What's Coming on Spacecraft: Next-Generation Distributed Satellite Bus Information Systems

L. H. Miller, M. M. Gorlick, D. L. Wangerin, C. A. Landauer
The Aerospace Corporation

29 February 2012

Presentation Outline

- “Standard” satellite bus hardware/software architecture
  - *Limiting factors: Weight, power, radiation*
  - *Key characteristics: “Inappropriate” complexity!*
  - *Survivability*
  - *Bus|payload firewall*
  - *Reminder: Terrestrial state of the art*
  - *Limitations*
- Distributed satellite bus hardware/software architecture
  - *Research goals*
  - *Related work*
  - *Software*
  - *Inter-device communications*
  - *Software architecture*
  - *Research approach: Distributed satellite bus architecture*
  - *COAST—COmputAtional State Transfer*
  - *Future work*

Space environment: Critical characteristics/concerns

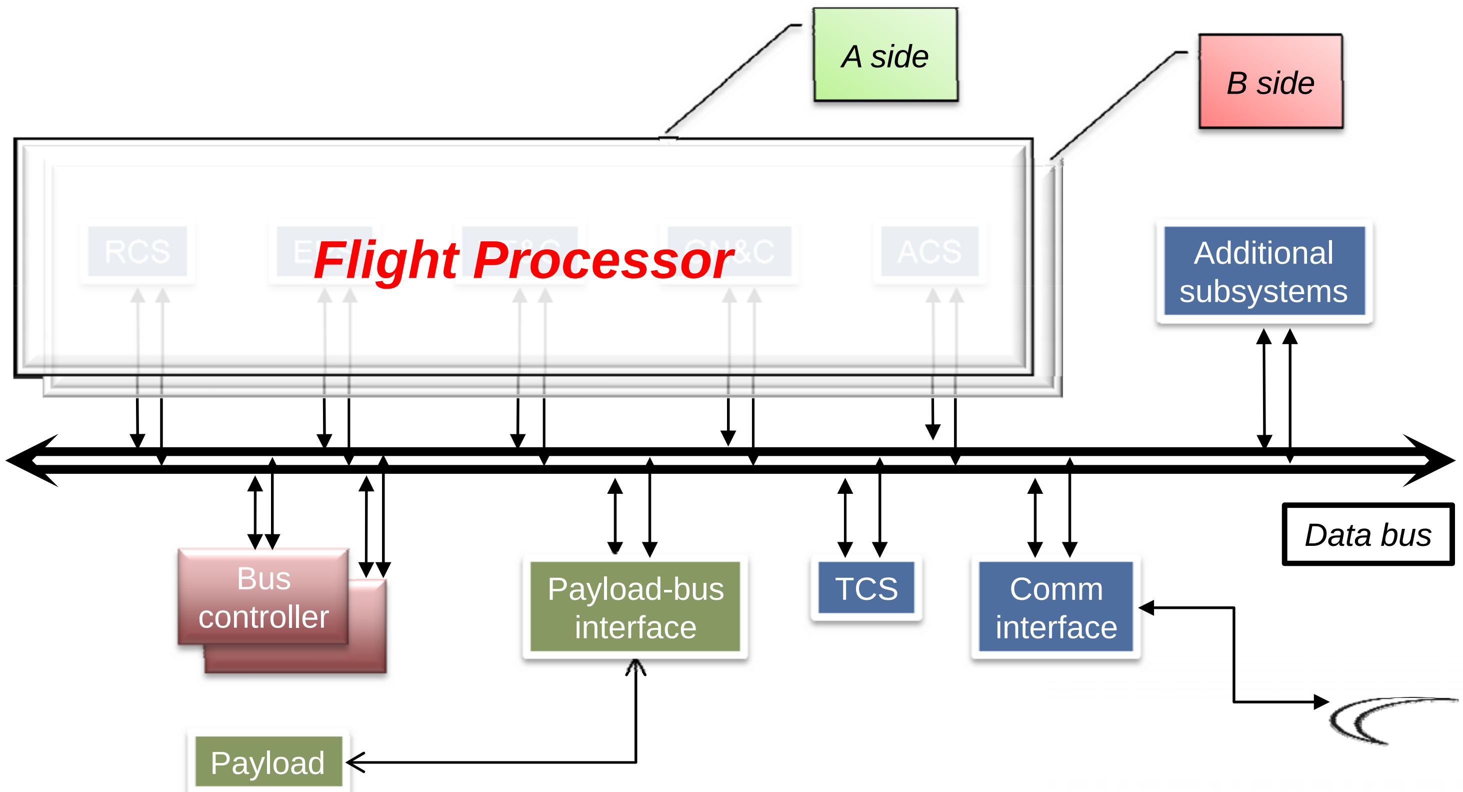
- Key limiting factors: Weight, power, thermal, radiation, processing power, memory, repairmen
- Critical concerns: Autonomy, security, availability, survivability
- Bus|payload firewall
- “Inappropriate” complexity!

