Effects of a Distributed Computing Architecture on the Emerald Nanosatellite Development Process

Julie Townsend, Bryan Palmintier, and Eric Allison

Space Systems Development Laboratory
Stanford University

Advisors: Prof. Robert Twiggs, Christopher Kitts

Abstract

Building satellites with greater capabilities on shorter timelines requires changes in development approach. Relative to previous satellite projects in Stanford's Space Systems Development Laboratory (SSDL), the Emerald Nanosatellite system is highly complex. Its mission requires numerous experiments and relatively sophisticated subsystem capabilities. To develop this system on a short two-year timeline required a new development approach to simplify system integration.

As a result, the Emerald development team adopted a modular distributed computing architecture. While this decision imposed many changes on Emerald's design process, the benefits of the distributed architecture for system integration and testing justified its selection. This approach has already affected the early stages of engineering model integration, and is expected to provide flexibility throughout construction and integration of the flight hardware. In addition the distributed architecture developed for the Emerald project will provide a useful tool for future development efforts in the SSDL and the small satellite development community.

Table of Contents

1. Introduction

Emerald is a two-year, two-satellite mission being pursued jointly by the Space Systems Development Lab (SSDL) at Stanford University and the Intelligent Robotics Program at Santa Clara University. Emerald's mission is to explore "Robust Distributed Space Systems", a phrase that encompasses many philosophies of distributed satellite system implementation. In particular, Emerald is a testbed for distributed science, scientific autonomy, distributed health monitoring, and emerging formation-flying technologies.

In keeping with the mission philosophy, the Emerald team has chosen to experiment with a distributed computing architecture for internal satellite processing, commanding, and data handling. This paper provides a description of the distributed computing architecture being implemented on the Emerald satellites. In addition, the paper discusses the effects of this design on the development and implementation of the Emerald satellite system and implications for future satellite development.

2. Distributed Computing Overview

Examples of distributed computing systems are everywhere in the modern world: the Internet, PC bus architectures (ISA, PCI, etc.), even the engine control systems of many cars. Such systems make it possible for many different design teams to easily come together with a cohesive product, and in many configurations provide significant performances advantages (such as parallel computing). Applying such an architecture to a satellite system also offers some unique advantages, that are briefly presented here.

Current applications

Many of the highest profile examples of distributed computing systems involve connecting general purpose computers (PCs, mainframes, etc.) together. However, the recent surge in distributed computing for specific small specific microntrollers provide a better analogy for such systems on satellites. Such Embedded systems are behind everything from automotive engine and accessory control and coordination, to industrial automation and control, to home entertainment systems, and more.

For space applications some aerospace corporations have historically used standard physical and data buses, but typically such systems still require significant reconfiguration of software and hardware in order to adapt to each mission, or are specifically designed with a single mission in mind. The recent push toward "Faster, Cheaper, Better" systems, has prompted further development of standardized architectures, but few have actually been implemented. For example JPL is developing the distributed X2000 architecture for deep space missions.

Advantages for Spacecraft

Adopting a modular, distributed bus architecture offers potential advantages throughout the entire lifecycle of a satellite (or satellite).

At the design stage, a distributed system can simplify the command and data flow within the satellite by clarifying which specific component is responsible for each task and what information exchange is required to make cause it to happen. Often a functional block from a signal flow diagram will map directly to a physical box on the satellite.

During development integration, a distributed architecture allows subsystems and experiments to tested individually and then incrementally integrated with other subsystems. This can happen even if crucial parts, such as the CPU, are delayed, provided the bus can be commanded by another system, such as a PC. It would also be possible to use the internet for "virtual integration" between remote locations. This would involve replacing the physical bus wiring with an internet connection.

On-orbit, a distributed architecture makes it possible for subsystems to share resources such as computational power and memory with-in a satellite and among multiple satellites. Also, a distributed architecture makes it possible to adapt to subsystem failures, even CPU failures (assuming a direct Comm to bus connection) while still continuing the mission.

And when extended to a multi-satellite mission, a distributed architecture allows multiple satellites to be inter-connected and used as a single "virtual bus." Resources, such as communication (and indirectly power), computation can be reallocated and shared among a fleet of satellites Experiments involving multiple satellites can be coordinated and controlled autonomously from a single satellite, as though the experiment involves only a single satellite. [i]

3. Emerald Architecture

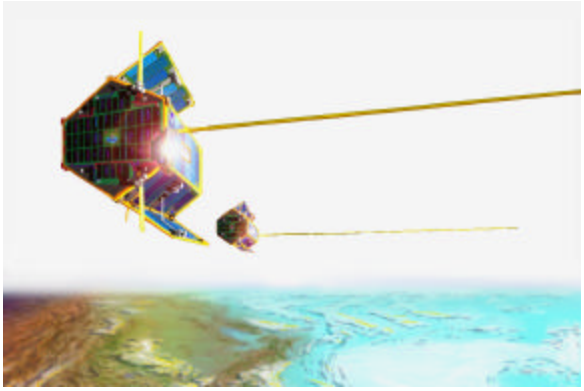

Figure 1. Artist's Conception of Emerald on Orbit (by W. Henning)

The Emerald Nanosatellite project is an entirely student managed, designed, built, and operated project building two satellites for launch in 2001. Emerald is a joint effort of the Space Systems Development Lab (SSDL) at Stanford University and the Intelligent Robotics Program at Santa Clara University.

