Architecture for Safety *(preliminary)*

for non-trivial high quality Real-Time Embedded Computers

Contents

1 Abstract

A good safety oriented architecture can increase confidence, reduce costs, and schedule for non trivial real-time systems. Performance and latency timing are especially key requirements for safety critical real-time systems. “Integrated Modular Avionics (IMA)” A is a proven architecture which guarantees performance and latency timing through strict memory and time management. IMA gives platform specific architects the tools to deliver specific platform requirements. IMA also gives platform independence for software modules, allowing the support of a large number of platforms.
In-spite of the high cost rhetoric for these more complex safety oriented systems, many projects became less costly and having better delivery schedules. IMA effectively breaks complex systems into easily manageable software applications, allowing much more functionality within a system.
Clear authority line and determinism are key to success on these systems. All potential impacts can be quantified and tested, or if possible eliminated. Quality and performance shall be addressed continually, but architectural visibility is critical. Testing is performed at every level during the product lifecycle, even when development is done.
All development processes benefit from a good architecture, including Agile, Waterfall (V-Model), and especially the traditional processes. The architecture presented here came from the traditional development process, hence you may need to adapt accordingly. B

1.1 Identification

June 26, 2013
Ty Zoerner, Infinite Delta Corp, `www.InfiniteDelta.com`

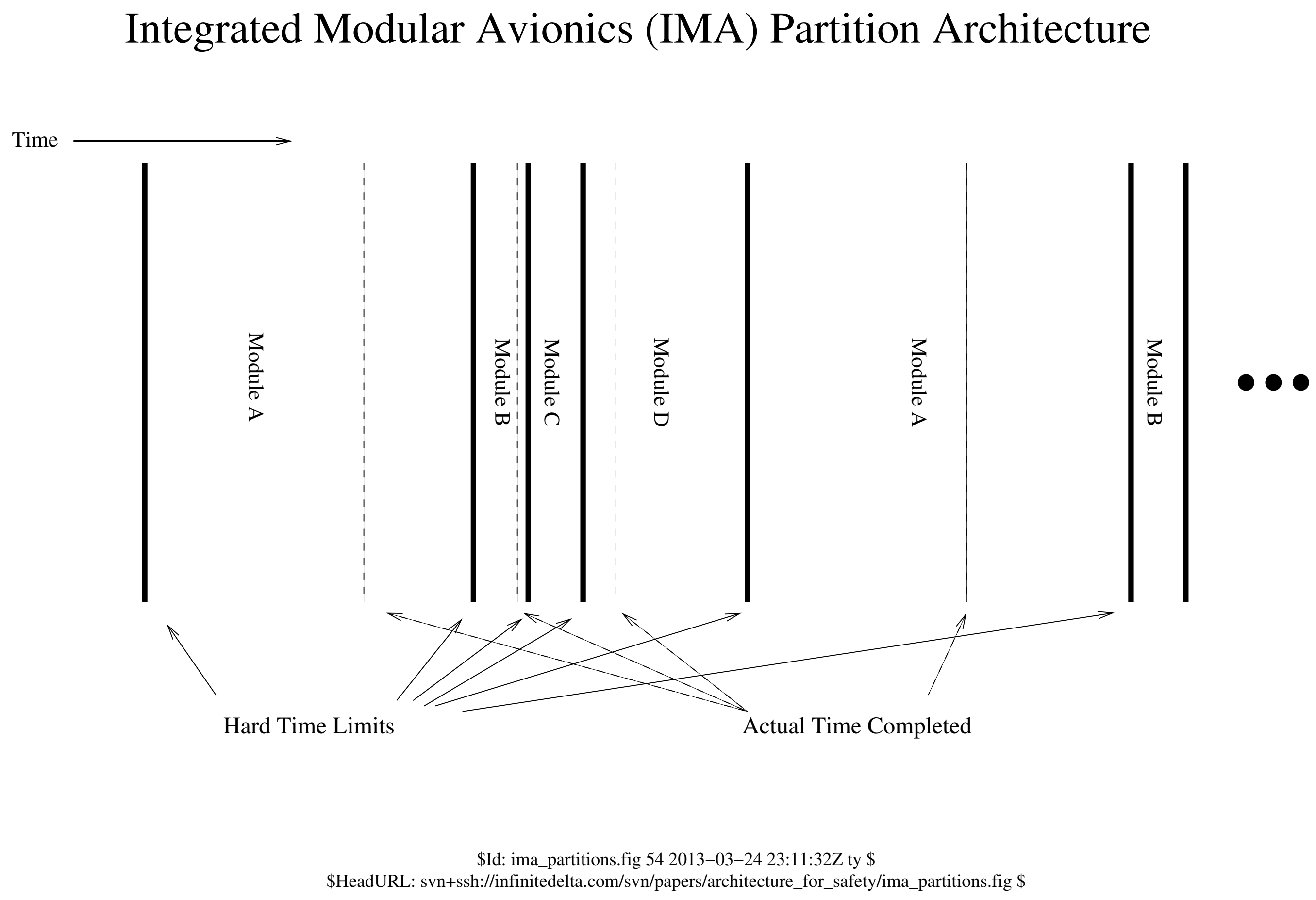
`$Id: architecture_for_safety.tex 58 2013-06-26 22:29:11Z ty $`
`$HeadURL: svn+ssh://infinitedelta.com/svn/papers/architecture_for_safety/architecture_for_safety.`

