Creating OpenRMSP:

Developing a DASH Reference Implementation

Overview

In recent years, we have seen an amazing transformation in the IT space. More computing power is available at lower cost, and it can be easily and economically purchased in scalable pieces, such as commodity servers or individual blades in a chassis. New technologies including virtualization also offer solutions to better utilize and compartmentalize computing power. The availability of all this optimized computing power is timely, as the demand from business for new services and capabilities is growing to keep up with today's high tech needs.

If computing power is getting less expensive to purchase, and the demand for power is on the rise, the simple conclusion is that IT budgets should be fine. But what these observations don't show is that, while capital expenditure per computing power is constantly decreasing, operational costs are skyrocketing:

- In 2000, the power needed to drive $1000 of server capital expense was 32 watts. By 2006, that number has risen to 125 watts, and it is projected to reach 482 watts by 2012.[1] As these systems use more power, they run hotter, pushing up the electric bill with cooling needs. Plus, energy costs are trending upward.
- Commodity servers and blades offer great flexibility, but they all need to be managed, creating an exponentially growing burden on personnel and tools. Additionally, activities like license management become more complex, requiring more IT staff time to control.
- Virtualization compounds the problem of "server sprawl" by converting one physical server into many virtual ones that must all be managed. While virtualization solutions can help manage virtual systems, they cannot completely alleviate the increase in complexity. License management, for example, becomes even more complicated. More software licenses are generally needed and the rules regarding how they can be used can be quite complex. Plus power management increases in complexity as virtual servers share physical hardware, and migrate dynamically from one device to another.

In response to these trends, standards produced by the Distributed Management Task Force (DMTF) such as CIM, SMASH, WS-Man, and now Desktop and mobile Architecture for System Hardware (DASH) are gaining in popularity and relevance. These standards offer advanced ways to manage systems in a non-proprietary fashion, allowing IT professionals, in theory, to integrate multiple vendors management solutions into one cohesive whole that can give them the necessary information and control to manage their infrastructure and simplify their IT operations.

As the DMTF technologies are beginning to see greater deployment in servers, the newer DASH standard is just starting to see interest in the market. Here is DMTF's description of DASH from their web site[2]:

The DASH Initiative is a suite of specifications that takes full advantage of the DMTF's Web Services for Management (WS-Management) specification – delivering standards-based Web services management for desktop and mobile client systems. Through the DASH Initiative, the DMTF provides the next generation of standards for secure out-of-band and remote management of desktop and mobile systems.

As IT professionals start to recognize the importance of proper management to contain operational costs of their servers, they are also starting to pay attention to the much larger scale issue of containing the costs of desktop management. Indeed, a very large percentage of IT resources is spent not in the data center but in maintaining and supporting desktop users. Desktop systems are often of greater heterogeneity than their server cousins (hardware and software), and also have to deal with users' activities, which can be unpredictable and possibly detrimental.

Unless the environment is very tightly controlled by IT policy, most companies' users have a bewildering mix of applications and functions on their PCs and laptops. Because of this and the distributed and remote nature of the managed entities, the desktop/mobile management challenges can be even greater than those faced by the data center, and can consume huge amounts of IT resources. By providing an open standard for the management of data collection and control, DASH allows various hardware systems, operating systems and applications to be managed in a uniform way.

To take advantage of these benefits, each PC needs to have a management agent to collect data and perform administrative operations. This might be handled in software or hardware, but hardware is more desirable as it cannot be compromised by OS failure and may not be impacted by some types of PC hardware failure. This is generally accomplished in some kind of expansion card or with hardware directly integrated into the motherboard.

What is OpenRMSP?

OpenRMSP (Open Remote Manageability Service Processor) is an open source project to implement the DASH 1.1 specification by the DMTF. The project contains various components:

- Firmware designed to run on next generation south bridge Manageability Service Processor (adaptable to other hardware) inside a PC or notebook.
- Host agent software that runs on the managed PC to interact with the firmware.
- Management client software that runs on the managed PC or on a remote system to monitor and manage the PC. Part of this includes the DASH SDK, which allows third parties to create management consoles and tools that interoperate with OpenRMSP.

The firmware and the host agent software work together to collect information from the system, then allow entities to interact with it following the DASH 1.1 standard. The two components also allow for remote management of the system per the methods defined by the DASH 1.1 standard.

Purpose of this White Paper

Raritan is committed to providing real value in open standards-based IT management. To support that commitment, we created this paper to raise interest and increase understanding about DASH and the OpenRMSP project. This paper will discuss:

- Value of using the OpenRMSP implementation in your own environment
- Architecture and capabilities of OpenRMSP
- Design and process of creating OpenRMSP

Note: This is not a development guide. If you are interested in the technical details of adopting this technology, information will be provided at the project's Web site as it becomes available:
http://dash-management.wiki.sourceforge.net/

Benefits of OpenRMSP Implementation

Adopting the OpenRMSP software for your application offers a multitude of advantages. While you may find many of your own benefits, here are some of the ones the team has uncovered:

- **Management Solution Control.** OpenRMSP implements the open DASH 1.1 standard, so you can interact and interoperate with any DASH–compliant devices. Additionally, since all the source code is freely available, you can modify it as you like to meet your needs.
- **No Licensing Fees.** OpenRMSP is open source, so there are no licensing fees to use it, copy it, deploy multiple copies or modify it yourself. This encourages quick adoption and leads to growth of a strong support community. It also offers a lower barrier to entry, as you can experiment with it in your environment without any financial investment and little is lost if you find it doesn't align with your needs.
- **Better Stability and Reliability through Community Growth.** A large customer community for any open source software helps to improve the stability and reliability of open source code. Over time, as the original developer or other community participants fix and enhance the code, the value and benefits of the product increase while the cost remains the same: zero! Further, as the source code can be reviewed by everyone, best practices are encouraged and issues are found and addressed.
- **No Vendor Lock-In.** Adopting an open source based management firmware stack has less lifecycle risk for the customers than proprietary software, as the control and ownership of the source code is not under a single company.