Emerald is part of the University Nanosatellite Program funded by the Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Project Agency (DARPA). For this program, ten applicant universities were each selected receive funding on the order of $100,000 to design and construct a 10-15 kg satellite on a two-year timeline. In addition to funding, a launch opportunity was provided for each satellite aboard the Space Shuttle in 2001. The goal of this program is perform creative low-cost experiments as a technology testbed for future developments in emerging fields such as distributed satellite systems.

Emerald's mission is to explore Robust Distributed Space Systems. To this end, experiments and demonstrations will be performed in autonomous operations, distributed science, and distributed health monitoring. A variety of new technologies for formation flying will be demonstrated, including

spaceborne GPS receivers, inter-satellite communication, drag-based position control, and a colloid microthruster, and ultimately, these components will be integrated for a demonstration of closed-loop formation flying. In addition, each Emerald satellite will integrate a new space mission architecture in which computing, control, and data management are distributed throughout a single satellite, and across a formation of closely flying satellites.

Sponsors for these experiments include NASA Goddard Space Flight Center, NASA Ames Research Center, Jet Propulsion Laboratory, Department of Defense, Boeing, Lockheed-Martin, Honeywell, and Stanford University laboratories such as the Space and Radioscience Laboratory (STAR lab), the Plasma Dynamics Laboratory, and the Aerospace Robotics Laboratory.

The bus design for the Emerald spacecraft is based on Stanford`s Satellite Quick Research Testbed (SQUIRT) microsatellite design.[ii] This satellite bus consists of:

- a 15 kilogram structure in a modular 18 inch diameter hexagonal configuration
- full-duplex ground-to-orbit communication in the amateur UHF and VHF bands
- half-duplex intersatellite communication in the amateur UHF band
- body mounted high efficiency GaAs solar panels with NiCd cells for power storage
- 5-volt and 12-volt regulated power distribution
- passive insulating and conducting elements for thermal control.
- light sensors and a magnetometer for attitude sensing

The Emerald team is currently building and integrating engineering model hardware. Flight hardware construction is planned for the summer to allow integration and testing to begin in fall 2000. Additional project information is available on the Emerald website at http://ssdl.stanford.edu/Emerald/.[iii][iv][v]

Distributed Data Handling

Overview

The Emerald distributed computing architecture was designed to explore the advantages of distributed systems. Limited development time and mass, volume, and budget constraints dictated a simple and easily implementable approach. Adaptability for future projects and missions was also an important consideration. This section provides a detailed description of the distributed data handling architecture being implemented on Emerald.

Architecture

To maximize integration ease and flexibility, the Emerald distributed architecture is built around a standard serial data bus with strictly defined interfaces. However, certain subsystems (Comm and GPS) required direct, non-standard connection to the CPU. The resulting hybrid configuration combines a bus topology for most subsystems with a centralized topology for Comm and GPS. This arrangement is illustrated in Figure 2.

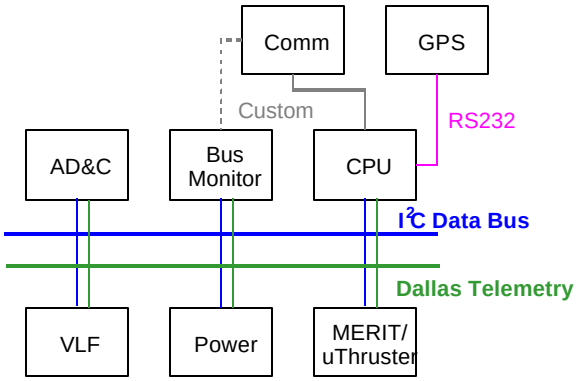

Figure 2. Emerald Data Bus Topology

The rationale behind this topology is as follows: The Emerald CPU and Modem are both manufactured by SpaceQuest and designed to work together through a custom interface. Inserting bus interfaces between these two components is unnecessary and would be inefficient. The GPS module merits a direct CPU connection due to its continuous data production and inter-satellite communication during relative position measurements. To prevent traffic on the data bus during relative position operations, GPS data is passed directly to the CPU. Finally there is a direct link from the Comm system to the Bus Monitor to allow signals to be sent directly to the I2C bus, bypassing the main CPU. For all other subsystems, this configuration offers a high level of modularity.

