Validation of Redfish – The Scalable Platform Management Standard

by

Priyanka Kumari, B.E.

A Thesis

In

Computer Science

Submitted to the Graduate Faculty of Texas Tech University in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCES

Approved

Dr. Yong Chen
Chair of Committee

Dr. Alan Sill

Dr. Yu Zhuang

Mark Sheridan
Dean of the Graduate School

May, 2017
ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor Dr. Yong Chen and my mentor Dr. Alan Sill for their persistent support, guidance, and encouragement without which this work would not have been possible. I thank them for providing me with the opportunity to work under their valued supervision. They were always there to help and guide me with my research and report. The experience has been enriching for me to grow both professionally and personally, and in a whole helped me become a better person. I would also like to thank Fatima Saleem from Intel and Paul Vancil from Dell for their support and invaluable guidance throughout my research. I am also thankful to DMTF for providing me the opportunity as well as all the needed resources to make this research successful. I really appreciate all their time and effort.

I would like to thank my committee member, Dr. Yu Zhuang, for his timely help, valuable suggestions and inputs. I would also like to thank TTU students Elham Hojati, Ali Nosrati and Ghazanfar Ali, for their continual suggestions on this research.

Finally, I must express my profound gratitude to my family and friends for the support and continuous encouragement throughout my study, research process and report writing. I couldn’t have accomplished this without them.
# Table of Contents

ACKNOWLEDGMENTS ........................................................................................................... ii

ABSTRACT ..................................................................................................................................... vi

LIST OF TABLES ....................................................................................................................... vii

LIST OF FIGURES ...................................................................................................................... viii

INTRODUCTION ...................................................................................................................... 1

1.1 History and Background ......................................................................................................... 1

1.2 Problem Statement ............................................................................................................... 7

1.3 Objective .................................................................................................................................. 8

1.4 Contributions ........................................................................................................................ 10

1.5 Organization of the document ............................................................................................... 11

DATA CENTER MANAGEMENT TERMINOLOGIES ....................................................... 12

2.1 Server Monitoring ................................................................................................................. 12

2.2 In-band and Out-of-band management ............................................................................... 16

2.3 Redfish Key Technologies .................................................................................................. 18

2.3.1 Baseboard Management Controller (BMC) .................................................................. 18

2.3.2 REST API ......................................................................................................................... 20

2.3.3 HTTPS .................................................................................................................................. 21

2.3.4 JSON ................................................................................................................................... 23

2.3.5 OData ................................................................................................................................. 25

2.4 Redfish Architecture ............................................................................................................ 27

2.4.1 Redfish Tree Structure ..................................................................................................... 27

iii
2.4.2 Redfish Operations ................................................................. 29
2.4.3 Redfish Operational Model .................................................. 30

2.5 Importance of Validation to a Streamlined User’s Experience ........ 30

CONFORMANCE TESTING .............................................................. 34

3.1 What is conformance testing..................................................... 34
3.2 Conformance testing process ................................................... 35

PLATFORM MANAGEMENT STANDARDS ........................................ 39

4.1 Intelligent Platform Management Interface ................................. 40
4.2 Alert Standard Format (ASF) .................................................... 43
4.3 Active Management Technology ............................................... 47
4.4 Desktop and Mobile Architecture for System Hardware (DASH) ..... 48
4.5 Systems Management Architecture for Server Hardware (SMASH) ... 49
4.6 Summary ................................................................................. 51

IMPLEMENTATION AND RESULTS .................................................. 57

5.1 Assertions and its Types.......................................................... 57
5.2 High Level Design .................................................................. 58
5.3 Evaluation .............................................................................. 62
5.4 Results .................................................................................. 64
  5.4.1 Validation of Redfish Result: .............................................. 64
  5.4.2 Comparison Result: ........................................................... 66

RELATED WORK ........................................................................... 68

6.1 IPMI Validating system ........................................................... 68
6.2 Data Center Manageability Interface (DCMI) test system .......... 70
6.3 DASH and SMASH Conformance Testing........................................... 71

