AIAA-2006-6127

A MESSAGE ORIENTED MIDDLEWARE FOR A SOFT REAL-TIME HARDWARE-IN-THE-LOOP SPACECRAFT FORMATION FLYING TESTBED*,†

Jason W. Mitchell‡ and David M. Zakar§
Emergent Space Technologies, Inc., Greenbelt, MD 20770-6334

Richard D. Burns¶ and Richard J. Luquette‖
NASA Goddard Space Flight Center, Greenbelt, MD 20771

The Formation Flying Test-Bed (FFTB) at NASA Goddard Space Flight Center (GSFC) provides a hardware-in-the-loop test environment for formation navigation and control. The facility is evolving as a modular, hybrid, dynamic simulation facility for end-to-end guidance, navigation, and control (GN&C) design and analysis of formation flying spacecraft. The core capabilities of the FFTB, as a platform for testing critical hardware and software algorithms in-the-loop, are presented with a focus on the implementation and use of a message-oriented middleware (MOM) architecture. The MOM architecture provides a common messaging bus for software agents, easing integration, and interfaces with the GSFC Mission Services Evolution Center (GMSEC) architecture via software bridge. The performance of the overall messaging service in the context of a soft real-time simulation environment is discussed.

Keywords: distributed computing, message oriented middleware, spacecraft formation flying, hardware-in-the-loop, real-time.

I. Introduction

Spacecraft formation flying is a concept that continues to attract significant attention; Fig. 1. The advantages of formation flying are manifold. The President's Commission on Implementation of United States Space Exploration Policy[1] identifies formation flying as one of seventeen enabling technologies needed to meet exploration objectives.

Both NASA and ESA are evaluating formation flying concepts for numerous planned missions. A brief list of currently planned missions include, NASA: Magnetospheric Multiscale (MMS),[2] Micro-Arcsecond X-Ray Imaging Mission,[3] Submillimeter Probe of the Evolution

†AIAA Modeling & Simulation Technologies Conference, August 21–24 2006, Keystone, CO.

‡Aerospace Scientist, Jason.Mitchell@emergentspace.com, AIAA Senior Member.
§Assoc. Software Engineer, David.Zakar@emergentspace.com, non-AIAA member.
¶Engineer, Code 591, Rich.Burns@nasa.gov, AIAA Member.
‖Engineer, Code 591, Rich.Luquette@nasa.gov, AIAA Member.

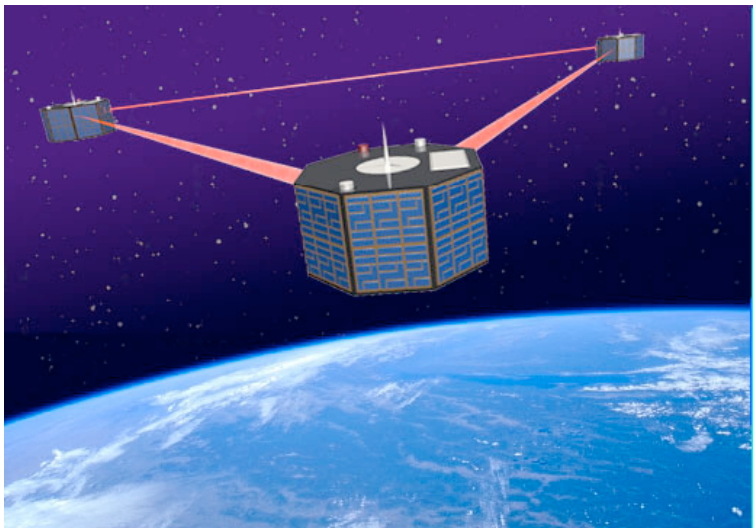

Figure 1. Artist's concept of formation flying spacecraft exchanging information via crosslink.

of Cosmic Structure,[4] Stellar Imager;[5] ESA: Darwin.[6] In addition, *precision formation flying* was chosen as one of the five candidate technology capability areas for the New Millennium Program's Space Technology 9 Project.[7,8]

To support the end-to-end guidance, navigation, and control design and analysis of formation flying spacecraft, the Formation Flying Testbed (FFTB) at NASA Goddard Space Flight Center (GSFC) allows formation flying navigation and control algorithms to be tested while interacting in real-time with the required flight hardware, such as relative navigation sensors and crosslink transceivers. By including hardware directly in the closed-loop testing, researchers and engineers can gain valuable information about the interaction and performance of their algorithms, and of the performance of the required hardware.

Supporting numerous low-level hardware interfaces as well as higher-level application program interfaces (API) provides a significant challenge in the FFTB. To facilitate the integration of software components, a Message Oriented Middleware (MOM) was implemented. Following a brief review of the FFTB hardware and software capabilities, the design and performance analysis of the MOM in the context of a soft real-time simulation environment is presented.

II. Facilities Overview

This section summarizes the hardware and software capabilities in the FFTB.

II.A. Hardware

Fig. 2 is a graphical depiction of connectivity for a sample Earth orbiting, two spacecraft simulation, where GPS is employed. The architecture of the FFTB is scalable with respect to the number of spacecraft, however certain hardware testing configurations limit that number to between two and four. The reason for these limits will become clear from the following description of the individual hardware elements.

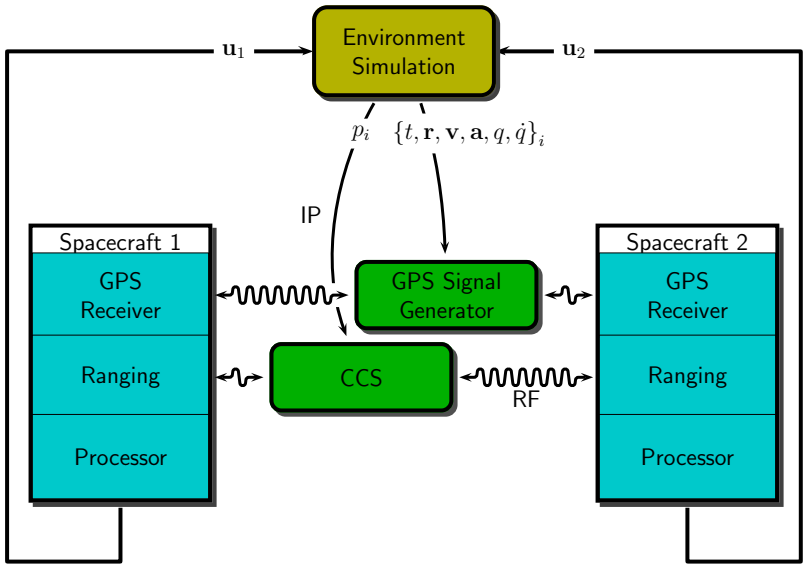

Figure 2. Information flow for sample two satellite Earth orbiting simulation employing GPS and RF ranging.