2 Introduction

Lessons learned from the complex safety oriented avionics systems can benefit industries with ambitions to manage safety, complexity, accountability, cost, schedule, and life-cycle. The system architecture has the primary responsibility for achieving target objectives. In spite of the high visibility the engineering development process, whether its Traditional (A-Team), Waterfall (V-Model), or Agile.
The avionics community has already and very successfully transitioned from a single vender, single core, single communication bus architecture, to a truly multi-vendor, distributed processor, multi-communication bus infrastructure. The latest Boeing 787 commercial aircraft is the latest example. Proposals such as “Exploring Modular Tickless Prioritized Preemptive RTOS for Avionics”D proposes further optimizations for safety, performance, and system costs.
“Integrated Modular Avionics (IMA)” A is the largest contributor to the avionics architecture success. This paper primarily draws from IMA and includes specific lessons learned to ease development in systems that are both complex and safety oriented.
The medical profession is now starting to become liable for system level “human factors”. The automotive industry can already prevent many collisions and in a few states are considered to support self guided vehicles. The automotive industry is preparing for the anticipated new rules requiring specific collision protections. Agriculture is one of the last to consider the more complex safety oriented systems, largely because of the lack of solid business cases and liability. The good news, is that much of the ground work has already been planted and only needs to be harvested.

3 Integrated Modular Avionics (IMA)

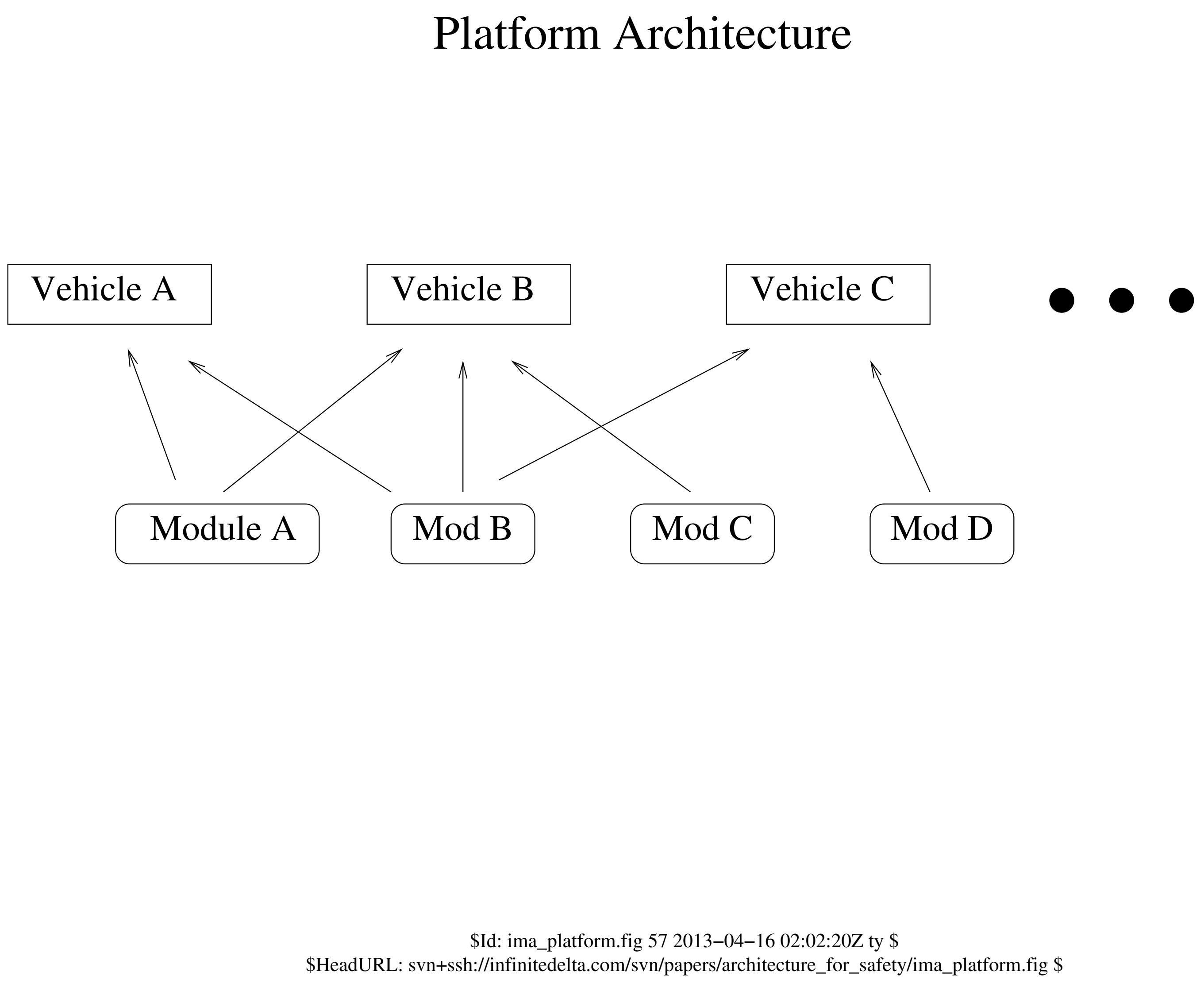
Each IMA software module is guaranteed a specified amount of system resources. IMA imposes time, space, and IO protections for each software module. In fact all resources that can impact the system have tight controls in order to satisfy performance.
Time spent within each module is closely monitored and limited by the operating system. Each of these time partitions effectively have a kernel level "Watchdog Timer" eliminating CPU starvation issues from other modules.


