

DEEP: Dallas EEPROM Equipment Profile for Rapid Integration and Automatic System Modeling

*Forrest Rogers-Marcovitz
Aerospace Systems Laboratory, Washington University in Saint Louis*

*Phelps Williams
Robotic Systems Laboratory, Santa Clara University*

Abstract

The Responsive Space Initiative, the ability for mission-specific payloads and support systems to be rapidly integrated within a short period, is a goal within the spacecraft community. However, as components are added to the spacecraft, the complex interactions between subsystems must be noted and, if possible, modeled. This process is extremely time consuming and if not done properly, can be a major contributor to spacecraft failure. A new paradigm is needed for Rapid Integration and System Modeling.

At the 2006 Conference on Small Satellites in Logan, Utah, Washington University and Santa Clara University demonstrated Rapid Integration and Testing by functionally combining their respective satellites, Akoya and Onyx. Both vehicles were connected via a common power and data wiring harness, allowing one spacecraft to operate any device on either vehicle. Despite possessing minimal prior knowledge of the other school's subsystems, functional integration was achieved in less than thirty minutes. This was accomplished by using a distributed computing architecture with a standardized interface and communication protocol; this architecture allows each subsystem to be developed separately and rapidly integrated into the spacecraft.

The Dallas EEPROM Equipment Profile (DEEP) Architecture extends this standardized bus to include improved support for rapid integration and system modeling. DEEP is a protocol standard using the Maxim/Dallas 1-Wire bus, which allows for low level control and monitoring of the spacecraft using commercial-off-the-shelf devices including memory and

sensor devices. DEEP specifies a standard with which a representation of subsystem functionality is encoded within the subsystem itself, allowing for the creation of a satellite-wide model paralleling the physical integration of the spacecraft. This allows for the creation of a stockpile of flight DEEP enabled subsystems, ready to be rapidly composed into a functional spacecraft. Each subsystem includes a subsystem model, with parameters such as thermal and power characteristics, allowing an anomaly management system to identify off-nominal conditions through model-based reasoning. Additional functionality includes automated ground operations and ground integration and test software generation, standard command planning, resource allocation, and other areas of command and control.

DEEP is currently being developed at Santa Clara University and Washington University in Saint Louis as part of the University Nanosatellite competition operated by the Air Force Research Laboratory. This paper describes the current success of both universities with rapid integration, current development of the DEEP architecture, and future advances regarding responsive space.

Introduction

Current spacecraft integration is a major barrier to the responsive space initiative. The goal is to reduce integration to under a week. Historically, spacecraft subsystems have been developed in isolation from other subsystems requiring lengthy Interface Control Documents (ICD) to interact with the rest of the spacecraft. Complex

spacecraft-level system modeling is needed so that all subsystems interact correctly. As subsystems are added, new ICDs need to be written, and modification must be made to all connecting subsystems, causing a cascading mound of paperwork. Many people are involved in developing, maintaining, verifying, implementing, and cross-checking ICDs, which consumes large quantities of time and resources. In addition to the ICD modifications, the intricate subsystem interactions must be changed in the system model.

System level changes are needed to meet the challenges of a responsive design and integration timeline; the goal is to reduce integration time to under a week. Modular subsystem design allows components to be designed separately from one-another and to be quickly swapped with similar components that meet mission-specific requirements. Standards are needed for modular design; these standards must be at the physical interface, wiring connection, and data protocol levels. One successful example is the Space Plug-and-Play Avionics (SPA) standard being developed by the Air Force Research Laboratory (AFRL). The AFRL, along with other organizations, is currently finalizing an AIAA published SPA standard [1] that has been used on the Lockheed Martin HexPak Testbed [2] and is currently being implemented as part of the AFRL Responsive Space Testbed effort.

As part of the SPA standard, the Satellite Data Model (SDM) provides unified plug and play mechanisms for applications to coordinate and share data, resources, and services without a prior knowledge of the physical location and properties of a the spacecraft components. XML Transducer Electronic Data Sheets (XTEDS), included with each component, provide characteristics of spacecraft hardware and software, and allows devices and applications to register to the Data manager. A Common Data Dictionary gives semantic or contextual meaning to the SDM/XTEDS elements. The Satellite Data Model along with SPA physical standards eliminates the need for the ICDs that consume time and money.