CONCLUSION AND FUTURE WORK ...................................................... 73

BIBLIOGRAPHY .................................................................................... 75
ABSTRACT

Data centers are at the core of modern software technology and play a crucial role in defining capabilities and success of an organization. The massive growth in size and scale of data and computing leads to an enormous growth in the size and complexity of clusters and data centers. Therefore, traditional management methods and standards like Intelligent Platform Management Interface (IPMI) are not sufficient to manage these modern scalable data centers. Redfish is a new standard for managing hardware in modern data centers and is anticipated to meet the expectations of end users to provide modern, simple and secure management of scalable platform hardware. It is essential to validate Redfish’s capability regarding the performance, scalability and security aspects as mentioned in the Redfish Specification. To validate Redfish services, we have designed a Redfish Conformance Test Tool (RCTT) in Python which performs compliance testing and confirms if the output matches as per the Redfish specification. The specification is divided into assertions and they are grouped into four divisions - Protocol Details, Data Model and Schema, Service Details and Security. RCTT covers all four divisions and helps users to re-instate their trust on the specification. Additionally, comparing Redfish with other standards will also help us in understanding how well these standards have evolved.
LIST OF TABLES

1. Hourly downtime cost per industry
2. Standards comparison results
3. REST vs SOAP comparison
4. TCP vs UDP Comparison
LIST OF FIGURES

1. Data Center Technology timeline .............................................................. 2
2. Evolution of data center management standards....................................... 5
3. Thesis Research and Implementation plan............................................... 9
4. Time taken to locate server outage ......................................................... 14
5. In-band and out-of-band management .................................................... 17
6. BMC in the heart of motherboard .......................................................... 19
7. Redfish Resource Map ........................................................................... 28
8. Redfish Operational Model .................................................................... 30
9. Conformance Test Processes - Generic .................................................. 35
10. Conformance Test Processes – Redfish .................................................. 37
11. Standards Benefits .................................................................................. 40
12. IPMI 2.0 Architecture ............................................................................ 41
13. Assertion categories ................................................................................ 57
14. RCTT Dependencies ............................................................................... 58
15. RCTT User Interface I ........................................................................... 58
16. RCTT User Interface II ........................................................................... 59
17. RCTT Flow Diagram ............................................................................... 60
18. Redfish Service Conformance Test Suite .............................................. 61
19. Assertion – Redfish Specification ......................................................... 62
20. Assertion Interpretation - Algorithm ...................................................... 62
21. Evaluation Flow Diagram ....................................................................... 63
22. Results ..................................................................................................... 65
23. RESTful interface................................................................. 66
24. WS-* based Interface ......................................................... 66
CHAPTER I
INTRODUCTION

1.1 History and Background

A Data Center centralizes an organization’s IT operations and equipment and is a facility to store, manage, and disseminate network’s data. The principle goal of any data center is to provide flexibility and scalability. This involves many factors such as site location, building location, floor layout, electrical system design, mechanical design and modularity. The need for data centers was felt in 1990s when individual systems could not perform tasks on large amounts of data (in Gigabytes or Terabytes), let alone its storage. Since then, data centers have come a long way and have greatly improved the usability of data as a whole.

Figure 1 below shows a timeline of some significant milestones, which led to the data center technology of today. The first digital computer - Electronic Numerical Integrator and Computer (ENIAC) - was built in 1946 for the United States Army to store artillery firing codes. This marked the beginning of the digital computer. In 1954, the first transistorized computer was developed. This provided a significant jump in terms of technology from the vacuum tubes used at that time [1] to more efficient transistor technology. Intel followed this by the introduction of the first general-purpose programmable processor, the 4004. This invention provided engineers with a way to use one device to run multiple piece of software.
ARCnet was introduced as the first LAN in the 1970s and was put into service at the Chase Manhattan Bank. Also, SunGard then developed the concept of commercial disaster recovery. Both inventions are central to the success of the modern data centers. Personal computers were introduced in the 1980s, which lead to a boom in the microcomputer industry. Network file protocol was introduced by Sun Microsystems that allowed the user to access the file stored in a network. 1970s and 1980s laid the foundation of datacenter networking.
Early 1990s saw microcomputers being used as "Servers" and the mainframe rooms became known as the "data centers". The ".com" surge in the mid-1990s let companies develop faster internet connectivity and non-stop operations. This resulted in multiple companies investing in hundreds and thousands of servers and developing their own server rooms. The datacenter as a service model started gaining popularity at this time. During the late 1990s, the concept of virtual PC was developed, and VMware started selling VMware workstations. At the same time, salesforce.com established the concept of Software as a Service (SAAS).

In 2002, Amazon Web Services (AWS) began development of cloud-based services, and in 2006 AWS started offering infrastructure services to companies in the form of web services, now known as cloud computing. Sun Microsystems introduced modular data center in 2007 that transformed the economics of corporate computing. With companies, such as Amazon, Google, Facebook and others investing huge sums of money to develop data centers to support their business, it was evident that datacenters had become a prime necessity for the modern technology development. The number of data centers sprawled across the globe. In 2015, over 500,000 data centers managed the 1.2 trillion GB of data created in the world. This exponential growth in the number and complexity of data centers also directed a similar growth in the systems that manage data centers and made them more critical.

