

The Standardization Process: What Works; What Doesn't

Doug Caldwell

Ecliptic Enterprises Corporation

398 W. Washington Blvd, Suite 100, Pasadena, CA 91103; (626) 798-2436 x404

dcaldwell1@EclipticEnterprises.com

ABSTRACT: For many reasons, it is often asserted that various segments of the space community need to adopt standards or need to create standards where ostensibly none exists. Phrases like “build standard spacecraft busses”, “create plug-and-play interfaces”, or “adopt commercial standards” have become almost commonplace utterances. However, such discussions often revolve around inappropriate comparisons with the commercial marketplace or misunderstandings of the standardization process, rather than solutions to clearly stated problems.

Substantial work has been expended on many standardization efforts that have not achieved their intended goals. Various parallel backplane computer bus standards can be used to illustrate how standards arise – or don't. This paper presents three examples of the standardization process. The Futurebus+ (Profile S) standardization activity is contrasted with the adoption of VMEbus and cPCI by the space community. The Ada programming language development, standardization, and subsequent mandate for use are contrasted with the grass-roots evolution of C and C++. The PC/AT bus and PC/104 buses are discussed in their market contexts and contrasted with the closed architecture of the Macintosh.

Finally, a framework for cost-effective standards development based on similarities with free-market entrepreneurial activity is presented.

BACKGROUND

Many discussions about the value of standards to the space community involve vacuous statements regarding the need for standard spacecraft busses, plug-and-play interfaces, or commercial standards. Such statements often reflect beliefs derived from inappropriate comparisons with the commercial marketplace rather than solutions to clearly stated problems.

Efforts to use standards in space applications are nothing new. In the mid-1960's, anticipating shrinking NASA budgets, research started on a general-purpose onboard computer for broad mission applicability to save recurring development costs. This effort led to the NASA Standard Spacecraft Computer, NSSC-1. In the 1980's the MIL-STD-1553B bus, developed for military aircraft, was adopted for use in space missions (e.g., by Cassini, in a heavily modified form, and TAOS).

What is new is that the nano/microsat community is maturing and moving beyond the types of simple missions that can be cost-effectively served by unique, non-standard solutions. Although it is time to start using standards, there are few available solutions that are properly matched to the mass- and power-

constrained applications characteristic of nano- and microsats. In this climate, where many people recognize the advantages of standards, it is tempting to say “we need to create some new standards!” However, without understanding how good standards come into being, it is too easy to waste considerable effort on well-intentioned but ultimately dead-end pursuits.

THE TYPICAL PROCESS

Standards have evolved in virtually every technical endeavor. Standards codify wisdom obtained through the lessons-learned of trial and error pertaining to well-defined problems. Some standards capture sufficiently important wisdom that there is no reasonable alternative; their use has often become mandatory through legislation. Wire your house according to the National Electrical Code lest you destroy equipment, electrocute someone, or burn down the house – and then face civil or criminal actions.

Most standards evolve. Usually, the primary knowledge or technology underlying a standard is initially practiced by a very small group. Over time, the practicing group expands and becomes more broadly known. Sometimes, the technology becomes a *de facto*

standard, despite having no formal recognition (e.g., Philips Semiconductors' I²C-bus) and its originators retain control. More commonly, at some point the group seeks to formally codify its work, either within a recognized standards body (e.g., IEEE, AIAA, ANSI, ISO) or by creating its own trade group (e.g., the PCI Industrial Computer Manufacturer's Group, PICMG, or the VMEbus International Trade Association (VITA) Standards Organization, VSO).

Technology rarely evolves in isolation. Often, other groups are co-evolving similar practices. Sometimes, one group will "see the light" and perceive that another's solutions are better. In such a case, the two groups will naturally merge. But many technical decisions are, in the end, somewhat arbitrary. With enough differences, inertia, and market penetration, there might not be enough similarity to simply merge; despite having much common ground, too many incompatibilities exist. A committee is created. Committee meetings become a forum for wrangling and compromise. A standard evolves that balances many competing objectives: technical, business, marketing, political, legal. Ultimately, language that is generally acceptable to all parties is written and a new standard is adopted (e.g., HDTV). Or not (e.g., Beta and VHS). The system isn't perfect but it usually works.

