OPEN SYSTEMS ARCHITECTURE - BOTH BOON AND BANE

Randy Black, Mitch Fletcher, Honeywell International, Inc., Space Applications, Glendale, AZ

Abstract

As with every major social revolution, the advent of the digital age was accompanied by growing pains. Many of the methods used in the first few decades were ultimately rejected for better and more proven approaches. Complex systems such as aircraft avionics or military weapons control systems are no longer the purview of the omniscient hardware guru or software wizard. These mystics have been replaced by processes that improve productivity and enhance long-term maintainability. However, in most cases application of these processes produces its own set of problems. This does not necessarily mean that the practice should be discontinued; the perfect solution may not exist. It is important that system architects select those practices that best meet their specific goals, while accounting for associated problems. Over the past several years managers have come to accept as axiomatic that the use of open systems reduces cost, decreases schedule, and eliminates risks to a program. However, the term, "Open Systems Architecture" invokes a variety of interpretations. To some it implies no proprietary components. To others it implies adherence to documented standards. Still others see it as implying plug-and-play features. This paper investigates several observed interpretations of the meaning of "Open Systems Architecture." Each interpretation exhibits both positive and negative aspects, which should be considered when architecting requirements for a specific system. Particularly for complex or long-life programs such as the NASA's Space Exploration Vision, there is a threshold beyond which the benefits of increased openness are outweighed by the associated costs. This paper concludes with a recommendation regarding the optimal placement of the open architecture threshold line for complex, long-life programs.

Introduction

During the past two decades rising costs and decreasing budgets have forced changes in the way companies and government programs approach high-tech projects. In an attempt to lower costs and decrease schedules, many developers are turning to Commercial-Off-The-Shelf (COTS) components. While rarely optimal, these COTS components are deemed "good enough" for the needs of the end user. While the term, "COTS" applies easily to bolts, washers, microchips, and power supplies; its applicability to overall systems is not as readily apparent. Several years ago one of the authors was a speaker at a major COTS conference. The topic was a tool to help integrate various COTS hardware and software devices into an overall system. The author was asked several times why the topic was being presented at a COTS conference.

Perhaps as a projection of the COTS concept into the systems domain, the term, "Open Architecture" has been applied to overall systems as a comparable solution to decreasing budgets. There are several perceived benefits that come to mind when open architecture is discussed, but the exact definition of "open" varies from individual to individual. In order to meet NASA's desire for open systems, Honeywell Space Applications commissioned a team of senior in-house engineers to perform an analysis of the various meanings of "open" and to recommend an approach that meets the needs of the customer. This paper details the results of their analysis and concludes with their recommendations. While the focus was primarily avionics architectures, the analysis applies equally well to most electronic systems. The team concentrated on openness as it applies to processors, I/O types, the system data bus and backplane, the software environment including operating systems and tools, and various mechanical properties of electronic systems.

Apparent Advantages of Open Systems

Intuitively there are several apparent advantages to an open system architecture. Generally these advantages respond to problems that have plagued systems in the past. Apparent benefits include solutions to problems associated

with single suppliers, NRE costs, maintenance costs, and upgrade costs.

Single Supplier. Open systems avoid the constraint of relying on a single supplier. Several risks are associated with a single supplier including the possibility of the supplier going out of business, the supplier increasing price due to their monopolistic position, and the supplier discontinuing support for older versions of a product.

NRE Costs. Open systems appear to reduce development costs. Using COTS components eliminates the need for new development. Open systems rely on the likelihood of multiple suppliers, fostering competition that leads to lower prices. Integration of COTS components are often handled by the suppliers, producing a list of compatible products for use on the project.

Maintenance Costs. Open systems increase the prospect that there is a pool of experienced users, decreasing the need for training efforts. Secondary support products such as development and maintenance tools are likely available as well. Support organizations, including supplier technical support, is often available for a yearly licensing fee.

Upgrade Costs. Many of the advantages associated with development and maintenance costs associated with open systems apply to upgrade costs as well. Competition drives product enhancements by suppliers at low or no cost to the project.

While all of these benefits seem intuitive, experience has shown that reality does not always match theory. When decisions are made on open system principles, it is important that system architects select those practices that best meet their specific goals, while accounting for potential associated problems.