Integrated Modular Avionics (IMA) Partition Architecture
Time
Module A
Module B
Module C
Module D
Module A
Module B
●●●
Hard Time Limits
Actual Time Completed
$Id: ima_partitions.fig 54 2013−03−24 23:11:32Z ty $
$HeadURL: svn+ssh://infinitedelta.com/svn/papers/architecture_for_safety/ima_partitions.fig $

ARINC 653 has become a standard IMA solution, which has a simplified POSIX interface and a rigid/fix partition scheme, however, better performance can be achieved through recent IMA proposals D.

3.1 Platform Architecture

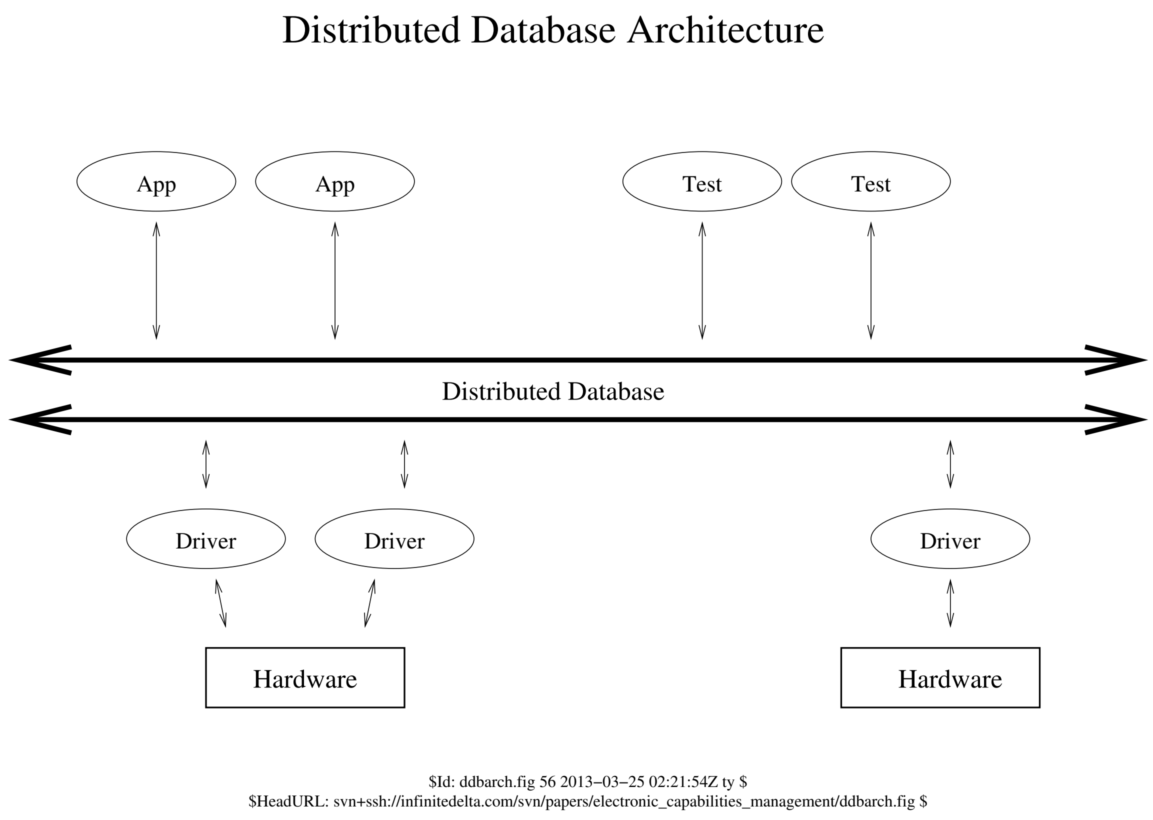
Traditional engineering processes reuses modules, often across many different vehicles.


Platform Architecture
Vehicle A
Vehicle B
Vehicle C
● ● ●
Module A
Mod B
Mod C
Mod D
$Id: ima_platform.fig 57 2013–04–16 02:02:20Z ty $
$HeadURL: svn+ssh://infinitedelta.com/svn/papers/architecture_for_safety/ima_platform.fig $

A multiplatform IMA architecture allows each platform architect to have control over their platform, with minimal impacts to other platforms. Some organizations have achieved less than ten (10) percent software for a new target vehicle.

4 Data Abstraction

Data abstraction planning has been one of the largest gains in distributed processing support and delivery schedules. Applications are forced to communicate through a low latency IPC such as a distributed database:


Distributed Database Architecture
App
App
Test
Test
Distributed Database
Driver
Driver
Driver
Hardware
Hardware
$Id: ddbarch.fig 56 2013−03−25 02:21:54Z ty $
$HeadURL: svn+ssh://infinitedelta.com/svn/papers/electronic_capabilities_management/ddbarch.fig $

Business logic, IO, user interfaces, etc are separated out into their own processes, or modules if an IMA is used. In some cases, even the POSIX kernel interface, has been replaced by one only supporting the approved IPC and low latency system calls.