Sometimes a committee is created to invent a standard where no existing practice precedes it. This is not a good way to start...

A CASE STUDY: FUTUREBUS+, PROFILE S

In the early 1990's, the VMEbus standard had established itself as a viable alternative to the approach of designing mission-specific parallel computer backplane bus solutions. JPL had selected it for the MESUR program (later to be known as Mars Pathfinder) and Spectrum Astro had used it in the MSTI-1 mission. Both of these used single-string primary computers and the flight systems were considered experimental and moderately high risk. Forward-looking members of the space computing community anticipated that the acceptance of a computer backplane standard within the broader community would hinge on addressing the high reliability and high availability needs of space users.

In 1992, as a result of interest in the space community, the IEEE Computer Society / Microprocessors and Microcomputers established IEEE standards working group 896.10 to define the "Standard for Futurebus+@ Spaceborne Systems - Profile S" for the stated purpose that:

Current backplane protocol standards do not address the unique requirements associated with spaceborne processing systems. This standard will establish and specify the requirements (bus, mechanical and electrical) to support implementation of spaceborne Futurebus+ based processing systems.⁽¹⁾

The working group reflected the diverse needs of the space community, drawing its membership from the spacecraft user/builder community (NRL, JPL, TRW, etc.), from the heavy launch industry (Atlas and Titan), and from the avionics and component suppliers (Honeywell, National Semiconductor, etc.).

The Futurebus+ backplane standard (IEEE 896 family) was selected because of its flexible high performance architecture, which supported 32-, 64-, 128-, and 256-bit wide data paths (in the same era that VMEbus supported 8-, 16-, and 32-bit data transfers). The working group defined physical and logical layers to support high-reliability and high-availability configurations. High reliability was supported by a dual-bus configuration to support block-redundant cold-sparing at the board level. High availability was supported by a triple modular redundant configuration to allow real-time operate-through capability. The committee defined a secondary serial bus on the backplane to provide the system manager with a backdoor for reconfiguring the system, and it worked with a companion standards group (1101.4) to define a conduction-cooled form factor.

In 1997, the standard was approved through the IEEE standards balloting process and officially became ANSI/IEEE Standard 896.10-1997.

IEEE standards must be reviewed at least every 5 yrs. This process requires, among other things, finding enough interested parties to perform the review. On February 7, 2003, IEEE Std 897.10-1997 was formally withdrawn. It is no longer an IEEE standard and is no longer endorsed by the IEEE. The author can find no evidence that any system was ever built using this standard.

What went wrong? Clearly, the features accommodated by the standard (block redundancy and operate-through voting redundancy) were desirable to the space community. Why were there no users at all, even among the organizations that had supported the standardization effort?

Futurebus+, Profile S, suffered from trying to create a single solution that would solve all anticipated problems: throughput, reliability, and availability. Any

user would have to pay a penalty for features that they didn't need. Worse, there was a degenerate case (single string, moderately high performance computing) that was adequately served by the existing solution (VMEbus) but which would pay a particularly high price for the extra features.

In the mid-90's, the majority of spacecraft desiring to use a standard parallel backplane bus used VMEbus in a single-string centralized processing architecture. The few users who wanted to use VMEbus but needed higher reliability solved this problem in the historical manner, by cross-strapping block-redundant larger functional assemblies. This approach provided sufficiently high reliability (or at least answered requirements to ensure no single point failures), and the finer-grained standby redundancy offered by the new Futurebus+ thus had little perceived value.

As far as operate-through high-availability systems were concerned, only the Titan launch vehicle family was entertaining an avionics upgrade. This potential user was eliminated by the EELV competition of the late 1990's

Finally, the engine of technological advance was addressing the one Futurebus+ feature of broad interest, data throughput. The same industrial community that advanced VMEbus also had performance growth issues. Although VME-64 offered an upgrade path, the PCI bus was even more attractive. The CompactPCI standard evolved by marrying an existing high-performance electrical standard to an existing, proven mechanical form factor. PCI variants now support as much as 1 GB/s throughput, far higher than envisioned by Futurebus+. There are even CompactPCI variants that support the high reliability needs of the telecom industry.