Dimensions of Openness

In system architectures, the definition of "open" is hazy at best. Some are more generally accepted than others, but none are universally acknowledged. Among the more common definitions are:

- Documented Standards
- Widely Used Standards
- Non-Proprietary Interfaces
- Plug and Play
- Commercially Available End Items
- Commercially Available Development Tools
- Long Life Availability
- Open Source Code for Software and Firmware

The following sections discuss each of these categories in details. Each category is described, with both benefits and potential issues outlined.

Documented Standards

Description. A documented standard is a defined set of protocols that is adequate for a company to build a compliant product. The standard must be sufficiently complete that one product built to the standard may interact seamlessly with another product built to the same standard. The standard itself may be controlled by an independent organization such as the Institute of Electrical and Electronics Engineers (IEEE), the Society of Automotive Engineers (SAE), or the Object Management Group (OMG). The standard may also be controlled by a single company, as long as it is commercially available to others, such as Microsoft operating system interface standards.

Benefits. Documented standards are available to everyone, eliminating many of the potential problems associated with single suppliers. Long life availability is another benefit of documented standards. While there may be few products that use the older standards, they are available for use as needed. For example, after several decades the Algol-60 programming language definition is still available. While not particularly practical for a new project, those who must maintain or upgrade older products have access to the standards to which the products were designed. If sufficiently complete and rigorously followed, documented standards can significantly reduce the time required to integrate components.

Potential Issues. One of the potential issues with documented standards is that cost consideration often force developers to not follow the standard exactly. Sometimes simplified versions are developed with less-critical capabilities left unsupported, placing integration with other

products at risk. Some developers add enhancements to standards. While a contract can dictate strict adherence to a standard, reality often intrudes upon that adherence. When use of an additional feature can cut costs and schedules significantly, most projects will select cost and schedule over dogmatic compliance to the standard. For example, Ada 83 (MIL-STD-1815A) was deficient in several advanced programming capabilities such as object-oriented programming and real-time concurrent programming. Compiler developers added enhancements to accommodate these deficiencies, greatly improving programmer efficiency in these areas. It was difficult for projects to strictly follow the Ada 83 standard when the result was higher cost and longer schedules. In this particular case, Ada 95 (ISO/IEC/ANSI 8652:1995) incorporated many of these enhancements, but this type of orderly progression of standards is not universal. Another potential issue with documented standards involves those standards not controlled by an independent body. Without independent support, standards often fade away over time.

Widely Used Standards

Description. Generally speaking, widely standards are controlled by an independent organization such as the IEEE, SAE, or OMG. While documented standards provide some degree of openness, not all documented standards are widely used. For example, very few applications follow the IEEE 1394a *backplane* standard, while numerous applications adhere to the IEEE 1394a *bus* standard. A broad continuum exists regarding the long-term support of widely used standards. For example, COBOL has remained an industry standard for decades, whereas Pascal enjoyed wide popularity for years before fading into obscurity. Other languages, such as Smalltalk and LISP have enjoyed long-term popularity in a niche market.

Benefits. One of the benefits of adhering to widely used standards is that there is likely a large base of experienced engineers familiar with the standard. This eliminates the need for extensive training and familiarization associated with learning a new system. Engineers are generally more efficient when working with familiar products. For example, when Ada was first introduced, very few engineers were familiar with the benefits provided by the language. In essence many FORTRAN programs were written using Ada syntax. However, as programmers became familiar with Ada and were able to take advantage of the features unique to the language, efficiency increased significantly. Another benefit of following widely used standards is that economic realities assure multiple suppliers will provide products that adhere to the standard. Competition among suppliers tends to drive costs down while improving product fidelity. In addition interfaces to similar products are created to form a larger customer base. These interfaces include bridges, gateways, drivers, Application Program Interfaces (APIs), and file format standardization.

Potential Issues. Widely used interfaces are not always sufficient to meet the technical requirements of a program. In these cases, the cost of adhering to a widely used standard may exceed that of more specialized alternatives. For example, MIL-STD-1553 is one of the most common bus protocols used by the Department of Defense. However, for multimedia applications requiring the movement of vast amounts of data, the MIL-STD-1553 bandwidth and protocol is insufficient to meet the application needs. Similarly, DOD-STD-2167A was designed prior to the popularization of object-oriented design techniques. Excessive time has been spent attempting to force fit object-oriented designs into the incompatible DOD-STD-2167A format. Long term life cycle costs may also suffer through the selection of widely used standards over more specialized approaches. For example, COBOL is still one of the most widely used computer languages in the world. However, for database applications, the cost of training COBOL programmers to utilize SQL would save considerable costs over the life cycle of the program.