  When dealing with proprietary solutions there is a high risk the provider's needs are not aligned with yours, and once heavily invested in such a solution, you have little recourse if it is not working well for you. In the case of management based on specialized hardware (like many remote management implementations), once you choose the software you must continue buying compatible hardware in order to keep reaping the benefits, creating another form of vendor lock-in. Proprietary solutions also may discontinue products, leaving you with an unsupportable old version or forced to invest in a newer version. With OpenRMSP, there is nothing to stop you or a third party from adapting it to any new or existing hardware in the market. This reduces the risk of obsolescence. There is also no risk of dealing with a special firmware supplier and getting locked into proprietary hardware solutions or suffering from hardware supply shortages. Also, the software cannot be discontinued as you can always use and modify the existing product.
- **Agile Enough to Adopt Emerging Standards.** By implementing a robust and flexible standard such as DASH, you create a base to advance into the next generation of management as it becomes available.

  The IT environment is changing rapidly as new technologies and challenges surface. Standards will need to evolve to meet new needs and provide management of new technologies. An open standard such as DASH is capable of being quickly extended to meet the needs of the rapidly changing IT environment. And by having an open source solution that anyone can enhance, OpenRMSP is capable of rapidly supporting new extensions by the work of the entire community. To allow end users to quickly take advantage of the existing and emerging benefits in OpenRMSP, a powerful SDK has been created. This allows rapid and efficient implementation and integration with other systems. And due to the emulation provided by the OpenRMSP project, new features can be easily implemented and tested first on a standard Linux system before they are ported to the target management hardware.

- **Easy Platform Adaptation.** Although the OpenRMSP project focuses on a few target environments, such as the Linux emulated system and the AMD Manageability Service Processor, it was designed from the ground up to be modular with clean separation in the architecture (see the next section). This allows other hardware to be easily supported by just creating some appropriate low level software to integrate with the desired hardware. This also offers opportunity to hardware manufacturers as a way to quickly test new management hardware.

Architecture of OpenRMSP

OpenRMSP is a complex project made of many different components that need to work together to meet specific goals. To understand how it works, a clear understanding of the architecture is necessary.

OpenRMSP Architecture Design Goals

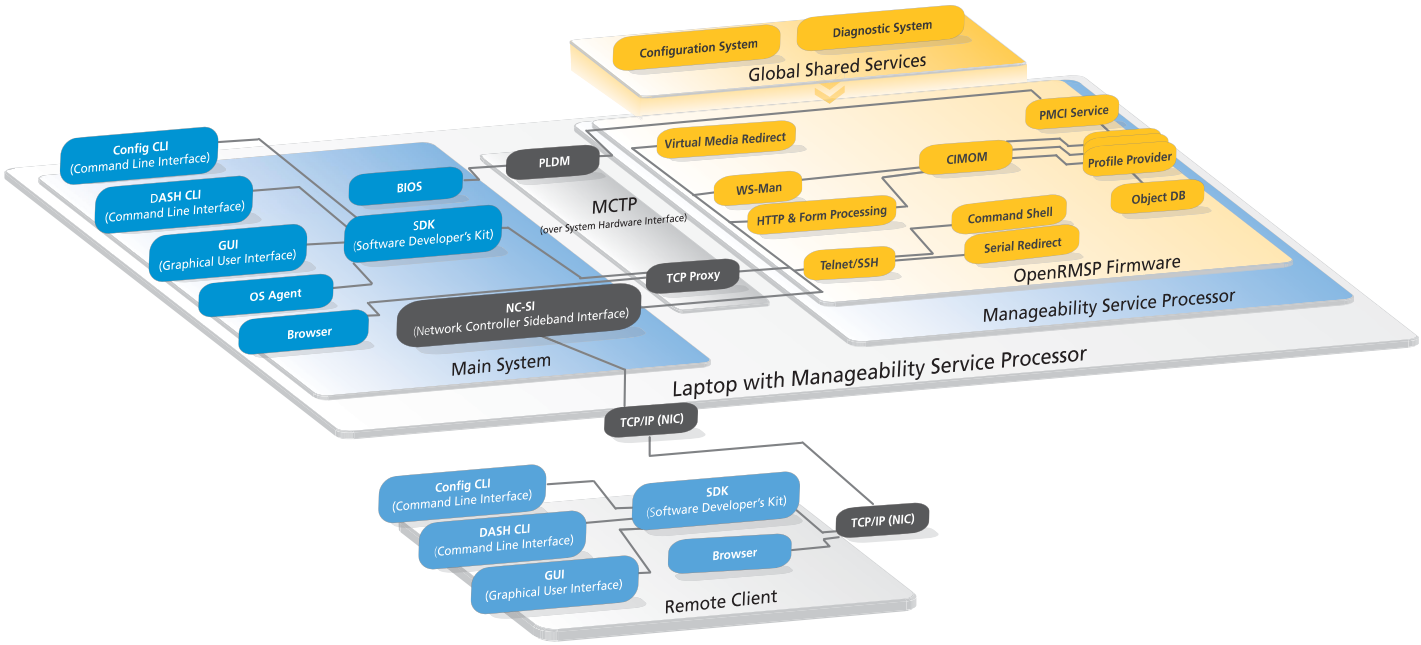
OpenRMSP Firmware architecture is based on a number of goals that guided the design of this firmware and the overall system. In priority order, they are:

- **Deliverability** – The product must be delivered as promised, meet all the requirements, be completed on time, and contain high quality.
- **Long Run Stability** – The firmware must run for long periods of time without any reboot or reset (possibly months or even years). In the case of other system failures, the firmware must be the primary access point, so a focus on the reliability of this component was critical.
- **Efficient and Standards-based Development** – Due to the accelerated pace of the project and the large number of geographically dispersed developers involved, the focus needed to be kept on well-known Linux design principles to increase the predictability and reliability of the development process.
- **Small Footprint** – As the OpenRMSP firmware is designed to run in embedded management hardware, the cost of RAM and flash memory was a key concern. The initial target was to run inside 8 MB of flash memory and 32 MB of RAM. An additional goal was to reduce that (for cost savings) in the future to 5 MB of flash memory and 24 MB of RAM.
- **Reasonable Performance** – As most of the components in this system are not performance critical, this goal was ranked lower than the others discussed; however, an acceptable level of performance was required.

> "We introduced a completely new architecture to meet stability and modularity requirements...The code as well as the knowledge we gained can be reused in any of Raritan's products..."
>
> *– OpenRMSP Software Architect*

OpenRMSP System Architecture Diagram

Below is an architecture diagram of OpenRMSP. The following sections then give a breakdown and description of the different components that make up the OpenRMSP system. This analysis is high level but it offers a view into what OpenRMSP is, how it works, and how it can integrate with third party systems.



OpenRMSP Firmware Components

OpenRMSP Firmware Components are designed to run on the Manageability Service Processor. The firmware runs on a small form factor version of Linux. Components include:

- **WS-Man** – This component implements the DMTF WS-Man protocol to allow the external entities to interact with the information in the CIMOM (CIM Object Manager). This is the primary external interface of the OpenRMSP firmware.
- **Telnet/SSH** – This offers Telnet and SSH services to support two types of serial connections:
  - Linux command shell: Access to the Linux command shell on the small form factor Linux on the OpenRMSP
  - Serial port redirection: Redirecting a serial port from the managed PC to a console application or other serial interface
- **HTTP & Form Processing** – This component allows a remote or host user with a Web browser to directly interact with OpenRMSP firmware without going through a machine interface such as WS-Man. This includes some setup and configuration for OpenRMSP along with some limited DASH Profile functionality.
- **Virtual Media Redirect** – Redirect allows a remote user to virtually mount media, such as local drives on the host, over the network as if they were on the remote machine. This can be done from the GUI application or the SDK.
- **CIMOM** – The CIMOM (CIM Object Manager) is the link to all the management information collected by the system and the point where modifications and commands can be dispatched to the system. It is accessed either via WS-Man or from scripts in the forms managed by the HTTP & Form Processing component.
- **Profile Providers (multiple)** – These components are individually responsible for different pieces of information in the system. For example, there might be a Profile Provider specifically to manage the temperature sensor on the chassis. The profile provider has the code necessary to talk to, retrieve information from, and control and manage the hardware that it represents. All Profile Providers then speak to the CIMOM via a standard communication format. They are analogous to device drivers for management hardware.
- **PMCI Service (Platform Management Component Intercommunications)** – This is an interface used by the profile providers to gain direct communication with the management hardware. Due to the complexity of this subsystem, we won't try to cover it in more detail here; for more information on this, review the project development documents.
- **Object DB** – This component holds data for Profile Providers that don't have any hardware interactions. Basically, it is a repository that allows reading and writing of values. For example, a Profile Provider might not be implemented yet to talk to the hardware, so instead it could use the Object DB. Anything written into a particular CIM Profile is saved and will be returned upon request, so you can write and test a management utility that modifies a configuration without the target hardware being available.
- **Configuration System** – This component is used to maintain configuration for the OpenRMSP system. It is accessed by various components within the OpenRMSP firmware. It is not directly visible to external interfaces.
- **Diagnostic System** – This is used to send messages between components. It is used by various components within the OpenRMSP firmware. It is not directly visible to external interfaces, although it can send events out of the system via the WS-Eventing mechanism (part of WS-Man) through an Indication Provider.

- **MCTP (Management Component Transport Protocol)** – The MCTP is defined by the DMTF as "a protocol for intercommunications among intelligent devices within a platform management subsystem. This protocol is independent of the underlying physical bus properties, as well as the data-link layer messaging used on the bus."[4]

  In OpenRMSP, this protocol is used to send messages between the main system and the MSP. The MSP can work with the BIOS on the main system by using PLDM running over MCTP, and can communicate with the OS layer components on the main system by using the TCP Proxy, which also runs over MCTP.