This paper focuses on a similar design architecture created to meet the needs of university-class satellites. The purpose of

university-class missions is to train students in the design, integration and operation of spacecraft, and this is accomplished by giving students direct control over the progress of the program [3]. They often use a modular design which reuses heritage subsystem designs. Typically, the missions are constrained to be small, low-cost satellites with short design timelines, allowing them to take higher risks with potentially larger pay-offs.

The Dallas EEPROM Equipment Profile (DEEP) Architecture was developed to meet the needs of university-class satellites. It offers a low-power, non-complex, inexpensive means of creating a plug and play system by using commercially-off-the-self components. While DEEP is designed specifically for university-class satellites, many of the DEEP concepts can be applied in more complex systems.

Past Rapid Integration and Testing Success

The Emerald Protocol suite has enabled past successes in rapid integration and testing. Excellent examples include parallel integration of two complete satellites within a two week time span and a 30 minute subsystem integration demonstration at the 2006 Small Satellite Conference. These events have shown the power of standardization being used in the university environment. Both will be expanded on later in this paper.

Santa Clara University's (SCU) Robotic Systems Laboratory and the Aerospace Systems Laboratory at Washington University in St. Louis (WashU) have had many successes in past Rapid Integration and Testing (RIT) demonstrations. SCU has developed the EMERALD and ONYX satellites. ONYX is designed to support a unique multispectral imager and provide a testbed for model based anomaly management through the entire space system. WashU has built the Akoya and Bandit satellites. Bandit's mission is to flight-test proximity operations technologies, including docking, safe navigation within 5 m of a target vehicle, on-orbit charging and image-based navigation. Akoya is the host vehicle that provides docking, recharging and ground communication capabilities.

Each satellite uses a distributive computing architecture where each subsystem is controlled by an individual microprocessor. Inter-subsystem communication uses the Emerald Protocol Suite [4][7][8] which extends existing I²C and Dallas 1-wire data protocols. The University of Texas-Austin also uses the Emerald Protocol Suite on their FASTRAC satellite. Figure 1 shows an example of this architecture on the Akoya satellite.

All subsystems send and receive commands based on the Emerald Data Protocol (EDP). The Inter-Integrated Circuit (I²C) provides the physical layer responsible for transmitting messages. To expedite message decoding, header information (source and destination address etc.) is included with every packet. A subsystem receives a packet bearing its address and deciphers the command. The command is interpreted and requisite action taken. .

Distributed computing architectures offer numerous advantages in the development of complex devices and systems. These advantages include well-defined interfaces, flexible

composition, streamlined integration, straightforward function-structure mappings, standardized components, incremental testing, and other benefits. The distributive processing architecture along with the Emerald Data Protocol allows subsystems to be developed separately and rapidly integrated into the spacecraft upon completion. Because of this architecture, the three schools saw three improvements: accelerated integration and training of new students; rapid modifications of existing systems; and school-wide collaboration among robotics projects [5].

At the 2006 Conference on Small Satellites in Logan, Utah, Washington University and Santa Clara University demonstrated Rapid Integration and Test by functionally combining Akoya and ONYX. Both vehicles were connected via a common power and data wiring harness, allowing one spacecraft to operate any device on either vehicle. Despite possessing minimal prior knowledge of the other school's subsystems, functional integration was achieved in less than thirty minutes.