Non-Proprietary Interfaces

Description. Some customers, particularly developers of long-life systems, are not as concerned about the standards being documented or widely used as they are about the potential long-term support of the standard. In these cases, owning or licensing the Intellectual Property (IP) from a third party is sufficient to meet their long term needs.

Benefits. Owning or licensing the IP for proprietary standards provides a means for the customer to maintain, expand, or modify the standard as the program needs dictate. Particularly with long-life programs, the cost of training for the unique standard is potentially low when amortized over the life cycle of the program. The benefit of this approach is that it allows a standard that best meets the program needs to be applied to all developers associated with the program. For example, a specific company may have a bus standard that is ideal for small satellites. NASA may wish to purchase or lease the standard, applying it to all satellite components. Over the course of developing dozens of satellites, the development cost savings associated with following an ideal standard may outweigh the costs of the additional training required.

Potential Issues. One issues that arises through licensing IP is that licenses normally have a limited duration. At the end of the licensing period, the risks become the same as those associated with a being forced to purchase from a single supplier. One of the potential risks is that the IP owner will increase the licensing cost to a level that is painful, but not fatal, to its continued use. From the IP owner's side, there are risks associated with licensing or selling IP. The IP may be sufficiently unique that the owner may be hesitant to allow other companies to study the IP. Even if the IP is licensed for a niche market, there is always the concern that reverse engineering will provide a competitor in the broader marketplace.

Plug and Play

Description. In some cases the use of standard interfaces has been refined to the point where devices can be inserted into a system and be recognized without user involvement. These "Plug and Play" devices can be boards, peripherals, software, or any component of an overall system. Both the device and system must strictly conform to the appropriate standard in order for this recognition to occur. For example, recent Microsoft operating systems recognize and are able to control various standard devices such as scanners, printers, and disk drives. However, these devices must adhere to Microsoft's standards in order to be recognized by the operating system. As a further refinement, some Plug and Play interface standards have been designed to allow a device to be installed or removed while this system is operational. This "hot swap" capability may be dynamic, where any compatible device may be inserted or removed from the system. This capability often requires a system reconfiguration to smoothly transition from one system state to another. The alternative is to have a static hot swap capability, where only "known" devices may be added or removed from a baseline configuration. A system reconfiguration is not normally required for static systems.

Benefits. Since Plug and Play and Hot swap capabilities are refined instances of documented, widely used, or non-proprietary standards, all of the benefits associated with the previous standards section apply. Additional benefits involve decreased integration time associated with Plug and Play devices, and decreased reconfiguration effort associated with hot swap devices.

Potential Issues. As with the benefits, the issues associated with Plug and Play and hot swap capabilities mirror those of the appropriate standards capabilities described above. Specific instances of Plug and Play and hot swap schemes have produced unique issues. For example, the IEEE 1394b bus standard allows for hot swap devices to be dynamically inserted into the system. When this occurs, the network identification number associated with each connected device is reconfigured. In the case of a loose wire causing intermittent contact with the network, the system can become tied up constantly reconfiguring the network addresses. While the IEEE 1394b standard defines alternate approaches, it is important to realize that each system requires detailed analysis prior to implementing Plug and Play and hot swap capabilities.

Commercially Available End Items

Description. One of the more common descriptions of an open system is one in which the architecture consists of readily-available parts, accessories, or enhancements provided by industry standard distributors. On an individual basis, COTS components provide a low-cost solution compared to the development costs of a custom product.

Benefits. Use of COTS components usually minimizes the overall system development costs. Market realities ensure the price of the components is sufficiently less than that required to design a new component. Generally, COTS components support several markets, allowing the high production volumes to lower costs. In addition, common components are likely to have a large support base. Research and development costs are can also be shared across markets.

Potential Issues. While COTS components tend to be designed for cost optimization, specific programs may have higher priority needs. Examples of these needs include high reliability, long life, radiation tolerance, and operation under extreme conditions. The cost benefit of COTS components must be traded against the program needs. Other system requirements may also drive down the use of COTS components. Coordination between redundant modules in a fault-tolerant system may require significantly more effort to achieve using COTS components than components designed for redundant operations. Similarly, integration and test of COTS components generally involves increased effort. The interfaces to COTS components are dictated by the supplier. Experience has shown that, despite documentation to the contrary, "compatible components often aren't." Depending upon the level of testing required, the black box nature of most components may increase testing cost or even prevent adequate testing. For long life programs, parts obsolescence is a major concern when dealing with COTS components. Very few components are produced unchanged for several years. The cost of upgrading a long life system with each new generation of COTS components is prohibitive.

