Distributed Computing on Emerald:
A modular approach for Robust Distributed Space Systems.[1]

Bryan Palmintier
Stanford Space Systems Development Lab (SSDL), Stanford University

Robert Twiggs
Space Systems Development Laboratory (SSDL), Stanford University

Christopher Kitts
Santa Clara Remote Extreme Environments (SCREEM) lab, Santa Clara University
Space Systems Development Laboratory (SSDL), Stanford University

Department of Aeronautics and Astronautics, Durand Building Rm. 250
Stanford, CA 94305-4035
(650) 723-6021, bryanp@leland.stanford.edu

Abstract—A modular, distributed bus architecture potentially offers significant advantages throughout a satellite's lifecycle. Specifically this architecture enables...

During Ground Integration:
- incremental integration of subsystems even if crucial parts, such as the CPU, are delayed.
- Using the internet for "virtual integration" between remote locations.

On-orbit:
- sharing resources with-in a satellite
- no cost redundancy, including directly commanding subsystems through the communication subsystem, should the CPU fail.

And when extended to a multi-satellite mission:
- Multiple satellites to be inter-connected as a single "virtual bus."
- Autonomous reallocation and sharing of resources across satellites, including adapting to subsystem failures.
- Autonomous experiment coordination.

This paper explores the benefits and challenges of single and multi-satellite distributed architectures and the unique strategies they enable. The data architecture for the two satellite Emerald mission is used as an example. Emerald uses an I2C serial bus to connect PICmicro based "smart" subsystems.

1. Table of Contents

2. Introduction

Emerald is a joint two year, two satellite mission between the Space Systems Development Lab (SSDL) at Stanford University and Santa Clara Remote Extreme EnvironMent (SCREEM) lab at Santa Clara University. Emerald's mission is to explore "Robust Distributed Space Systems." As part of this mission, Emerald is experimenting with a modular, distributed computing architecture and its trade-offs.

Additionally, this paper provides an overview of distributed computing and how it relates to small satellites.

3. Overview of Distributed Computing

What is Distributed Computing? It means a lot of subtly different things to different groups of people. For example, distributed computing can mean computers working together across the internet, a single computer with multiple processors, or even a system of sensors where each sensor has its own microcontroller for data analysis.

This section explains how the term distributed computing is applied in this paper. It includes a discussion of terminology and key concepts and then goes on to describe distributed computing and its trade-offs in the context of satellites.

[1] 0-7803-5846-5/00/$10.00 © 2000 IEEE

Terminology

In this paper, "Distributed Computing" refers to any decentralizing of the computing power of a system. In other words moving the centralized computing responsibilities away from one central location (such a CPU box, board, or even single chip module) and dividing it between multiple locations, typically for some form of performance improvement. Under this definition all of the examples given above fall under distributed computing.

The most common distributed computing arrangement is a "network," in which a diversity of computational nodes is connected together, often with an emphasis on sharing data. In a network arrangement it is clear to users that there are multiple machines, and these multiple machines may be performing totally different functions at any given time. There is also lots of interest among computer scientists in "distributed computing environments" in which a collection of computers (typically on some sort of network) appear to the user as a single computer. [1, 2]

When, the computing is distributed solely for computational (number crunching) performance, this paper refers to "parallel" computing. In this case, a single larger problem is split apart such that multiple computing resources can work on the problem simultaneously. Parallel computing is an active area of research, enabling more and more computationally expensive problems to be considered. For space applications, such computationally expensive operations are often done in the ground segment, though parallel processing is being used on-orbit for data reduction of remote sensing and other applications.[3]

In another type of distributed computing , an entire category of tasks, such as interfacing to specific hardware, interacting with the outside world, or highly specific computations, is off-loaded to "smart" subsystems or "smart" peripherals. Each smart subsystem contains its own computational power, allowing the main CPU (if any) to take on a coordinating role, free from the burden of menial tasks. This architecture is comparable to object-oriented computer programming, in which data is stored along with the functions that operate on it. As object-oriented hardware, smart peripherals keep sensors, signal conditioning hardware, and some data processing and analysis software at the data source.

Standardization

A standardized interface forms the foundation for all but the most simple distributed computing systems. Without it, a distributed computing system quickly degrades into either a single, centralized system, or a collection of independent and non-interacting devices. Such standardization can take place at many levels, from detailed hardware specifications to high-level "user" interfaces.

