Source of Acquisition
NASA Ames Research Center

Software Architecture of Sensor Data Distribution In Planetary Exploration

Charles Lee[1], Richard Alena[2], Thom Stone[3], John Ossenfort[1], Ed Walker[2], Hugo Notario[2]

[1]SAIC / NASA Ames Research Center
[2]NASA Ames Research Center
[3]CSC / NASA Ames Research Center

Abstract: Data from mobile and stationary sensors will be vital in planetary surface exploration. The distribution and collection of sensor data in an ad-hoc wireless network presents a challenge. Irregular terrain, mobile nodes, new associations with access points and repeaters with stronger signals as the network reconfigures to adapt to new conditions, signal fade and hardware failures can cause:

- Data errors
- Out of sequence packets
- Duplicate packets
- Drop out periods (when node is not connected)

To mitigate the effects of these impairments, a robust and reliable software architecture must be implemented. This architecture must also be tolerant of communications outages. This paper describes such a robust and reliable software infrastructure that meets the challenges of a distributed ad hoc network in a difficult environment and presents the results of actual field experiments testing the principles and actual code developed.

1. Introduction

Sensor data are an essential part of planetary surface exploration. Sensor data arises from status monitoring, system health assessment as well as scientific sampling. All are required for the success of the mission. Sensor data is required for all aspects of a mission:

- Planning requires sensor data as input to determine location, environment and distances.
- Scheduling requires sensor information for calculating duration, time, position, and routes
- Operation requires sensor information to calculate location, progress, health status etc.

Sensor data are real time streams and are time critical. Parallel processing of sensor data to produce useful information introduces reliability issues. The two major causes of data loss are the burden of communications overhead and packet drops plus the difficulty of multithreaded programming. Packet loss in wireless systems can be caused by many factors such as:

- Congestion
- Moving out of range of base station
- RF interference
- Multi-pathing
- Obstacles to line of sight
- Routing problems

Transmitting sensor data accurately over a wireless infrastructure to single or multiple receivers in very difficult environments where packet loss and even loss of signal are the norm, are the problems that we addressed using software techniques at the applications level. Although this work was done for surface communications it could well have application to other data management areas, such as orbital asset management and spacecraft tracking and control. Satellite constellations usually consist of many instruments and sensors producing multiple steams to multiple consumers of the data. Satellites can experience periods of a high rate of data errors and periods of loss of signal. Like our surface data, spacecraft sensor data can

have highly distributed users, some on different planets.

We experimented with different schema frameworks to determine a method to find an optimal system for robustness and reliability of sensor data for surface communications.

In the end we selected Message Oriented Middleware (MOM) for our distributed infrastructure [3]. Message Oriented Middleware is a category of inter-application communication software that presents an asynchronous message-passing model as opposed to a request/response model.

A MOM has the following attributes:

- Fast
- Reliable
- Asynchronous
- Guaranteed message delivery
- Receipt notification
- Transaction control

As far as the client software is concerned, MOM is indistinguishable from real-time processing [4]. The primary advantage of a message-oriented communications protocol is the ability to store, route, and resend a message that needs to be delivered.

Most MOM systems provide persistent storage to hold messages until they are successfully transferred. This means that it is not necessary for the sender and receiver to be connected when data are created. This is useful for dealing with faulty connections, unreliable networks, and timed connections (where communications is only available during predictable periods). It also means that if a receiver fails to receive a message for any reason, the sender can continue unaffected, since the messages will be held in the message store and will be transmitted when the receiver reconnects.

MOM systems present two messaging models:

- Point to point:
- This model [2] is based on message stores known as queues. A sender sends a message to a specified queue. A receiver receives messages from the queue. A queue can have multiple senders and receivers, but an individual message can only be delivered to one receiver. If multiple receivers are listening for messages on a queue, the underlying MOM system usually determines which receiver will receive the next message. If no receivers are listening on the queue, messages remain in the queue until a receiver attaches to the queue.
- Publish Subscribe:
- This model [1] is based on message stores known as topics. Publishers send messages to a topic. Subscribers retrieve messages from a topic. Unlike the point-to-point model, many subscribers can receive the same message.