“Standard” satellite bus responsibilities

- ACS – Attitude Control Subsystem
- TCS – Thermal Control Subsystem
- TT&C – Telemetry, Tracking & Commanding
- RCS – Reaction Control Subsystem
- EPS – Electrical Power Subsystem
- GN&C – Guidance, Navigation & Control
- Comm – Communications
- FMS – Fault Management Subsystem
- Bus management & control
- Payload interfaces

And these are just the high level responsibilities

“Standard” Satellite Bus Hardware/Software Architecture



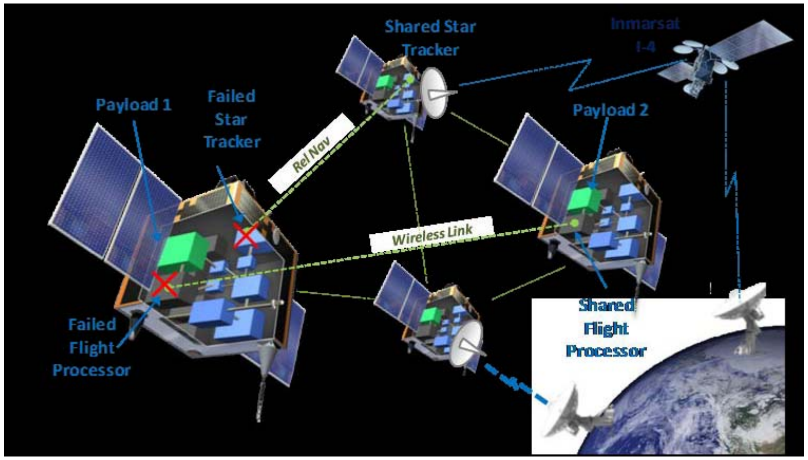
Real satellites tend to be far more complex, with corresponding software complexity

Research Goals

- Overcome “Key Limiting Factors” described on slide 3
- Provide a simple, systematic, understandable, and verifiable approach to “Critical Concerns”
- Provide a unified approach to both bus and payload processing
- Simplicity through commercially available, standard parts

Background: Related Work

- <u>DARPA F6</u>: “Fractionated” spacecraft with functionality distributed across a cluster

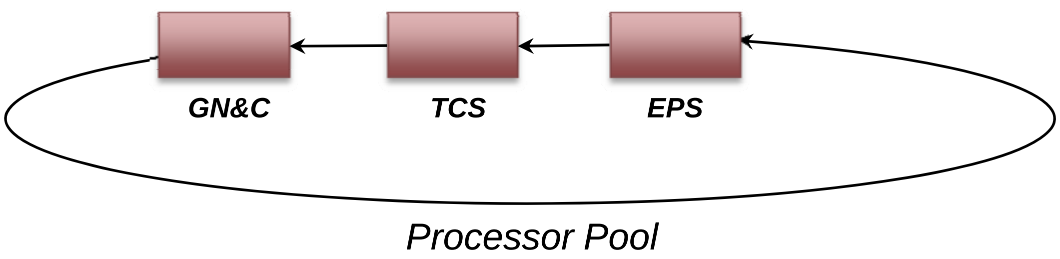


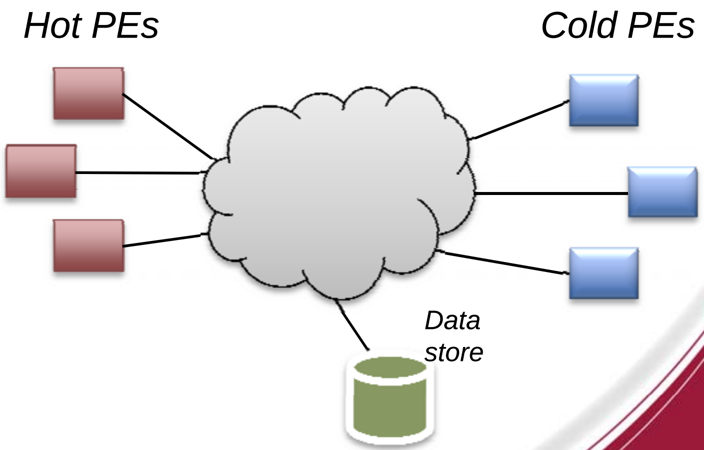
- <u>ORS</u> (Operationally Responsive Space): Fast[1] turnaround from concept to launch
- <u>PnP</u> (Plug and Play Satellite): Construction from standard parts
  - *“The era of the huge military satellite programs that cost tens of billions of dollars appears to be over.”*

[1]Three tiers, with Tier 1 providing capability from minutes to hours.
[2] *Defense Industry Daily*, Jan 28, 2010.