- **PLDM (Platform Level Data Model)** – The PLDM is defined by the DMTF as "an effective data and control source for mapping under CIM. PLDM is targeted to provide an efficient access to low-level platform inventory, monitoring, control, eventing, and data/parameters transfer functions such as temperature, fan, voltage, event logging, and boot control."[4]

  In OpenRMSP, PLDM is used to allow the MSP firmware to collect data from components residing on the system main board. These could include fans, temperature sensors and other low-level monitoring and control systems. These are accessed by using the PLDM protocol over the MCTP protocol.

Host/Remote Components

Host/Remote components run either in the host or the remote as normal PC processes. If you want to adopt OpenRMSP, this is generally the best point of integration. These components include:

- **Config CLI (Configuration Command Line Interface)** – This component allows configuration of the system using a command line, scriptable interface. It includes things like firmware update of the OpenRMSP firmware.
- **DASH CLI (DASH Command Line Interface)** – CLI allows direct access to query DASH MAPs via scriptable CLI commands.
- **DASH SDK (Software Developer's Kit)** – The SDK is used by the other host/remote components to interact with the OpenRMSP firmware on the Manageability Service Processor. This intermediary component is often the best point to integrate with for a third-party application that leverages the OpenRMSP product. This component offers a number of services, including:
  - DASH CLI (Command Line Interface): Scriptable tool and command shell for accessing DASH functionality.
  - High Level API: High level functions that can be called to interact with the system. This includes discovery of MAPs, discovery of profiles, capabilities and services and finally access to functions like power control, basic inventory, boot path selection and others.
  - Low Level API: Lower level functions that the high level functions are built on top of. Provides low level access to the CIMOM. Can be used to extend beyond DASH 1.1 to other CIM-based standards such as SMASH.

  Versions of this SDK are built for Windows and Linux. The API is written in C++ and C.

- **OpenRMSP GUI (Graphical User Interface)** – The OpenRMSP GUI is an application that allows manual interaction with the management firmware. It is able to both collect data for display and send commands to the management system. It can also receive notifications from the management system and display alerts to the user.

Versions of this software are built for Windows and Linux. The software is written in C# and runs on the .Net 2.0 platform for Windows and Mono for Linux.

- **Browser** – This is just the standard Web browser on the system, such as Internet Explorer, Firefox, Safari, or Chrome. Because the firmware has a Web server with form support, a browser can connect to the server directly and interact with it through any functionality that it publishes on its Web interface. This interaction does not require the SDK or any installed software so it can be conveniently performed from any remote system (subject to routing and authentication, of course). Currently this provides access to administrative functions for the OpenRMSP firmware and a few DASH profile dialogs.

> "…the interoperability of the different parts of the complete project is very impressive."
>
> *– OpenRMSP Software Developer*

Host-Only Components

Host-Only Components are designed to run on the managed PC itself, on the main CPU and installed operating system. Unlike the Host/Remote components, these are not useful except on the managed PC itself.

- **OS Agent (Software Agent Interfacing the firmware to the OS)** – This component allows the OpenRMSP firmware to get information and perform operations that require the support of the operating system (currently supporting Windows and Linux). Some examples of this agent's capability are:
  - Power management: Get the current OS state at the request of the firmware, or change the power management state (e.g., go to sleep) at the request of the firmware. Battery status is another example of power management information supplied by the agent that is valuable to the firmware.
  - Status: The agent sends heartbeat messages to ensure that the firmware knows the OS is operating properly.
  - Version and other useful information: The agent sends useful management information to the firmware, such as the type and version of the operating system.
- **TCP Proxy** – The TCP Proxy emulates a TCP/IP network connection to allow the host software components to talk to the OpenRMSP firmware using the same protocols that the remote system can use. This reduces the number of interfaces and ensures that most host applications can also be used on a remote client. The TCP Proxy uses MCTP (Management Component Transport Protocol), which is part of the DMTF PMCI specification suite, to pass information between the host applications and the OpenRMSP firmware.
- **BIOS (Basic Input/Output System)** – This is the standard BIOS running on any server or PC that hosts the OpenRMSP software.

Feature Set of OpenRMSP

OpenRMSP is a robust and flexible software project that has many desirable capabilities. The following are some that focused on the solution features that allow it to be conveniently adopted and the DASH features that offer a high level of compliance with the DASH 1.1 specification along with rich DASH functionality.