Each subsystem that interfaces to the I2C bus is more than just a passive input/output device. The subsystems are “smart”. Each utilizes a PICmicro® microcontroller to perform application-specific data collection and processing tasks. Smart subsystems allow the CPU to interact with the system via high-level commanding. For example, instead micro-managing data collection in any given subsystem, the CPU can send a command such as “c(1)”, collect data every second until I say stop, or “c(1,34),” collect 34 sets of data at a rate of 1 Hz. Abstracting out these functions to such a high level relieves the CPU of responsibility for low-level

subsystem functionality, reserving its processing power for other tasks. Such abstraction also allows hardware modifications in the subsystems to be more transparent to the CPU.

Data Bus (I2C)

The data bus protocol selected for the Emerald distributed data handling system is the Inter-Integrated Circuit (I^2C). This protocol is used in Audio/Visual equipment, on PC motherboards, and in "smart" batteries.

Low level: I2C

I2C is a synchronous serial protocol that uses only 2 wires, one for data (SDA) and one for clock (SCL). It also requires a common ground. It operates at 100 kbps (kilo *bits* per second) in standard mode.

Both lines utilize wired-AND connections, which allow for arbitrary numbers[†] of devices and support "hot swapping", adding/removing devices without interrupting the bus.

Communication is always initiated by a master, which also drives the clock. Each I2C message consists of an arbitrary number of 9 bit "words." These words are 8 bits of information (supplied by the current "transmitter") plus one acknowledge bit (from the "receiver). The first word of the message sets the address (7bits) and the communication direction (Read or Write). In read mode, after the first acknowledge, the slave begins transmitting and the master becomes the "receiver."

For more information and details, see the I2C specification [vi] or *The I2C Bus from Theory to Practice* [vii].

High Level Messaging

I2C standardizes many layers of the communication protocol. Specifications exist for a wide range of tasks, from basic reading and writing to complex multi-master support and arbitration. However, because it does not specify any data integrity checks, the Emerald team has developed a simple, error checking message format.

In the Emerald format, commands packets coming from the CPU (or any other master) are fixed to 5 bytes in length, plus a field for a return address and checksum (rolling 8bit sum). Reply packets include a field for length plus a variably sized data portion and finally a checksum byte for error detection.

A simple acknowledgement system was also developed. From the master's perspective, a complete communication includes sending a command packet, waiting for an acknowledgement byte, and then, if indicated, requesting a data packet in reply, and finally acknowledging that the data packet was successfully received. The complete message formats is shown in Figure 3.

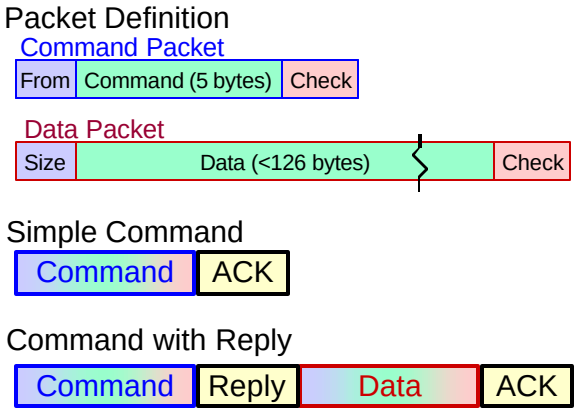

Figure 3: Emerald Message Format

Certain standard commands are defined for commanding of all subsystems on the bus. These include functions for checking subsystem status, synchronizing time, and querying the subsystems for a list of defined commands. This last feature is very attractive because it allows subsystems to change and expand their functionality without requiring reprogramming in the main, coordinating CPU. [i]

Telemetry Bus (Dallas 1-wire)

In addition to the higher bandwidth command/data bus described above, Emerald will have a low bandwidth telemetry bus that uses the Dallas 1-wire protocol.

This multi-layer configuration is similar to the configuration chosen by JPL for the X2000 data bus. X2000 uses IEEE 1394 (Firewire) instead of I2C for data transfer and I2C instead of Dallas 1-wire for telemetry and control. [viii]

Dallas 1-wire supports an arbitrary number of temperature sensors connected directly to the two shared wires of the bus, providing simple temperature telemetry collection over the entire satellite. Standalone analog to digital converters (DS2450) are also available, allowing other types of telemetry (such as

[†] The number of devices is actually limited by the electronic signalling requirements of the I2C bus, such as maximum capacitance and resistance, but this is not a problem for even the maximum number of nodes ever considered for Emerald (~20)

bus voltages and currents) to seamlessly connect to the 1-wire bus in the same fashion.

