

White Paper

Rapid Embedded Software Development in the New Storage Industry

Introducing the EUREKA™ Architecture And the Firmware Foundry™ Approach to Intellectual Property Reuse

Abstract:

The depth and speed at which fundamental transformations in the data storage industry are occurring have reached the proportions of a global revolution in the industry. This sea change is the impetus for deep rethinking about the way storage products need to be constructed. Because storage products in this new era require increased sophistication in the internal services that they provide, and because the degree of performance and capability in off-the-shelf computing and Operating System platforms has crossed a beneficial threshold, there is shift from storage product development being a predominantly hardware-oriented development paradigm to being software-dominated.

Conventional methods of storage product development are oriented toward the limited approach of crafting “point solutions” on a product-by-product basis, with little IP reuse. This hampers the ability of storage product vendors to be adaptable to the rapid shifts in demand that are occurring in the marketplace, to be innovative in the types of products they can deliver, and to foment new business initiatives.

Breakthrough Systems offers a set of core technologies that radically accelerates the embedded software development process specifically for storage products, vastly improves the level of embedded software reuse, and offers an infrastructure built for flexibility, adaptability, extensibility, and reliability – deftly responsive to the technical needs of this new era in the data storage industry.

Introduction – Adapt or Die

“It is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change.” – Charles Darwin

“The more rapid the pace of change, the more dire the consequences of stubbornly sticking to old ways.” – John S. McCallum

“Adapt and overcome.” – USMC motto

The data storage industry is undergoing a series of fundamental transformations in response to tidal shifts in the way the world regards data. New data storage demands in this era are biased toward:

- globalized data distribution
- policy-based and proactive management capabilities
- content-based data organization and content-addressable storage
- catastrophe-proof levels of fault tolerance and self-healing subsystems
- 24x7 availability
- stronger security, encryption, and authentication
- automated audit trail
- gigantic storage capacities
- ever higher-throughput and lower-latency data flow
- increased cross-platform interoperability
- virtualization and emulation across heterogeneous device types and feeds
- support for a wider variety of transport and networking protocols
- valuation of storage as TCO value to the enterprise instead of cheapest \$/GB

Individually, changes in any of these areas would constitute evolution within the industry; occurring together as they are, all at the same time, they have ushered in nothing short of a revolution in the industry. Impacts on the storage industry span the spectrum from host bus adapters, switching and bridging components, individual storage devices, bus and interconnect architectures, storage appliances, storage subsystems, and conjure up devices and information flow topologies as yet unforeseen.

Trend #1: The data storage industry guru Fred Moore observes: “We are entering into the next phase of the evolution of the data storage industry. This phase will be marked by a new set of rules and a value system far different from that of the past.”¹

Advancement in the previous era of the industry has been marked for decades by the principles of “faster, bigger, cheaper, and tinier”. This has succeeded wildly – to the point where raw storage itself is asymptotically approaching zero cost: storage recording density increases at about 60-100% every year, and drive performance increases at about 10% per year (a doubling rate of about 8 years), while storage pricing has been falling between 30-40% annually. Device form factors have continued to shrink, from 5” and 3.5”, down to 2.5” and 1” sizes for drives, and 8U down to 4U, 2U, and now even 1U for rack-mount storage blades and appliances.²

At the same time, there continues to be an unabated appetite for ever-greater quantities of raw storage worldwide. Application data growth is compounding at the annual rate of 125%, the value of data stored in these storage systems is growing exponentially, and digital data is growing proportionally 50-70%, yet storage management deployment lags this radical storage growth by 2x to 10x.³ Clearly, the status quo in data centers and other deployments worldwide is headed for a crisis as this gap widens. Fred Moore again observes: “The biggest problem in the storage industry continues to lie in implementing effective, bullet-proof storage management and data protection strategies.”⁴

As the exponential curve of storage in deployment is climbed, the sheer enormity of the data under management begs entirely **new methods** for categorizing the data, and managing and extracting the information it contains. Stalin’s famous epigram that “Quantity has a Quality all its own” is applicable to storage systems in that the management techniques that work for moderately-sized volumes of storage become ineffective as those grow without bound.