2. Overview of Message Services

Typically a message service is implemented using a Java framework. A message-driven bean (MDB) is an Enterprise Java Bean (EJB) that functions as a message consumer. Unlike session beans or entity beans, clients cannot access message-driven beans directly. Also, unlike session beans and entity beans, a message-driven bean does not have remote or home interfaces. The only access a client has to a message-driven bean is through a JMS (JAVA Messaging Service) destination (topic or queue) to which the message-driven bean is listening.

A MDB must implement two interfaces:

[1] ***javax.jms.MessageListener***--This interface defines the *onMessage* callback method. When a message is put on the queue/topic, the *onMessage* method of the message-driven bean is called by the EJB container and passed the actual message.

[2] ***javax.ejb.MessageDrivenBean***--This is the EJB interface that contains the EJB lifecycle methods: ***ejbCreate()***--called by the EJB container when the message-driven bean is created ***ejbRemove()***--called by the EJB

container when the message-driven bean is destroyed or removed from the EJB pool

setMessageDrivenContext(MessageDrivenContext context)--called prior to *ejbCreate* and passed the message-driven context by the EJB container

A message-driven bean must declare deployment information about itself in a deployment-descriptor file named *ejb-jar-xml.* The EJB container handles the duties of subscribing the bean to the topic or connecting it to the queue based on information placed in the deployment descriptor. The context has runtime information, for instance transaction data. The diagram in Figure 1 illustrates the interactions between a JMS message, a client, a topic, an application server, an EJB container, and message-driven bean instances.

As mentioned before, message-driven beans do not have remote or local interfaces as with session beans and entity beans. Message-driven beans are not located by client classes, and client classes do not directly invoke methods on them. All access to a message-driven bean is through a JMS topic or queue that directs messages at the message-driven bean through the EJB container. The EJB container ultimately passes the JMS message to the message-driven bean through the bean's *onMessage* method. All message-driven beans must implement the *javax.ejb.MessageDrivenBean* and *javax.jms.MessageListener* interfaces, as the example illustrates.

The Java Message Service (JMS) provides a standard Java-based interface to the message services of a MOM of a MOM schema from another vendor.

Messaging systems are classified into different models that determine which client receives a message. The most common messaging models are:

- Publish-Subscribe Messaging
- Point-To-Point Messaging
- Request-Reply Messaging

Not all MOM providers support all these models.

Publish-Subscribe Messaging

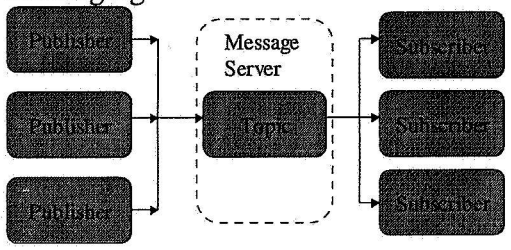


Figure 1 Publish subscriber Messaging

When multiple applications need to receive the same messages, Publish-Subscribe Messaging is used. The central concept in a Publish-Subscribe messaging system is the "Topic". Multiple Publishers may send messages to a Topic, and all Subscribers to that Topic receive all the messages sent to that Topic. This model, as shown in Figure 1, is extremely useful when a group of applications want to notify each other of a particular occurrence.

In Publish-Subscribe Messaging, there may be multiple Senders and multiple Receivers.

Point-To-Point Messaging

When one process needs to send a message to another process, Point-To-Point Messaging can be used., This may or may not be a one-way relationship. The client to a Messaging system may only send messages, only receive messages, or send and receive messages. At the same time, another client can also send and/or receive messages. In the simplest case, one client is the Sender of the message and the other client is the Receiver of the message.

There are two basic types of Point-to-Point

Messaging systems. The first one involves a client that directly sends a message to another client. The second and more common implementation is based on the concept of a Message Queue. Such a system is shown in Figure 2.