In addition to telemetry, the Dallas system will be used to provide subsystem on/off signals using serially addressable switches (DS2405, DS2406, and/or DS2406). When commanded by the CPU, each switch will be used to drive a P-channel power MOSFET, which sources power to a subsystem.

Baseline Subsystem Processor (PICmicro®)

The Microchip PIC16C77 was chosen as a baseline subsystem microcontroller for interfacing each subsystem to the data and telemetry buses. For the distributed architecture to work effectively, it is not necessary for all subsystems to use the same processor. As long as the standard bus interface is observed, processor differences are transparent to all other subsystems on the bus. The PICmicro® standard was selected to simplify baseline subsystem development, and subsystems is free to deviate from it as they see fit.

PIC16C77 Micro Controller

The PIC16C77 is a single *chip* computer, requiring only an external clock source. It resides in a 40-pin package with 8 kwords of ROM and 384 bytes of RAM. This particular PICmicro® has many built-in peripherals, including I2C capabilities and an 8 channel A/D converter, both of which are very convenient for Emerald's application. Another attractive feature is its low power consumption: less than 100 mW at max speed (20Mz) and less than100µW in its low-power sleep mode. The PICmicro® also has a simple "RISC" instruction set, which allows fast and efficient execution. In addition, the PIC16C77 is a close relative to the PIC16C73, which has been used in research for fault-tolerant computing on small satellites. [ix]

One of the major shortcomings of the PICmicro® is its lack of support for external memory. The Emerald team has worked around this deficiency by interfacing a simple custom SRAM capable of storing up to 2.5Mbytes of data for subsystems that require additional storage. Less memory intensive subsystems will store data in an external serial EEPROM.

Software

A common library of low-level hardware routines, as well as standardized I2C bus interfacing has been developed to simplify subsystem programming. The library includes support for:

- I2C, including high level protocol
- RS232, for debugging
- A/D conversion at up to 18kHz
- Timer control for synchronizing fast events
- Real Time clock base for slower events, timestamping, and scheduling.
- External memory support (SRAM, EEPROM)
- Simplified interface for translating I2C commands into subsystem function calls
- Standardized help command to query for valid function names.
- Standardized, run-time configurable "loop" structure.

4. Designing for a Distributed Architecture

In addition to the design of the distributed computing architecture itself, this architecture required new design approaches in other aspects of the design. Specifically, the distributed computing architecture was a central consideration in the design of all subsystems and experiments interfacing with the data bus, the satellite test port interface, and the equipment panel layouts.

Motivation

Traditionally, missions in SSDL have centered around a small number of experiments (2-3) requiring specific hardware, of which a single experiment took top priority. They have also been developed under largely open-ended schedule without the pressures of a proposed launch. (Sapphire [x] and Opal [xi]). Emerald, however, was different: the mission included a relatively large collection of experiment specific hardware and experimental subsystems (at least 7), all with comparable importance. Plus, the team faced an aggressive two-year time schedule. As with many university projects, Emerald involved a wide diversity of students (from undergraduate to PhD), with different levels of commitment (volunteer, part of a class, research assistants) and lengths of involvement ranging from one quarter to the entire project. To complicate matters further, the Emerald team is based at two universities, who work cooperatively to develop the two satellites, rather than contributing one entire satellite each.

To successfully navigate these challenges required a modular approach to development and integration. Ideally, this approach would also allow subsystems to be de-scoped or removed if they fell behind schedule with minimal impact on the rest of the system. In addition, the team desired a system that would allow thorough subsystem testing and validation at remote

locations. A distributed computing architecture offered all of these features.

Protocol Selection

The prime objectives in selecting the bus protocol were to maximize the freedom to add and remove subsystems from the bus and minimize wiring harness mass and volume. Only synchronous serial protocols were considered because they provided the desired balance of simplicity, reduced wiring, and speed. Additionally, protocol support for multiple nodes sharing control of the bus (multi-master) was desired, though not strictly required, for some experiments. We looked in more detail at three simple protocols commonly used in embedded systems:

- *Controller Area Network (CAN):* Used in automotive industry and industrial control.
- *Serial Peripheral Interface (SPI):* Widely used general purpose
- *Inter Integrated Circuit (I^2C):* Used in Audio/Visual equipment, motherboards and "smart" batteries

For Emerald's needs, the CAN protocol was overly complex, and the SPI protocol did not offer enough expandability. I2C offered a simple system with all the features that Emerald required, and commercial components supporting the I2C protocol were easily available. For these reasons, Emerald chose to use an I2C bus.