5 Validation and Verification

Validation and verification is required for all safety critical systems, but can be scaled to benefit most systems. A Test Driven Development (TDD) system captures the essence of code verification. Verification adds the following for each target platform:

- Module level test.
- Partial system level test.
- Full system test.
- Requirements certification.
- Different safety level certifications.
- Inter and intra organization contract completions.
- Liability tracking and authorization.

6 Restricted Development Environment

A good, but restrictive development environment can significantly decrease the cost and extend the potential life of a system.

- Enforces standards.
- Allow software development to run to complete, without the target hardware being available.
- Allows multiple platform verification.
- Minimizes vendor obsolescence issues.
- Enforce complexity rules can keep modules maintainable.
- Supports latency and performance tools.

7 Example

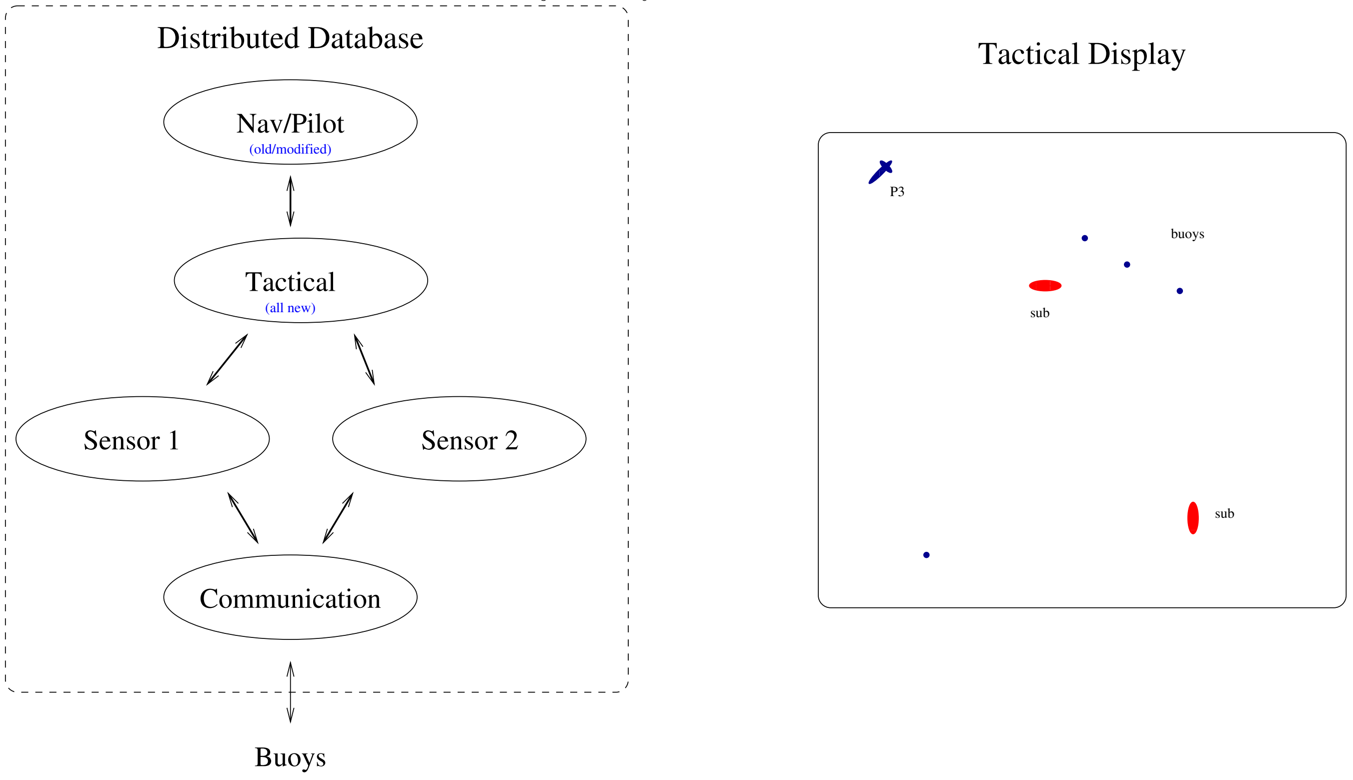
The following examples highlight some avionic architectural benefits.

7.1 P-3 B

Anti-submarine and surveillance aircraft. (1961-present)

7.2 P-3 Avionics Distributed Database

Avionics Distributed Database for five major subsystems:

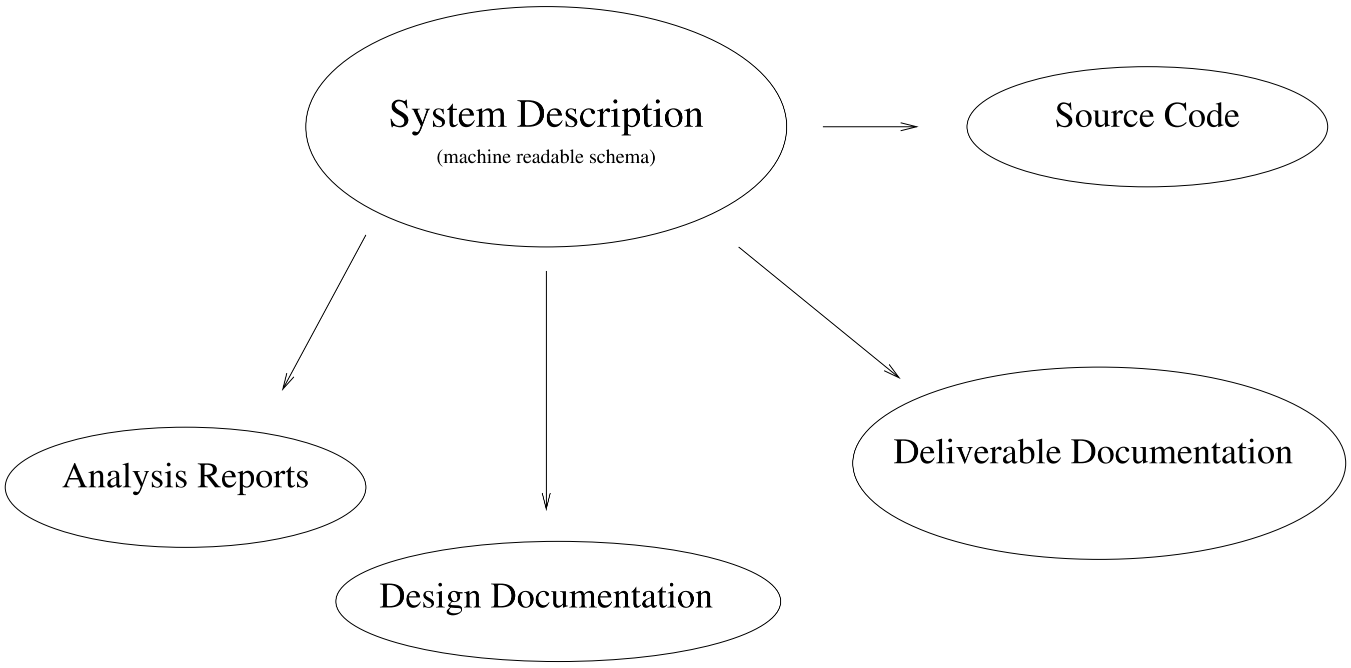


$Id: ddblayout.fig 53 2012−11−08 12:06:26Z ty $

$HeadURL: svn+ssh://infinitedelta.com/svn/papers/electronic_capabilities_management/ddblayout.fig $

- Critical Tactical hardware not expected for a full year.
- Support for many different communication buses.
- Expected high amount of subsystem requirement changes.
- A distributed database paradigm was selected.
- Hardware, and message parsing kept at the architecture level.
- Module and system level verification tests can be developed without hardware.