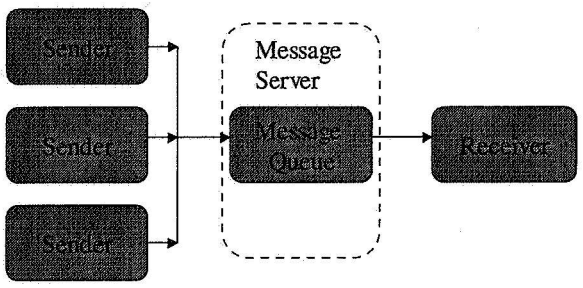


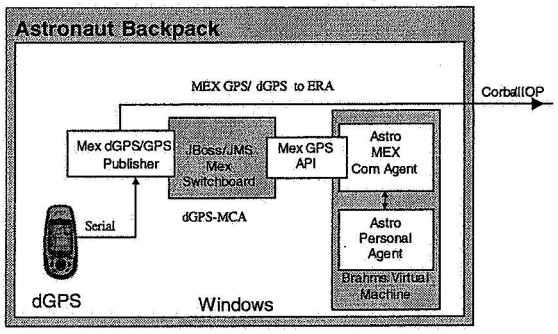
Figure 2. Point to Point Messaging

The critical aspect of Point-to-Point messaging is that, even though there may be multiple Senders of messages, there is only a single Receiver for the messages.

JMS Parent	Publish Subscribe Domain	Point To Point Domain
Destination	Topic	Queue
Connection Factory	TopicConnection Factory	QueueConnection Factory
Connection	TopicConnection	QueueConnection
Session	TopicSession	QueueSession
MessageProducer	TopicPublisher	QueueSender
MessageConsumer	TopicSubscriber	QueueReceiver, QueueBrowser

3. ARCHITECTURE DESIGN

We selected Publish Subscribe architecture for data distribution. In our project requirements, the data is to be distributed to multiple remote clients and the publisher may publish the data to a remote machine. These requirements are satisfied with Publish Subscribe architecture.



Figure

As Figure 3 shows, an Astronaut carries a backpack and our software runs on a computer in the backpack. A GPS unit is connected to the computer and the data are distributed to the JMS server by using the GPS server model. The client will access the data by subscribing to the topic using the API provided by the GPS server developer.

The Biological information from the Astronaut is also distributed by the architecture as shown in the Figure 4.

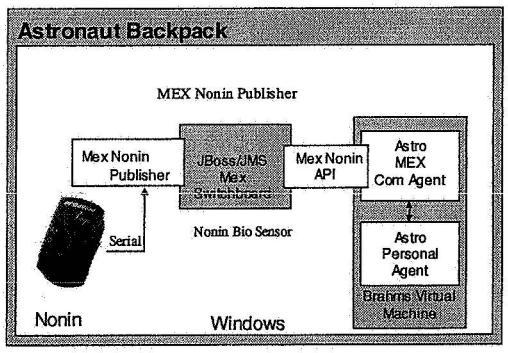


Figure 4. Biosensor architecture

We have (3) collaborated with a Robotic Rover team (Extra Vehicular Activity Robotic Assistant (ERA)) team from Johnson Space Center in several exploration field tests (see below). The sensor data needs to be distributed to the ERA server so that the robot can perform commands such as:

- Follow the astronaut
- Take a picture of an area that is of interest to the astronaut
- Take a picture of an astronaut
- Make a voice note
- Open a sample bag for a specimen

Since the ERA team is using a CORBA (Common Object Request Broker Architecture) framework [6] for their distributed object model, we needed to distribute the data across a CORBA object by connecting our CORBA client with the sensor and pushing the data to the Rover object running on CORBA ORB (Object Request Broker).

The architecture of the data distribution to the ERA server is shown in Figure 5. The ERA has a server called Executor to accept the data and store it in local memory for a finite period of time. We need to push the data at a rate that refreshes the data before the memory times out.