At the hardware level, the nitty-gritty electronic (or optical) signaling details, and physical interconnections, such as connectors, wires, and pinouts, can be specified. While at the user interface level, the standard commands and parameters recognized by all devices and the "look and feel" of interactions maybe detailed. Between are protocol layers for arbitration, data transfer, error detection and correction, routing information and so on.

An important standardization concept is that of the data bus. In its most general terms a bus is a collection of hardware and software that allow multiple computing resources to be connected together, typically using a well defined set of interfaces and protocols. A fairly simple example is the serial bus that connects your mouse to the computer. (probably either RS232 or PS/2) This particular bus allows for only two devices ("single drop") which simplifies the protocol. A more complicated example is the ISA (or PCI) bus that allows you to plug expansion cards into your computer. This bus is more complicated since it deals with multiple nodes, (multi-drop) many more data lines, and much higher speeds.

A "data bus" should not be confused with a "spacecraft bus," which refers to the supporting subsystems on a satellite (CPU, Comm, Power, Structure, etc.) as opposed to the mission specific payload. Often, both are referred to as just "the bus" leading to further confusion. This paper avoids using "bus" alone unless the meaning is readily apparent.

Modularity

For Emerald, the ultimate goal of standardization is modularity. In a modular architecture, component interfaces are standardized to the level that components can be readily connected together. This requires all of the standardizations from hardware specifications to the command ("user") interface details to match.

Such a configuration allows one device to be transparently replaced by another, similar device as long both conform to the same interface. Likewise one implementation of a given subsystem may be an increase in performance, a bug fix, or an extension of capability.

This concept forms the foundation of software engineering and is wide spread in terrestrial systems in the form of standard fasteners, recording media, etc.

Historically, spacecraft are often one of a kind items, making modularity rare.

It is true that there are many standard interfaces used on satellites, but typically they don't specify all the levels from hardware to user interface, requiring customization with each new design. Recent interest in large constellations of satellites and the steady push toward cheaper systems, has forced engineers to think seriously about standardization, modularity, and mass-producability.

Architecture[1]

The architecture of a system refers to the physical and logical framework used to interconnect components. It determines the pathways for inter-subsystem data transfer and may have a large bearing on both wiring harness size and modularity.

Centralized Architecture— In a centralized architecture, as seen in Figure 1, every subsystem has a dedicated connection to a central node (typically the CPU). Such architectures typically require large wiring harnesses and significant software and hardware changes to the central node when adding or changing the attached subsystems.

Note that it is possible to use a centralized architecture for distributed computing, if the subsystems themselves have processors. This is fairly common in satellite architectures which may use dedicated RS232/422 channels (COM ports) to connect smart subsystems to the main CPU.

Ring Architecture— A "ring" architecture, as seen in Figure 2, is a distributed architecture in which each subsystem is connected to two others to form a ring. Communication is typically facilitated with a "token" that is passed around the ring from one subsystem to another. This architecture usually requires a small wiring harness. Changing the attached subsystems, may require some small modification to member nodes, and can interrupt communication on the bus.

Bus Architecture— A "bus" architecture, as seen in Figure 3, is another distributed architecture in which each subsystem is connected to a shared, common, standardized data bus. This architecture usually results in a small wiring harnesses and makes it very easy to add, remove, or change the attached subsystems, with minimal changes to other nodes.

Hybrid Architectures— For real systems, including satellites, it is common to combine these architectures into a "hybrid" architecture. Hybrid architectures allow data transfer to be tailored to specific types of needs. For example, within a modern desktop computer, the disk drives maybe connected using a bus architecture; the Modem, sound card, and other peripherals may use another bus architecture, while the graphics card may be connected to the CPU in a centralized architecture. Emerald uses a hybrid architecture.

Layered Architectures— In some situations it is also useful to connect some or all of the subsystems to multiple independent data buses. These independent buses form "layers" of the overall data bus architecture. For instance, satellite subsystems may be connected to a slower data bus for command and controller and also to a faster bus for science data transfer. Or, they may be connected to multiple identical bus layers for redundancy. Emerald uses a two-layered architecture.