II.A.1. GPS Receivers

Currently, for Earth orbiting applications, the FFTB includes GPS receivers in-the-loop: two Orion receivers and four PiVoT receivers. The Orion receivers, from the German Space Operations Center, possess a direct relative navigation feature, allowing exchange of GPS measurements via direct RS-232 cable connection in the laboratory configuration.

II.A.2. GPS Simulator

A pair of Spirent® STR4760 GPS signal generators provide four Radio Frequency (RF) outputs. These outputs stimulate the GPS receivers. Each receiver's antenna pattern is configurable.

II.A.3. Crosslink Channel Simulator

The Crosslink Channel Simulator[9] (CCS) simulates the space environment for RF signals used for inter-spacecraft communication and ranging. The CCS applies delay, Doppler shift, and attenuation to an RF crosslink between two spacecraft, see Fig. 2. At present, the CCS supports S-band RF input frequencies between 2.0 GHz and 2.5 GHz, and provides two addressable channels.

II.A.4. Crosslinks

Using the CCS, spacecraft communication hardware that broadcasts and receives in the S-band range covered can be used in simulation. In recent applications, crosslinks have been used for inter-spacecraft communication and RF ranging.

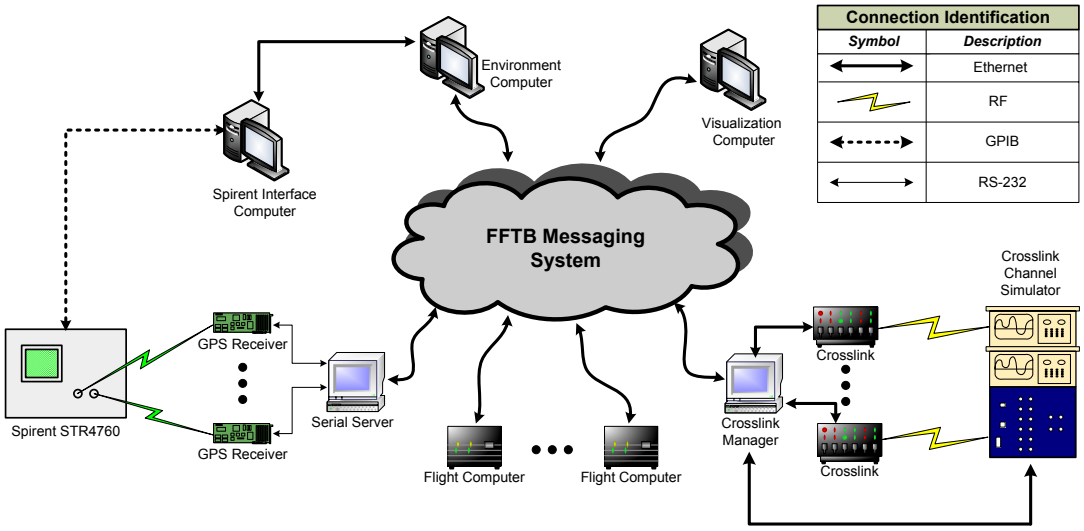

Figure 3. Hardware interface connectivity.

II.A.5. Computational Resources

Currently, there are four flight computers with single processor HyperThreaded Pentium 4 (6-series) units, clocked at 3 GHz, and 1 GB of RAM. To emulate the computational power of typical flight computers, these computers can be stepped down using Intel's SpeedStep® technology via the Linux `cpufreq` module.[10] The 6-series processors offer eight frequency steps for clocking (kHz): 3000, 2625, 2250, 1875, 1500, 1125, 750, 375, so providing a range of on-board computing resources. This is clearly a compromise to allow for easy and speedy development while attempting to provide some realism in expected computational power during simulation.

Spacecraft trajectory and attitude truth are computed on an AMD X2 4800+ with 2 GB of RAM.

II.B. Connectivity

Within the current architecture of the FFTB, hardware components are connected as shown in Fig 3. As new hardware becomes available, the low-level interfaces are developed as needed. Higher level integration, which is handled by the FFTB Messaging System, is described in the later sections.

II.B.1. Serial

The Orion and PiVoT GPS receivers are connected to the serial server, shown in Fig 3, via RS-232.

II.B.2. IEEE 488

As seen in Fig 3, the GPS signal generators are connected to the interface computer via the General Purpose Interface Bus (GPIB).

II.B.3. Ethernet

The FFTB internal Ethernet network is a Gigabit, switched, Class C network, that can operate at full-duplex between capable hardware. For hardware with a fixed Ethernet interface of 10/100 Mbps, the network is fully auto-negotiated. Critical hardware components with Ethernet interfaces are constrained to a single switch to guarantee a common back-plane and reduce latency. Multiple switches can be linked with mini-GPIC modules.

II.B.4. RF

Radio frequency components are generally connected via RG-142 cable. For the case of static crosslink ranging tests, high loss RG-174 is sometimes used in place of the CCS.

II.C. Software Architecture

The FFTB software systems drive soft real-time hardware-in-the-loop simulations. The simulation software components are dividend into the following areas:

II.C.1. Environment

The environment system provides the simulation with truth state information, and provide for the generation of output to be consumed by sensors, e.g. RF signals, noise model corrupted states. Currently , the environment is composed of: Trajectory and Attitude, GPS Signal generation, and RF space environment modeling.

Trajectory and Attitude: The Spacecraft Trajectory and Attitude Real-time Simulation (STARS) drives the truth spacecraft environment in the FFTB. STARS provides truth orbit and attitude trajectories to the overall simulation at a 10 Hz update rate, synchronized to a 1Pulse Per Second (PPS) signal provided by the Spirent® hardware previously described. Maneuver inputs are accepted by STARS in a vehicle local reference frame.

GPS Signal Generator: The SimGEN® software package drives the Spirent® GPS Signal Generators. STARS feeds true vehicle state information directly to SimGEN®, which simulates the GPS satellite constellation to produce realistic RF GPS signals that stimulate receivers used in a simulation. The RF signal generators also produce a 1PPS synchronization signal that is distributed to other devices that require a 1PPS timing discipline. The SimGEN® software is updated by STARS with truth state information at 10 Hz, and allows truth state data to arrive up to 20 ms early and 80 ms late. This data acceptance window represents the soft real-time aspect of the simulation since the data delivery deadline floats.