It should be noted that, despite being more advanced, cPCI has not displaced VMEbus in space applications. Many spacecraft use a hybrid system architecture, using cPCI for high-performance memory and I/O operations, while using VME for low-speed I/O connectivity. This largely boils down to simple economics. The high-performance RAD750 processor is only available in cPCI; if one wants its performance, one must support cPCI. But most such users have legacy VME I/O boards that are more than adequate and whose migration to cPCI is not economically justified. Since both buses have common board form factors, two different buses can coexist in the same chassis with minimal overhead.

RULE #1: SOLVE THE RIGHT PROBLEM

Probably the first rule of creating something with an eye toward its becoming a standard is to solve the right problem – or at least *a* right problem. If you have a technical problem to solve, either

- a) yours is a unique problem and it really doesn't matter how you solve it (with respect to its becoming a standard solution) because no one else will care; or,
- b) it is a general problem and one or more others have already solved it adequately, in which case you should probably consider adopting the already-implemented solution; or,
- c) it is a general problem (or, better yet, you're on the cutting edge and you can see that it will become a general problem), no one has come up with an adequate solution, and you can do better.

If the goal is to create a standard, the requirements-definition process should mirror that of creating any product that is to be sold in the free market. Potential users should be identified who have the substantially the same problems. The first solution should probably be good enough to start getting an initial-adopter community working together (rather than trying to hit a home run with the "perfect" end solution), a community that is willing to help work out the kinks (read: beta testers) before moving to a larger audience or market. Iteratively evolve. If your solution builds on an existing standard or technical approach, make sure you understand everything you're buying into; often there is more to a standard than the naive user is aware.

As a counterexample to this rule, consider the creation of "plug-and-play" solutions for space systems. Clearly, significant time and effort go into the space system integration process. Commercial solutions to the component integration problem involve the "plug-and-play" features of both PCI and USB. It would seem that a similar approach would benefit the space community. But there are significant differences.

The integration problem faced by the consumer electronics industry is how to allow component A manufactured by random company A1 to be plugged into component B sold by random company B1 – in the presence of unrelated random components C, D and E, not to mention random software programs F, G, and H (and six variants of operating system M). And the integrator (consumer) is generally not tech-savvy. A call center that can handle all possible variations and help the myriad frustrated customers is costly. System components that carry integration information with them and automatically communicate during the integration process is a reasonable way to communicate

between manufacturers – effectively cutting the nominal integrator (the consumer) out of the process. The enormous front-end software cost associated with the USB device drivers supporting this seemingly simple process (for devices of any complexity) are worth the investment for companies whose alternative is to actively support thousands (or tens of thousands) of customers after the product is in the field.

The aerospace system integration problem is very different. Component A is produced by a specific company. Component B is produced by another very specific company or perhaps the integrator. The integrator has the ability to take pains to know what the relationship is between A, B, and all the other system components. The integrator is almost assuredly tech-savvy and probably highly educated. If configuration information is needed during integration, the cost to transfer this information using ancillary media is not great. When problems are encountered during integration, there are reams of technical documentation and, usually, an army of technical personnel who can be consulted. The aerospace integration problem is in the lack of maturity of the interfaces when the two sides meet – and this problem cannot be solved by “plug-and-play” architectures. In fact, the extra work expended in trying to support “plug-and-play” in its commercial sense will exacerbate the problem by taking resources away from ensuring the interface’s maturity.

CASE #2: ADA vs C

Faced with an explosive growth of software costs in military procurements, the US Department of Defense (DoD) initiated the Higher Order Language Working Group (HOLWG) in 1975. Among other things, over 400 programming languages and dialects were believed to be in use in DoD projects at that time. This group worked throughout 1975 to establish requirements for a new programming language that would address software engineering of large, complex projects with attention to language standardization, validation, reliability, maintainability. In 1976, the HOLWG evaluated 23 languages against the requirements. The C language, first documented in 1974, was not among the 23. In early 1977, the HOLWG concluded that no existing language was suitable and that a new language needed to be developed.