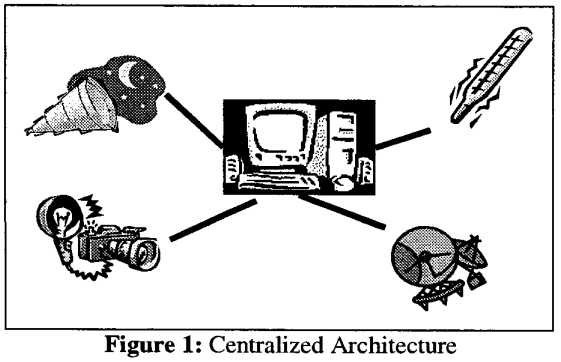

Figure 1: Centralized Architecture

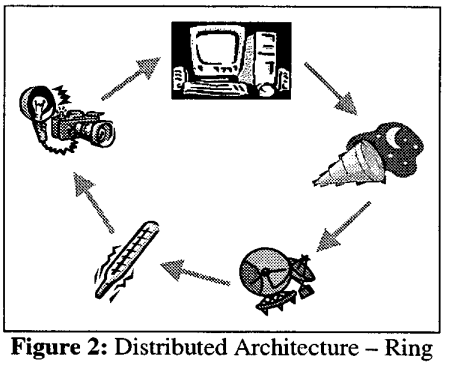

Figure 2: Distributed Architecture – Ring

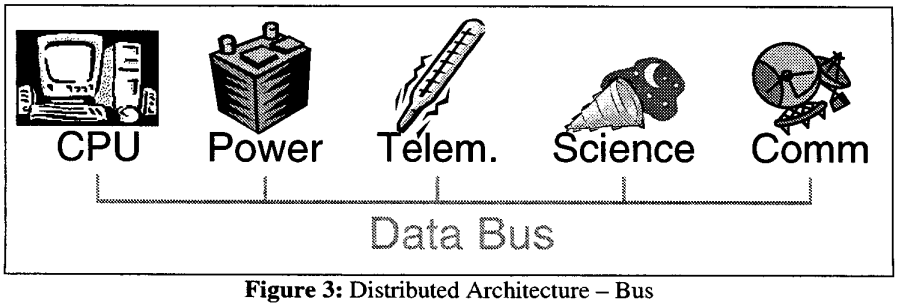

Figure 3: Distributed Architecture – Bus

[1] For further reading see *Space Mission Analysis and Design*. [4]

4. EMERALD OVERVIEW

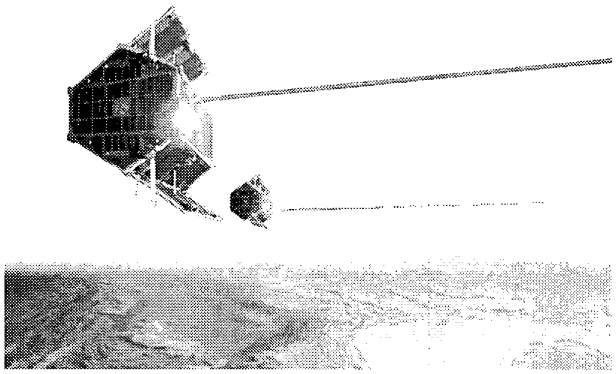

Figure 4: Artist's Rendition of Emerald in Orbit.

The EMERALD Nanosatellite project is an entirely student managed, designed, built, and operated project to build 2 small satellites to launch in 2001. EMERALD is a joint effort of the Space Systems Development Lab (SSDL) at Stanford University and the Santa Clara Remote Extreme EnvironMent lab (SCREEM) 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 universities were chosen to build ten 10-15kg nanosatellites and each was given $50,000 per year for two years plus a Space Shuttle launch. The goal of this program is to use commercial and new technologies to perform creative low-cost experiments as a stepping stone for future space missions.

Emerald's mission is to explore Robust Distributed Space Systems. It will experiment with new space mission architectures in which computing, control, and data management are distributed through out a single satellite, and across a formation of closely flying satellites.

The Emerald mission goals are to:

1. Explore the integration and operations effects and possibilities of a distributed architecture.
2. Test the robustness of such an architecture
3. Validate the usefulness of distributed space systems by investigating Very Low Frequency (VLF) radio emissions from lightning-induced upper atmospheric phenomena. [4]
4. Test the robustness of individual electronic and electro-mechanical components in the space environment. (MicroElectronics Radiation Inflight Testbed or MERIT)
5. Test the on-orbit performance of an advanced Colloid microthruster. [5]
6. Verify subsystem-level technologies necessary for future formation flying missions. This will include:
   - using low-power differential Global Positioning System (GPS) receivers for sub meter level relative position sensing
   - inter-satellite communication for GPS and on-orbit inter-satellite control
   - simple drag-panels for coarse relative position control.
   - integrating the operation of these payloads in order to experiment with simple closed loop relative position control. [6]
7. Extend low-cost satellite development techniques critical to fielding multi-spacecraft fleets.
8. Investigate aspects of autonomous operations for a distributed, multi-satellite mission.