Commercially Available Development Tools

Description. Commercially available development tools are considered open if the output conforms to a specified standard. This definition allows multiple suppliers to provide interchangeable tools. For example, a certified Ada 83 compiler produces low-level code in accordance with MIL-STD-1815A. At an interface level, the Extensible Markup Language (XML) defines a common standard for sharing data between tools, allowing tools from multiple suppliers to interact seamlessly. While these two examples follow public standards, the standards need not be public. In order to expand their marketplace a company may choose to allow others to build to a proprietary interface standard. For example, Green Hills Software, Inc. provides a set of Ada development tools [1] for rival Wind River Systems' Tornado® integrated development environment [2].

Benefits. The benefits associated with commercially available development tools are primarily due to the lack of dependence on a single supplier. The developer has the option of migrating to alternative tools if a supplier goes out of business or makes unacceptable changes to technical or business approaches. Particularly for the more widely used development tools, the assumption is that long term support will be available. Decades of support for COBOL compilers bolters this assumption.

Potential Issues. The use of commercially available development tools is fraught with potential hazards, particularly for applications requiring formal certification. There are three major categories of potential issues associated with development tools: 1) Certification / Qualification; 2) Product Evolution; and 3) Version Incompatibility.

Certification / Qualification. When required, one of the most costly phases of software or hardware development is certification. For software this often involves testing to a wide range of input data, demonstrating that testing has covered all potential decision paths in the code, and proving no unused code exists in the program. Despite following the same standard or API definition, the exact output of individual tools often varies. For example, even though two Ada 83 compilers are certified to MIL-STD-1815A, the low-level code produced by the compilers will differ. Likewise, the output of the same compiler running on different processors produces low-level code suitable to each processor. Ada code that has been certified using one compiler on a particular processor will require re-certification if the compiler or processor changes. Additionally, commercially available development tools are often driven by time to market rather than a goal of zero defects. In these cases, the certification effort is

often extended due to errors in the development tools.

Product Evolution. One of the concerns associated with commercial development tools is that they are constantly evolving regardless of the source of the applicable standards or API definitions. Open standards such as LINUX are public for the very purpose of promoting improvements and enhancements. Even tightly controlled standards managed by organizations such as IEEE, SAE, and the American National Standards Institute (ANSI) are constantly being reviewed and updated. The previously cited example of the evolution of Ada 83 to Ada 95 demonstrates the point. This same evolution occurs due to unofficial upgrades as well. Companies seek to improve their market share by providing features that are beneficial, even if they do not follow the associated standard. Features added to the HyperText Markup Language (HTML) by one browser supplier are difficult for web developers to ignore, even though other browsers do not support the added features. Product evolution reduces many of the open system benefits associated with commercially available development tools. Lack of consistency over time limits re-use of components, as well as requiring system architects to maintain familiarity with changes in standards and products.

Version Incompatibility. Associated with product evolution is the issue of version incompatibility. As successive versions of tools and standards are introduced, improvements generally take priority over backward compatibility. There is no easy solution to version incompatibility. One option is to continue working with outdated versions of the commercial development tools. Understandably, supporting previous versions is not a high priority for product suppliers. The users of outdated versions generally are required to pay a substantial fee to maintain support from the supplier. A second option is to migrate to the newer version while only using capabilities that are unchanged from the older version. Unfortunately, backward compatibility is normally only provided for a limited number of generations. In addition, backward compatibility rarely results in identical outputs, leading to the need for recertification. As previously indicated, forcing developers to ignore improved features in the newer version is extremely difficult, particularly if use of the features can significantly decrease cost and schedule. The third option is to migrate completely to the new version of a tool. This requires recertification for each migration. Particularly for standards or products that evolve frequently, this can be extremely costly.

Long Life Availability

Description. For long life programs, component availability over time is often a major concern. Of necessity, components must be available throughout the life of the program to support maintenance, modification, and enhancements. Many of the issues discussed previously describe roadblocks to component availability. However, several workarounds have been successfully utilized in the past. One workaround involves purchasing components in volume for use throughout the life of the program. Another workaround is to purchase the appropriate IP and use it as necessary for long life support. For example, the AMD29050 processor used in the avionics system of a major aircraft has become obsolete. Honeywell purchased the IP and created an ASIC to replace defective processors. Another option that minimizes the impact of obsolescence is to select components with a clear long term growth path. While backward compatibility may not be maintained throughout the product life cycle, required modifications to the overall system are minimized when there is some level of commonality between successive generations of component upgrades.