The old data center management methods hit their plateau with the volume and complexity of data centers and the network administrators needed a more intelligent and automated infrastructure management system. The current platform management
systems, which include IPMI and its altered versions are limited by scalability, simplicity, performance, and most notably security. This requires development of new management standards to manage hardware in modern data centers.

**Background**

With the evolution of data centers in the early 1990s, leading companies—Compaq, Dell, Intel and HP also developed an organization DMTF, Distributed Management Task Force (formerly called Desktop Management Task Force), to remedy the data center management problem. Initially, the organization focused to develop software based specifications for systems management. DMTF initially developed Desktop Management Interface (DMI), which provided a large set of pre-defined object classes for a system, that included hard disk, Central Processing Unit (CPU), memory and system chassis [2]. It also included common objects for system events such as disk failure and chassis intrusion and allowed management of system components from a central location [3]. The next specification was Common Information Model (CIM), which added object-oriented features, and hence allowed a greater reuse of code for common objects. However, these software-based solutions could only be used when Operating System is on; if the system crashes, administrators cannot identify the root cause of the failure. Figure 2 below gives an overview of the evolution of standards.

With the advancement of systems, the requirement for robust remote systems management capability, became essential. Different organizations started developing their propriety solutions; the solutions moved from software level solutions to add-ins
and then to embedded controller based solutions. The solutions also added features that would enable cards to operate even if the server lost power or if OS crashed and even allowed the server to be managed in a pre-OS state. These solutions were complimentary to the software solutions. However, the tools and agents had to be rewritten with every new release.

To remedy this problem, Intel, Dell, HP and NEC created the Intelligent Platform Management Interface (IPMI) in 1998 as a specification to provide systems management capability in hardware. IPMI provides a common message-based interface to access all

![Figure 2. Evolution of data center management standards](image-url)
the controllable features in a system. It includes a rich set of pre-defined commands and interfaces that are organized by type of operations and provides methods to access system event log, power control and hardware watchdog. Several releases and enhancements have been made to IPMI since its original 0.9 specification to the latest release 2.0.

IPMI is one of the widely-adopted standards by the computing industry for the server management and server health monitoring. Other platform management implementations such as Data Center Manageability Interface (DCMI) focus on server management in High Density Data Centers and are modified versions of Intelligent Platform Management Interface (IPMI). Such standards enable customers to save time, maximize IT resources and manage multi-vendor environments.

However, as discussed before, Data Centers we see today are more complex, more interdependent and more critical than ever before. Addressing massive-scale data and computing is a big challenge nowadays. The massive growth in size and scale of data and computing has led to an enormous growth in the size and complexity of clusters and data centers. This has led to new requirements in platform management systems in terms of performance, scalability, security, simplicity and interoperability, among others. IPMI, the widely-used standard for out-of-band management, is limited across multiple measures and this requires a new standard that can conform to the new requirements.
1.2 Problem Statement

With an increase in the usage of data computing to make business decisions, every company is investing in its computing infrastructure, making it larger and more complex. In addition, the success of the business is becoming more dependent on the availability and performance of this infrastructure. However, the IT budgets do not increase at the same pace as that of the infrastructure requirement. This requires IT organizations to devise new ways to effectively as well as efficiently manage their computing environments.

IPMI is used as the standard management platform for most of the computing infrastructures. However, IPMI which was introduced in 1998 is limited in multiple traits, required by the modern computing infrastructure. Firstly, IPMI is out-of-date and doesn’t include complexity to define modern architectures such as multi-node servers. Secondly, IPMI doesn’t provide the required performance levels in terms of parallel communication protocols or control beyond servers, which are demanded with the increasing complexity of the infrastructure. Thirdly, IPMI is unable to scale at the same pace and meet requirements of modern scale out servers. Fourthly, there are interoperability issues with IPMI as many vendors have designed their own version of IPMI. E.g.; HP has ILO, Dell offers DRAC, IBM offers IMM2. This confines data center managers to limit their operations to one type of server. Finally, IPMI also lacks modern security best practices, which is required for running commodity servers on a cloud scale. All the above-mentioned limitations in IPMI led the industry to consider and develop a new standard - Redfish API, as a significant improvement over IPMI.
Redfish claims to be more secure and take advantage of today's more memory-rich baseboard management controllers, to provide a richer set of information which is easily interpreted by scripts and administrator programs, to be human readable allowing non-experts to understand the API, to include the scalability requirements of the modern infrastructure, and to have better performance and control than IPMI. All these claims haven’t been validated yet and the specification of the Redfish needs to be tested thoroughly and the results need to be cross verified. At the same time, the new standard needs to be compared with the existing standard and required to check against all the aspects which include security, flexibility and scalability.