7.2.1 Distributed Database Process

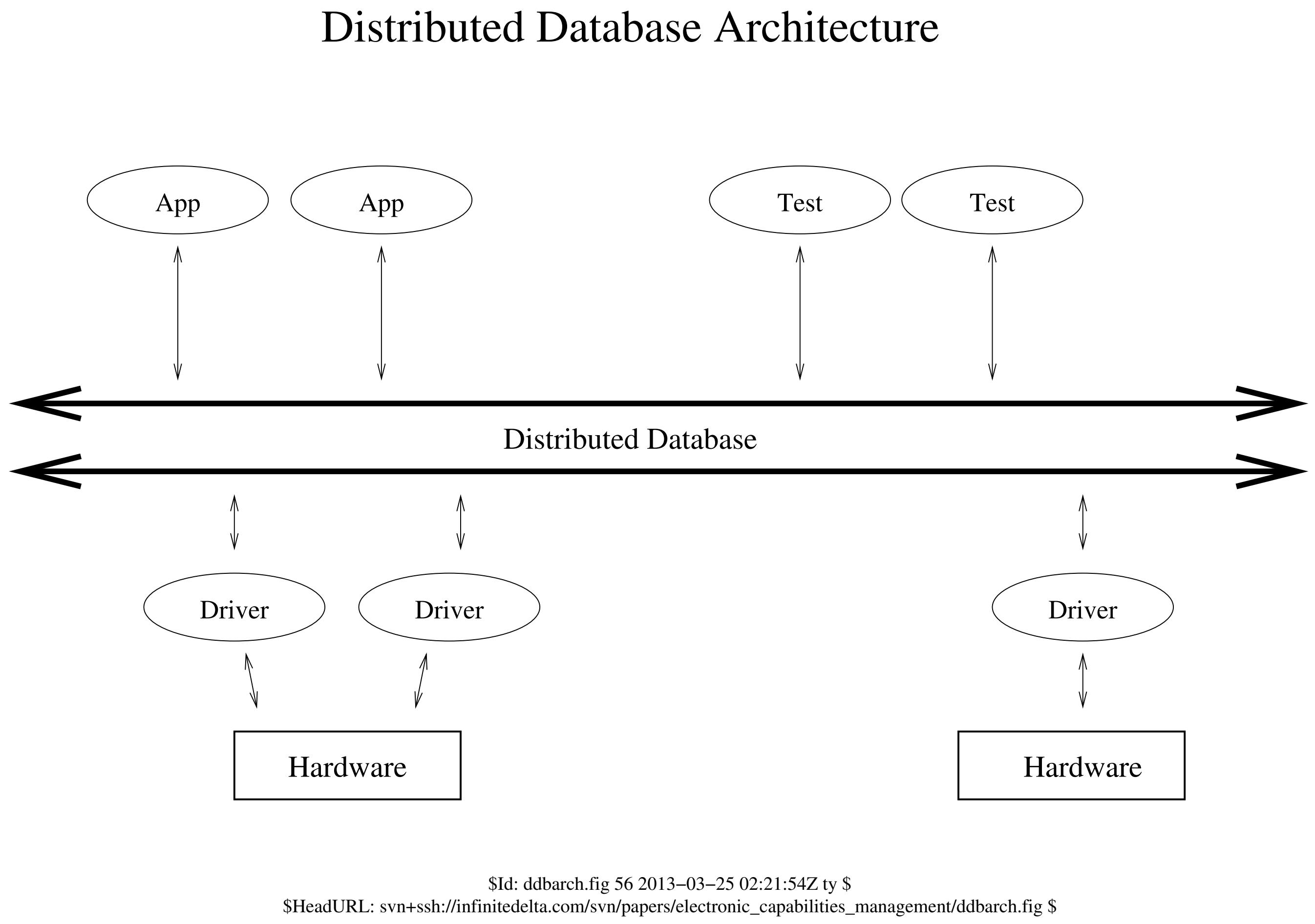


$Id: ddbschema.fig 29 2012–03–01 09:35:58Z ty $

$HeadURL: svn+ssh://infinitedelta.com/svn/papers/electronic_capabilities_management/ddbschema.fig $

- A machine readable distributed data schema drives the entire system layout.
- Meets DO178B determinism requirements for the full system performance.
- Almost eliminates continual integration meetings and the associated developer errors.
- The Distributed Database infrastructure only required 500 hours of effort.

7.2.2 Architecture


Distributed Database Architecture
App
App
Test
Test
Distributed Database
Driver
Driver
Driver
Hardware
Hardware
$Id: ddbarch.fig 56 2013−03−25 02:21:54Z ty $
$HeadURL: svn+ssh://infinitedelta.com/svn/papers/electronic_capabilities_management/ddbarch.fig $

7.2.3 Results

- Generated analysis, critical source code, and documentation.
- Averaged three data layout changes a day for the first six months.
- Systems engineers are now in the driver's seat, allowing the software groups to run.
- Software had necessary resources to meet their commitments.
- Bandwidth was used much more effectively.
- Two weeks of integration of the new Tactical Display processor, 85 percent of the system was operational.
- Project was completed within three years with the combined navigation and tactical systems only requiring 20 man years of effort.

Quality is more often seen then measured. F

8 Planning

Planning the system architecture and infrastructure is the most critical step of a safety oriented system which will be required to support a complex system. Most organizations either purchase a pre-validated architecture and infrastructure, or perform this step very few times. In any case they merely reuse it and enhance the system as they proceed.
Personally, I strongly dislike the pre-purchased solutions. The tendency is to skip critical in depth analysis, lifecycle requirements, and local ownership of the system. Traditional quality F places a high value on ownership and responsibility.

8.1 First time architecture

Clarify all of your current and future application requirements of the architecture. What are their timing, performance, and memory requirements. Typically, even when developers don't know their own specific requirements, they will know what they need from the architecture.
Clarify all of your current and future device driver requirements of the architecture. This needs to include the **what** is needed, not necessarily the **how**. Include all expected hardware failure support that is expected.
Clarify how you are going to quantify the system performance to certify deterministic behavior for critical portions of the system. Include heavy load and worse case requirements to the application developers. These are the worst case test cases scenarios which need to be run on each target system to help quantify the determinism.
Mission Replay needs to be a core architectural decision, including actual control disparities.
Get a sign off from each group, including applicable auditors for a proposed architecture.

8.2 Local Development

Running the applications on the local system is the most productive. The individual development systems must be able to support several simultaneous development environments, in order to support several projects at different points in their lifecycle. These stations must support full regression testing, mission replay testing, along with any module testing.
Strict control over the libraries is required. A two phase linking system can assure conformance.
Properly setup, few actual target systems are required to support many groups, internal and external.

8.3 Failing to plan, is planning for failure...

Architecture above all needs to understand the end vision, and plan for it.

9 Approach

Approaching a potentially complex safety oriented system architecture starts by

- Identifying Risk.
- Investigating the expected lifecycle.
- Determinism.
- What validation is required, and where.
- Define a clear line of authority.
- Define or eliminate all potential dependencies.

9.1 Risk

The core avionics safety specification DO-178 identifies crew activity loading as a very high priority concern for both safety and mission effectiveness. This applies throughout the product lifecycle, especially engineering.
Effective process and architectural effort is required to keep the entire system clean and manageable at all levels. Everyone should be able to understand any portion of the system, with the only a few exceptions where knowledge of specific algorithms are required within an application. Although the safety and validation people can be extremely intelligent and organized, they are required to defend the product to higher authorities.
Hence, make the infrastructure defend-able for the entire product lifecycle.

9.2 Lifecycle

How much and how extensive are the anticipated requirement changes during development and product life? Are there different lifecycles within a product, and how are these supported? Specifically, are there any known planned obsolescence dependencies.

9.3 Determinism

Determinism is another key element for the critical processes. System validation can not even start without an infrastructure that supports a quantifiable ability to complete critical processes within schedule.
Note, that forcing too tight determinism across the entire system is neither required, or desired.

9.4 Validation

Validation can be a large and costly aspect of any project. Validation alone can detour many organizations from considering safety related product considerations. However, early planning can keep validation costs minimal.
Every architectural decision must include an answer to the following question: Could this in anyway affect a safety critical operation? Keeping these answers can greatly help if or when you start safety validation.

9.5 Clear line of authority

Avionics safety has a military clear line of authority organization which permeates down to application level. Clear responsibility is given to each system component including: OS, drivers, infrastructure, and every application. Specifically most interfaces, including user interfaces, are separated from the system business logic. Often, the different components are given levels of scrutiny depending on their assign criticality.