Research Approach: Distributed Satellite Bus Architecture

- Pool of processors connected through Ethernet
  - *Every device/box/subsystem talks IP/TCP*
- Tens to 100s of processors
  - *Powerful, cheap, commercial quality*
- Redundancy through fast reassembly
- Mobile code, zero latency, inherently survivable





Inter-Device Communication – Distribution, Redundancy

- Ethernet-based TCP/IP
  - *Endpoint interconnects between processors and the network constructed from commercial devices*
- Massive processor redundancy
  - *Tens to hundreds of distributed, inexpensive, low-power, Ethernet-ready, commercial micro-controllers*
  - *If several processors die we simply don't care*
- Advantages of a distributed spacecraft bus include
  - *Eliminating processors and buses as single points of failure*
  - *Ample reserve processing power*
  - *Eases recovery due to failure of spacecraft bus peripherals*
  - *Applying computational resources to compensate for the shortcomings or degradation of bus peripherals*
  - *Simplifying physical devices*
  - *Enlarging the design space for spacecraft buses*

Further Advantages

- Reductions in cost and schedule for spacecraft development and integration
- Encouraging more generic, highly modular, spacecraft buses
- Reducing spacecraft weight and mechanical complexity by eliminating custom wiring harnesses and replacing them with standard PoE (Power on Ethernet) cabling— integrating into a single cable power distribution and network communications
- **Reducing the development cost of spacecraft bus software**
- Employing open source software for spacecraft for including operating systems, cryptography, and application libraries
- Encouraging the development of standard network command and control interfaces for common spacecraft bus devices (such as gyroscopes, reaction wheels, sun, star, and earth sensors)

COAST Architectural Style

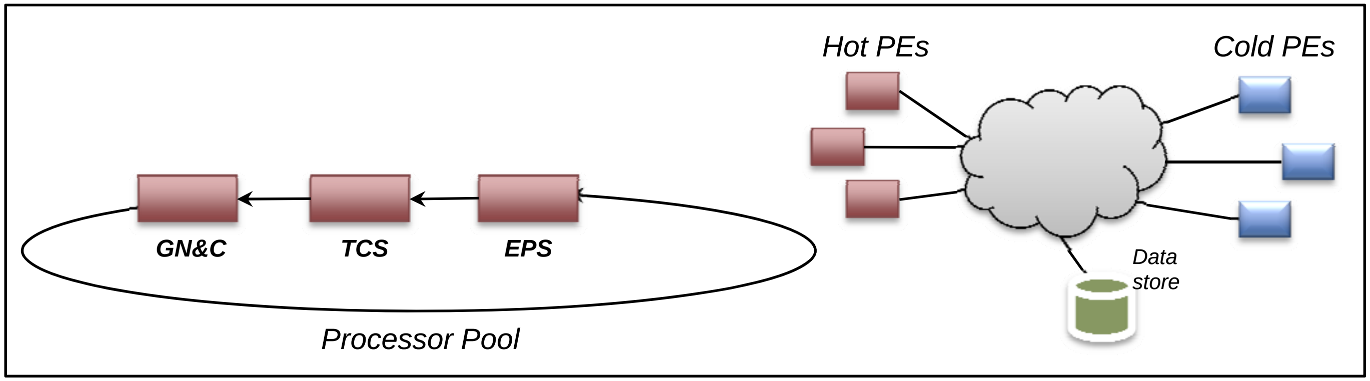
- COAST supports:
  - *Self-healing – Capable of detecting faults and recovering autonomously*
  - *Hot update – Changes applied without halting system execution*
- System properties supporting these include:
  - *Rapid transfer of executing computations between processor s*
  - *Fine-grain update of running software*
  - *Isolation for safety and robustness*
  - *Inexpensive, dynamic, setup and teardown of subsystem software*
  - *Multiple, simultaneous execution of subsystems and applications for hot failover without loss of critical state, data, or execution continuity*

How It Works

- Motile — mobile code language
- Island — peering infrastructure for computation exchange
- Motile programs move from island to island on demand
- Powerful security and safety mechanisms built into the language and core infrastructure
  - *Capability-based security everywhere always*
  - *Impossible to circumvent*
  - *Mobile computations may be restricted in time, space, function, and authority*

Application to Spacecraft Bus Hardware

- Canonical spacecraft bus architecture (from slide 9)



- Bus processes distributed to general purpose processors
- Processors are monitored through well known heart beat or similar means
- On processor failure, processes migrate to processors from the processor pool
- State maintained in replicated data stores
- Design supports simple, safe software upload process
- Conceptually simple, elegant, fault tolerant
  - But some really tough engineering to make it all work

References

J. Erenkrantz, M. Gorlick, G. Suryanarayana, and R. Taylor. “From representations to computations: the evolution of web architectures,” in *Proceedings of the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 255–264, Dubrovnik, 2007.