1.3 Objective

The objective of this thesis is to validate Redfish claims on how far the new standard has proven itself to be a better choice for the industry and developing a conformance test tool for the Redfish. The conformance test tool will validate whether Redfish works as per the specification or not. In the end, the study will also develop a comparative analysis on various standards including Redfish and show the incremental advantages of the technologies used in Redfish.

To achieve the objective, initial focus was centered on developing an understanding of platform management standards. IPMI and Redfish standards were the first two standards studied. This was followed by understanding different terminologies in the standards such as JSON, OData, HTTPS, RESTful APIs etc. to have a thorough understanding of the concepts. To further enhance the learning on standards, functioning of IPMI and Redfish was studied by learning about BMC controller, different protocols
used in the communication between server and controller, channels used for in-band and out-of-band management as well as difference in operations of various standards. After this, Redfish specification was analyzed to review all the claims and a list of assertions (an excel file) were derived based on the claims. This was followed by testing Redfish mock up server with an initial conformance test tool. RCTT was developed then with more number of assertions and by keeping up-to-date with the current version of Redfish specification (v-1.0.5). Once RCTT was developed, the test cases and claims were tested initially on the mock-up. All the assertions were then tested on Redfish-enabled real server provided by Dell and the results were noted for each assertion. Figure 3 gives an overview of thesis research and implementation plan executed.

Figure 3. Thesis Research and Implementation plan
1.4 Contributions

This section summarizes the key contributions of the presented work in this thesis. Starting with a thorough understanding of the history of standards, this thesis helps understand the needs that led to the development of different platform management standards and below is the summary of contributions from this research.

- This thesis provides a comprehensive analysis on the key technologies, which were used to develop Redfish standard and how those technologies supersede the existing methods.
- This research also provides a complete picture of the Redfish architecture, which helps understand the key elements of Redfish and also on how redfish works.
- A conformance test tool is proposed in this thesis to validate Redfish specification. The initial tool was provided by Intel, which was further enhanced in this research with the latest version of Redfish specification as well as with many more number of test cases covering more assertions.
- The testing was performed not only with the mock up data but also on the real-time server provided by Dell. This helped to develop both the methods, which can be used to test standards in two ways.
- The results of this testing would help users reinstate their trust on the specification. In addition, testing during the developmental stages helps in making changes on the go, thence making it viable-both economically and resourcefully.
• Finally, this research not only provides results from the conformance testing for a particular standard, but also covers an extensive study on different standards and the comparisons between them.

1.5 Organization of the document

The organization of the thesis report starts off with section-2, providing an insight into data center management terminologies followed by section-3, which provides an overview of the conformation testing methodology. This is succeeded by section-4 that does extensive study on different management standards and compares them. Section 5, entails the implementation and discussion of results emerging from the conformance test tool and the corresponding comparisons while section-6 speaks about the associated work. Finally, section-7 concludes the research and explores the scope of future work in this domain.
CHAPTER II

DATA CENTER MANAGEMENT TERMINOLOGIES

Before we dwell further into the details on different data management protocols that have been introduced in past few years, it is highly important to know why data center management protocols are important for data centers. A data center consists of thousands of assets, starting from servers, storage and network devices to the equipment that supports infrastructure such as Computer Room Air Conditioners, PDUs and UPSs. The data centers have experienced massive growth due to an influx of data analytic projects. This growth immersed with increased data has resulted in data centers facing new challenges.

2.1 Server Monitoring

One of the attributes of the data center management protocols is to help monitor server health [6]. Proper functioning of server is critical for a data center to support the network. Most of the businesses which work with servers rely considerably on the stability of the I/O network and their infrastructure. The cost of downtime is one of the major concerns. 59% of Fortune 500 companies experience a minimum of 1.6 hours of downtime/week. This translates to $46M/year in just unutilized labor for fortune 500 companies [7]. This doesn’t include the loss due to lost sales, damaged reputation. Another research firm calculated the average hourly cost per industry as shown in the table 1 [8]. Downtime effects the productivity badly. This in turn has a negative impact on the service of the organization and causes great dissatisfaction to the customers,
leading to bad reputation. Because the market is very competitive, a small dent in the reputation can cause a huge loss to the business. It is necessary to make sure that this doesn’t happen. And the simple solution to avoid this is to incorporate server monitoring.