9.6 Dependencies

Minimize or remove all dependencies where possible. Anything that can make your effect your critical applications requires strict scrutiny. Even third party software, back group applications, specific libraries all can impose lifecycle issues. Bus centric designs lock systems down and obstruct system level testing, encourage a record/structure/class abstraction to the infrastructure levels. Minimize the need for system level understanding to support application level development. The "Need-To-Know" paradigm can benefit development productivity. Especially minimize manual steps, preferring the rigor of what a good and complete set of tools can supply.

10 Prepare for an IMA RTOS,,,

Architecture changes are the most expensive, being prepared can have huge savings.

- Prepare as though an IMA RTOS will be used.
- Enforce a strict RTOS application interface.
- Support RTOS constructs such as abort-able processes.
- Move most thread applications to either stand alone processes or abort-able tasks.
- Support a zero-copy communication policy where possible.
- Architect for a distributed system including multi-core, GPU, DSP and FPGA components.
- Understand the limitation of hardware, including cache, shared memory, buses, etc.

Unlike applications, infrastructure changes can be very expensive as a product matures. Normally, it is so prohibitive, that a new product is started instead of supporting the old. Being prepared is critical.

10.1 IMA RTOS

A good IMA RTOS combines the best features of the small embedded RTOS and full up OS's with loadable applications. Prepare to migrate to an actual IMA RTOS by enforcing a strict kernel interface for the applications to use. Even if it entirely out of scope for now, use a library to connect to your current host system.

- Separately linked and loadable applications.
- Each application can be individually certified.
- The specific combinations of applications can be certified as a whole.
- Running of applications is strictly controlled by the managing authority. Do not attempt control or place limits within the applications.
- Strict time and memory limitations are placed on each process.
- Encourages moving away from threads to full up processes.
- Robust and easy to use timing, event, and IPC mechanisms.

10.1.1 Future IMA

An advanced IMA RTOS proposal D which includes:

- Multicore support.
- More rigorous and yet flexible timing and control certification.
- Support for event driven abortable processes. The kernel does not perform a save/restore for a context switch. The application is expected to complete.
- Data and event logging is integrated.
- Multiple levels of security (MLS).
- Embedded distributed database.

11 Detailed Considerations

11.1 Keeping applications very small

Break up requirements and applications as much as possible. Let the OS protect for memory and timing issues.

11.2 Keeping RAM up

Today, more than ever, it is now easy to keep RAM up while the system is effectively off, without coping it off somewhere.

11.3 Minimal or no allocations and Zero Copy

Take advantage of a deterministic design and hardware memory protections. Allow applications direct access to data with limited life cycle.

11.4 Abort-able processes

Most of the older, and some of the new RTOS's support abortable processes. These are process never intended to context switch back into. They are only initiated, static memory is maintains, but stacks are clear and started for each instance.

11.5 Process control

Starting and running applications are a safety concern. If an unauthorized, unlimited high priority application is started, it can kill the system determinism.

11.6 Security distributed operations

Individual symmetric keys allow multiple and independent security on a distributed platform with very little performance loss.

11.7 No strings attached

Do not use strings for anything. Enumerate everything, especially events. Take advantage of GCC's different link sections to strip out the actual Event logger strings in the product. Only record the address in the event log and only parse it back at the lab.

11.8 Embedded Distributed Database

Embedded Distributed Database is a real time lock-less virtual shared memory scheme using a simple data interface. Applications move from a message based paradigm to an event driven database paradigm.

- Each system maintains only data their applications require.
- Infrastructure is responsible for maintaining the database on each system.
- Applications needs only minimal system level knowledge.
- Applications become very portable, testable, and validatable.

11.9 Mission Replay

Mission replay is one of the most valuable tools I've ever encountered, especially combined with IMA and embedded distributed database. Consider when events are fully incorporated with the mission data. Features vary, but can include:

- Ability to repeat both field and simulated test scenarios.
- Develop accurate navigation and vehicle control models.
- Validate individual applications, models, and infrastructure.
- Replay data can be very valuable to many high level user applications.
- Effective commercial test strategy with minimal expense.

11.10 No Change, No regression testing

No regression testing, when executable images are not changed. Minimal development dependencies on other developers. Automate version tracking for post flight diagnostics.

11.11 Modeling

Modeling often slips back into product for flight optimizations. Mission Replay can be used for Model Verification and updating. There is a strong tendency for different groups to create the same model with only slightly different requirements. Hence, trust is lost on most, with further bad visibility to the customer.

11.12 Development process optimizations

Productivity time reporting to optimize development process. How much time are we spending waiting for different things, building, reviewing, etc.
Yes using existing libraries can cut initial development time, however, what is the cost for the product lifecycle. Keeping a clean system seriously helps lifecycle issues.

11.13 Linux as a development environment

Linux as a development and test environment, not necessarily the long term target solution.

12 Summary

Take what you can from the avionic lessons learned. An embedded system architecture designed for safety, can increase quality and still cut cost and schedule, especially as the complexity increases. Expected gains can include:

- System quality and reliability.
- Deterministic system performance.
- Allows true parallel development.
- Predictable engineering performance.
- Better use of talent, teams, and organizations.
- Multiplatform support.
- Multivendor support.
- Liability mitigation and tracking.
- Dependency management.
- Lifecycle support.

