the configuration server – in reciprocal fashion.
- To prevent the configuration server from becoming a single point of failure in any continuum, one or more backup configuration servers may be initiated. Additional MAMS message traffic allocates responsibility among these servers.
- To defend against the possible loss of important MAMS traffic in unreliable networks, an additional (optional) MAMS function is supported: nodes may periodically re-advertise their existence and message reception preferences to the message space.

The RAMS protocol is implemented only at RAMS gateways, which are application nodes that serve a special purpose: each one acts on behalf of the other nodes in its message space to forward messages to and from the nodes in other message spaces of the same application instance (i.e., in other AMS continua). To accomplish this purpose, each RAMS gateway does the following:

- Forwards locally issued subscriptions to the RAMS gateways in remote message spaces.
- On receipt of a subscription from a remote message space, subscribes locally to messages on the indicated subject.
- On receipt of a published message, forwards exactly one copy of that message to every other RAMS gateway from which it has received at least one subscription to this message's subject.
- On receipt of a forwarded message, re-publishes the message locally.

D. Quality of Service

A core concept of the AMS design is reliance on the capabilities of underlying transport services, e.g., TCP/IP. It would not be unreasonable to say that AMS does little more than simplify applications' use of these services, which directly or indirectly utilize communications hardware in order to copy message data from the memory of one entity to that of another. The quality of the communication service (QOS) provided by AMS is therefore wholly dependent on the capabilities of underlying transport services: AMS itself does not perform data retransmission, bandwidth management, etc.

The role AMS plays in providing appropriate communication service quality to applications is twofold:

1) AMS provides facilities that enable an application node to specify the *mode* of transport service to use when sending messages on a given subject to that node. Transport service mode is determined by two node-specified variables: (a) whether or not assured delivery is required and (b) whether or not the order of transmission must be preserved when delivering received messages. This specification indirectly causes selection of a transport service that supports the specified mode.
2) AMS enables two node-specified transmission control parameters – *priority* and *flow label* – to be passed through to underlying transport services adapters, to modify the manner in which those services are directed to convey messages. Priority may be interpreted by the transport service as means of ordering data transmission, retransmission, etc. Flow label might be translated by the transport service adapter into some combination of bandwidth reservation parameters, transmission latency limits, etc.

In all cases, the result of specifying service quality parameters in AMS function invocation will depend on the manner in which the standard is implemented and locally configured for operation.

When an AMS message is published, a separate copy of the message is individually sent to each subscriber. This enables AMS to apply to each distinct transmission of the message the service quality parameters that were specified by the recipient at the time it subscribed to messages on the indicated subject.

Multi-transmission of a published message minimizes the necessity of compromising on QOS – each subscriber receives the level of service it asks for – but it is not without cost: transmission overhead increases with the size of the subscribing population. When there is a large group of subscribers to messages on some subject, it may be advisable to adopt the same strategy on which many commercial messaging systems are based: transmit the message just once, to an IP multicast address.

For this purpose, one application node acts as the agent for the group: in its subscription to the message subject it specifies a transport service mode that maps to the IP multicast transmission service, and it specifies a flow label that maps to the particular multicast address on which messages on this subject are to be directed. All other nodes in the group merely “invite” messages on the same subject, specifying the same transmission mode and flow label. Each published message transmitted to the subscribing node is then automatically be delivered to all other members of the group as well. Note that no modification of AMS itself is required in order to provide this capability.

III. Notes on Applying AMS

AMS is designed to be usable without modification in a wide range of application environments, on the theory that most variations in the operational requirements imposed by these different environments can be accommodated by using different transport services underneath AMS.

A. Terrestrial Applications

In order for AMS to be usable for terrestrial applications, such as the operation of missions’ ground data systems, it must function well in the local area networks and wide-area networks that characterize terrestrial data communications. Running AMS over UDP/IP and TCP/IP provides asynchronous messaging functionality that benefits from the reliability and scalability of the Internet protocol suite.

A special case that requires additional attention is messaging between application modules running on computers that are separated by firewalls. In this scenario, intensive and ongoing firewall adjustment can be avoided by establishing distinct AMS continua behind each firewall boundary. Remote AMS can then be used to relay messages over a relatively static grid of secure and easily monitored point-to-point connections.

B. Spacecraft On-board Applications

Many on-board applications require “hard” real-time performance in message transfer. AMS has no intrinsic mechanisms for assuring bounded message delivery latency, but this performance profile can be provided by running AMS over transport services that do support such mechanisms. For example, an AMS transport service adapter might access the 1553 bus (or Firewire, Spacewire, etc.) directly.

As noted earlier, AMS can best be thought of as an extension of application functionality – simplifying applications’ utilization of transport services that already meet the applications’ communication requirements – rather than as a transport system in itself. Any available on-board communications mechanism that supports connectionless data interchange among individually addressable flight software entities should be able to support AMS operations.