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 university laboratories such as Stanford's Space and Radioscience Laboratory (STAR lab), Plasma Dynamic Laboratory, and the Aerospace Robotics Laboratory.

The bus design for the Emerald spacecraft will be based on Stanford`s Satellite Quick Research Testbed (SQUIRT) microsatellite design.[7] Specifically Emerald's satellite bus consists of:

- a 15 kilogram structure in a modular 18 inch diameter hexagonal configuration
- 9600 bps, half-duplex communication in the amateur UHF band
- body mounted high efficiency GaA solar panels, with a 5/12V bus
- Passive magnetic attitude stabilization
- and simple, passive thermal control.

Currently Emerald is building and integrating its engineering model in hopes of having flight hardware completed and ready for testing during the second half of 2000.

For further details on the project, please visit the EMERALD website at http://ssdl.stanford.edu/Emerald/.

5. EMERALD DISTRIBUTED COMPUTING DESIGN

On Emerald, we hope to test as many of the advantages of distributed systems as possible. Yet, the architecture has to be simple enough to fit within the tight time (2 years total until delivery), mass (15kg each), volume (1.7 cu. ft.), and power constraints imposed by the mission. But, it is also desired to develop a system that would be readily adapted to future missions. The goal of the SQUIRT program is to develop satellites on a one year time frame. Hopefully, the standardized interfaces and subsystems from Emerald will allow the next SQUIRT to meet this goal by plugging together the desired pre-developed bus subsystem components and adding the appropriate experiments.

Architecture

To maximize integration ease and flexibility, the Emerald distributed architecture is built around a standard serial data bus with well defined interfaces. However, certain subsystems (Comm and GPS) required direct, non-standard connection to the CPU. The resulting hybrid configuration combining a bus topology for most subsystems with a centralized topology for Comm and GPS is shown in Figure 5.

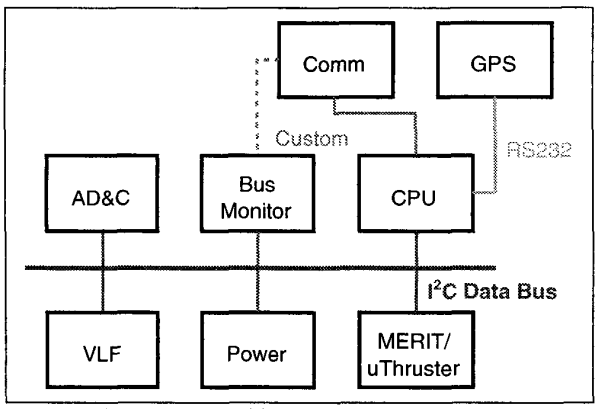

Figure 5: Emerald Data Bus Architecture

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. There is no reason to attempt to insert an I2C converter only to have to convert back again. For the GPS, the RS232 format would not be hard to convert to I2C, however, during relative position measurements, the GPS system will be sending and receiving data from the other satellite(s) continuously. Instead of cluttering up the I2C bus with this data, it 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 of the subsystems on the I2C bus, is more than just a passive input/output device. The subsystems are "smart" in that each contains its own microcontroller to provide application specific higher level data collection and processing. Smart subsystems allow for "high-level" command interfaces. For example, instead of having to ask the AD&C system every second or so for data, the CPU can just 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. By abstracting out these functions to such a high level, the CPU does not have to worry about how the subsystem is going about a certain task, allowing the CPU to focus on more relevant things. Such abstraction also allows hardware modifications to be more transparent.

Data Bus (I2C)

The specification for the data bus requires careful consideration since it forms the backbone of the distributed architecture. There are numerous existing standards for both high and low level aspects (and everything in-between) of distributed computing systems, especially in distributed industrial control applications. [8][9][10] However, these standards typically assume computational and power resources not available in small satellites. Therefore, we looked at simpler communication protocols, typically used at the chip and board scale (instead of the room or facility scale). This allowed us to meet power, voltage, and computational constraints, but required designing some of the higher level details from scratch.

Protocol Selection— We decided to only consider synchronous serial protocols since 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 exeriments. We looked in more detail at three simple protocols commonly used in embedded systems:

- *Controller Area Network (CAN):* Used extensively in automobiles and also in industrial control applications, but seemed overly complex for our needs. Historically, CAN has been targeted toward the industrial scale user; however; increasing commercial availability of stand alone CAN chips and microcontrollers with integrated CAN support is beginning to change that.