Solution Features

OpenRMSP offers great flexibility in ways that your users and applications can utilize the power of OpenRMSP, including:

- **Browser** – The simplest way for a user to get basic access right into the system, immediately and from anywhere.
- **DASH GUI** – A simple and effective GUI to manually review and adjust DASH configuration. This is an effective way to test and verify application behavior that is being integrated through some other means. The DASH GUI also offers virtual media mounting.
- **DASH CLI** – This scriptable command line interface allows a straightforward way to programmatically access and manipulate the DASH information.
- **Configuration CLI** – This scriptable command line interface allows a straightforward way to programmatically access and manipulate the OpenRMSP configuration and perform maintenance such as firmware upgrades.
- **DASH SDK** – Giving high-level control to perform all major DASH 1.1 operations, and low-level control to manipulate the CIMOM, the SDK offers the most feature-rich and capable way to take advantage of OpenRMSP in your products. For a full-fledged integration, this is generally the correct integration point.
- **WS-Man** – The DMTF WS-Man interface can be used to gain access to all the standard capabilities of OpenRMSP through the network. As this interface is standards-based, it is the simplest way to use OpenRMSP systems with DASH-compliant management systems.
- **Profile Providers** – Creating additional Profile Providers allows support for additional types of custom hardware and expands the information and capabilities the OpenRMSP provides. OpenRMSP supports standard protocols to interact with the system, including PMCI[4], PLDM[4], KCS, HCMI, ASF!, and MCTP[4].
- **Multiplatform Support** – OpenRMSP GUI, SDK, CLI, and agent components are designed to run on both Windows and Linux, providing greater flexibility in the host support and client environment. Linux client systems can communicate with Windows host systems and vice versa.
- **Emulation Support** – OpenRMSP can emulate the DASH profiles that don't have the hardware necessary (or haven't been implemented yet). This allows for more independent development and test work and the ability to deal with heterogeneous hardware. The OpenRMSP project is actually a few implementations, including multiple emulations, a prototype hardware implementation, and a final hardware implementation.
- **Provisioning Tools** – OpenRMSP supplies provisioning and setup tools to configure the system.

DASH Features

The table below lists all the DASH 1.1 profiles and describes the support provided in OpenRMSP:

DASH 1.1 Profile	Description*	OpenRMSP Implementation Notes
Profile Registration	Registers the profiles used in the system	Included in OpenRMSP
Base Desktop Mobile	Autonomous profile for describing desktop or mobile systems	Included in OpenRMSP
Physical Asset	Physical component, chassis, card, FRU representation	Included in OpenRMSP
Boot Control	Boot sequence representation and configuration	Included in OpenRMSP
Power State Management	System power state representation and control	Included in OpenRMSP
Software Inventory	Representation of software/firmware identification and version information	Included in OpenRMSP
CPU	Processor representation and configuration	Included in OpenRMSP
System Memory	System memory representation	Included in OpenRMSP
Fan	Fan status and component representation	Included in OpenRMSP
Power Supply	Power supply status and component representation	Included in OpenRMSP
Sensor	Sensor status and component representation	Included in OpenRMSP
Role Based Authorization	Role and privilege representation and management	Included in OpenRMSP
Simple Identity Management	User identity representation and management	Included in OpenRMSP

DASH 1.1 Profile	Description*	OpenRMSP Implementation Notes
Indications	Subscription, listener destination, event filter and indication representation and management	Included in OpenRMSP
Battery	Battery status and component representation	Included in OpenRMSP
BIOS Management	BIOS configuration and control	Included in OpenRMSP
DHCP Client	DHCP client configuration and control	Included in OpenRMSP
DNS Client	DNS client configuration and control	Included in OpenRMSP
Host LAN Network Port	Network port/LAN configuration and control	Included in OpenRMSP
Ethernet Port	Ethernet port configuration and control	Included in OpenRMSP
IP Interface	IP Interface configuration and control	Included in OpenRMSP
OS Status	OS representation and management	Included in OpenRMSP
Opaque Management Data	Opaque data representation and management	Included in OpenRMSP
Software Update	Software/firmware installation and update	Included in OpenRMSP
KVM Redirection	KVM (Keyboard, Video & Mouse) console redirections management	Not Implemented in OpenRSMP
Media Redirection	Media redirections management	Not Implemented in OpenRSMP
Text Console Redirection	Text console redirections management	Included in OpenRMSP
USB Redirection	USB redirections management	USB Media redirection in OpenRMSP

** Profile Descriptions taken from DMTF DASH 1.1 White Paper.[3]*

OpenRMSP also supports:

- **Integration with Active Directory and Kerberos** – For robust remote authentication and authorization, OpenRMSP can use Active Directory and/or Kerberos for AA.
- **DASH 1.1 MAP Discovery (through GUI applet or SDK)** – OpenRMSP can search for and discovery MAPs (Manageability Access Points) through a range of IP addresses or a local subnet broadcast.

Development / Test Environment and Technologies

Creating a large and complicated project like OpenRMSP requires a strong development team and strong tools and processes to be successful. With over 20 people on the Raritan team alone and additional support from other development teams in other organizations, our project needed effective processes and tools to keep everyone working together effectively. At the same time, a careful balance was needed to ensure the team was not overburdened with overhead that would reduce efficiency and flexibility.

While every project is different and there are many ways to attack the complexity of developing such a project, here are some of the environments and technologies that the OpenRMSP team employed to create an effective and successful software product. In the following section, we will look at some of the processes and practices that were used to drive the project forward.

Development Environment

The development environment was a standard Linux environment, consisting of a gnu tool chain, gnu debugger, tftp loaded kernel, and nfs mounted root file system. All of this ran on an ARM processor. Additionally, there were two emulation environments: an x86 build that ran most of the applications, and an x86 user mode Linux build, which ran the kernel and all the system libs and applications.

Most of the project development took place on an evaluation board that was not identical to the final hardware, as the final hardware was not available until later in the development cycle. The software had to be designed to be flexible to be able to adjust quickly once the hardware became available.