Benefits. The benefits of long life availability primarily relate to avoiding the need to constantly replace outdated parts while incurring the necessary development and integration costs.

Potential Issues. Market concerns prevent most components from remaining static indefinitely. Even the described workarounds present potential problems. In most cases the support infrastructure for products are short-lived, requiring the program to provide its own support infrastructure. Purchase of sufficient quantities of components to last the program duration engenders its own potential risks. One risk involves the shelf life of the spare components. If the components break down over time, then the deterioration rate must be a part of the initial spare count calculations.

In addition, components may fail at a higher rate than anticipated, exhausting the spares reserve prior to the end of the program. In the case of a workaround involving purchase of the IP, the problems of the support infrastructure are magnified. Not only must the original product be supported, but any unique aspects of the rebuild process, components, or facilities must also be supported.

Open Source Code for Software and Firmware

Description. For software and firmware, free access to the source code is considered by some to define an open system. There are three major approaches to making the source code available. In the first case the proprietary rights are retained by the original developer. Customers and others can examine the code, but maintenance and modification is controlled of the developer. In the second case the proprietary rights are licensed. In this case the source code may be modified and maintained freely or subject to some restrictions. For example, software may be freely licensed in a niche market such as satellites, but may remain proprietary for use within the automotive market. In the third case the proprietary rights are sold to the customer who becomes the owner of the code.

Benefits. If the proprietary rights are purchased the customer becomes the owner, allowing for long term maintenance, modifications, and enhancements, regardless of the status of the original developer. With licensing or purchasing of the proprietary rights, the customer controls any modifications or enhancements to the source code, eliminating potential conflicts between the developer and customer.

Potential Issues. It is important to remember that purchasing the proprietary rights to source code does not necessarily guarantee the availability of a compatible development environment. Along with the source code, control of a compatible development environment must either be purchased or developed. Maintenance of a development environment can often be as costly as development of the application software or firmware itself. In addition, licensing of proprietary rights normally are limited to a specific time duration. At the end of the duration a new license must be negotiated. While not always the case, many suppliers choose this time to increase the rates significantly, since alternate solutions are often not as prevalent as when the initial license was negotiated.

Conclusions

For each category of open systems, there are both benefits and potential issues. There is no universal approach to open architectures that is guaranteed to reduce cost or schedule. Each requirement driving toward increased openness should be carefully analyzed and a cost-benefit analysis performed. Only those open architectural principles that will truly result in decreased cost or schedule should be imposed on a project.

The team of Honeywell senior systems engineers performing an analysis of open system architectural principles concluded that the following categories are most likely to produce significant savings and should be seriously considered when designing a new system:

- Widely Used Standards
- Non-Proprietary Interfaces
- Commercially Available Development Tools

The other categories may also produce significant savings on a case by case basis, but generally are less likely to be cost effective. In all cases a serious analysis should be performed to determine the optimal level of openness for each project. The bottom line is, open systems do not always provide cost or schedule relief, and in many cases cause increases over the life of the program.

References

[1] *AdaMULTI/Ada 95 for Tornado*® retrieved September 8, 2006 from http://www.ghs.com/products/rtos/AdaMULTI_Tornado_VxWorks.html

[2] *Tornado II for VxWorks* retrieved September 8, 2006 from http://www.windriver.com/portal/server.pt?space=Opener&control=OpenObject&cached=true&in_hi_ClassID=512&in_hi_OpenerMode=2&in_hi_ObjectID=720&

Acknowledgement

Special thanks to the team of senior Honeywell Systems Engineers who performed the analysis upon which this paper is based: Peter Alejandro, Randy Black, Clint Browning, Ray Crum, Dave Dopilka, Mitch Fletcher, Paul Milo, and Phil Scandura

Email Addresses

Randy Black
Autonomous System & Robotic IPT Lead
Human Space Enterprise Team
Honeywell - Aerospace Electronic Systems
randy.black@honeywell.com

Mitch Fletcher
Chief Systems Engineer
Human Space Enterprise Team
Honeywell - Aerospace Electronic Systems
mitch.fletcher@honeywell.com