- *Serial Peripheral Interface (SPI):* Simple and widely used, but doesn't scale well to arbitrarily sized networks since it requires additional wires for address lines. Also, multi-master support requires extending the baseline specification.

- *Inter Integrated Circuit (I^2C):* Used in Audio/Visual equipment, on PC motherboards, and in "smart" batteries. Easily scales to networks of arbitrary size, including built-in support for multiple masters.

For these and other reasons, Emerald chose to use the I2C bus. Conveniently, there are many simple, off the shelf devices available for I2C, (digital I/O, A/D converters, EEPROM, microprocessors, etc.)

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. Faster modes (400kbps and 3.4 Mbps) are also specified, but fewer devices support these modes. The protocol is specified to a fairly high level from reading and writing to multi-master support and arbitration.

Both lines are connected wired-AND which allows for arbitrary numbers† of devices and to 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 arbitary number of 9 bit "words." These words are 8 bits of information (supplied by the current "transmitter") plus an acknowledge (from the "reciever). 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 "reciever."

For more information and details, see the I2C specification [11] or *The I2C Bus from Theory to Practice* [12].

High Level Messaging— I2C standardizes many layers of the communication protocol, from electronic signaling to multi-master arbitration and simple data acknowledgement. However, it does not specify any data integrity checks; so, Emerald developed a simple, error checking message format.

Specifically, commands coming from the CPU (or any other master) are fixed to 5 bytes in length, plus a field for a return address and CRC (cyclic redundancy check, for error detection). Reply messages include a field for length plus a variably sized data portion and finally a CRC byte for error detection. These message formats are shown in Figure 6.

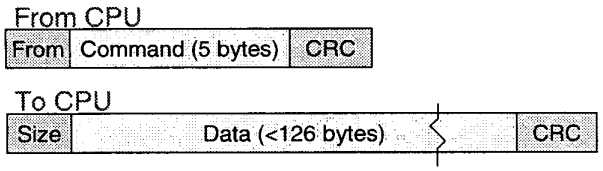

Figure 6: Emerald Message Format

Additionally, certain commands are standardized across subsystems. These include functions for checking subsystem status, synchronizing time, and querying the subsystems for a list of defined commands. This last feature is very important since it allows subsystems to change and expand without requiring code changes in the main, coordinating CPU.

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.

† 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)

In many ways, this multi-layer configuration is analogous to similar to the configuration chosen by JPL for the X2000 data bus. X2000 uses IEEE 1394 instead of I2C for data transfer and I2C instead of Dallas 1-wire for telemetry and control. [13]

The 1-wire protocol was chosen because:

- 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 [14]

Dallas 1-wire allows an arbitrary number of temperature telemetry nodes by simply connecting all of the sensors to two shared wires. Furthermore, standalone analog to digital converters (DS2450) are also available, allowing other types of telemetry (such as bus voltages and currents) to seamlessly connect to the same bus.

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

Baseline Subsystem Processor (PICmicro)

To eliminate the overhead and duplication of each "smart" subsystem developing its own computing hardware and software, the Microchip PIC16C77 was chosen as a baseline subsystem microcontroller.

Having a standard processor allows low-level hardware and interface functions to be developed once and then shared across the subsystems.

Note that for the Emerald architecture to work, it is not necessary that all subsystems use the same processor. In fact, both the VLF and radiation testbed experiments are considering upgrading to alternative processors. But, since the interface to the I2C bus, including most of the experiment specific commands, remains unchanged, the rest of the satellite will not notice the change.

PIC16C77 Micro Controller—After considering other options (68xx, COP, 8051),the PICmicro microcontroller family was chosen as the baseline for subsystem processors. Specifically the PIC16C77 was chosen because it

- is a single *chip* computer, only external hardware required is a clock source. (40 pin package 8kwords ROM, 384bytes RAM)
- is toward the top a wide and diverse family (from 8 to 68 pin, with 512 to 8k ROM and 25 to 384 bytes of RAM)
- Has many built-in peripherals such as 8 channel A/D, I2C, UART, and more
- Consumes little power, <100 mW at max speed (20Mz) and has a low power sleep mode (<100μW)
- Supports a secondary crystal time base that runs even in sleep mode to provide accurate time data.
- Has a close relative (PIC16C73) that has been used in research for fault-tolerant computing on small satellites.[15]
- Has readily available inexpensive (or even free) development tools including a device programmer, C compiler and demo board each available for under $100
- Is used (along with the 68HC11) in the graduate mechatronics course at Sanford (ME218)
- Has a simple "RISC" instruction set allowing for fast efficient execution.