Crosslink Channel Simulator: The CCS software provides the interface to control the hardware described previously. This control is performed with User Datagram Protocol (UDP) messages over Ethernet. The CCS interface software is updated at 10 Hz with truth state information provided by STARS, and computes a set of channel parameters that include Doppler shift, signal delay, and power attenuation that are sent to the CCS. This software component also contains user selectable models that can modify the computed channel parameters to include Earth ionospheric effects, transmit/receive antenna gain, etc.

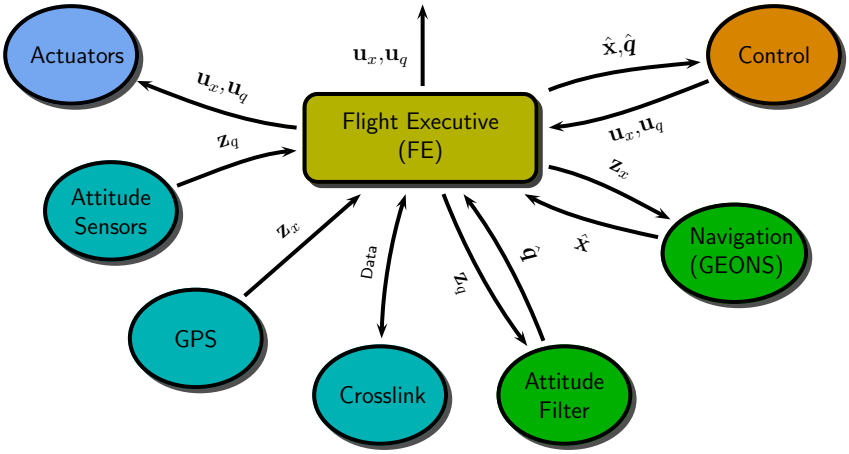

Figure 4. Flight Executive (FE) component connectivity diagram.

II.C.2. Flight Software

From Fig. 2, each spacecraft flight computer instantiates a Flight Executive (FE), composed of a collection of Java/C/C++ programs and MATLAB scripts; see Fig. 4. The FE accepts GPS or other sensor measurements and processes them with GEONS[11] for orbit determination, navigation and control. A flexible formation controller interface accommodates control algorithms written in MATLAB, Java, C and C++.

In the absence of the crosslink hardware and the CCS, a software crosslink ranging model is available to augment position measurements to improve formation knowledge and control. Fig. 2 shows the information flow for a two satellite Earth orbiting configuration using RF ranging via crosslink and GPS.

A single FE instance runs on each of the flight computers described in the previous section.

II.C.3. Visualization

Full 3-D scene visualization of the real-time simulation data is accomplished with the Satellite Tool Kit®[12] in combination with FFTB adapter software using STKConnect. Full scene 3-D visualization can also be performed with the *Jatalizer*, leveraging the Java Astrodynamics Toolkit[13] (JAT) to perform visualization. In addition to full scene visualization, the FFTBPlot tool can be used to graph simulation data.

III. Message-Oriented Middleware

As seen in the previous sections, connecting all of the components in the FFTB into a cohesive hardware-in-the-loop simulation presents a significant integration challenge. To address this challenge, a Message-Oriented Middleware (MOM) layer called the FFTB Messaging System (FMS) was introduced into the FFTB. The introduction of the FMS significantly reduces the development burden of integrating software components into the existing software infrastructure. This is particularly useful for collaborators that wish to test algorithms within the FFTB, allowing more time for testing and evaluation, and reducing component integration time.

However, the benefits of easy integration and extension must be balanced against the

real-time requirements. Consequently, characterizing and understanding the performance of the FMS is essential to maintaining simulation capability.

III.A. Design

The core components of the FMS were selected to satisfy the following fundamental goals: low cost; readily available and easily managed connectivity; well defined, yet flexible messaging structure; and high-performance message brokering for distributed computing. In addition, it was decided that any third party applications must include source code, allowing in-house customization to meet FFTB specific needs. This decision and the desire for minimal cost motivated the consideration of Free Open Source Software (FOSS) products and open standards. Connectivity, messaging structure, and brokering are discussed in the following.

III.A.1. Connectivity

Messages are exchanged over Gigabit Ethernet. The ubiquity of Ethernet, low cost, extensive cross-platform support, and standard interface provide a low-risk and robust connectivity solution. Further, the FFTB was historically connected via Gigabit Ethernet where possible, and its expanded use continues leveraging that previous investment.

In real-time systems, Ethernet is frequently too slow to meet an application's latency requirements. Fortunately, the FFTB operates in a soft real-time mode as described in § II.C.1, and the update rate of 100 ms is quite large compared to the average round-trip latency of a single backplane switched Gigabit network. Table 1 clearly indicates that even the maximum latency incurred does not occupy a significant portion of our update rate. Thus, Gigabit Ethernet satisfies the latency requirements.

III.A.2. Message Structure

To enable the FMS to easily exchange data between heterogeneous platforms, individual messages are formatted in XML version 1.0,[14] with entries composed as empty-element tags containing (*key,type,value*) attributes. A simple message could be:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<msg>
 <e key="key4" typ="I" val="12" />
 <e key="key2" typ="F" val="1234.5" />
 <e key="key3" typ="D" val="1234.12345678" />
</msg>
```

Dissecting the example above, the message begins with a valid XML prolog, including version and encoding information. This prolog is required for a well-formed XML message[‖]. The `<msg>` start-tag then follows the prolog. Next, an unordered set of empty element-tag entries falls between the start-tag and the `</msg>` end-tag. Each entry has the following form:

```
<e key="some-key" typ="some-type" val="some-value" />
```

where attributes within an entry are strictly ordered, and must appear as shown above: `key`, `typ`, `val`.

While XML is an obvious choice to manage message data semantics because of the depth of its framework, there is a price to pay for the benefit gained since every external data

[‖] see § 2.8 of Ref.[14]

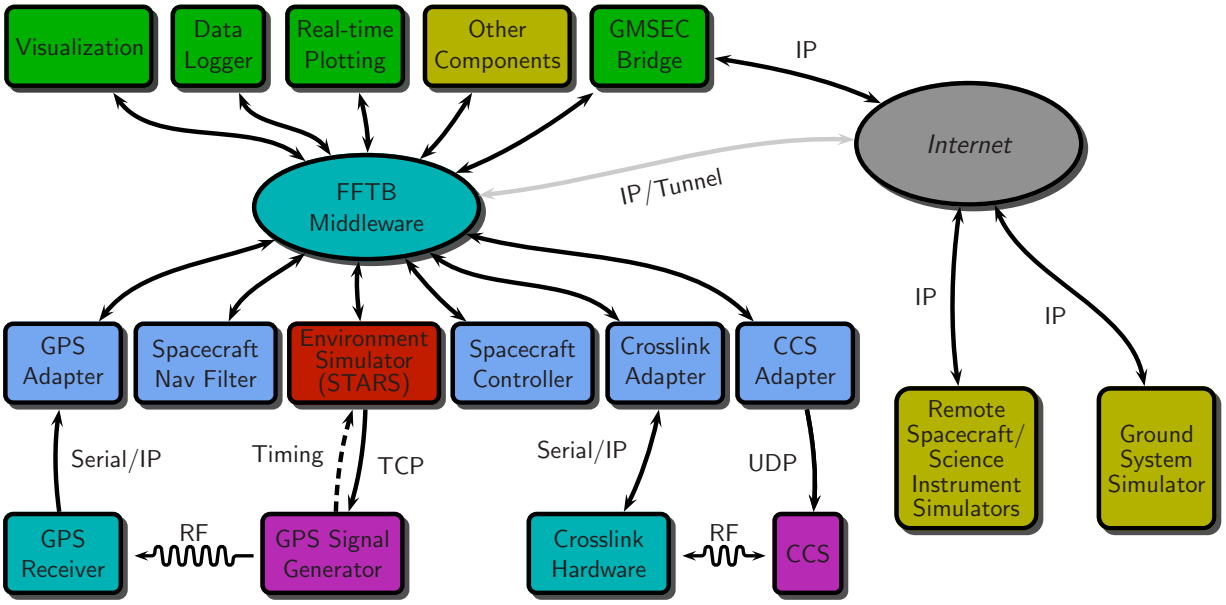

Figure 5. FFTB Messaging System software architecture.

exchange requires adding semantics before sending and parsing the semantics after receiving. This additional overhead can be significant.

III.A.3. Message Brokering

Managing numerous point-to-point, or unicast, network connections is very cumbersome and error prone. Each time a new consumer wishes to receive a message, both the new consumer and the message producer must be modified for the point-to-point connection. While there are techniques that can reduce this development burden, e.g. IP-multicast protocols, there is still a considerable effect on the ease of systems integration. Other issues to consider are delivery reliability, group membership information, message ordering, etc. Rather than addressing each of these problems individually, a better solution would be a general purpose messaging service to broker messages, so that distributed applications are easier to build. After surveying several message service solutions, the freely available Spread Toolkit[15–17] was selected.

The Spread Toolkit website[15] states:

> Spread is an open source toolkit that provides a high performance messaging service that is resilient to faults across local and wide area networks. Spread functions as a unified message bus for distributed applications, and provides highly tuned application-level multicast, group communication, and point to point support. Spread services range from reliable messaging to fully ordered messages with delivery guarantees.
>
> Spread can be used in many distributed applications that require high reliability, high performance, and robust communication among various subsets of members. The toolkit is designed to encapsulate the challenging aspects of asynchronous networks and enable the construction of reliable and scalable distributed applications.

The Spread Toolkit, version 3.17.3, interface[18] provides the following message ordering options: none, FIFO by sender, casual order,[19] and total order; and the following message

reliability options: unreliable, reliable, and safe. As a Group Communication System (GCS), supports Extended Virtual Synchrony (EVS) and Open-Group (OG) semantics. With OG semantics, a client is not required to be a member of a group to send a message to that group. For EVS, client notification is not required for changes in group membership. The benefit of EVS and OG semantics are, generally, faster performance and improved scalability. Of course, since membership notifications are not required, algorithms must be designed to tolerate unanticipated changes in group membership.

III.A.4. FFTB Messaging System

The FMS architecture, Fig. 5, uses a customized form of the Spread Toolkit as the foundation of a *publish/subscribe* messaging system. In the publish/subscribe model, publishers export their messages to clearly defined subject names following the naming convention prescribed by the GSFC Mission Services Evolution Center (GMSEC) Interface Specification Document.[20] The subscribers listen to the desired subject**, receiving each message so published. The naming convention states that subject names will be a dot (.) delimited string, consisting of upper-case alphanumeric characters, the underscore (_), and the dash (-). The subject elements are strictly ordered as: System, Mission (constellation or scenario name), Satellite ID, Message Type, Message Subtype, Component Name, Message Name. Individual messages are formatted as discussed, with message definitions maintained by the FMS Interface Control Document[21] (ICD).

Although the FMS follows the GMSEC subject naming convention, it is not GMSEC compliant. Currently, there is no free-ware/open-source GMSEC compliant middleware available. However, FMS and GMSEC messages can be exchanged, Fig. 5, via a GMSEC bridge.

III.A.5. FMS Language and Platform Support