Trend #2: Although there has been rapid advancement in storage capabilities, at any given instant in time there is great uncertainty as to the next direction that the storage industry will be taking.

In the last decade we have seen the mechanisms of market hype and faddism at work. Network processors were touted as the new wave in embedded architectures, only to collapse spectacularly, leaving leading-edge firms scrambling to repurpose their NP designs and first chip rollouts to other applications. From 2000-2001 InfiniBand was the darling interconnect protocol, taking the industry by storm and headlining in industry rags; by late 2002, IB is all but abandoned, with Intel, IBM, HP, Microsoft scrapping their chips and stacks,

leaving only one company worldwide actively offering IB products; then in 2005, it has a dramatic resurrection in interest, with scores of vendors peeling cobwebs off their IB designs and QLogic making a major investment in early 2006. The industry has been further pockmarked by the back-and-forth of rancorous religious wars (e.g., SAN vs. NAS, SATA vs. SAS, inband management vs. out-of-band, etc.), exacerbating the disparate prevailing views on technologies and approaches.

Forecasting questions abound: Will iSCSI live up to its potential as the dominant storage interconnect topology? Will SAS- and SATA-based storage arrays supplant the need for Fibre Channel as the industry-standard interconnect for large storage systems? Hierarchical Storage Management (HSM) has been reconceived more broadly as Information Lifecycle Management (ILM) – will the tenets of ILM be another transitory news item, or become widely adopted as the organizational model of choice for next-generation storage architectures? Will tape storage finally die?

Some questions about the course of emerging technologies are answered resoundingly; others remain perpetually unanswered, or are reversed radically. Some technologies and methodologies fall abruptly by the wayside, suffering the fates of the Pet Rock and 8-track tapes; others have the staying power to become the state-of-the-art, and jell into standards. One thing remains certain: the nature and direction of the systemic transformations in the storage industry are **impossible to anticipate** reliably very far in advance.

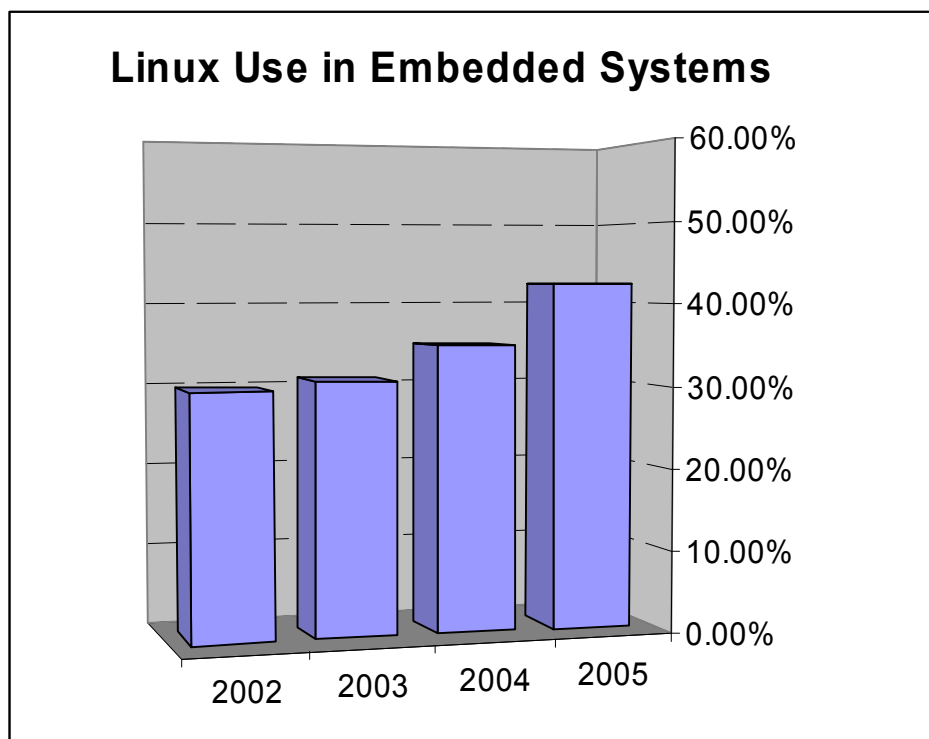
Trend #3: Concurrently with these events in the storage industry, costs for computing hardware in the digital industry in the large have continued to plummet. The density of transistors in semiconductors doubles every 18-24 months, spurring a net performance improvement for processors at the rate of 30-35% per year, yet despite these gains the price of compute power is falling at between 25-35% per year. Off-the-shelf systems and hardware components have become staggeringly powerful.