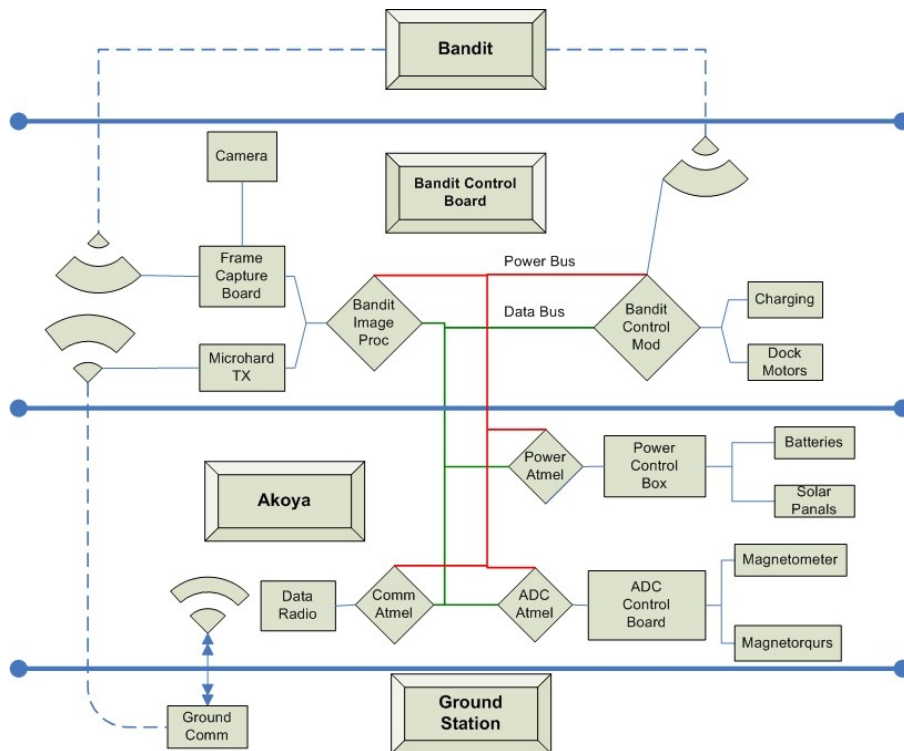


Figure 1 - Akoya's Distributive Computing Architecture

In March 2007, SCU and WashU completed parallel flight integration processes in preparation for the AFRL/AIAA University Nanosat-4 competition Flight Competition Review (FCR). Both schools completed integration in two weeks. At FCR for the AFRL/AIAA competition, WashU and SCU repeated the demonstration from the 2006 Conference on Small Satellites in slightly over 2 minutes. This was possible only through modular design and use of the Emerald Data Protocol.

Motivation

Although the integration was completed in less than thirty minutes, the process could have gone smoother if additional tasks were automated. More specifically, all interactions between subsystems must be coded before integration because each subsystem is not necessarily aware of all other subsystems and their capabilities. DEEP is an approach to remove this dependency.

An additional challenge each system has is a hard coded address which limits the extendibility of the system in a number of ways. First, there could be an address conflict during integration or, if an address is changed, any interacting systems must be recompiled with the new address. Second, in order for one system to talk to another, the commands accepted and replies given by the second system must be known in advance. Third, there is no way for a new system to be announced when it joins the network. For a basic plug-and-play network, each device must: announce its presence and

capabilities, discover who else is a member, and selectively interact with other members of the federation. DEEP was developed to eliminate pre-integration coordination and create subsystem-level embedded operational intelligence.

DEEP Concept

DEEP fulfills two particular objectives. DEEP enables rapid spacecraft assembly and assists in generating an entire spacecraft model. The implementation of common services such as turn subsystem X on or off may functionally consist of a complex sequence of events. Through abstraction of this functionality to a higher level, “turn subsystem X on” is associated with that complex sequence of events. This abstraction is held within the subsystem it is specific to. Each subsystem can now share this information with other subsystems on the bus and awareness of available services is achieved through the sharing of this information. Another way to think about this is as a functional model of the subsystem embedded in the subsystem itself.

While such functional models may not be required for rapid assembly, this information allows a complete model of the integrated system to be constructed, completing the second DEEP objective. A variety of operationally relevant tasks utilize this model such as automated ground-station software generation and model based anomaly management. Additionally, the integration and test process can make use of this model information in the same manner.