Development Technologies

To keep the project organized and moving forward efficiently, a number of readily available technologies were adopted by the OpenRMSP team. Here are some of them and what benefits they have for the team:

- **Doxygen** – Doxygen generates implementation reference documents by finding comments in code. To properly use this software, comments must be written and they need to follow a defined format. But by ensuring that the development team adhered to the tool's requirements, precious time was saved as nobody had to constantly create and update the documentation. Additionally, the information was more accurate because the author of a function or other component was the one writing the description instead of a third party, and because it was updated live every time the code was changed.

- **Twiki** – Twiki is one of a number of freely available wiki software packages. These packages provide a collaborative place to collectively store and share information. Wikipedia.org is an example of a public wiki. But many other wikis exist as well, and many are kept private for projects. The OpenRMSP team used Twiki to store many important project artifacts, including project documentation, project task and schedule tracking, project procedures and rules, application notes on various technical issues, links to external resources like standard specifications, and much more. While much of this is private, portions of the wiki are publicly accessible.
- **Subversion** – Subversion is a freely available source code control tool. This keeps a repository of all the source code in the project and tracks changes to it. This protects the code and allows developers to find issues that were introduced at a particular time and version, along with many other valuable benefits.
- **Eventum** – Eventum is a change-request tracking system. This freely available system is used to track bugs discovered in testing along with feature requests and enhancements. All change requests are related to a particular build so it is easy to find out when a particular issue was introduced or fixed. In this case, separate projects were kept for the OpenRMSP firmware, the SDK, and the host software.
- **Tinderbox** – Tinderbox is used to visualize the status of builds. It also notifies developers who have made changes if a build fails so they can quickly remedy the situation.

Test Types

Testing in any large-scale development project is critical, and OpenRMSP used various test approaches to meet this need, including both automated and manual testing.

- **Build Testing** – Build testing was used to ensure the software can always be built. It was performed by checking to make sure the entire package builds correctly by an automated system reviewing build errors.
- **Unit Testing** – Unit testing was designed to find errors at the module or function level, using simple tests designed to exercise the scope of the module's behavior. Unit testing used the CppUnit framework to test every module each time it was built.
- **Use Case Testing** – Use case testing was the procedure of using the system as a normal user would and ensuring it behaved properly. It employed Python scripts to run through specific use scenarios to ensure the entire system behaved as expected. Some use case testing also needed to be handled manually by the testers.

Test Environment

The test environment included a number of systems. The build and unit test systems ran Fedora Core 8, Fedora Core 9, Ubuntu 8.04, and Windows XP, some with an actual IMC ARM-based evaluation board (to run the MSP firmware), and some using just emulation of the MSP.

For system testing, virtual machines were used, including 3 Fedora Core 9 systems and 3 Ubuntu 8.04 systems on 2 hardware systems, and virtual machines of Windows 2008 Server, Windows XP, and Windows Vista running on an additional hardware system. Additionally, 3 of the ARM-based evaluation boards used for the MSP development were available for target tests.

Test Technologies

- **TestFramework** – TestFramework is an internally developed module that runs on the OpenRMSP firmware and orchestrates the running of the automated tests whenever needed (e.g., when a new version is built).
- **CppUnit** – CppUnit is a freely available unit testing framework similar to JUnit but designed for C++ source code. This ensures that individual classes, functions, and modules fulfill their role correctly. By ensuring that each individual entity is unit tested, the overall system quality is greatly improved.
- **OpenTestMan** – OpenTestMan is an open source validation suite that is designed to test interoperability of MAPs claiming to be DASH 1.1 compliant. As OpenRMSP strives to be DASH 1.1 compliant, this is a critical part of testing in the system.
- **WinRM** – WinRM stands for Window Remote Management command line tool. This tool is provided in Windows Vista and Server 2003 and allows for access to CIM data, among other things.
- **Python** – Python, a common scripting language, is used for a number of use case tests. The use case behavior is written in script and the results are checked to ensure that the system performs the use case correctly.
- **ProTon** – ProTon is an additional internal project to create a test tool as an alternative to OpenTestMan based on PDD files. This offers additional test options to the team.
- **Testlink** – Testlink is a freely available, Web-based software for managing test cases. It keeps track of all the tests to be performed and can organize them into test plans, track results, make reports, etc.

"The development team is more distributed around the world than what we were before. That required us to be much more formally organized with respect to interfaces, design, tasks, components, and test... …our organization level had to improve a lot...."

– OpenRMSP Software Architect

Team Processes and Practices

The environments and technologies discussed in the last section help to give structure to the task of creating a large software project such as OpenRMSP. However, the tools and technologies alone were not enough to keep the activities on track. On top of the tools and technologies that supported the developers, clear processes and best practices had to be defined. This ensured the team behaved in a predictable and effective fashion, despite being made up of so many different individuals, some of whom are geographically remote.

Some of the tools and technologies discussed already defined workflows that guided the development process. But beyond which practices the tools encouraged, the team also agreed on and followed a number of processes to ensure a successful project. In this section we review some of the policies, processes and practices that the team developed in order to function at their best. Some of these were the result of experiences during the OpenRMSP project, while others were the result of many other projects with Raritan prior to this one.

Processes for the team were documented either in Twiki (project private wiki site) or in files accessible from Twiki. This documentation was binding on all team members.