The COTS initiative from government systems in the early 1990s has spurred parallel trends in the commercial systems arena, and the shift toward increasing reliance on commodity hardware has continued to accelerate. Halfway through the first decade of the 21st Century, a performance/price threshold has been crossed, not only for CPUs, but also for memories, interface chips, and controller hardware, such that this commoditization phenomenon is now ubiquitous in the

computing industry as a whole and in the storage industry in particular. “Most new system deployments will be on commodity hardware platforms using Intel or Intel clone processors.”⁵

Intelligent silicon has become so cheap that it is no longer economical for all but the largest of high-end enterprise storage companies to develop their own proprietary hardware products. In the Open Systems marketplace, hardware development is fast turning into the loss leader for proprietary storage vendors. New storage directions and next-generation storage products are becoming principally about software, not hardware.

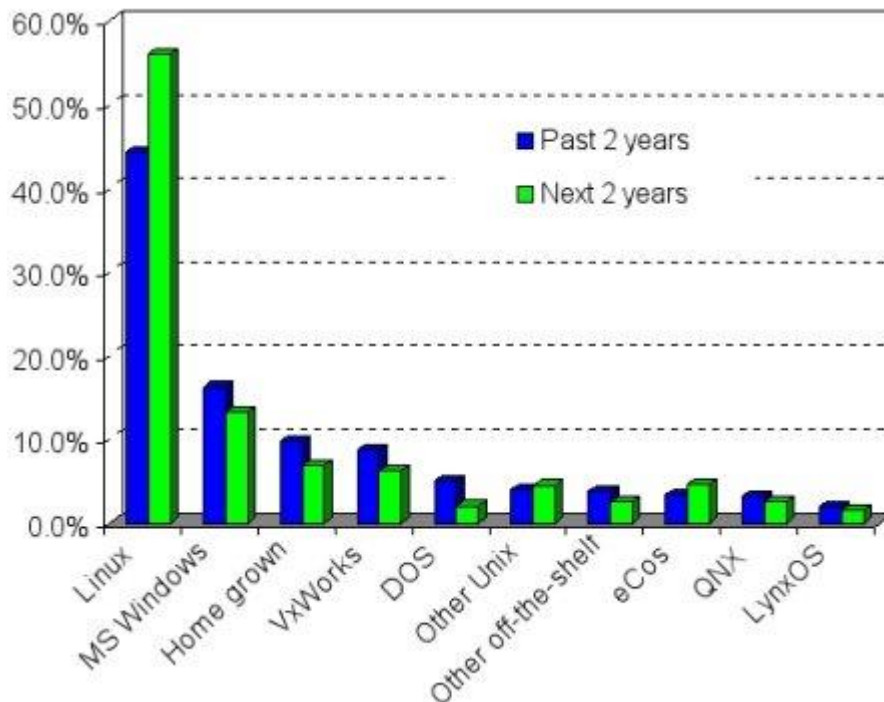
In the same way that hardware has become commoditized, so has Operating Systems and other infrastructure software. The emergence of Linux as a dominant entrant into the OS mix has taken place gradually over the last decade, but the biggest upturn in its use as an embedded OS platform is coincident with the release of the 2.6-series kernel in 2004.



Source: www.linuxdevices.com

Other open-source platforms are also available, leading to the rapid decline in the number and prevalence of embedded deployments using relatively costly commercial real-time OS's such as vxWorks or QNX, or using proprietary "home-grown" RTOS's.

"For decades, embedded systems consisted of special-purpose hardware running special-purpose software... Today, embedded systems are built on general-purpose operating systems (GPOS's) intended for embedded systems, such as MontaVista Linux, or completely general operating systems, like Windows ..."⁶



Embedded RTOS Usage Distribution – 2005

(Source: www.linuxdevices.com)

GPOS's are feature-rich development and execution platforms, providing driver and subsystem support for all of the latest interconnects, devices, middleware, and other software that in past times would have to be developed from scratch.