To reserve the bandwidth of the I2C bus for commanding and data handling, the Dallas 1-wire bus was added to handle telemetry and power-switching tasks. The 1-wire protocol was selected for the following reasons:

- It is simple, requiring only 1 power/data line plus a common ground and using a simple asynchronous serial signaling style (like RS232).
- A standalone temperature sensor (DS1820) is available. The sensor comes in a tiny three-pin package and each unit, like all Dallas 1-wire devices, comes preprogrammed with a unique hardware ID.
- The shared power/data line causes device power to be cycled frequently, minimizing the potential damage of Single Event Effects in the radiation environment of space. [xii]

Subsystem Design

Design of 'smart' subsystems was, for SSDL, an adventure into uncharted territory. Implementing subsystems as standalone units that could operate based on high-level commands required an entirely new approach.

For smart subsystems, each design team is required to design microcontroller software in addition to electronic and mechanical hardware. This issue immediately surfaced as a potential stumbling block for distributed computing implementation. Because satellite design teams in SSDL generally suffer a shortage of experienced programmers, it seemed unlikely that each subsystem design team could be provided with a capable software designer. To address this issue, a standard software set was designed to provide basic interfacing functionality. This standard software set removed most of the complexity of the subsystem-level software design, leaving the subsystem design team responsible only for implementing hardware and application specific functionality.

Additionally, the use of a standardized data bus interface allowed the design team to define a common set of interfacing hardware. This was a change from past projects in which commonality between individual subsystems was rare. The Emerald design team took advantage of this commonality by designing a standard interface board for use with all subsystems and experiments. Included in this standard circuit are the PICmicro® microcontroller and supporting hardware, power switching circuitry, and latch-up protection circuitry. These standard boards will be produced in large numbers (about 50) to supply interfaces for both the engineering model and flight hardware. This further freed the individual subsystems to work exclusively on application specific hardware, and allowed some fairly sophisticated functionality -- such as multi-channel latch-up protection circuitry to protect against radiation-induced Single Event Effects -- to be designed once and incorporated into all subsystems.

Main CPU Design

By design, the distributed architecture simplified the requirements on the satellite's main CPU. In its simplest form, the CPU is only required to be a message router among the ground, other satellites, and the distributed subsystems and experiments. As a result, the CPU team has been able to look into implementing more sophisticated user interfaces and autonomy experiments. Most importantly, this has allowed the team to survive some unexpected delays and complications with the CPU hardware.

Test Interface Design

The distributed architecture also complicated the test interface design on Emerald. In previous SSDL satellites, with one centralized processor, a single serial

connection to the CPU was sufficient to provide a full view of system functionality. For Emerald's distributed architecture, however, CPU observation tells only part of the story. Full system understanding requires access to the distributed portions of the satellite as well.

As a result, in addition to the CPU debug port, Emerald includes an additional test interface that allows full access to both the I2C data bus and Dallas 1-wire telemetry bus. Full characterization of the system also requires access to each subsystem's PICmicro® microcontroller. To allow for this access, the standard interface board used by each subsystem includes its own debugging and re-programming port. These individual debugging ports will be accessible during satellite integration and testing, but not when fully assembled, due to wiring constraints.

Component Layout Considerations

Efficient component layout was also influenced by the distributed computing architecture. Space was left along opposite interior corners to run power buses separately from the data bus. Subsystem and experiment boxes were arranged such that data and power connections cluster toward the center of the tray, making sure that enough space was left for connections and flight connector protectors. In addition, care was taken to ensure that all PICmicro® debugging and reprogramming ports are accessible when the satellite side panels are removed, without removing any boxes from the equipment trays.

5. Implementation

Modular Development

For the design, construction, and testing of Emerald subsystems, the distributed architecture provides a nice division among subsystems and between the subsystems and the main CPU. This modularity has allowed each subsystem and experiment development effort to progress independently, significantly easing coordination among the large diverse team as a whole.

The modular configuration also allows one device to be replaced transparently by another, similar device as long as both conform to the same interface. This makes upgrading a given subsystem with an increase in performance, a bug fix, or an extension of capability easy, as long as the upgraded system conforms to the standard.

Impact of Distributed Computing on Integration

One of the areas in which the distributed architecture stands to make the greatest impact is that of integration and test. The serial data bus architecture allows easy, "plug and play" addition of subsystems and experiments to the system as they become available. Additionally, under I2C subsystems can even be "hot-swapped," that is added or removed from the system without affecting the rest of the bus in anyway. This eases the task of integrating the electronics of each of the subsystems.

The PICmicros® in each subsystem can be queried by the CPU for a list of valid commands. This allows the CPU to be ignorant of the details of the individual subsystems. On startup, the CPU compiles a table of subsystem commands and their parameters by asking each subsystem for a brief command help listing. This table can then be a) passed to the ground for operations planning, b) used to check ground commands for validity, and/or c) be autonomously used by the CPU to implement high level planning objectives by relating the physical commands with desired effects, using keyword matching and artificial intelligence.

