Advanced Middleware for Space Plug-and-play Avionics (SPA) Architecture Standardization and Rapid Systems Integration

Presented by:
Grant K. Holcomb
ATK Space Systems
grant.holcomb@atk.com

August 12, 2009

Outline

- **Operationally Responsive Space (ORS) Mission**
- **Space Plug-and-play Avionics (SPA) Definition**
- **ORS/SPA Software Requirements**
- **Problem Statement**
- **Solution Summary**
- **Middleware Defined**
- **Middleware Characteristics**
  - Metadata Defined
  - “Signals,” “Nodes,” and “Attributes” Defined
  - Node Functionality
  - Journaling (Integrated Feedback Loop)
  - Advanced Data Modeling
- **Middleware Usage Example**
- **Conclusion – Middleware Advantages**

Operationally Responsive Space (ORS) Mission

The unclassified April 17, 2007, report to the Congressional Defense Committees entitled "Plan for Operationally Responsive Space" presents the ORS mission:

- **Develop** and deploy selected responsive Intelligence, Surveillance, and Reconnaissance (ISR) and communication **capabilities to augment,** surge, and/or reconstitute **existing assets**
- **Enhance space situational awareness** to characterize and enable better protection of on-orbit assets
- **Provide rapid launch capabilities** along with the command and control capabilities; responsive ground elements; and tactics, techniques, and procedures to ensure responsive space operations
- Integrate capabilities vertically and horizontally to **provide decision makers** with access to **the information they need to execute their missions**

Space Plug-and-play Avionics (SPA) Definition

The American Institute of Aeronautics and Astronautics (AIAA) July 15, 2008, draft guide entitled "Standards Development Guidebook for Space Plug and Play Avionics" defines Plug-and-Play (PnP) avionics:

"In the vision of an adaptive avionics framework, **a new discipline involving machine-negotiated interfaces** can be used to permit the elements of a complex system to transparently contribute information that accelerates the integration process by reducing or eliminating error-prone human interpretation. Such adaptive avionic interfaces, **by process of electronic self-configuration / self-organization**, could allow for rapid space vehicle construction. The placement of sensors and actuators on the spacecraft would not be restricted to specific, predetermined locations. In the terrestrial electronics industry, **this capability is called "Plug-and-Play (PnP)."** The SPA approach fully supports an à la carte method of constructing arbitrarily complex arrangements of virtually any sensor or actuator type. This behavior makes the network not only easy to expand and modify, but also makes it robust to component failure from either natural causes or from deliberate attack."

ORS/SPA Software Requirements

As summarized below, a detailed review of the AIAA SPA related documentation establishes requirements for the software based capabilities essential to support the ORS mission:

- Fault Tolerance
- Computing Topology Independence
- Common Data Understanding
- Network Topology Independence
- Operating System Independence
- Autonomous Processing
- Automation of PnP for both hardware and software components

Problem Statement

- In order to facilitate their automated integration into a coherent solution, all sub-components of a satellite system (hardware or software) need to possess a common interface that can be discovered on demand from any location on a network

- There is no network-centric platform independent standard that enables the automated discovery of the interfaces of a system's sub-components—*that has no dependency on the network location of the sub-components or the order in which they become available to the network*

- There is no network-centric platform independent standard that enables the automated instantiation, monitoring, and control of network distributed software processes

Solution Summary

A library of software building blocks (middleware) can be assembled and made available to all vendors, which can be used to establish a standard suite of mechanisms that:

- Facilitate the rapid interface definition and implementation for any hardware or software sub-component designed for use in a Space SPA architecture
- Enable the automated discovery of any hardware or software SPA sub-component interface from any location on a network regardless of the order the sub-component becomes available to the system
- Enable the automated instantiation, monitoring, and control of network distributed software processes
- Will ultimately meet or exceed all ORS/SPA software requirements

Middleware Defined

- A library of software building blocks that a programmer uses to develop system software
- The layer between the hardware (processors, routers, sensor, actuators, storage devices, etc.) and all system specific software
- An abstraction layer that encapsulates hardware or system software processes that may change over a long life cycle
- If the roads in a city were a system and vehicles were the data, the middleware would be a citywide interconnected traffic light system that ensures all traffic flows smoothly and efficiently

Middleware Characteristics

The middleware must possess the following characteristics to facilitate PnP and establish a standard for fully supporting ORS/SPA small satellite mission requirements:

- **Metadata Based** – Metadata provides a proven mechanism for creating non-proprietary data structures and software that has no dependency on any computing platform or operating system while also facilitating the efficient implementation of highly-automated processes
- **Discoverable Variables, Methods, and Objects** – The basic building blocks of all software provided by the middleware must be remotely discoverable from anywhere on a network and include a mechanism that efficiently notifies remote processes when any changes occur
- **Feedback Loop** – Critical for robust autonomous solutions, the middleware building blocks must include an integrated feedback loop
- **Data Modeling** – To accurately model complex processes the middleware building blocks must support the dynamic establishment of complex data flow hierarchies across a network

Metadata Defined

- Data that adds context and meaning to data
- Abstraction layer for data, which eliminates the proprietary relationship between data structures and source code
- The Extensible Markup Language (XML) is the primary metadata format
- Example of XML formatted metadata used to define a name:

```
<Name>
        <First Name>Maurice</First Name>
        <Last Name>Martin</Last Name>
</Name>
```