Commoditization of hardware and software evens the playing field for storage vendors – it no longer takes a firm with the development might of, say, IBM or HP to offer innovative storage solutions. Vendors can now focus on perfecting the value-added "secret sauce" that makes their particular storage solution special in the marketplace rather than the commonplace infrastructure upon which it runs.

These three major factors in the storage industry: emerging **new market imperatives and paradigms**, **strategic uncertainty**, and **platform commoditization**, are the primary-colored brushes that paint an entirely new picture of how storage products must be developed in this new era. The remainder of this paper introduces a new approach toward developing embedded storage software/firmware built expressly to address these factors.

Storage Firmware Development State-of-the-Art: Pretty Ugly

It is clear that in this new era the most successful storage product development companies will need to be organized around **flexibility** and **rapid redevelopment** to be responsive to changing market demands and emerging technologies. However, long experience in the industry has shown that the embedded storage software development efforts in most firms are not optimally organized to have the sea legs of nimble responsiveness under their efforts, and are ill-equipped to operate in this dynamic environment.

As we have seen, the storage industry is experiencing extremely rapid growth and high product line metamorphosis. The average lifetime of a storage product is about 12-24 months. For disk drives in particular, this lifetime shrinks to 9 months. As Time-To-Market increasingly becomes the dominant factor in these brisk market segments, this means that the process of storage product development has devolved into a state of constant scramble – development schedules are compressed into a matter of a few months in which to design, implement, and test products from conceptualization to ship.

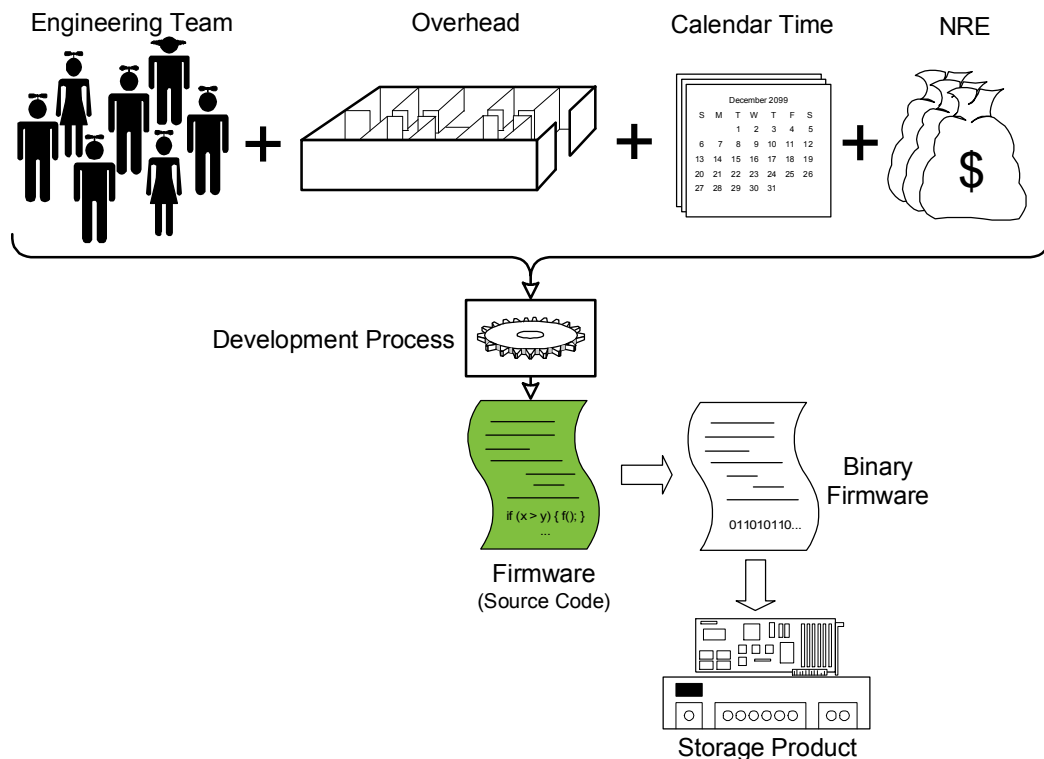
Also, in the pre-commoditization era, product firmware development typically overlapped the development of the product hardware, and the two halves of the product were integrated together in final stages of the development cycle. Now, with the hardware element of the storage product increasingly being an off-the-shelf component, the bulk of the calendar impact and onus of development pressure is increasingly on the software team.