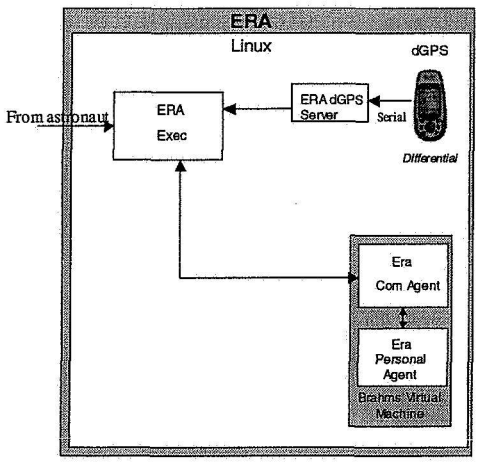
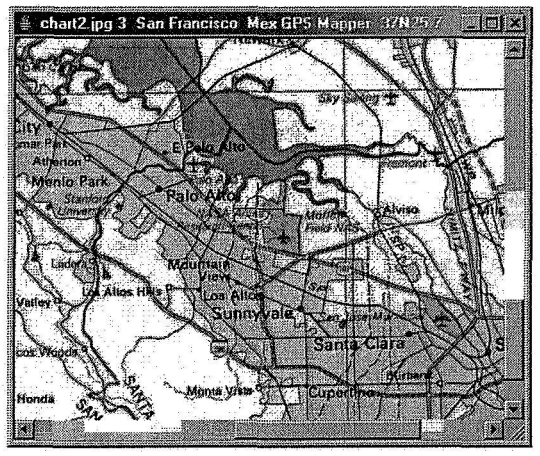


Figure 5. The ERA CORBA server.

The subscribed client will receive the stream of data by intercepting the message listener. An example of this type of client is the Rover monitor. It can show the movement of the rover on a map in the real time. The figure 6 shows the monitor screen from a pre-field test at Moffett Field, CA.



The circle with cross is the moving cursor showing the rover location by interpreting the coordinates received from subscribing to the GPS topic.

4. RELIABLE AND ROBUST SURFACE COMMUNICATIONS

The architecture we have described has the capability of providing reliability and robustness during short outages. However, some issues are not addressed directly by our architecture itself. Of most concern to us are longer duration network outages in severe environments.
The low power output and delay sensitive protocols of 802.11, especially when used over multi-hop long haul point to point circuits, is prone to fading. Add the problems caused when equipment moves out of line of sight or is subject to routing difficulties as equipment moves and long duration outages will occur.

We have the tested our methodology during simulation tests at the Mars Desert Research Station in Utah in the spring of every year from 2002-2005. These "Mobile Agent" expeditions tested interactions between astronauts and robotic assistants. They were a collaborative effort between several NASA Centers. We used

802.11b to communicate between astronaut, robot and the base station. Some of these links were over several kilometers. Repeaters were put in temporary locations and subject to wind and rain. A satellite link sends the data from base camp to researchers at their home institutions.

As the astronauts and robots move they come in and out of wireless signal coverage. Under such conditions, with short-term outages the norm, data distribution becomes unreliable. Connections can be lost either between the astronaut and the JMS server, or from the ERA robot to the JMS server (or both). Any of these data interruptions will keep the sensor data from reaching the proper destination.

We devised a software workaround to mitigate this impairment.

We established a retry loop that continues to test the path until it has recovered. To prevent the retries from impacting data collection from the sensor, tying up CPU usage, or other resources we set a counter to wait a period of time (in seconds) before the connection is retried. Of course the data is stored until connectivity is established again.

The logic path used for implementing timeout loops in the subscriber model is as follows:

```
In the processing of data loop
If (reconnectCounter==0)
        DoReconnect();
        Publish();
Else
        ReconnectCounter--;
Endif

When (ConnectionException)
        SetReconnectCounter;
 End loop
```

Another important issue is that when SerialConnection class is used to acquire data from the COM port, it can't be interrupted by other tasks, as the data flow received is a continuous real-time stream. To prevent interruption we provide separate threads for data distribution and the SerialConnection class that is dedicated to acquiring and storing the data in memory for further processing.