The Spread Toolkit is written in ANSI C, supports many common platforms, and has language bindings for C, C++, C#, Java, Perl, Python, and Ruby. The FMS libraries are available in both Java and C++, and include support for XML marshalling and unmarshalling as well as other supporting functionality, e.g. input file parsing, state file reader, etc. While the FMS is primarily targeted to Linux, it should be easily portable to any POSIX compliant operating system. Recently, initial support for Perl has been added, and work is underway to provide a Python binding.

IV. Testing

Our objective in evaluating the performance of the FMS is to categorize both message throughput and latency, quantifying relative effects of bus and parsing delay. In a soft real-time simulation environment, such as the FFTB, minimizing message latency is of critical importance. Recall, the truth state updates at 10 Hz, i.e. every 100 ms. Thus, losing 30 ms to a slow middleware implementation could be catastrophic.

Amir et al.[17] provide a basic architectural outline for Spread as well as baseline benchmarks for raw messaging performance. It is clearly of interest to verify this performance,

**GMSEC *Subjects* in this context correspond to Spread *groups*.

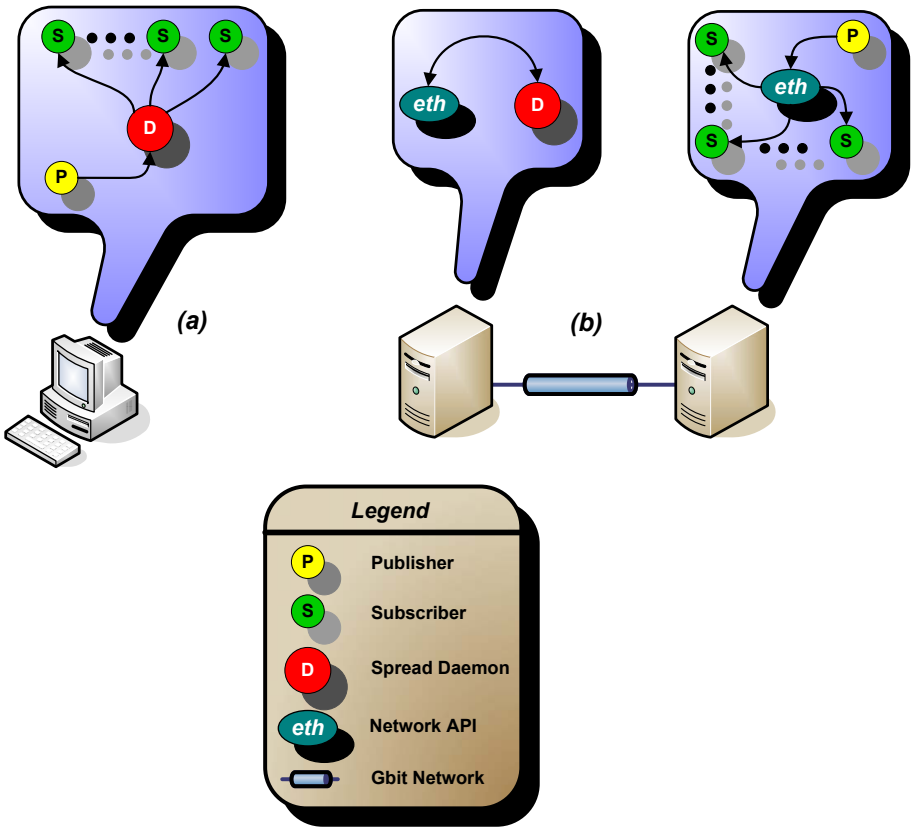

Figure 6. Architectural diagram of test configurations.

and to understand the effects of message marshalling and unmarshalling that are added by sending XML messages.

Of the performance claims[17] made for the Spread Toolkit, we are most interested in:

1. Message latency is small, $\mathcal{O}(10\,\text{ms})$, even for large messages;
2. Message size has little effect on average throughput for messages larger than about 1 KB, up to the maximum message size tested, i.e. 20 KB.
3. Message delivery latency is unaffected by the number of spread groups.

These claims clearly mark Spread as a high-powered messaging system, and very useful in simulation environments that require low latency.

IV.A. Testing Setup

The general architecture of the test setup can be seen in Fig. 6. In Scenario 6a, a computer ran a single instance of a Spread dæmon, with one publisher and a number of subscribers. This scenario is referred to as *Loopback* in the following text, since it used the the loopback network interface. In Scenario 6b, the Spread dæmon ran on one computer, while the publisher and subscribers ran on a separate computer. This scenario is referred to as *Network* in the following text because the two computers were connected with Gigabit Ethernet via a

Packets (#)	Min (μs)	Avg (μs)	Max (μs)	Mdev (μs)
10^2	34	38	45	6
10^3	34	44	215	12
10^4	34	45	221	13

Table 1. Gigabit Network Round-Trip Time (RTT) Latency reported by the `ping` command.

single backplane Gigabit switch. This network has an average latencies as shown in Table 1, for a conditioned connection.

For the scenarios shown in Fig. 6, the following hardware was used:

Computer 1: AMD Athlon 64 X2 4400+ CPU and 2 GB of RAM, with the Fedora Core 4 x86-32 operating system;

Computer 2: AMD Athlon 64 3200+ CPU and 2 GB of RAM, with the Fedora Core 3 x86-64 operating system.

Computer 1 was the single computer used in Scenario 6a. For Scenario 6b, Computer 2 ran the Spread dæmon, while Computer 1 ran the the single publisher and varying number of subscribers.

The following tests were run for each of the scenarios in Fig. 6:

Send measured the average time taken to send messages of varying size from the client to the Spread dæmon only. No subscribers were serviced in this test.

Send-receive measured the average latency to send and successfully receive 10^3 messages of varying size. In addition, the number of subscribers varied from 1–9. The publisher and each subscriber ran as separate threads, and each XML message was parsed by the subscriber before the timer was stopped. This test is representative of the flight software implementation in the FFTB seen in Fig. 4.

Throughput measured the average data rate for a range of messages sizes and number of subscribers. To establish an average throughput, the total time to publish each message 10^4 times to a number of subscribers was recorded. Thus, the average data rate was computed as the total amount of data transported to the subscribers divided by the total measured time. The publisher and subscribers in this test were executed as individual processes, which is representative of simulation component integration in the FFTB.

In each test, message sizes varied from 100 Bytes to slightly less than 10 MB. All messages were of the `AGREED` delivery type for total order.