The software lifecycle of storage products is one of frenetically-paced *ad hoc* rollouts. Each new product is another “point solution”, solving a specific purpose, and the embedded firmware contained therein is developed mostly from scratch with each rollout, with a minimum of borrowing from the code of previous rollouts.

Since it comes at the end of the development cycle, product testing is usually the sacrificial lamb of these abbreviated development schedules, and TTM-driven mandates often coerce teams into making unwise compromises in the depth or breadth of validation and verification. Certainly in comparison to other embedded products, such as medical systems, avionics, facilities control, etc., storage systems are notoriously under-tested. Product reliability predictably suffers, debugging is left to the end-user, and the increased support overhead further burdens NRE for the product lifetime as a whole.

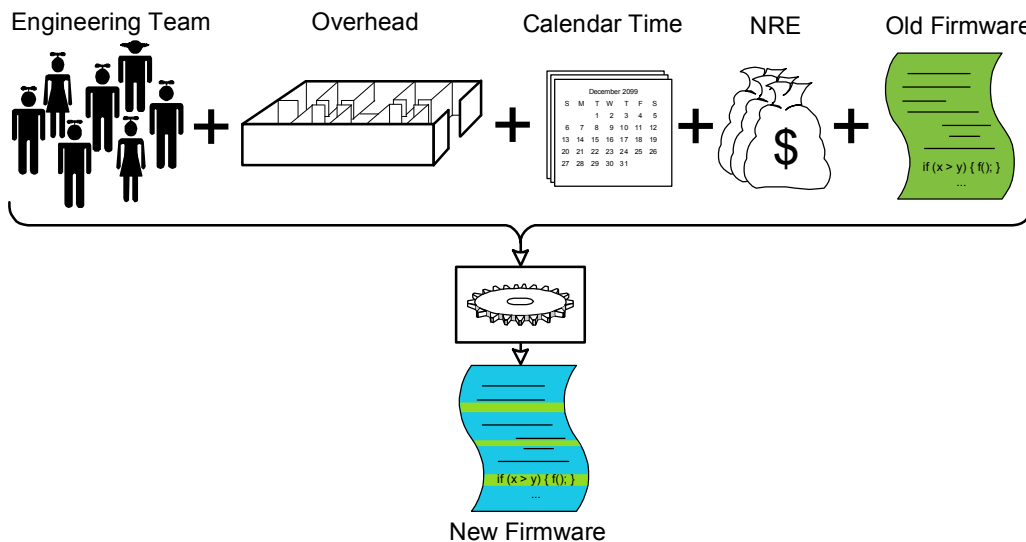
This unfortunate state of affairs is the *de facto* “best practice” in for the storage industry, and is summarized as follows:

- ① New Product Development: From product requirements, an engineering team engages in standard software development process over a project schedule, culminating in delivery of a firmware version for embedding into the storage product.



The “final” version of embedded firmware, thus produced, enters into the carefully managed state of code maintenance phase, and becomes essentially “artifact” code. Innovation on that version firmware is at an end. Code modifications are permitted on an absolute “bug-fix only” basis by a team, different from the original development team, who may or may not understand the original design, architectural intent, or implementation nuances of the code.

- ② Subsequent Product Development: From product requirements for a new storage product, the software development process proceeds very much like for the previous product. To whatever extent is possible, the firmware from previous products is cannibalized opportunistically, using whatever parts and pieces of the old code might be adopted and adapted in the new point solution.



Typically, the degree of code reuse is minimal because the original code, developed in haste, is insufficiently generalizable or applicable to the new product, or is poorly-structured; thus, “reuse” in this process is mostly an act of salvage. Code reuse is by serendipity only.