General Processes and Practices

Project Roles

To focus the team's activities properly, individuals filled out different roles. The OpenRMSP team contained the following roles:

Role	# of Individuals	Purpose
Program Manager	1	Central contact point, customer advocate, coordinates cooperation with outside teams
Project Manager	1	Track tasks and keep project on schedule
Software Architect	1	Design and support the software architecture
Lead Engineers	2	Lead individual development teams (Firmware and Host/SDK)
Software Engineers	12	Create the OpenRMSP software
Test Engineers	2+	Ensure product quality and fitness (number of testers varies according to product activities)

Note: This only includes Raritan OpenRMSP team members and does not represent the effort of the supporting partners in this project.

Project Communications

Project data was contained mostly on the private Twiki site. For an individual within Raritan to gain access to this site, it must be approved by the project management for the project. Public communications were posted in different areas of the wiki and on SourceForge.

In addition to many specific topics, basic information about the project structure, acronyms, architecture and other critical topics were available to team members. That makes it much easier for new members to the team to quickly understand what was going on, how things were done, and what the project entailed.

Development Processes and Practices

- **Development Process** – OpenRMSP used a modified version of the XP development process. Through team members' experience with other Raritan projects in the past, an effective agile development process evolved and was used in OpenRMSP. The OpenRMSP process emphasized certain XP principles such as:
  - **Test-Driven Development** – Automated unit testing was written by developers before or during initial coding, not afterwards. Unit tests were written for all code. System tests were written by testers during the development process. More on this below.
  - **Continuous Customer Feedback** – Due to a regular and frequent release schedule, the team received constant feedback from the stakeholders and was able to quickly adapt the design to meet the needs that arose.
  - **Collective Code Ownership** – The team was comfortable with the code and used defined coding practices, so any team member could work effectively on any area of the code.
  - **Iterative Development** – The team planned and delivered frequent releases at 4-week intervals. Each release had a clearly defined set of features that needed to be delivered in production quality.
  - **Regular Refactoring** – The team didn't become overwhelmed in designing the solution up front, but used experience and judgment to create solid designs that met the current needs. If the designs were not sufficient for future needs, the team refactored the code to provide the additional functionality at the time it was needed.
- **Object-Oriented Software Design** – One area in which the team was determined to advance itself was into greater use of object-oriented software design. Due to embedded systems limitations and legacy code inertia on previous projects, the team had a limited ability to employ these technologies in the past. With OpenRMSP offering a fresh start, the team jumped immediately into object-oriented languages, specifically C++ and C#. Although the team could have accomplished this project using other technologies, this proved to be a great opportunity to take advantage of these concepts.
- **Change Request Process** – One of the key processes in the development progress was the Change Request (or CR) process. This defined how changes were handled, including everything from fixing defects discovered in testing, enhancing existing features and usability and adding new functionality. At every state change in the life of a CR, the team member responsible could add notes to ensure the reason for the state change was clear.

  The tool Eventum was used to track change requests. Whenever a change was needed, any individual could create a new change request to begin the process. Then the new CR would be reviewed by the CRRB (Change Request Review Board) prior to continuing the process.

The CRRB was a small group of team members who had the responsibility to manage the flow of new changes into the project. It included the Program Manager, Project Manager, Software Architect, Lead developers and optionally additional developers and testers as needed depending on the nature of the change request.

If the new CR was approved by the CRRB, it became an "Open" CR and the CRRB assigned to a developer, set its priority, and targeted it for a specific release. The developer then fixed the CR and it was verified by test. Once the developer believed the CR to be fixed, they set the state to "Fixed" and they checked in their new source code. The source code modifications were associated directly to that CR. The developer also provided unit testing code for the automated systems to test the fix. From there it went into the test processes.

If a developer was unable to fix a CR, they could set it to one of the following states instead:

- **Functions as Intended** – The product was working as it was designed to do. The change was rejected, although possibly some additional actions needed to be taken, such as better documentation of the behavior and/or reconsideration of the design.
- **Cannot Reproduce** – The developer was unable to reproduce the problem in order to troubleshoot it. In this case, the developer would start by working closely with the person who reported the CR to see what could be done to replicate the undesired behavior. If the developer, testers and original reporter could not reproduce the issue, it would be closed by the CRRB in this state. The CR could be reopened if the problem recurred, so it remained in the system as a permanent record of any effort that went into investigating the issue in case that information was useful in a future investigation.
- **Will Not Fix** – The CR's value was too small in comparison to the effort involved in remediating it, so it was not worthwhile to fix it.

  If the CRRB was in agreement with the developer's assessment, they would change the state to the "Verified" version of the state (e.g., "Cannot Reproduce" to "Verified Cannot Reproduce"). If a CR was fixed in a way that was not intended by the original request, the CRRB could make the final decision on whether the fix was acceptable. The CRRB could reopen the CR (set to "Open" and return to the developer) if the approach taken to fix it was not satisfactory.
- **Release Management** – All development of new features in OpenRMSP took place on the main branch of the codebases, or "trunk." Daily snapshots were taken and built into internal builds for the tester to use for verifying CRs. Weekly snapshots were also taken; those could be shared with the customer representatives for review. Both daily and weekly snapshots included an automated list of all CRs fixed since the last build of that type. As these snapshots were published automatically, they undergo automated testing only and did not have manual verification prior to release.