This modular, standardized interface further enhances and eases integration by allowing other devices, such as a desktop PC, to connect to the data bus to simulate non-existent nodes. For example, such a PC node could be used to fully test and debug a subsystem, before it is connected to the common data bus. The PC node can also stand-in for the CPU system to continue to exercise the other subsystems while the CPU is adjusted off-line.

Experiences to Date

The modularity of the design has benefited the Emerald project in several ways. With the interface defined, and the requirements clearly laid out, the development of individual subsystems can proceed in isolation. This has been especially beneficial, because of the collaboration between Stanford University and Santa Clara University. The problems of tight integration of different teams of people over long distances are partly mitigated by the modularity inherent in the design of all the subsystems.

The development of the PICmicro®-enhanced subsystems has proceeded more quickly than the development of the main CPU. So, even though the main CPU has been unavailible, a PC has been used for stand-alone testing and debugging of several of the subsystems.

Basic integration using the distributed system has also been demonstrated in the lab. The same PC interface was connected to multiple subsystems simultaneously, and used to communicate with a prototype of the attitude control system (developed at Stanford) and mechanisms control block (developed at Santa Clara). Additionally, a prototype of the bus monitor experimental hardware was used to verify that the PC interface communicated according to the Emerald I2C protocol. (See Figure 4)

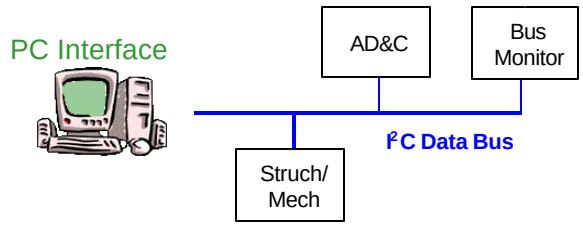

Figure 4: Integration Test Setup

Thus, the modularity provided by the distributed architecture allowed an efficient division of labor across geographical distances and the potential of the distributed architecture to aid integration has also been demonstrated.

Future Impact of Distributed Computing on Emerald Integration

The potential of Emerald's distributed architecture to ease integration has yet to be fully exploited. The continuing development of the CPU is expected to lag behind that of the other subsystems. For this reason, the PC node will be used extensively, not only to debug and test the individual subsystems, but also to stand in for the CPU and command all of the other subsystems at once. This capability allows the development of the subsystems to proceed at a desirable pace, while still providing assurances that the integration is indeed valid. The validity of the PC-based integration is assured by strictly adhering to the interface standard, providing the modularity described above.

6. Orbital Operations

Routine Operations

During normal spacecraft operations, the distributed computing system frees the CPU from responsibility for low-level hardware control. Because the subsystems have the intelligence to operate their own hardware, the main CPU can control them using simple, high-level directive commands. Thus, while data collection or some other activity is being carried out within a subsystem, the CPU processing capabilities are free to attend to scheduling and other system-level tasks.

Virtual Data Bus

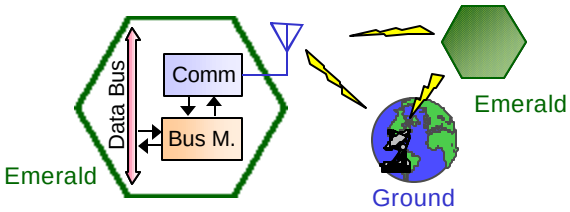

Figure 5: Virtual Data Bus Concept

Just as the internet is used to bridge two local area networks (LANs) on the ground, the inter-spacecraft communication link can be used between the two satellites to form an inter-satellite RF (radio frequency) bridge, or Virtual Data Bus, on orbit.

On Emerald the Communication system is connected to the Data Buss both through the main CPU and the Bus Monitor experiment (which otherwise is a passive monitor). Such a configuration opens up a whole collection of operations possibilities, such as:

Fault Tolerance.

Should one satellite's CPU fail, the entire satellite may not be lost. Smart subsystems on the data bus may be directly commanded from the ground. This concept is illustrated in Figure 5. Since the subsystems recognize high level commands, a few command bytes passed from the ground is enough to run a potentially complicated procedure.

In a multiple satellite system, like Emerald, there is no reason that direct communication system to bus commands have to come from the ground. Instead, as seen in Figure 6, the subsystems can be controlled by another satellite. In this example Satellite B's CPU has died, yet the inter-satellite missions are not lost, since Satellite A's CPU can still control all of satellite B's subsystems. This inter-satellite control allows complex operations sequences for the injured satellite (B) even, if the satellite is out of range of a ground station.