One of the major short comings of the PIC, lack of support for external memory has been worked around using a simple custom SRAM interface capable of storing up to 2.5Mbytes of data for the experiments. Other, less memory intensive subsystems will rely on an external serial EEPROM for storage.

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

- I2C including high level protocol
- RS232 for debugging
- A/D converter at up to 20kHz
- Timer Control for synchronizing fast events
- Real Time clock base for slower events, timestamping and scheduling.
- External memory (SRAM, EEPROM) support
- Simplified I2C command to associated function call interface.
- Standardized help command to query for valid function names.
- Standardized, run-time configurable "loop" structure.

6. Emerald Distributed Computing Experiments

This section describes the large range of distributed computing experiments and technical demonstrations we hope to conduct with Emerald. It should be noted that while the technical details of these experiments are specific to the Emerald architecture, the concepts generalize to other modular distributed computing architectures.

Integration

Given the ambitious schedule, large number of experiments, and student environment, improved integration is not just a nice idea, but a necessity.

The serial data bus architecture should make it very straightforward to add subsystems and experiments to the system as they become available. It should be as easy as plug and go. And 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.

Furthermore, since the PICmicros can be queried for a list of valid commands, the main CPU doesn't have to know anything about the individual subsystems. On startup, the main CPU compiles a table of subsystem commands and their parameters by asks 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 also allows other devices, such as a desktop PC to connect to the data bus to simulate non-existant nodes. In the simplest case, such a PC node can be used to fully test and debug a subsystem, standalone, 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.

The data bus architecture also enables "Virtual integration" across internet. With virtual integration, two physically separated data bus fragments are connected together using a PC LAN (Local area network) or the internet. To the devices on either fragment, all of the data bus nodes, including those on the other fragment, appear as though they were local. This requires a PC node to be connected as a messenger on each network fragment. Messengers from the physically separated data buses would then exchange data across the internet, forming a bridge between the data buses. Virtual integration will allow the Emerald team to conduct earlier and more frequent integration tests between subsystems being developed at the two schools.

Figure 7 shows virtual integration using the Internet, with a PC standing in for the CPU node.

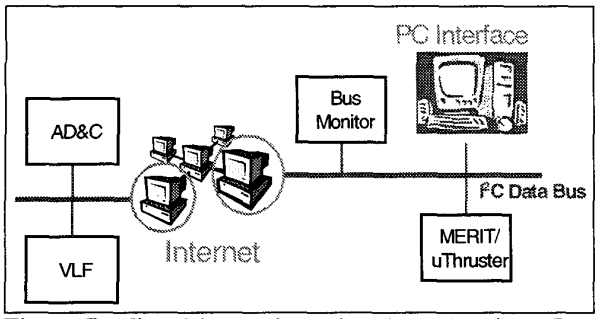

Figure 7: Virtual integration using the web, with a PC standing in for the CPU

One of the major challenges to implementing both virtual integration and stand-alone testing is time: PC serial interfaces are typically slower than the 100kbps of I2C, especially when using an inexpensive, commercially available I2C to RS232 converter. And over the internet, it gets worse. When connecting from on campus (T1), the data rate is typically not the limiting factor . However, the roundtrip packet time is often on the order of a few msec, even for nearby computers, requiring some clever buffering and command handling.

The minimum success criteria for this part of the experiment is to conduct one virtual integration session, and (can be independent) have a PC successfully stand-in for the CPU during some portion of integration.

Virtual Data Bus

Just as the internet was able to bridge two networks 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. Such a configuration opens up a whole collection of operations possibilities, such as:

- *Fault Tolerance.* Should a satellite's CPU fail, the entire satellite may not be lost. Smart subsystems on the data bus may be directly commanded from the ground through a direct link from the communications system to the data bus. Since the subsystems recognize high level commands, a few command bytes is enough to run a potentially complicated proceedure. On Emerald this is enabled by a direct connection from the Communication system to the Bus Monitor and from there to the I2C bus.

- *On-orbit Fault Tolerance.* In a multiple satellite system there is no reason that direct communication system to bus commands have to come from the ground. Instead, as seen in figure 8, 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.