Clock synchronization and bias issues in calculating message latencies was avoided in the Send-receive test by hosting both the publisher and subscribers on a single computer. For this case, all clients, processes and threads, shared the same clock, and if necessary could read the CPU's read-time-stamp counter (RDTSC) for higher precision timing. Time was measured using the POSIX.1b monotonic timer, e.g. `CLOCK_MONOTONIC`, which provided a resolution of 4 μs.

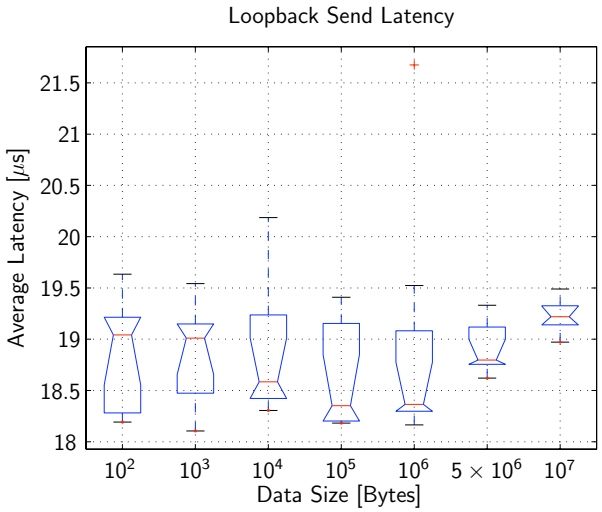

Figure 7. Loopback send latency by message size.

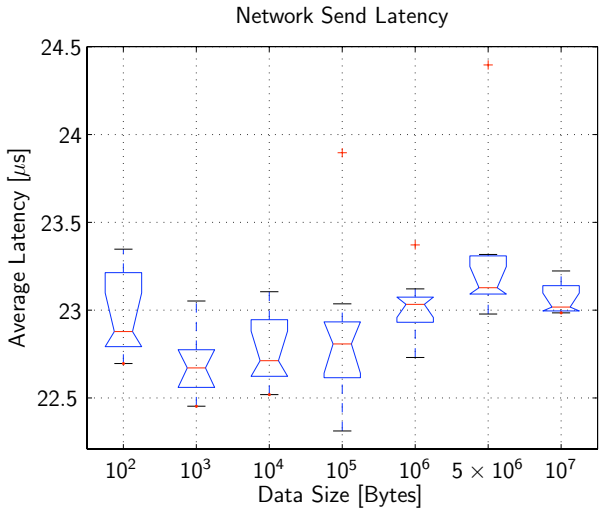

Figure 8. Network send latency by message size.

V. Results

V.A. Introduction

The data summarized in the section makes frequent use of notched box and whisker plots, i.e. *boxplots*. Each box, seen in Figs. 7–12, has lines at the lower quartile, median, and upper quartile values, with whiskers extending to the extreme value within 1.5 times the interquartile range (IQR). Individual values that lie outside the the whisker range of the median, approximately 2σ, are termed *outliers* and are marked with a plus-sign, +. The notches provide an indication of the robustness of the estimated uncertainty of medians when comparing boxes at a 5% significance level. Thus, if the notches of two boxes overlap, we conclude, with 95% confidence, that the true medians do not differ.

V.B. Send-only

The send-only Loopback and Net latency test results are summarized in Figs. 7 and 8.

From Fig. 7, we see that, except for message sizes larger than 10^6 bytes, the send-only latency is relatively constant and quite small as compared to the 100 ms execution window. For messages larger than 10^6 bytes, we see a clear increase in median latency, however it is relatively small, but is unfortunately below the single query resolution of the system clock. Considering worst case latency, we see that it is bounded from above at less than 22 μs.

In Fig. 8, the effect of network interaction can be seen on the send-only latency. The median latencies appear to differ over the range of message sizes, but as before, the variation of the latency is less than the single query clock resolution. We expect to see some variation in latency as the message size increases beyond the maximum network packet size and fragmentation occurs. As before, considering the worst case latency, we find it is less than 25 μs.

Comparing the worst case between Figs. 7 and 8, we find that the network send-only performance is only approximately 5 μs worse than the loopback performance. This may seem very optimistic given the round-trip-time (RTT) network latency data found in Table 1. In this comparison, network activity must be considered. The latency testing was performed

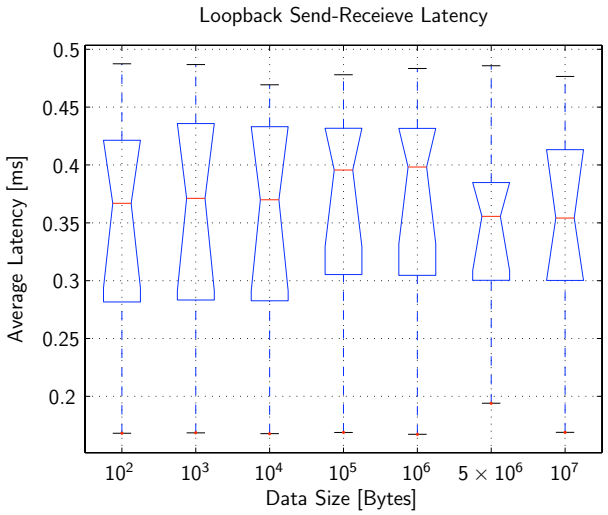

Figure 9. Loopback send-receive latency by message size.

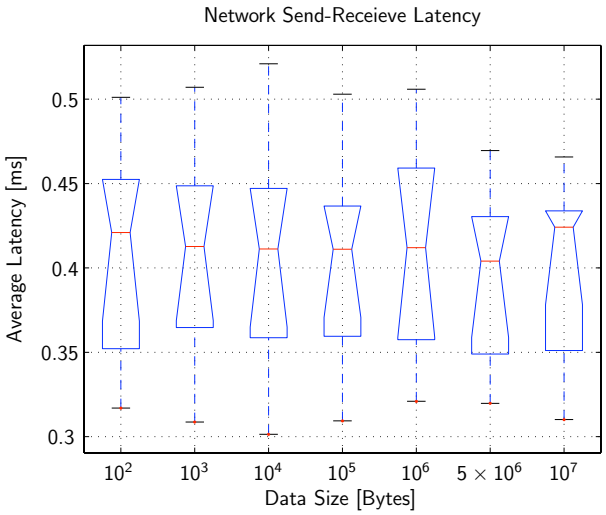

Figure 10. Network send-receive latency by message size.

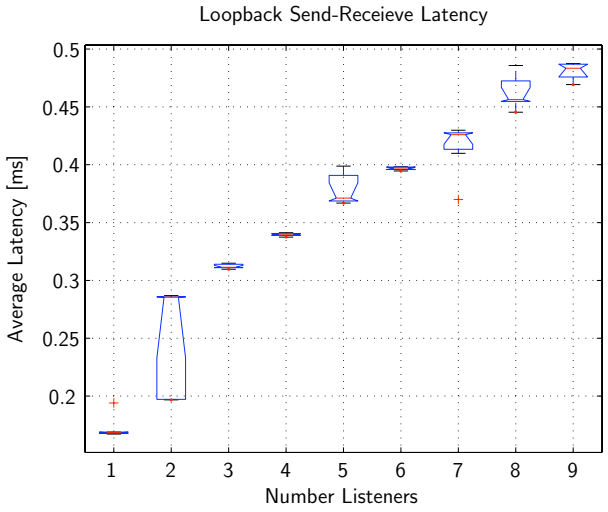

Figure 11. Loopback send-receive performance by subscriber load.

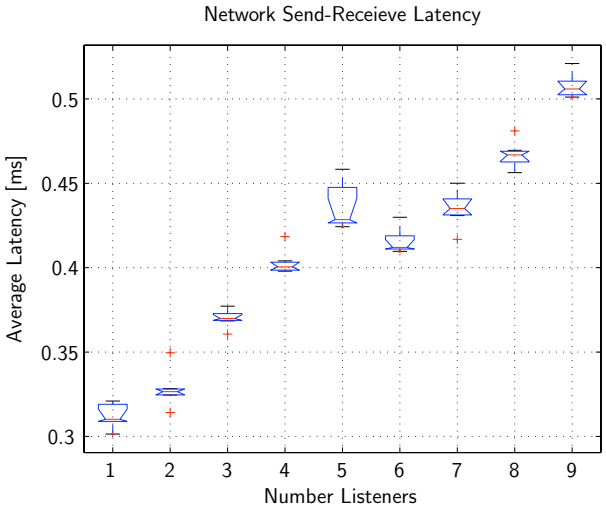

Figure 12. Network send-receive performance by subscriber load.

with a dedicated quiescent network. However, the data provided in Table 1 was collected with light simulation activity. With this, and recognizing that send-only is one-way latency and not RTT latency, the performance appears to meet expectations.

V.C. Send-receive

The send-receive latency summaries are shown in Figs. 9 and 10. Each boxplot in these figures represents the distribution of median latencies of each message size for the prescribed number of subscribers. Both figures indicate that for 1–9 subscribers, median message latency is relatively constant with respect to message size, and the effect of interacting with the network is minimal. The worst case latency is less than 500 μs for both loopback and network cases. This generally agrees with Amir et al.[17]

Figs. 11 and 12 provide a slightly different view of the send-receive latency. In this case, we aggregate median latency for all message sizes by number of subscribers. For both loopback and network cases, each subscriber adds finite latency. Although the added latency

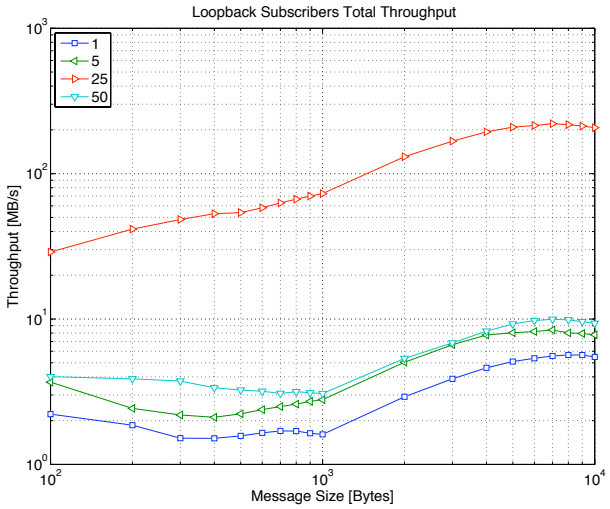

Figure 13. Loopback throughput by message size and number of subscribers.

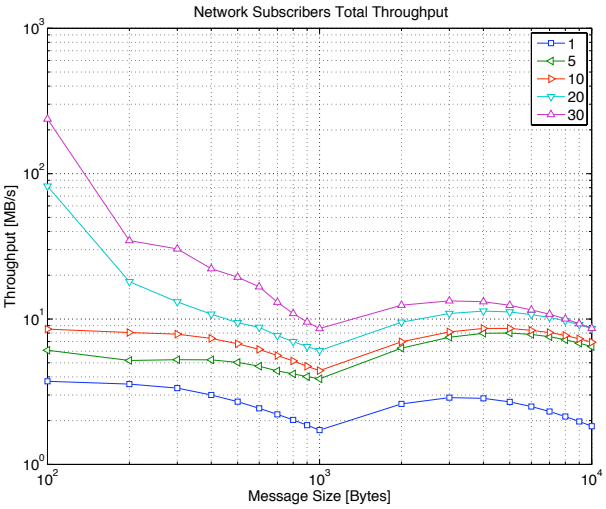

Figure 14. Network throughput by message size and number of subscribers.