Clearly, since software development is such an NRE-intensive and error-prone activity, if there was some method of reliably reusing a significant proportion of the already-proven code from one version of embedded firmware within a subsequent product, development time would be much shorter, cheaper, and more reliable.

Software Reuse: A Brief Overview

Eureka! A high degree of **software reuse is the missing piece** that would enable software developers to meet the special needs of the new era in the storage industry that we have been elucidating.

The software engineering field is not immune to hot trends of its own, and perhaps the hottest topic in the mid-1990's was this concept of software reuse. Once touted widely as a panacea, this concept has since suffered criticisms and failures on a number of fronts:

- Building in general-purpose functionality into software can bloat code size, and slow down its execution.
- It is harder to understand and maintain software developed using this approach because of the indirect approach at solving the problem at hand.
- Despite the best of intentions, supposedly "reusable" code components had to be rewritten anyway on subsequent projects because they were insufficiently generalized in their design, required too much information about implementation internals, suffered from poor performance, or because of NIH ("not invented here") mentalities. The dream of "software ICs" being plugged together, Lego-like, into product code across the software development spectrum was shown in practice to be largely a utopian fantasy.
- Product A is highly specialized or sufficiently different from a follow-on Product B that there is insufficient functional overlap to warrant much code reuse of Product A firmware in Product B anyway.
- Design and implementation for reuse costs more than developing a point solution, by a multiplicative factor; the tradeoff would be for long-term TCO vs. short-term ROI, and management will almost always rule in favor of the short-term bottom-line.

It's true... various software engineering projects have suffered from one or more of these liabilities. Why, then, is this notion, seemingly from a past era of software engineering history, explored as beneficial?

First, it is not the predetermined fate of all software development initiatives to fall prey to the pitfalls enumerated above. Because there are countless ways to drive a car off the road doesn't mean that it is the fate of all cars to wind up in the ditch. Software reuse requires a skilled driver who can steer carefully between insufficiently generalized designs and wastefully over engineered solutions, who can make wise choices on what specific portions of the software architecture

benefit from this approach, and who can carry out the entire process with discipline and efficiency.

Secondly, code reuse is hardly a blast-from-the-past fad. Trends are driven by the novel and alluring, and because the software engineering community is no longer abuzz with this once-new idea doesn't mean that it is now archaic. Microwave ovens were once an amazing new technology, and are now as mundane as newspaper. Software development projects that take place in a less feverish environment than those in the embedded storage industry have benefited immensely from disciplined use of these techniques, benefits that have real bottom-line impact on engineering budgets and calendars.

Finally, embedded storage software development and the current state of affairs in the storage industry creates a proving ground perhaps ideally suited to take full advantage of the benefits of software reuse. A fortuitous factor in this regard is the industry's embrace of standards – while many other software sectors continue to support custom interchange formats, protocols, and interfaces, the storage industry has consolidated these into a small subset which are used almost ubiquitously. The key to this unprecedented cooperation is the SCSI suite of standards, which overarches and unifies a number of other higher- and lower-level interface architectures into an orderly common hierarchy, and which has grown and evolved over the decades to accommodate the latest technologies.

What the studied application of code reuse to the storage firmware development process enables is a new conceptualization for storage product development – commoditization of the storage software stack. In the same way that the industry is moving toward using premanufactured hardware componentry and open-systems RTOS's for basic platform engineering, it is now possible to extend those inherent advantages down into what used to require custom firmware.

Just as no sane storage product developer would choose to develop their own proprietary BIOS, real-time operating system, or TCP/IP stack anymore, so has it become nonsensical to develop and redevelop the same core infrastructure for low-level controller chips, interconnect protocol stacks, basic data flow management, standard SCSI support, and generic device control for each new product. It is now possible to offer a **storage firmware infrastructure** that is adaptable, full-featured, high-performance and (above all) reusable, rendering custom development of these capabilities a thing of the past

The Breakthrough Systems Approach

Breakthrough Systems introduces the Embedded Universal Representation & Exchange Key Architecture (EUREKA™) and the Firmware Foundry™ toolset.