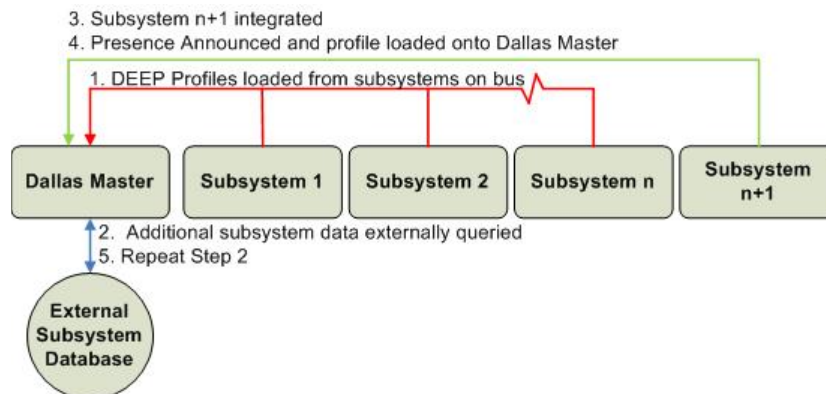


Figure 2 - DEEP Flow Diagram

Regardless of hardware implementation, three elements must be present for the DEEP concept to work:

- A standardized communications bus
- A common memory device directly accessible on this bus
- A common memory structure (hardware profile) across all memory devices

Provided this structure exists, information is held dynamically, updating to reflect changes in bus state (Figure 2). This is true because any connected node can temporarily become master of the bus and obtain the profiles of all other nodes on the bus, parallel maps of the bus can be maintained at each network node, and each hardware element has its own model of the services available on the bus.

Many interesting variations arise from this concept. Suppose a single data bus wire harness connects two trays of subsystems in a spacecraft. If this connection is broken, subsystems on both trays would update to maintain valid lists of available services, potentially permitting continued operation despite a critically damaged state.

Utilizing this framework, a functional map of the bus is generated. For example, suppose two communication systems were integrated on a bus. The first is designed for general bus telemetry gathering tasks and transmitting this data to the ground at a low data rate. The second system is designed for high data throughput but has high pointing requirements and is designed to transmit payload data to communication satellites in geostationary orbits (GEO). The model of these two hardware elements would capture these unique design traits. The system model would allow for unintended configurations to be automatically recognized. If the first communication system ceases to function, a model of the system would instantly present a secondary route for the bus telemetry formerly reported by the first system. This action might occur through the existing GEO link or through an attitude adjustment for a direct ground link.

This capability can be used to augment ground test software. For the most basic degree of

ability, the software would only need to be able to recognize and exercise advertised hardware functionality. Presumably, the operational capacity of the spacecraft will be a specifically sequenced subset of these advertised functions. Ground operations software could once again take advantage of this framework and automate much of its creation as well.

DEEP additionally provides the flexibility for transient services to exist in a hardware system. Interesting examples of this could be other DEEP enabled spacecraft and ground systems. For example, suppose a satellite in a low earth orbit (LEO) comes into communications contact with a ground-station. DEEP enables the satellite to recognize that the services of the ground segment are now available by sharing its own profiles with those available on the ground. If the ground-station provides the ability to synchronize a system clock, the satellite recognizes the availability of this service and can utilize it. The same functionality that enables rapid integration provides an opportunity to transcend the traditional system definition. An unmanned aerial vehicle (UAV) utilizing live imagery from a LEO satellite for navigation purposes does not need to understand these distinctions. It can simply recognize an additional navigational aid which happens to be available on a cyclical basis, due to orbit.

DEEP was initially conceived without knowledge of standards such as IEEE 1451, the Transducer Electronic Data Sheet. In retrospect, significant similarities are present between the two. However, critical differences exist: DEEP does not limit itself to a standard definition of Transducers as IEEE 1451 [6][9] specifies. DEEP leaves this open to the system designer. Any hardware element with an interface to the standard bus and a memory device that is accessible on this bus can be DEEP enabled. This means any component of the spacecraft could be DEEP enabled from an Attitude Determination and Control subsystem to the tray it is affixed to.

An Example of DEEP Implementation

A demonstration implementation was constructed to show the potential of DEEP at SCU. Due to time constraints, sacrifices were made in this implementation and the full potential of the technology was not completely brought to fruition. Despite this, the results were impressive and have inspired continued development of the DEEP concept.