is not strictly constant, it does appear to be a linear trend. For the worst case, loopback latency was less than 500 μs, with the network latency under 550 μs.

Amir et al.[17] makes no direct statement about the effect of the number of subscribers, i.e. *group members*, on message delivery latency for a single dæmon. Further, the test configuration of Amir et al.[17] is quite different from our usage since each computer runs a Spread dæmon that is network linked. In our usage, a single Spread dæmon instance served all subscribers that were members of a single group. Amir et al.[17] does include a test that measures the latency in join/leave operations on a single dæmon instance as a function of subscribers, i.e. *group size*. The join operation incurs the highest latency, and it increases quadratically from approximately 1 ms for 1 subscriber up to approximately 35 ms for 10^3 subscribers. The quadratic growth occurs because the subscriber notification size also grows linearly along with the linearly increasing number of subscribers. Message delivery latency should, however, increase linearly since the subscriber notification size is fixed as is the message size. For the comparatively few subscribers shown in Figs. 11 and 12, the expected linear increase in latency manifests. Fortunately, the worse case for both loopback and network latencies for nine (9) subscribers remains small relative to the update window of 100 ms. Clearly, additional data collection over a broader range of subscribers is required to fully characterize the latency per subscriber.

V.D. Throughput

The average throughput by message size and number of subscribers is shown in Figs. 13 and 14.

In Fig. 13, we see that loopback throughput increases up to 25 subscribers, then falls dramatically for 50 subscribers. This indicates that the system was oversubscribed. For the case of 25 subscribers, the maximum theoretical capacity of a Gigabit network easily exceeded, even for relatively small messages sizes. Thus, Spread performs well on loopback throughput.

Fig. 14 summarizes the network throughput. Considering the overload of 50 subscribers seen in Fig. 13, the number of subscribers was changed for the network test to: 1, 5, 10, 20,

30. The network throughput, as expected, generally increases with the number of clients, particularly for messages smaller than 10^3 bytes. Interestingly, for 30 subscribers, efficient delivery and clock resolution combine to give the appearance that the network capacity was exceeded. Overall, for messages less than 10^3 bytes, it appears that additional delivery capacity remains. However, for messages greater than 10^3 bytes, and particularly for the largest message size, 10^4 bytes, it appears that the throughput saturates near 10 MB/s. This is consistent with the throughput stated in Amir et al.[17] However, since their testing used a 100Mbps network, rather than a Gigabit network, it appears that our testing is limited by our single Spread dæmon configuration. Fortunately, this is not a significant problem at this time because our message sizes are generally less than 2 KB.

VI. Conclusions

This work presents the implementation and performance of a message-oriented middleware used in a soft real-time, hardware-in-the-loop simulation environment. The message-oriented architecture provides a common messaging bus for software agents, eases component integration, and supports the NASA GSFC Mission Services Evolution Center (GMSEC) architecture via software bridge. The custom messaging system is composed of the Spread Toolkit for message brokering, and a GMSEC motivated publish and subscribe paradigm that exchanged XML messages.

The performance of the messaging system is found to agree reasonably well with the performance described for the Spread Toolkit. In addition, the overhead of the XML marshalling and unmarshalling has little effect on message delivery latency. The configuration of a single message broker has potential scaling problems, however the amount of data collected is insufficient to conclusively identify message/subscriber scaling issues.

References

[1]Aldridge, Jr. (Chairman), E. C., "A Journey to Inspire, Innovate, and Discover," Tech. rep., President's Commission on Implementation of United States Space Exploration Policy, June 2004, `http://tinyurl.com/d28rx`, Accessed August 5, 2006.