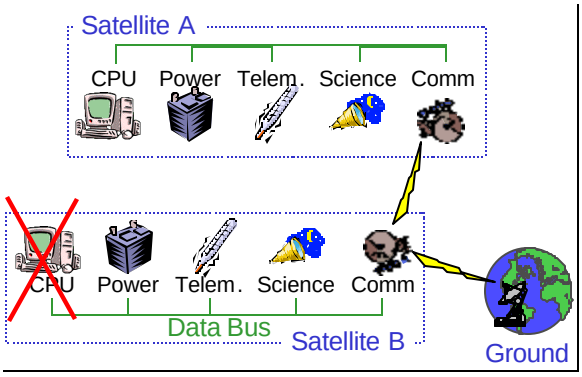

Figure 6: On-Orbit Fault Tolerance Concept

Inter-satellite resource sharing.

The virtual data bus also allows sharing common resources. This can be used for fault avoidance or as part of standard operations. For example, if one Emerald spacecraft has anomalous power levels, it could transfer data to the other satellite using the low power inter-satellite link, letting the other, healthy satellite supply the high power required for downloading to the ground. Similarly, memory or even computation resources could be shared using the virtual data bus.

Experiment Coordination

The virtual bus will also be used to coordinate experiments between the two Emerald spacecraft. For example the CPU on one Emerald can request the GPS position of the other to determine the effectiveness of the drag panel. Also, the VLF experiment from one Emerald will coordinate directly with the VLF experiment on the other to synchronize operations.

Design Considerations

To support virtual bus experiments, Emerald is using the same communication system (437.5 MHz, 9600 baud, half-duplex) for both ground to satellite and inter-satellite communication. This allows one satellite to "log-on" to the other and initiate commands just as the ground station does. Data addressing is handled by the amateur radio AX.25 protocol, which incorporates both a sending and receiving call-sign into each packet. An additional receiver (145.8 MHz) on each satellite allows the ground to send commands to the satellites even if the inter-satellite communication is tying up the main communication channels.

The Bus Monitor connection from the Communication system to the data bus will be isolated from the main communication on the same frequency by a fire code and a different communication protocol. When a certain sequence of bits is uploaded, the CPU will ignore the next data and instead allow the Bus Monitor to interpret the received signal (at 1200 baud) and drive the I2C bus. A similar procedure will allow the bus-monitor to also drive the transmitter to verify that the command was completed successfully, download data, or to initiate communication with the other satellite.

One of the major challenges for implementing a virtual data bus is time. The inter-satellite communication link is much slower than the on-board data bus, so messages must be packetized, cached, or the data bus must be temporarily slowed down to the speed of the inter-satellite link. Slowing down the data bus has the advantage of being simple, but it does tie up the data bus, potentially delaying other messages from getting through (latency). When the data is cached, this latency only applies to the command being sent inter-satellite. However, handshaking, such as the I2C acknowledge bit, becomes tricky.

As a baseline, Emerald will slow down the data bus when operating as a virtual data bus. This slowing down of the data bus is readily supported by the I2C protocol, since it allows slave devices to hold down the clock line until ready to proceed. When the virtual bus is active, the data bus will be slowed down on both satellites, but during normal operations (intra-satellite) it will still run at full speed

7. Future Applications

In SSDL

Future satellite projects in SSDL will benefit from Emerald's distributed system development. For projects similar in size and scope to Emerald, development time will be greatly reduced. Beginning with existing Emerald hardware, future student teams can quickly prototype a baseline system by removing Emerald's experiments from the bus and replacing them with their own. Once the baseline has been achieved, the modular nature of the bus components themselves will simplify performance upgrades. Revised and augmented subsystem models can replace existing hardware as they are completed without compromising the functionality of the rest of the system. The distributed computing architecture provides enough flexibility that the existing satellite design can be upgraded to meet future mission requirements rather than reinventing the design from scratch.

Emerald's distributed computing architecture will also be applicable to projects on a smaller and less complex scale than Emerald itself. For example, the SSDL would like to provide master's degree students with the opportunity to develop and build very simple satellites from conception to operation in a single year. The distributed computing architecture developed for Emerald could be instrumental in shortening the development process enough to make this vision a reality. Each student could begin development with a small version of Emerald's distributed architecture, mastered by a microcontroller and combined with a generic set of bus components (power, comm, etc). Beginning with this sort of baseline would allow the students to concentrate on payload development and integration, fabrication, testing, and operations; a more feasible set of tasks for a year's work.

For projects on a larger scale or with significantly higher complexity, the distributed computing architecture developed for Emerald would still provide a good starting point. However, use of I2C for the main data bus may need to be reconsidered if a higher bandwidth of data transfer is desired or required.