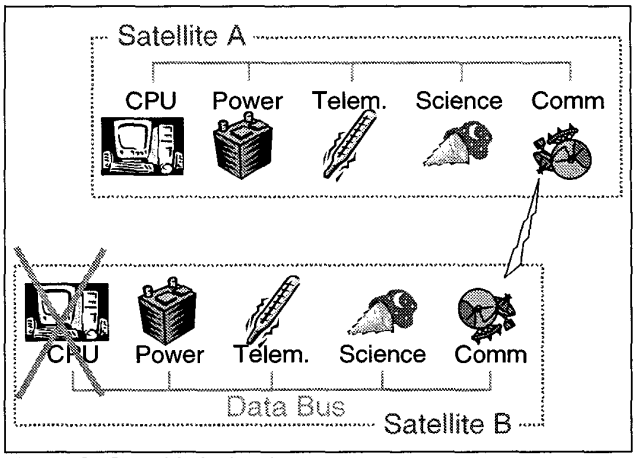

Figure 8: On orbit fault tolerance

- *Inter-satellite resource sharing.* In a multiple satellite configuration, the virtual data bus also allows sharing resources multiple satellites have in common. This can be used both for fault avoidance or as part of standard operations. For example, a spacecraft with anomalous power levels, could transfer data to another 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.

- *An orbiting network.* Taken a bit further, the CPU and high powered down link communication system could be launched as one core satellite. Then at a later date launch a delayed experiment or many such experiments, as additional satellites. On orbit the instrument satellites would communicate high bandwidth data through the core satellite to the ground.

As with virtual (internet) integration, one of the major challenges for implementing a virtual data bus is time. The inter-satellite communication link is (typically) 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 ends, but

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.

The minimum success criteria for this portion of the experiment is to send and verify one direct ground-to-subsystem command, bypassing the CPU.

Bus Monitor

Another portion of the on-orbit experiment is determining how well the I2C based data bus performs in real life. To this end, a passive "bus monitor" will be flown on both Emeralds to collect meaningful statistics on the data bus throughout its mission. Possible statistics include: Total data throughput, #messages in a given period, number of errors, etc. The bus monitor will time tag its data so that the data can be correlated to operation mode. An extension to this project would actively monitor the Dallas 1-wire power switching network to determine which subsystems are on at a given time. In the spirit of "smart" subsystems, these statistics will be stored internally until requested for download by the CPU.

The minimum success criteria for this portion of the experiment is to Collect and download single set of bus statistics.

Distributed System Autonomy

The Emerald mission will provide several demonstrations of advanced and cost-effective autonomous operations techniques, which are enabled by the distributed architecture.

Distributed Beacon—The Emerald vehicles will carry an enhanced version of the beacon-based health monitoring system that has been incorporated into the Sapphire and Opal spacecraft. A basic beacon-based health monitoring system is composed of an on-board software production rule system and a transmitter capable of broadcasting low data rate tones. This system determines and periodically broadcasts a very high level health status message. These broadcasts are received by a network of low-cost automated receiving stations developed by SSDL. The stations forward the health messages to a central mission control complex which automatically pages an on-call operator in the event of a vehicle anomaly. Initial experimentation has shown that this system is capable of drastically lowering the cost of nominal health monitoring. [16] The Emerald enhancements to this system will include a) the use of more robust model-based health assessment techniques, b) an inter-satellite beacon capability, and c) a single space segment level beacon broadcast to ground.

Science Execution—In addition, an on-orbit intelligent execution system is being developed for the VLF science payload. This system will provide synchronized control of the VLF systems on each Emerald satellite thereby allowing 'space segment level control' in which a single ground command initiates collaborative actions on both spacecraft. In addition, the ability to detect unplanned opportunistic VLF science events is being developed. This will allow the satellites to detect such events on their own and to subsequently coordinate data collection activities on their own. Additional capabilities involving on-orbit science data processing may also be explored.

Migrating Position Control—Finally, an autonomous ground based navigation control system will be used to command satellite positioning when the on-orbit system is not functioning. This may occur do to component failures, power limitations, or because the vehicles are out of range of the intersatellite communications system. In the current design, the enhanced beacon system may be used to indicate the status of the on-orbit navigation system. Based on this information, the ground-based system will engage itself in order to compute and execute position control commands. [17]

7. CONCLUSIONS

Distributed computing on Emerald is supported by a modular bus architecture, inter-satellite communications and smart subsystems. This architecture, like any engineering design, has its trade-offs. Below, these trade-offs are presented in the context of a general satellite system.

Advantages

During ground based subsystem development, integration, and testing, the modular, distributed architecture enables:

- Stand-alone subsystem design and testing.
- Incremental integration, as subsystems become available, even if traditionally crucial subsystems, such as the CPU, are delayed.