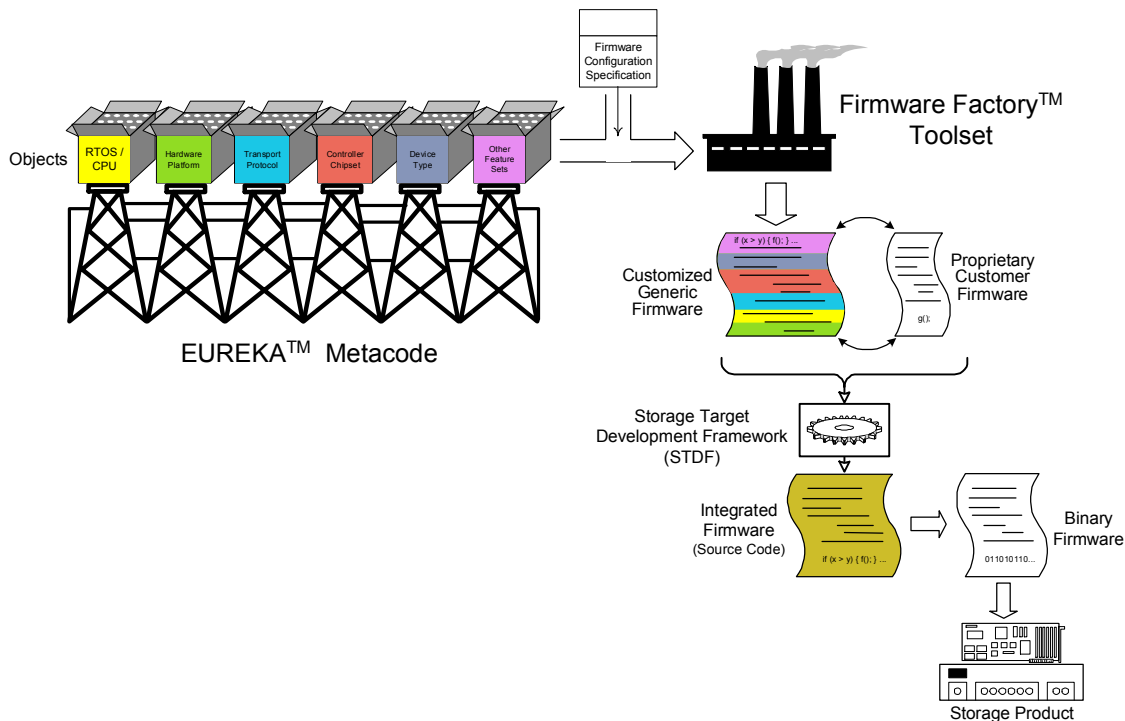
EUREKA™ is an object-oriented embedded storage software architecture that provides the fundamental infrastructure under which firmware for storage products can be built. The objects implemented in EUREKA are organized within a framework that implements code for a wide variety of execution platforms, storage architectures, interface chipsets, and other design alternatives.

EUREKA is an organizational framework and a body of “metacode” objects that support a large span of the universe-of-possibilities that constitute the embedded storage product domain. For any given storage product that may be built; only a subset of this metacode universe is needed. A selection process takes place, and an automated toolset called the Firmware Foundry™ combines the specific collection of chosen metacode into a cohesive body of code – generic firmware customized for a specific storage product. In this way, EUREKA is somewhat akin to a “gene pool”, and the customized generic firmware that is generated by the Firmware Foundry can be thought of as a uniquely individualized “organism” – functioning code that serves as the basis for a storage product.

Depending on the subset of functional components selected from the EUREKA framework, this generic firmware can be as full-featured as desired – it can be configured to be robust and rich enough to completely execute the functionality of certain storage products (e.g., standard SCSI storage devices, virtualized appliances, switches, etc.) without any added external code. Typically, though, the business initiative for a new storage product involves the inclusion of proprietary IP that adds specific market value to the product – new ideas that remain under the confidentiality domain of the storage vendor. EUREKA objects and layers have been meticulously crafted to provide the necessary hooks and interfaces to integrate the generic firmware generated by the Firmware Factory seamlessly with proprietary firmware provided by the customer.