  When the features were complete for a particular release, the code was branched to ensure that it could be stabilized and the quality guaranteed. New features continued to be added on the trunk but did not impact the current release; only defect fixes would be applied to the release branch. Fixes made in the release branch would later be applied back to the trunk. Multiple releases made along a release branch were labeled as minor releases, whereas new branches that were created with additional features were identified as major releases. Official releases, both major and minor, underwent full automated and manual testing prior to publishing, and included lists of all changes since the last release, as well as a list of known defects that had not yet been addressed.

- **Release Scheduling** – As this project was working closely with a primary customer, the following strategy was used to keep the project on schedule. Once a major or minor release was given to the customer, the customer had one week to decide how long they would need to test it. Once the customer determined the length of testing needed, they began testing it and filing CRs for any issues they discovered. In parallel the CRRB started to review the CRs and got developers working on them for the next release. Once the agreed-upon testing duration was reached, the CRRB would not accept any new CRs for the upcoming release. Any concerns discovered after that time would be targeted for later release. This prevented an endless cycle of new issues being discovered while others were fixed, which could indefinitely delay the release and impact the project schedule.

Test Processes and Practices

In the OpenRMSP project, the team aggressively pursued more robust test practices than in prior projects. A large focus was placed on automated testing and making sure everything was unit tested to guarantee a high quality product.

- **Build Process** – Builds were automatically created with daily and weekly snapshots as mentioned in the prior section, and could also be manually triggered as needed for special purposes. The build system automatically performed the build testing and unit testing on the build when it was created (see below). The build system assigned a version number (manually set) and a build number (automatically incremented). Assuming everything succeeded, it also created release notes based on the information of fixed CRs from Eventum and copied the output to the appropriate places.
- **Build Testing** – The build system automatically checked for any build errors, and if any identified the source files responsible and sent notifications to the developers.
- **Unit Testing** – After the build completed successfully, the build system automatically ran all the regression tests that had been defined for the system. All CRs would be tested by the appropriate automated test code. If the automated testing indicated that the CR passed, the CR would be changed from "Fixed" to "Built Fixed." If the CR failed, it was returned to the developer and placed in the open state.
- **Use Case Testing** – Use case testing was performed either by Python scripts automatically after the build or by a tester manually performing the operation in question.
- **Manual Testing** – When the automated system identified a CR as "Built Fixed," it passed into the test team for human validation. In the daily task meeting, all CRs that were identified as "Built Fixed" from the prior night's build would be assigned to one of the testers. The tester needed to both manually run the automated regression test to ensure the result was accurate, and then manually perform the steps that caused the issue to see if it was truly fixed. The tester may then, at their discretion, perform additional testing that they felt was appropriate to ensure the issue was truly resolved. Any additional testing and its results needed to be documented in the CR.

  If any of the test activities failed, the CR was returned to the developer in the "Open" state, along with notes from the tester as to what the issue was. If everything was satisfactory, the tester changed the state of the CR to "Verified Fixed" and assigned the CR back to the reporter.
- **Final CR Closure** – A reporter received their CR back in any of the "Verified" states (Verified Fixed, Verified Won't Fix, Verified Functions as Intended, Verified Cannot Reproduce). The reporter would then verify they were satisfied with the results of the CR. Assuming they were satisfied, they may close the CR. If the reporter had further concerns, they could return it to the open state.

Conclusion

OpenRMSP was a huge project with an excited and dedicated team. Hopefully, this paper has given some insight into the OpenRMSP project, from its technology, the factors in its success, down to the individual people who made it possible.

We encourage you to take advantage of the enthusiasm and hard work of this team by leveraging the OpenRMSP product in your own applications. Visit the OpenRMSP project for more detailed information on how to adopt OpenRMSP into your applications:

http://dash-management.wiki.sourceforge.net/

Should you have any issues regarding this open source, please communicate it through the Sourceforge site directly, or send an email to openrmsp@raritan.com.

Our engineers are willing to reply and help you resolve your problems.

By doing so, you may avoid many of the pitfalls that the OpenRMSP team had to face and overcome. We also hope you have an understanding for the effort required to tackle the creation of the DASH implementation from scratch.

If neither of these dissuades you from trying to tackle creating a DASH implementation on your own, then we hope some of the experiences here will aid you in your own projects.

"Everything you do with passion has positive personal benefits."

– OpenRMSP Software Architect

Bibliography

[1] **THE UPTIME INSTITUTE:**
The Economic Meltdown of Moore's Law and the Green Data Center, 2007.

[2] **DISTRIBUTED MANAGEMENT TASK FORCE, INC.:**
Desktop and mobile Architecture for System Hardware (DASH) Initiative.
http://www.dmtf.org/standards/mgmt/dash/.

[3] **DISTRIBUTED MANAGEMENT TASK FORCE, INC.:**
Systems Management Architecture for Mobile and Desktop Hardware White Paper, Version 1.1.0, 2007.
http://www.dmtf.org/standards/published_documents/DSP2014_1.1.0.pdf.

[4] **DISTRIBUTED MANAGEMENT TASK FORCE, INC.:**
Platform Management Component Intercommunications Architecture, Version 1.0.0a, 2007.
http://www.dmtf.org/standards/published_documents/DSP2015.pdf