[2]Smith, D. and Colón, G., "Magnetospheric Multiscale Mission," 2006, `http://stp.gsfc.nasa.gov/missions/mms/mms.htm`, Accessed August 5, 2006.

[3]White, N. E. and Newman, P., "Micro-Arcsecond X-ray Imaging Mission," 2006, `http://maxim.gsfc.nasa.gov/`, Accessed August 5, 2006.

[4]Leisawitz, D., "Submillimeter Probe of the Evolution of Cosmic Structure," 2006, `http://space.gsfc.nasa.gov/astro/specs/`, Accessed August 5, 2006.

[5]Carpenter, K., "Stellar Imager," 2006, `http://hires.gsfc.nasa.gov/~si/`, Accessed August 5, 2006.

[6]European Space Agency, "Darwin Mission," 2006, `http://tinyurl.com/86n5h`, Accessed January 27, 2006.

[7]Fisher, Diane, K. and Leon, N. J., "New Millennium Program's Space Technology 9 (ST9) Project," 2006, `http://nmp.jpl.nasa.gov/st9/`, Accessed August 5, 2006.

[8]Beasley, D. and Hupp, E., "NASA Selects Advanced Technology Providers," NASA News, July 2006, `http://tinyurl.com/aztru`, Accessed August 5, 2006.

[9]Hunt, C., Smith, C., and Burns, R., "Development of a Crosslink Channel Simulator," *Proceedings of the IEEE Aerospace Conference*, Vol. 2, 2004, pp. 1322–1328.

[10]Brodowski, D., "Linux kernel CPUfreq subsystem," 2006, http://tinyurl.com/75ulj, Accessed August 5, 2006.

[11]NASA GSFC Mission Engineering and Systems Analysis Division, "GEONS Open Architecture Solutions for Onboard Orbit Determination in any Orbit," 2006, http://geons.gsfc.nasa.gov/, Accessed August 5, 2006.

[12]Analytical Graphics, Inc., "Satellite Tool Kit," 2006, http://stk.com/, Accessed August 5, 2006.

[13]Gaylor, D., Berthold, T., and Takada, N., "Java Astrodynamics Toolkit," 2006, http://jat.sf.net/, Accessed August 5, 2006.

[14]"Extensible Markup Language (XML) 1.0," Tech. rep., World Wide Web Consortium (W3C), June 2004, http://tinyurl.com/3kxga, Accessed August 5, 2006.

[15]Spread Concepts, LLC, "The Spread Toolkit," 2006, http://spread.org/, Accessed August 5, 2006.

[16]Amir, Y., Danilov, C., and Stanton, J., "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication," *2000 International Conference on Dependable Systems and Networks (DSN 2000) (formerly FTCS-30 and DCCA-8), 25-28 June 2000, New York, NY, USA*, IEEE Computer Society, 2000.

[17]Amir, Y., Danilov, C., Miskin-Amir, M., Schultz, J., and Stanton, J., "The Spread Toolkit: Architecture and Performance," Tech. rep., The Johns Hopkins University Distributed Systems and Networks (DSN) Laboratory, 2004, http://tinyurl.com/9r3u6, Accessed January 20, 2006.

[18]Spread Concepts, LLC, *The Spread Users Guide*, 2006, http://tinyurl.com/f3rkc, Accessed August 5, 2006.

[19]Lamport, L., "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, Vol. 21, No. 7, 1978, pp. 558–565.

[20]Dan Smith, "Goddard Space Flight Center Mission Services Evolution Center," 2006, http://gmsec.gsfc.nasa.gov/, Accessed August 5, 2006.

[21]Gaylor, D. and Bamford, B., "FFTB Messaging System Interface Specification," Tech. rep., Emergent Space Technologies, Inc., December 2005.

[22]Jackson, R. and Lawshe, C., "Terrestrial Planet Finder," 2006, http://planetquest.jpl.nasa.gov/TPF/, Accessed August 5, 2006.

[23]Weaver, K. and Mattson, B., "Constellation-X Observatory," 2006, http://constellation.gsfc.nasa.gov/, Accessed August 5, 2006.

[24]European Space Agency, "X-ray Evolving Universe Spectroscopy," 2006, http://tinyurl.com/9e23m, Accessed August 5, 2006.

[25]Leitner, J., "A Hardware-in-the-Loop Testbed for Spacecraft Formation Flying Applications," *Proceedings of the IEEE Aerospace Conference*, Vol. 2, 2001, pp. 615–620.

[26]Dvorak-Wennersten, M., Banes, A. V., Boegner, G. J., Dougherty, L., Edwards, B. L., and Roman, J., "PiVoT GPS Receiver," *Proceedings of the ION GPS Conference 2001*, 2001, pp. 855–861.

[27]Haas, L., Abousalem, M., and Murphy, J., "The Ashtech G12-HDMA: A Low Cost, High Performance GPS Space Receiver," *Proceedings of the ION GPS Conference 2000*, 2000, pp. 350–354.

[28]Naasz, B. J., Burns, R. D., Gaylor, D., and Higinbotham, J., "Hardware-in-the-Loop Testing of Continuous Control Algorithms for a Precision Formation Flying Demonstration Mission," *18th International Symposium on Space Flight Dynamics*, 2004.

[29]Antonucci, R. and Waktola, W., "Middleware Evaluation and Benchmarking for Mission Operations Centers," *2005 Ground System Architectures Workshop*, 2005, http://tinyurl.com/d2qbn, Accessed January 20, 2006.

[30]Antonucci, R. and Waktola, W., "Middleware Evaluation and Benchmarking for Mission Operations Centers," (internal report).