Figure 3 – AVR-Sat Hardware

As mentioned previously, DEEP requires a standardized communications bus, a common memory device directly accessible on this bus, and a common memory structure (hardware profile) across all memory devices. It is important to note this implementation only requires conformity with the communications bus and memory device. The subsystem designer is left the flexibility to define the subsystem beyond this requirement. As a demonstration of this functionality, 4Kbit EEPROM devices were added to an existing avionics package developed at Santa Clara

University. The avionics hardware, named AVR-Sat modules (Figure 3) were designed around the Emerald Protocol suite, a standard communications and interface definition also developed in the University environment.

Dallas 1-Wire devices were chosen because the support framework was already well implemented on existing hardware. Additionally, low cost, high availability, and usage simplicity made this hardware a suitable choice.

Instead of maintaining a network image at each node, a centralized approach was taken. This was a key factor in simplifying implementation. A subsystem dubbed the Dallas Master was created to perform this role. This subsystem was responsible for periodically polling the bus in order to update the internal map of subsystem connectivity. The profile of each DEEP-enabled device was stored in this device, which permitted the Dallas Master to service requests such as “Turn the Communications System On.” The requesting subsystem element does not need to know any details specific to this operation in order to command its occurrence.

The device profile stored in the small EEPROM device onboard each hardware element is populated as seen in Figure 4.

Following a generic header, a DEEP Version value is present, enabling support for future modifications. The I²C address, the primary address needed for commanding of the hardware specified by the Emerald Protocol suite, is listed. An important future modification would be to design around the static nature of embedding an address in the DEEP profile. Functionality

Byte	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
0	D	E	E	P	DEEP Version #		I ² C Addr	Dev Count
0008	Subsystem Name							
0010	Hardware Version Number							
0018	Hardware Configuration Number				Hardware Serial Number			
0020	Timestamp of Programming				Header CRC16		Data CRC16	
0028	Dallas Name String 0							Type
0030	Dallas ID 0							
0038	Dallas Name String ...							Type
0040	Dallas ID ...							

Figure 4 - DEEP Hardware Profile

similar to Dynamic Host Configuration Protocol (DHCP) would free the bus of this constraint but the complexity of this issue made it unfeasible for this sample implementation. Another simplification made in this approach was to limit all of the control elements to be Dallas 1-Wire devices. The device count field in the profile captures the count of Dallas devices that are available on the hardware this profile is describing. An 8-byte ASCII encoded name follows. This embeds a human readable identification tag of the device which is generally helpful during development.

The Hardware Version number captures a specific hardware design revision. This will allow for a higher fidelity model to be referenced for information specific to this hardware revision. The excessive bit length of this value enables future expansion of the meaning of this value. The Hardware Configuration Number provides numerically referenced configuration bins for the designer to utilize. For example, if there is a certain grounding configuration that is being utilized on a collection of hardware, that distinction could be captured in this field. It is important to note that that type of detail is not specifically held within that value; an external reference is assumed to exist to maintain this contextual information. The implementation of this external reference, be it an XML document, SQL database, or similar technology, is not specified and should be chosen based on requirements external to DEEP. The Hardware Serial number is a unique identifier of each hardware element maintaining the same Subsystem Names. This value is not necessarily unique between Subsystems and, similar to the other fields, does not need to be globally coordinated.

Following a typical UNIX style timestamp, two separate CRC check-sums are included. The intention here is to decouple data held within the following payload section from that of the header. The payload section maintains Dallas 1-Wire addresses, an ASCII description of their functions, and a more general type identifier. This type identifier allows an interpreting subsystem to immediately classify that device and automatically generate usage guidelines. Very specific categories were created for this sample implementation, (Table 1). Should

corruption occur in the payload component, it can still be retrieved from a remote mirror of the data similar to the other specifics of the hardware which, under this implementation, need to be externally referenced anyway.

Device Type Table	
Value	Description
00	Latch-up V1 5V Control
01	Latch-up V2 12V Control
02	Latch-up V2 5V Alt
03	Board Temperature
04	INFOsw (Non-DEEP Memory)
05	Temperature 5V Regulator
06	Temperature 12V Regulator
07	Current Solar Panels 1-4
08	Current Solar Panels 5-8
09	Power Regulated
10	Power Batteries 1-2
11	Power Unregulated
12	Temperature Battery 1
13	Temperature Battery 2

Table 1 - Device Type Classification

Five hardware elements configured with a DEEP Profile including a Dallas Master subsystem compose the demonstration, (Figure 5). Any of the five hardware elements could be integrated and removed from the bus and the internal map of the configuration onboard the Dallas Master would immediately recognize this change. The Dallas Master could accept generic commands by specifying only a general description of the hardware, such as communications system, and the type of device to modify along with the status or state desired.