C. Mission-wide Applications

The Remote AMS protocol was designed specifically to support message exchange across interplanetary space. The operation of a RAMS gateway balances efficient utilization of expensive deep space communication links with fine-grained subscription specification:

- Only a single copy of any message is transmitted on the interplanetary link between two RAMS gateways, no matter how many subscribers are registered in the destination continuum.
- Upon receipt of the message at the destination continuum, re-publication is subject to the domain constraints asserted by local subscribers.

For example, node A in a Mission Control continuum might subscribe to “low-battery” messages from the wide-field camera only, while nodes B and C subscribe to all “low-battery” messages. A “low-battery” message published by the laser altimeter in the Mars Orbiter continuum would be received by nodes B and C only, while one published by the orbiter’s wide-field camera would be received by all three nodes – but in each case only one copy of the message would be transmitted over the communication link between the orbiter and the mission control center.

IV. Prototype Implementation

A. Architecture

A prototype implementation of AMS has been developed at JPL. The implementation has the following components:

- A daemon process (*amsd*) that can function as a configuration server, as the registrar for a single zone of a single message space, or both, depending on the values of command-line parameters.
- A library (*libams*) that implements the AMS application programming interface. This library is typically provided as a “shared object” that is dynamically linked with application code at run time.

All of the code is written in the C programming language. It relies on a common infrastructure library named “ICI”, derived from elements of the CFDP implementation that was part of the Deep Impact mission’s flight software. The ICI infrastructure base includes a portability layer (called "platform") that simplifies the compilation and execution of common code in a variety of operating environments including Linux, VxWorks, and Interix (Windows Services for Unix). It also includes a memory management system, called "PSM", which provides dynamic management of a privately allocated block of memory. This may be useful in environments such as spacecraft flight software where the dynamic management of system memory (*malloc*, *free*) cannot be tolerated.

Using this implementation of AMS for application development and operations entails the following steps:

- Creating one or more management information base (MIB) files, in XML format. All daemon and application processes must load their management information bases from mutually consistent MIB files.
- Linking application code with *libams* and *libici*.
- Starting one or more *amsd* daemon processes, to establish the required MAMS framework.
- Starting the application processes.

B. Performance

Evaluation of this prototype implementation of AMS by NASA, ESA, and CNES has only recently begun, and the findings are far from complete. Some preliminary benchmarking performed in JPL's Protocol Test Laboratory has yielded the figures presented in Table 1 below. These results were obtained from a simple two-node system – a single subscriber and a single publisher – running on a Gigabit Ethernet, where each of the nodes was hosted on a dual-core 3GHz Pentium-4 system running Fedora Core 3 with an 800 MHz FSB, 512MB of DDR400 RAM, and a 7200 rpm hard disk. Real-world systems running on slower networks will clearly not exhibit this level of performance, but these results at least suggest that the additional processing imposed by the publish/subscribe model is not necessarily incompatible with efficient operations.

Number of messages sent	*Size of each message (bytes)*	*Messages exchanged per second*	*Data rate (Megabits/sec)*
10,000	20,000	5,337	814
100,000	2,000	25,739	393
1 million	200	107,910	165
10 million	20	154,335	23

Table 1   Preliminary AMS Performance Benchmark Results

Note that, as message size decreases, AMS processing overhead becomes a more significant factor in effective data rate. Processor performance becomes the primary constraint on data rate and, conversely, network performance is no longer limiting.

V. Conclusion

The AMS definition effort is still in its early stages. The specification has been published as a CCSDS Proposed Standard ("white book")[6], but it may be modified significantly as it advances to Draft Standard and, finally, Recommended Standard status over the next 12 to 18 months. Evaluation of the prototype implementation in a variety of operational scenarios should provide valuable insight into the validity of the decisions underlying the protocol designs.

At this point, however, it appears that substantial progress has been made toward the goal of developing a new international standard for asynchronous message exchange in deep space mission operations.

Acknowledgments

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1]Consultative Committee for Space Data Systems, "CCSDS File Delivery Protocol (CFDP) Recommended Standard", Issue 3, June 2005, URL: http://public.ccsds.org/publications/archive/727x0b3.pdf.

[2]Eugster, P., Felber, R., Guerraoui, R., and Kermarrec, A., “The Many Faces of Publish/Subscribe,” *ACM Computing Surveys*, Vol. 35, No. 2, Jun. 2003, pp. 114-131.

[3]Sun Microsystems, “Java Message Service Specification Version 1.1”, March 2002, URL: http://java.sun.com/products/jms/docs.html.

[4]Object Management Group, “Data Distribution Service for Real-time Systems, V1.1”, December 2005, URL: http://www.omg.org/technology/documents/formal/data_distribution.htm.

[5]Object Management Group, “CORBA Event Service Specification Version 1.2”, March 2001, URL: http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm.

[6]Consultative Committee for Space Data Systems, “Asynchronous Message Service (AMS) Proposed Standard”, December 2005, URL: http://public.ccsds.org/sites/cwe/sis-ams/Public/ams%20white%20book.pdf.