Developed and honed over the span of an entire decade in the storage industry trenches, BSI has made the up-front investment into a powerful Key Architecture built for code reuse – an investment that that storage product developers can't afford to make, because they're trapped in the frenzied “point solution” lifecycle.

Here's how it works...



The development process starts with a storage product developer determining a number of key architectural characteristics of the storage product, across various dimensions:

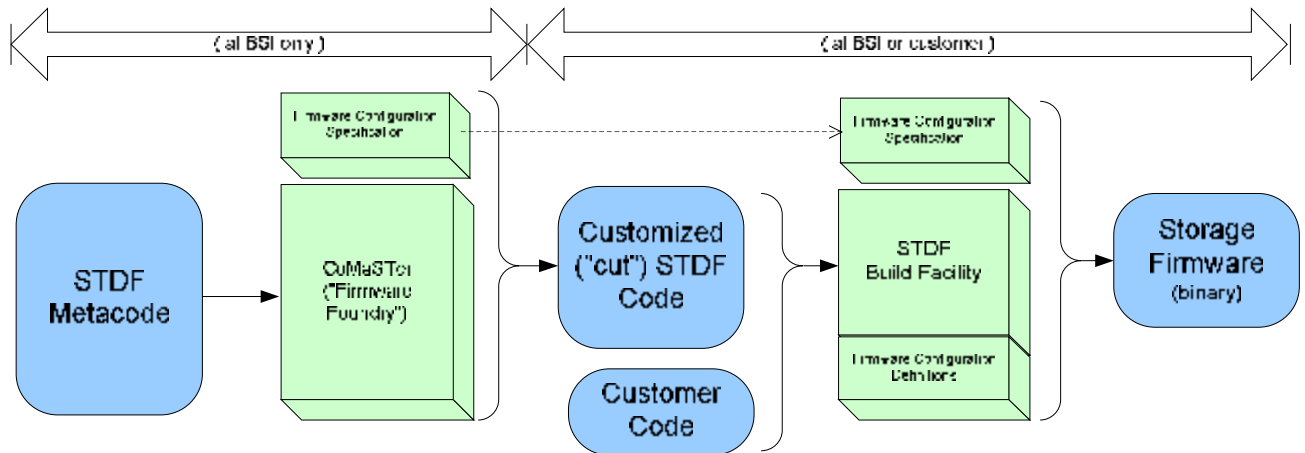
- what processor(s) the product firmware will run on
- what RTOS will be used
- what code development toolset will be used
- what basic hardware platform interfaces will be required, such as bus architecture, memory architecture, DMA capabilities, interrupt handling, etc.
- what storage transport protocol(s) will be used – e.g., Fibre Channel, iSCSI, SAS, InfiniBand, etc.
- what interconnect controller chipsets will be used (e.g., QLogic, Marvell, proprietary ASICs, etc.)
- what levels of the various interconnect and storage architecture standards are needed
- what storage virtualization capabilities are required
- what peripheral device types and standards levels are applicable
- what specific device emulation characteristics are needed

- what speeds and feeds are necessary for core data paths vs. management information paths
- what external management capabilities are required
- other specific feature sets and capabilities

These characteristics comprise a Firmware Configuration Specification, which is essentially an *ala carte* list of features and specific customizations that will be built into the generic firmware for a storage product. These choices are submitted to BSI, who uses them internally as a “work order” specification to the Firmware Foundry toolset. The Firmware Foundry performs the software text processing magic necessary to select the appropriate subset of the EUREKA metacode, and reconstitute it into a standalone body of generic firmware source code.

The generic firmware is then either provided to the customer in source, partial-source, or object library form, along with a development framework called the Storage Target Development Framework (STDF). This framework serves as a rich development environment, which the storage vendor development team uses to develop the proprietary value-added portion of the storage product.

STDF Foundry and Customer Code Integration Process



References

¹ Fred Moore, "Storage: New Game, New Rules", Horison Information Strategies, 2003.

² Ibid.

³ Ibid.

⁴ Fred Moore, "A Storage Manifesto", Storage Technology Corporation, 2001.

⁵ Enterprise Computing Association (Encompass US), "Computing Commoditization", 2004

⁶ Electronic Design, "GPOS's Add Security to Embedded Systems", 2005