Many of these achievements have been very expensive to learn, but are now becoming standard engineering practices. Finally, ignorance of system level safety issues may no longer be accepted in many industries. Recent federal legislation is now adding "human factors" liability to the medical industry.

13 Avionic Safety Definitions and Clarifications

All avionics follow these basic rules regardless of the specific architecture.

13.1 Accountability

A clear line of responsibility is required in every aspect. Every vulnerability has coverage. Architecturally, this requires an extremely clean interprocess interfaces with well defined memory and timing controls and limitations.

13.2 Auditors

Safety, quality, traceability, accountability, etc., are auditors which adds considerable value to the product. Typically, when consulted early, they can prevent major re-work, or certification issues later. They also can be your greatest nightmare when consulted late in a project.

13.3 Dependency Management (IMA)

IMA dependency management is performed at the platform architecture level. Each module has resource requirements, often including other modules. These are resolved at platform architecture level.

13.4 Deterministic

Deterministic is a software validation point stating a time requirement is certifiable. Typically, used regarding intervals or on specific rates. However, it also refers to responses to specific events, and combined system behavior. Example, a serial communications application has a requirement to acknowledge a message within 100 μS. We can verify this application on many systems, however, we can't validate it to be deterministic without understanding, qualifying, then controlling all potential issues that may cause the requirement to fail.

13.5 Integrated Modular Avionics (IMA) A

Moved avionics from a single executable image on a platform to support any number executables running under a deterministic RTOS. Originally, an entire system was certified to a specific safety level, but IMA now supports multiple safety levels and Multiple Security Levels (MLS) E sharing a single platform, minimizing the need for a federated architecture and verification procedures. This can be huge cost and schedule savings for the product lifecycle.

13.6 Mission Replay

Mission Replay allows re-running the part or all of the business logic on the ground. This typically keeps interface logic and user interface separate from the vehicle business logic. This aspect allows the ability to repeat most control failures back at the lab. Mission Replay also supports training, product evaluation, and control model validation. Missions exposing system errors can be included into the test suite to minimize repeated issues.

13.7 Modeling

Avionics vehicle modeling are not only used for aircraft design and planning, but must also be used during flight to support changing mission or weather conditions. Hence, these models need to undergo all of the validation and verification of the flight management systems using them.

13.8 Module (IMA)

An IMA module can consist of any number of software processes and IO resources depending on the specific IMA solution selected. A module delivers one or more requirements to the system. Typically, a platform architect can select from many modules to support a platform. Many, but not all modules can be on multiple platforms.

13.9 Safety Levels

Avionics has several safety levels all revolving around how bad the outcome might be. A DO-178C safety levels C:

Catastrophic - Failure may cause multiple fatalities, usually with loss of the airplane.

Hazardous - Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.

Major - Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries)

Minor - Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change.

No Effect - Failure has no impact on safety, aircraft operation, or crew workload.

We can often, through backups, support higher safety levels then our own system. Example, multiple user interfaces can be rated lower than the system flying the aircraft, since each user interface "is a backup" for the other. This further allows different verification levels.

13.10 Signature (IMA)

A Signature, as used in this document, allows software modules to be loaded and run. Typically under conditions requiring approval. Example, some military aircraft can not fly into commercial airspace. Those that have been certified, can but must follow commercial airspace rules.

13.11 Single Event Upsets

Single Event Upsets (SEU) I are soft errors which either hardware and/or software needs to support. SEUs can occur 100 times more often at higher altitudes, and even 300 times more often for aircraft.

13.12 Traceability

Traceability connects requirements, source code, hardware, and maintenance to as an artifact for the system validity.

13.13 Validation

Are you building the right thing? Many development processes mix or closely associate Validation and Verification. However, for avionics, architecture validation is a non-starter. An architecture shall be validated before the project is started. Architecture validation may include RTOS, communications, system calls, libraries, and device driver interfaces.

13.14 Verification

Are you building it right? Verification is your testing, complete with record-able, and repeatable artifacts. Depending on safety level, it may require independent validation. When possible, avionics abstracts to the data interface keeping IO, specially User IO, away from business logic.

13.15 References

A. Integrated Modular Avionics (IMA)
`http://en.wikipedia.org/wiki/Integrated_modular_avionics`
`http://ftp.rta.nato.int/public//PubFullText/RTO/EN/RTO-EN-SCI-176///EN-SCI-176-04.pdf`

B. Delivering Capabilities through Requirements Management a process investigation
`http://www.infinitedelta.com/wp/electronic_capabilities_management.pdf`

C. DO-178C, Software Considerations in Airborne Systems and Equipment Certification
`http://en.wikipedia.org/wiki/DO-178C`

D. Modular Tickless Prioritized Preemptive RTOS `http://www.infinitedelta.com/wp/avionics_rtos.pdf`

E. Multiple Security Levels (MLS): `http://en.wikipedia.org/wiki/Multilevel_security`

F. Metaphysics of Quality: `http://en.wikipedia.org/wiki/Robert_Pirsig`.

G. Flying Cheap: `http://www.pbs.org/wgbh/pages/frontline/flyingcheap/etc/script.html`

H. Dr. Tom Herald "Affordable Architectures", Oct, 2011
`http://incose.org/chicagoland/library.aspx`

I. Single Event Upsets (Soft Errors) `http://en.wikipedia.org/wiki/Soft_error`