Lessons Learned:

Field tests took place at the Mars Desert Research Station (MDRS), in an isolated area in Utah. A satellite link connected to the NASA Research and Education Network (NREN) through Glen Research Center in Ohio. Astronauts (fully suited) are paired with robotic assistants. They communicate with each other over wireless links and the robot responds to voice commands from the astronaut. The robot contains a mobile WLAN repeater. The activities are monitored from a base camp several Kilometers away. Additional repeaters are on ATVs nearby and on hilltops.

Experiences from the 2003 Mobile Agents field season:

There was an attempt to integrate the equipment (robots, wireless, etc.) and software from the many groups from ARC and JSC before the outing but lack of time and travel resources from the participants forced us to do a very superficial integration effort. The various groups were also in development stage until just before the deployment.

It was almost expected that we would experience problems and would have to redesign in the field.

The first week of the two-week period was marked by problems related to the substantial delay of the receipt of packets as well as frequent connectivity drops. The software responsible for GPS location service failed to correctly establish coordinates when the GPS real time data from the backpack were delayed. This made testing astronaut to robot to operations center voice recognition and command processing impossible. We had additional problems with the backpack

computers overheating and routing on our multi-hop wireless system.
We took several steps that mitigated the problems.

- We implemented an NTP (RFC 1305) Timeserver to timestamp all GPS and biosensor data. This made it possible to correctly correlate the location of the astronaut or the robot with a time series.
- We moved the sensor message to a computer that had less network traffic and less other processing
- We fully implemented our publish subscribe middleware

These measures and further tuning of the wireless infrastructure to fix some routing problems and adding an additional fan to the backpack led to several successful simulations.

Experiences during the 2003 field season were more positive (even with some rather severe weather and dust storms). Before deployment we redesigned the astronaut backpack adding better ventilation to accommodate an updated rugged laptop computer.
The laptop had more memory and a mobile Pentium processor and a wider operating temperature range.
The WLAN complexity (mobile access points on ATV and mobile robots with repeater sites on far away) was reduced and multi-pathing and channel overlap were reduced. Routing between the elements was rationalized to prevent loops. This led to higher bandwidth and better throughput.
We tested our distributed sensor architecture by placing the JMS server on different computers on the system. All worked as designed although response was still slow but tolerable.

The 2005 season built on the successes of 2003. We optimized our software and redesigned the client API. The client is the "subscriber" to our sensor data message server. One improvement was to have the software "sleep" when no messages are in the queues and awake when they are available. This cut the CPU and memory requirements substantially. The Sensor data have frequency ranges of from 1-10 HZ the "sleep" cycle left the resources for the other software tasks. Improvements were also made in the voice loop software, the mobile agent software itself and the voice recognition software. These upgraded produced improved performance and seamless transition in and out of wireless coverage.

6. CONCLUSION

The field tests and experiments show that the distributed components model that utilized JMS architecture is very suitable for real time sensor data distribution. It produces reliable and robust data stream to multiple clients in real time. The publish subscriber model is very scalable even for processing data from many sensors. For publishing data from multiple sensors,
message beans, and topics, can be easily created for each occurrence of a sensor.
Longer duration network outages as are common in the field can be easily mitigated by simple software modifications.
These techniques have relevance to the situation where multiple assets are distributed on the ground and in orbit and sensor and other data are to be distributed to multiple consumers locally or on Earth.

REFERENCES

[1] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec,"The Many Faces of Publish/Subscribe", *ACM Computing Surveys*, Volume 35, Issue 2, pp 114-131, June 2003.

[2] L. Garces-Erice and E.W. Biersack and P. Felber and K.W. Ross and G. Urvoy-Keller. "Hierarchical Peer-to-Peer Systems", *Parallel Processing Letters*, Volume 13, Issue 4, December 2003.

[3] Jameela Al-Jaroodi, Nader Mohamed, Hong Jiang, and David Swanson, "Middleware Infrastructure for Parallel and Distributed Programming Models in Heterogeneous

Systems", *IEEE Transactions On Parallel and Distributed Systems,* Vol. 14, No. 11, November 2003.