- "Virtual integration" using the internet to connect subsystems and partial buses from remote locations.
- Simplified late subsystem addition, and minimized effects of removing a subsystem that is not ready on time.
- Transparent subsystem redesign and expansion.
- Future satellite missions to directly re-use existing modular subsystems, allowing designers to focus on mission specific hardware.

On orbit, it enables:

- subsystem storage and computing resource sharing.
- built-in redundancy with minimal additional hardware, for example directly commanding subsystems through the communication subsystem, should the CPU fail.

And when extended to a multi-satellite mission, it enables:

- all of the subsystems from multiple spacecraft to be connected as a single "virtual bus."
- reallocation and sharing of resources between spacecraft
- space segment reconfiguration to adapt to subsystem failures.
- autonomous experiment coordination between satellites
- unique operations scenarios that combine these ideas with the ground segment.

Disadvantages

However, the modular architecture is more complex since:

- It requires some effort to standardize nodes.
- Careful thought is required when setting up the data bus to ensure that enough data gets through fast enough and that there is no lost data.
- Events (such as data storage) from separate subsystems may require synchronization or time stamping.

The modular architecture may also:

- Require duplication of expensive or one-of-a-kind hardware.
- Use additional resources such as mass, volume, and power.
- Introduce additional failure points or modes.
- Reduce performance (or increase cost) for high radiation environments, since there are more nodes to protect either with shielding or expensive radiation hardened parts.

These draw backs may require a longer lead time or higher up-front costs, which may only be justified if the system (or its derivatives) will be used repeatedly, or if there are a large number of units to produce. Furthermore, with careful design most of these disadvantages can be worked around and may even turn into advantages.

8. 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 in SSDL, and SCREEM are thanked their tremendous effort in developing the Emerald spacecraft. Special thanks goes to Lee Boyce, Jim Cooley, Pascal Stang, Caleb Kemere, Vlad Beffa, Greg Hutchins, and the rest of C&DH team for their ideas and help in developing the bus and its components and to Freddy Pranajaya and Zsolt Kiraly for supporting distributed computing as a baseline architecture in SSDL.

References

[1] Dennis Howe, *Free On-line Dictionary of Computing*, 1998 [Available Online: http://nightflight.com/foldoc/, Oct 20 1999].

[2] *Distributed.net: The largest computer on earth*, 1999 [Available Online: http://distributed.net/, Oct 25 1999].

[3] Scalable Spaceflight Digital Signal Processing system, Innovative Concepts, Inc [datasheet].

[4] C. Kitts, "A Small/Micro-/Pico- Satellite Program for Investigating Thunderstorm-Related Atmospheric Phenomena", In *Proceedings of the 12th Annual AIAA/USU Conference on Small Satellites, Logan, Utah*, September, 1998.

[5] M. Cappelli and F. Pranajaya, *"Colloid MicroThruster Research"*, Stanford University Research Report, July, 1997.

[6] C. Kitts, R. Twiggs, F. Pranajaya, Bryan Palmintier, and J. How, "Emerald: A Low-Cost Spacecraft Mission for Validating Formation Flying Technologies" In *Proceedings of the 1999 IEEE Aerospace Conference, Snowmass, CO*, March 1999.

[7] 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.

[8] *Fieldbus Homepage*, Dec 1998 [Available Online at http://cran.esstin.u-nancy.fr/CRAN/Cran/ESSTIN/Fieldbus/page_norm.html#Fieldbus, Dec. 17, 1999]

[9] Synergetic Microsystems, Inc., *Online Fieldbus Comparison Chart* [Available Online: http://www.synergetic.com/compare.htm, Dec. 17, 1999]

10 Wade D. Patterson, *VMEbus Frequently Asked Questions*, 1999, [Available Online: http://www.vita.com/vmefaq/index.html, Dec. 17, 1999]

[11] *The I^2C-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]

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

[13] 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.

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

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

[16] Swartwout, Michael A., Carlos G. Niederstrasser, Christopher A. Kitts, Raj K. Batra, and Ken P. Koller, "Experiments in Automated Health Assessment and Notification for the SAPPHIRE Microsatellite", In *Proceedings of SpaceOps '98: The 5th International Symposium on Space Mission Operations and Ground Data Systems, Tokyo, Japan*, June 1-5, 1998.

[17]C. Kitts, F. Pranajaya, J. Townsend, and R. Twiggs, "Emerald: An Experimental Mission in Robust Distributed Space Systems," In *Proceedings of the 13th Annual AIAA/USU Conference on Small Satellites, Logan, UT,* August, 1999.