Based on a revised set of requirements, an RFP was released in April 1977. Seventeen proposals were received. Four contractors were picked to build prototypes. A single winner was selected in May 1979 and the language was officially named “Ada.” After ANSI review and subsequent revisions, Ada became

ANSI/MIL-STD-1815A-1983 and then ISO-8652:1987. Subsequent revisions led to “Ada95” which is formally a joint ISO/ANSI standard, ISO-8652:1995.

In 1983, the US Department of Defense (DoD) mandated that all mission-critical applications with more than 30% new code in the result be written in Ada. Waivers were often granted. Early compilers were very slow. The run-time overhead added significant burden in the embedded, real-time applications that Ada targeted. Few programmers had experience with Ada; large projects paid a substantial premium for this skill. In 1997, as DoD started embracing COTS standards, it effectively eliminated the Ada mandate.

Various studies indicate that total life-cycle costs for software projects implemented in Ada are substantially lower than those implemented in C/C++ (or any other language), that defect rates are substantially lower, and that overall product reliability is substantially higher. Yet today, for embedded real-time software development projects (the primary target for the Ada effort), there is about ten times the demand for C/C++ personnel as there is for Ada. Among many software developers and managers, there remains the perception that Ada isn’t the right tool for the very applications for which it was developed.

RULE #2a: DON’T EXPECT TO LEGISLATE BEHAVIOR

The “Ada mandate” did not result in its acceptance. In fact, the mandate might have had the perverse effect of reducing its acceptance, since software engineers (like most creative people) tend to resent such directives. The waiver process was porous enough that it was relatively straightforward for many projects to find the appropriate rationale for exemption (not unlike obtaining sole-source procurement authorizations). Many reasons provided real enough arguments: existing code base, performance requirements, personnel availability, project schedule.

In free markets, a large number of individual decisions collectively select the winners, whether they are tangible products or intangible things like standards. The small satellite community is closer to being a free-wheeling marketplace than the larger aerospace industry is. In theory, more ideas can be explored in less time with small sats than with much larger craft. With many more technology innovators and system-level decision makers, the small sat community should act like a marketplace of entrepreneurs and arms-length buyers, allowing grass-roots bottoms-up technology to

drive standards evolution rather than relying on a top-down requirements-driven procurement process.

RULE #2b: EXPECT TECHNICAL PROGRESS

Technology happens. Ada took eight years to develop. During this time, C was used to implement UNIX. With its roots in research and academia, C spread through the higher education world. By the time Ada was formally standardized, C was a *de facto* standard that had been learned by eight or ten classes of computer science students. By the time of the Ada mandate, C was in widespread use and C++ had been created. Despite the DoD's goal of reducing the amount of training needed by each program, the reality (or at least the perceived reality) was that projects using Ada faced a not-insignificant up-front cost of workforce training that C-based projects avoided. Without a long-term vision (or with a self-contained project budget that didn't account for long-term benefits), it was difficult to justify this investment.

Whether you're building the perfect widget or the perfect standard, other things are happening in the world while you're developing. Keep the development cycle short and get the word out that you have the best technology. Build a user community and get feedback early. You cannot be successful in a vacuum.

CASE #3: IBM PC/AT, PC/104, APPLE

When IBM engineers created their PC (and, later, the PC/AT), they needed a low cost parallel bus for I/O on their new motherboard. They didn't set out to create an interface standard. They simply needed a solution and built something that was good enough to get the product to market. They didn't even document the internal details of the bus for external use (e.g., bus signal timing). Despite this, third-party vendors introduced a plethora of add-on products for the PC. Later, entrepreneurs built alternate motherboards to host the interface, based on the existence of a market of such plug-in I/O boards. For years, the only way of determining compliance with this *de facto* standard was to plug a board into a real IBM PC and see if it worked. An entire industry grew up around this unplanned standard.