[4] Angelo Corsaro, and Douglas C. Schmidt, "The Design and Performance of Real-Time Java Middleware", *IEEE Transactions On Parallel and Distributed Systems,* pp1155-1167, Vol. 14, No. 11, November 2003.

[5] Charles Zhang and Hans-Arno Jacobsen, "Refactoring Middleware with Aspects", *IEEE Transactions On Parallel and Distributed Systems,* pp1058-1073, Vol. 14, No. 11, November 2003.

[6] Victor Fay-Wolfe, Lisa C. DiPippo, Gregory Cooper, Russell Johnston, Peter Kortmann, and Bhavani Thuraisingham, "Real_Time CORBA", *IEEE Transactions On Parallel and Distributed Systems,* Vol. 11, No. 10, October 2000.

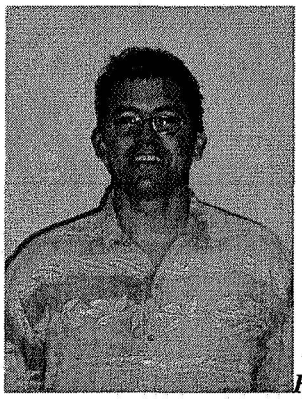
[7] Wenbing Zhao, Louise E. Moser, and P. Michael Melliar-Smith, "Unification of Transactions and Replication in Three-Tier Architectures Based on CORBA", IEEE Transactions on Dependable and Secure Computing, pp 14- 23, Vol. 2, No. 1, January-March 2005.

Charles Lee, employed by SAIC, is the Technical Lead on Mobile Agents project at NASA Ames Research Center. He holds a Ph.D. in systems engineering and computer science from Oakland University, in Rochester, Michigan. Completed research projects includes robust GPS switchboard on-demand services that provide GPS information with awareness of loss and the ability to regain wireless network connections, and a store and forward architecture to maintain data continuity in the event of network connection loss. In addition, Charles Lee developed distributed agents that serve sensor information through a publish and subscribe architecture in heterogeneous computer environments, and a mapping and planning system that provides location and orientation of mobile rovers and astronauts on topographic maps for navigation planning and real time monitoring. Other work includes join development of custom software to provide access to avionics data for Advanced Diagnostics System (ADS) and System Health Monitoring applications, and collection and organization of International Space Station (ISS) data sets by fault scenario, along with liaison with ADS developers and users in the design of data interfaces, user interfaces and tools relevant to ADS on ISS. He developed the first version of Caution and Warning cube visualization software that handles the command and data handling events for fault detection, and then was became Strider application.

John Ossenfort is a network/systems administrator at the NASA Ames Research Center, specializing in wireless communications. He has been acting systems administrator for the IMT Lab and the MEX testbed for the past three years and has accompanied the Mobile Agents team on four field simulations in the Arizona/Utah desert, assisting in all aspects of wireless network design, deployment, troubleshooting and maintenance. He is currently working on a wide array of projects, from Martian subsurface drilling simulations to Integrated Systems Health Management for the International Space Station. John has a dual BA degree in Anthropology and East Asian Studies from Washington University in St. Louis. He currently resides in Los Gatos, CA.

Ed Walker is a hardware and networking specialist with the Intelligent Mobile Technologies (IMT) team at NASA Ames Research Center. He is a member of the team developing and testing the capabilities of the Mobile EXploration (MEX) testbed. The MEX System is a model for human planetary exploration that incorporates rugged computing, long-

range wireless communication and mobility in support of planetary explorers. He graduated Foothill College in Los Altos, California with AS degrees in Data Communication & Network Management as well as Enterprise Networking.



Hugo A. Notario: Since 1994, I have been heavely involved in software and harware architecture.
I accomplished my bachelor degree in Industrial Engineering with an option of electronics in 1988. Also, in 1997 I achieved an A.A. in Computer Service Technology.
In the past 5 years I have earned several technical degrees in software and networking.
I am currently working on my second bachelor in Computer Science. I joined NASA/Ames
around two years ago in which I have been involved in Sensor Data Distribution in
Planetary Exploration.