- Software developers on any computing platform using any programming language can create applications that can read XML formatted ASCII text files in order to utilize recorded data
- Metadata is essential for establishing long life cycle, technology independent, and highly automated solutions

“Signals,” “Nodes,” and “Attributes” Defined

- **Signal** – The middleware must provide a common efficient mechanism for notifying remote network listeners that a change has occurred within a variable or object. The mechanism is called signaling (also called an event or messaging). The format of the signal should be metadata based (XML formatted ASCII)

- **Node** – A Node is a reusable software building block (object in memory) that implements the common signaling mechanisms. Any software object definition should be able to inherit this Node functionality. The Node should also include a mechanism that “publishes” the format of the respective object’s internal data structure on demand

- **Attribute** – An Attribute is a Node used to define a single atomic variable (smallest sized data element managed by software). An Attribute is used to replace the atomic variables defined by the typical programming language. This makes the smallest building block of any system visible from any location on a network. The middleware should include predefined Attributes necessary to replace current variable types

Node Functionality

Inherent and Supported Functionality	C++ Object	Middleware C++ Object
Remote Discovery	NO	YES
Remote Instantiation	NO	YES
Remote Modification	NO	YES
Remote Change Notification	NO	YES
Metadata Formatted Persistence & Signaling	NO	YES
Hierarchical Data Modeling	NO	YES
Distributed Data Modeling	NO	YES
Redundant Data Modeling	NO	YES

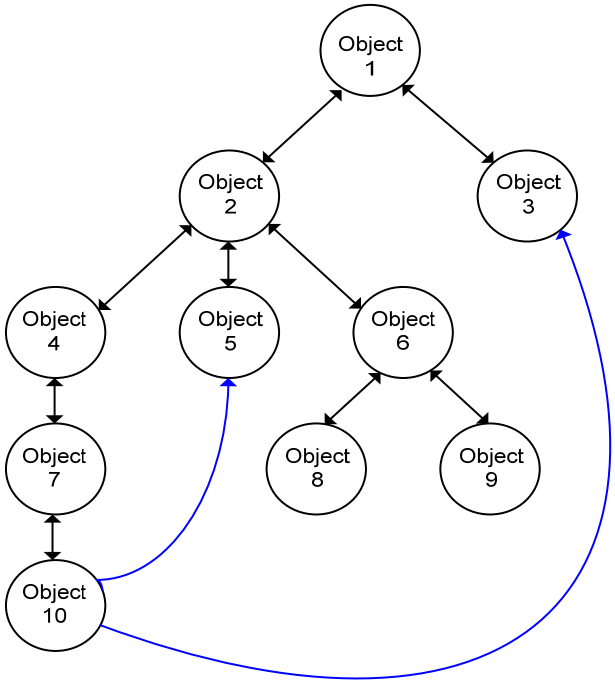
Journaling (Integrated Feedback Loop)

As a result of the common signaling (event or messaging) mechanism, the metadata formatted signals sent by Nodes can be recorded and analyzed in a highly automated fashion. In the middleware, a recording of these signals is called a "Journal." A Journal represents a system-wide feedback loop that facilitates the following:

- **Trend Analysis and Security** – Automated journal analysis can surface unanticipated trends or patterns, particularly by users, which can drive changes to security policy
- **Performance Analysis** – Process timing/frequency is measurable
- **Systems Reliability** – Failures can be identified and responded to
- **Simulations** – Journals can be modified and played back into a system representing an integrated simulation capability (symmetrical)

Advanced Data Modeling

Through the Node base class, all middleware objects share a common mechanism for building complex data models (data structures, protocols, process flow, workflow, etc.)



- A network-centric, recursive, thread-safe implementation of the Node base class supports the rapid creation of lists, trees, graphs, and cyclic graphs of objects of any type, purpose, combination, complexity, and *network distribution*
- Combined with the metadata formatted signaling, any model can be simulated to any degree as required for design test, verification, and validation (integrated simulation and QA)

Middleware Usage Example

C++ Object

```
struct Person {
    Person();

    std::string     mFirstName;
    std::string     mLastName;
    int             mAge;
    bool            mEmployee;
};
```

Middleware C++ Object

```
struct Person : public Node {
    Person();

    StringValue     mFirstName;
    StringValue     mLastName;
    IntegerValue    mAge;
    BooleanValue    mEmployee;
};
```

Conclusion – Middleware Advantages

- Facilitates rapid integration of sub-components, makes dissimilar technologies interoperable, and normalizes data and control flow
- Reduces both the number of lines of source code and the time it takes to design, implement, and test a deliverable
- Hardware and software processes can change continuously over time to support improvements and new functionality on demand
- Lowers the barrier of entry for all vendors while providing the freedom to innovate and produce solutions they own and sell, knowing these solutions will reliably integrate into future satellites
- Establishes economic incentives and a competitive technology base that accelerates innovation and reliability while rapidly lowering sub-component costs
- Has the potential to (1) establish industry wide standardization, (2) accelerate satellite integration, (3) increase reliability and survivability of satellites, and (4) measurably and cost effectively support ORS goals.