Like the PC/AT bus, the PC/104 standard arose serendipitously. PC/104 originated as "Mini Modules," created at Ampro to provide modular expansion I/O capability for PC motherboards used in embedded systems, where the card-edge finger connector of the original PC bus was not mechanically sound. Later,

CPU modules were created to replace the larger motherboards, allowing whole computers to be built in the now-familiar 3.8" x 3.6" footprint. Many manufacturers sell a wide variety of module types and it is easy to create new functions and products.

In contrast to IBM, Apple retained intellectual property control over its Macintosh architecture. Third-party vendors were carefully vetted and few were allowed to develop plug-in hardware. While the Mac's architecture is generally considered cleaner than the PC's and this has led to fewer system integrity problems, the closed system approach has hindered its adoption by many market segments, such as instrumentation, automation, and embedded systems.

Today's PC contains several communications standards, each serving different purposes. The PC AT and EISA buses have been supplanted by PCI (due more to plug-and-play configuration considerations than performance). Specialized buses serve video displays (the Advanced Graphics Port) and hard drives (the IDE/ATA and, now, Serial ATA). Within the space community, the need for a general-purpose parallel bus has been answered by both VMEbus and cPCI. Is there no special problem within our industry that justifies a simplified interface standard like SATA?

RULE #3: YOU CAN'T PLAN EVERYTHING

As the previous examples show, not every carefully planned standards activity results in an accepted, widely used standard. And not every accepted, widely used standard results from formal standards activity. In fact, it is probably the case that more standards in use today originated from individual or small-group activities that later got formally blessed as standards than originated as committee efforts. Technology happens.

A STANDARDIZATION CHECKLIST

Thirty and forty years ago, NASA and DoD dominated technical direction in the (U.S.) space community. Today, there are many more players who at least have the potential to direct the technical evolution of space components and systems. In this free-market, survival-of-the-fittest environment, technologists can increase their likelihood of creating respected and accepted solutions that ultimately become standards by thinking like entrepreneurs who are developing products.

The following questions are similar to those that one should pose when trying to create a marketable product.

This list is not intended to be exhaustive. Rather it is intended to serve to start the process.

Problem Statement

What problem am I solving? What confidence do I have that I understand how others have or might have addressed this problem?

Problem Type

Is this a technical problem, a process problem, or a personnel problem? Can it be solved with a widget or might it be better solved with a standard procedure? Can the solution be packaged in a way that is easily conveyed to its intended audience? Can it be “sold”?

Market / User Community

Who cares? Whose problem is this? Does anyone else actually have the same problem? Can it be stated in general terms that describe a target market or community? Are my assumptions about the intended audience correct? Have I really researched the target market or community?

Market / User Value

What is the likely perceived value of the solution in its intended market? Is that target market likely to accept the costs associated with my solution, whether direct financial costs or indirect costs (e.g., mass, power, complexity, etc.)?

Host Organization Value

What will my organization get out of this? Will we give it away to enhance other aspects of our business? Will we try to protect it and license it, risking that others might object to such apparently self-serving behavior and then develop their own solutions?

Implementation Approach

How will I get a prototype built? Who will be my beta-testers? Do I expect to have the resources in my own organization to do this internally, or might it be better to immediately seek outside partners? Would potential customers be the best partners, or potential competitors? (Note that establishing a standard with a potential competitor validates the existence of a market, creates a unified front, and might make raising capital easier.) How quickly can I get the ideas developed? Will the need still exist when I’m done?

End Game

How will I know that I’m done? What are my success criteria?

CONCLUSIONS

The small satellite community has reached the stage where many individuals and companies recognize the potential value of using standards to reduce their spacecraft development efforts, costs, and schedules.

This community has unique problems that are not adequately addressed by existing commercial or aerospace standards, leading to the conclusion that new standards are needed. But naïve approaches that simply set out to establish such standards risk being overtaken by other solutions. Standards evolve and become accepted in many different ways. Some approaches are more likely to be successful than others.

Given the nature of the small spacecraft community as a dynamic collection of many disparate thinkers, we might be best served by a market-driven approach to standards development and adoption, rather than the requirements-based approach that has often driven military and aerospace technology efforts in the past.

REFERENCES

- (1) <http://standards.ieee.org/cgi-bin/status?896.10-1997>