In general

Unlike the data bus systems used by some major manufactures to simplify satellite development which are aimed at larger scale missions, the simple sort of system being developed for the Emerald project could help make space more accessible to small businesses and individuals. Simple and easily available distributed architectures could significantly reduce research and development costs for small businesses interested in one-time satellite design. These developers could benefit from the baseline bus design in much the same way as the university students.

As more and more technologies are proven to support formations of closely flying satellites, the distributed bus concept will be helpful in coordinating the development of multi-satellite fleets. Standardized data interfaces would ease satellite integration in a situation where parts of each satellite in the fleet are designed at distant locations or by several different companies. In the case where each satellite in the fleet is designed and constructed independently, a standard data interface would simplify inter-satellite interactions.

8. Conclusions

Adoption of a distributed computing architecture for the Emerald project caused significant revision in SSDL's satellite development process. Defining the architecture, creating standard interfaces, and designing intelligent subsystems required extra effort in the design and hardware/software prototyping phases of development.

However, this extra design work has already begun to pay off during the early stages of engineering model integration. The distributed architecture has allowed testing and integration of finished subsystems even though development of the main CPU has been uncontrollably delayed. The Emerald team expects to benefit from the modularity of the distributed architecture in this way throughout the remaining integration and testing phases of development.

In addition, the effort spent in designing Emerald's distributed architecture is expected to shorten the design process for future SSDL satellite missions. The modular, expandable nature of the distributed system will provide future design teams with both a baseline design and a functional prototyping platform.

Distributed architectures have great potential to make space more accessible to a wider variety of satellite developers by reducing research and development effort and simplifying system integration. Based on the benefits observed through the Emerald's development process, the Emerald design team expects distributed computing systems to become an enabling technology for faster and cheaper satellite development.

9. Acknowledgements

The authors wish to thank AFOSR and DARPA for their commitment to supporting university-based spacecraft development projects. Thanks also to Microchip Technology Inc. and IAR Systems Software Inc. for their generous support in the form of PICmicro® microcontrollers and development systems. Finally, the students at Stanford and Santa Clara are thanked for their tremendous effort and dedication in developing the Emerald spacecraft.

References

[i] B. Palmintier, et. al., "Distributed Computing on Emerald: A modular approach for Robust Distributed Space Systems" In *Proceedings of the 2000 IEEE Aerospace Conference*, Big Sky, MO, March 2000.

[ii] C. Kitts and R. Twiggs, "The Satellite Quick Research Testbed (SQUIRT) Program", In *Proceedings of the 8th Annual AIAA/USU Conference on Small Satellites, Logan, Utah*, August 1994.

[iii] C. Kitts, et. al., "EMERALD: A Low Cost Formation Flying Technology Validation Mission", In *Proceedings of the 1999 IEEE Aerospace Conference*, Snowmass, CO, March 6-13, 1999.

[iv] C. Kitts, et. al., "Emerald: An Experimental Mission in Robust Distributed Space Systems," In *Proceedings of the 13th Annual AIAA/USU Conference on Small Satellites*, Logan, Utah, August 23-26, 1999.

[v] J. Townsend and E. Allison, "The Emerald Nanosatellites:Two Student-Built Satellies as a Testbed for Distributed Space System Technologies", In *Proceedings of the 5th International Symposium on Small Satellite Systems and Services*, La Baule, France, June 2000.

[vi] *The I²C-Bus Specification*, version 2.0, December 1998 [Available Online: http://www-

us.semiconductors.philips.com/acrobat/various/I2C_BUS_SPECIFICATION_2.pdf, Oct 30 1999]

[vii] D. Paret with C. Fenger, *The I2C bus : from theory to practice,* Wiley, Chichester ; New York, 1997.

[viii] N. Paschalidis, "A Remote I/O (RIO) Smart Sensor Analog-Digital Chip for Next Generation Spacecraft" In *Proceedings of the 12th AIAA/USU Conference on Small Satellites,* 1998.

[ix] D. W. Caldwell "A Minimalist Hardware Architecture For Using Commercial Microcontrollers In Space" In *Proceedings of the 16th Digital Avionics Conference,* 1997.

[x] R.Twiggs, and M. Swartwout, "SAPPHIRE - Stanford's First Amateur Satellite", In *Proceedings of the 1998 AMSAT-NA Symposium*, Vicksberg, MI, October 1998.

[xi] J. Cutler and G. Hutchins, "OPAL: Smaller, Simpler, and Just Plain Luckier", In *Proceedings of the 14th Annual AIAA/USU Conference on Small Satellites,* Logan, Utah, August 2000.

[xii] D. W. Caldwell, "A Distributed Spacecraft Thermal Control Architecture Using The Dallas Semiconductor Microlan Products*"* In *Proceedings of the 16th Digital Avionics Conference,* 1997.