Figure 5 - DEEP Demonstration Unit

Conclusion

The DEEP architecture builds on past RIT successes at both Washington University and Santa Clara University. DEEP provides dynamic system modeling while decreasing integration and testing time. Inherent in this idea is the notion of embedding a functional model of the hardware within itself.

While a demonstration implementation has been constructed, much future work exists. DEEP will be extended to allow dynamic addressing with more sensor types added. Sensor operating limits will allow the use of Anomaly management algorithms. Embedded command lists with response types will allow for subsystem interaction and dynamic ground control and modeling.

DEEP will be integrated into the existing Akoya and ONYX satellites and used on future satellites built at the two labs. Akoya and ONYX have dedicated payload space for the other school. Because the payloads adhere to the Emerald and DEEP protocols, no a priori knowledge will be needed for integration. While DEEP is designed specifically for university-class satellites, much of its design implementation can be applied to more complex spacecraft systems.

Acknowledgments

The following people helped, developed and supported the DEEP concept: Bryan Palmintier, Christopher Kitts, Michael Swartwout, and Lane Haury.

This work has been sponsored through a variety of sources to include the National Science Foundation via Grant No. EIA0079815 (development of the distributed command and data handling architecture at SCU), the USAF/AFRL via grant FA9550-05-1-0249, and the SCU Technology Steering Committee via grant TSC209; any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the USAF, or of Santa Clara University.

References

[1] AIAA SPA Committee on Standards, Draft Standards: Development Guidebook for Space Plug and Play Avionics

[2] Orogo, C.D, Flaggs, D.L., Enoch, M., "Javabased Plug-N-Play (Flight) Control Systems for Responsive Spacecraft", Paper No. RS4 2006-6002, 4th Responsive Space Conference, Los Angeles, CA, Apr. 2006.th

[3] Swartwout, Michael, "Twenty (plus) Years of University-Class Spacecraft: A Review of What Was, An Understanding of What Is, And a Look at What Should Be Next," *Proceedings of the 20th Annual AIAA/USU Conference on Small Satellites*, SSC06-I-3, Logan, UT, 14-17 August 2006.

[4] B. Palmintier, "The EMERALD Protocol Suite: Design and Implementation of a Modular, Distributed Architecture for Small Satellite Command, Telemetry, and Power Systems," Engineer Thesis, Stanford University, June 2004

[5] Swartwout, Michael, Christopher Kitts, Pascal Stang and E. Glenn Lightsey, "A Standardized, Distributed Computing Architecture: Results from Three Universities", *Proceedings of the 19th Annual AIAA/USU Conference on Small Satellites*, SSC05-VI-6, Logan, UT, 8-11 August 2005.

[6] IEEE Std. 1451.2. IEEE Standard for a Smart Transducer Interface for Sensors and Actuators- Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, Institute of Electrical and Electronics Engineers, Inc., Piscataway, New Jersey 08855, 1997.

[7] Palmintier, B., Kitts, C., Stang, P., and Swartwout, M., "A Distributed Control Architecture for Small Satellite and Multi-Spacecraft Missions," *Proceedings of the 16th AIAA/USU Conference on Small Satellites*, Logan Utah, August 2002.

[8] B. Palmintier, R. Twiggs, C. Kitts, "Distributed Computing on Emerald: A modular approach for Robust Distributed Space Systems" In Proceedings of the 2000 IEEE Aerospace Conference, Big Sky, MT, March 2000.

[9] Conway, P.; Heffernan, D.; O'Mara, B.; Burton, P.; Miao, T. "IEEE 1451.2: An interpretation and example implementation"; Instrumentation and Measurement Technology Conference, 2000. IMTC 2000. Proceedings of the 17th IEEE Volume 2, 1-4 May 2000 Page(s):535 - 540 vol. 2