Table 1. Hourly downtime cost per industry

<table>
<thead>
<tr>
<th>Industry</th>
<th>Hourly cost (M$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brokerage Service</td>
<td>6.5</td>
</tr>
<tr>
<td>Energy</td>
<td>2.8</td>
</tr>
<tr>
<td>Telecom</td>
<td>2.0</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>1.6</td>
</tr>
<tr>
<td>Retail</td>
<td>1.1</td>
</tr>
<tr>
<td>Healthcare</td>
<td>0.6</td>
</tr>
<tr>
<td>Media</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Server monitoring is a preventative measure. Monitoring the servers will help us detect if there are any issues with the server, even before they cause any major issues which can affect the productivity or hinder growth of the business. Server monitoring is the process of keeping track of the servers in a continual manner and to scan them on a designated network. Monitoring is also done by performing scans on the network to find if there are any failures or if there are any irregularities with the infrastructure. This is performed by making use of a server monitoring software.

If a server/network crashes, it causes a great increase in cost and wastes a lot of time. During such a downtime (which effects productivity and invites bad reputation), it will be difficult to allocate more resources (cost and staff) to get things back up and running again. By incorporating server monitoring, we can pick up the issues with the
server in its early stages, before it causes any major damage to the productivity. Server monitoring is a key factor in making sure that the service is available as expected.

A Digital Realty Trust survey [9] provided a data which mentioned that only 26% of data center managers were able to locate a server, which had gone down, within minutes. It would take less than four hours for 58% of the data center managers to locate the server while 20% of the managers would require more than a day to locate the server. Due to this inability, the mean time to repair (MTTR) for the data center equipment increases, which further decreases the overall availability of the server. The Figure 4 displays the results of the survey on approximate time it takes data center managers to locate if a server goes down.

![How long could it take to find a server that has gone down?](image)

**Figure 4. Time taken to locate server outage**

Without a data center management protocol, it will be extremely difficult to manage the data centers manually. The manager or administrator would have to be present in the data center physically at all the times to take immediate action in case of
any faults. In addition, manager would be required to switch the server on physically if the server fails. Manager will also have to check many other attributes such as temperature, power of each server, fan speeds, cooling system among others to avoid any failures.

Different data center management protocols provide important information that help managers or administrators to manage data centers in a better way and even remotely. The management protocols provide real-time information and even send alarms if something goes wrong for e.g. in case of high temperature, it notifies the administrator and the administrator can remotely power off the server. Some of the protocols even help to take out the information from the server while in off state. So even when the server is down, and there is no way of communicating with it, some of the protocols will help to connect to the server and change the state to “ON” state while providing required information.

To monitor the power usage of IT equipment, there are two defined primary methods. The first method uses rack power distribution units (PDU) with outlet-level metering capability and SNMP being the communication protocol employed. A prominent drawback to this method would be that the outlet level monitoring-inclusive rack PDUs tend to be more expensive than their standalone power-draw monitoring counterparts. Another demerit associated with this method is that an efficient management system needs to be used to monitor the power-draw associated with the dual corded devices that distribute their power between two dissimilar rack PDUs.
Polling power readings directly from the device is another method employed for monitoring IT equipment. Thence, the communication protocol commonly employed is IPMI (Intelligent Platform Management Interface). However, there are issues that plague this method - Issues ranging from the inability of all IT equipment at supporting IPMI and equipment manufacturers customizing the IPMI via extensions catering to their tastes/requirements, to, the need for an efficient management solution at handling these varying multitudes of protocols, which inherently do the same thing, but are unique in terms of their predominant shackles to the OEM-controlled specifications. Examples of such protocols would be IBM Remote Supervisor Adapter (RSA), Dell Remote Access Controller (DRAC), Sun Integrated Lights Out Manager (ILOM), and HP Integrated Lights Out (iLO). Protocols such as SNMP, WMI, and WBEM are adept at providing additional information as deemed necessary. Eventually, a device supporting one of these protocols might not provide power usage information which is indispensable to a data center.

This necessitates a well-defined standard governing the hassle-free transference from one vendor specific platform to another. Redfish provides an open industry standard specification catering to the needs of end users for a simplistic yet modern and secure management of scalable platform hardware [10].

2.2 In-band and Out-of-band management

There are two main ways in which devices are monitor, managed and maintained in the network - In-band management and Out-of-band management. In-band management [11] refers to monitoring devices on the network by using in-band channels
and software that is installed on the system. In-band management hence requires the operating system to be booted for system to be monitored. In-band management hence doesn’t allow to reinstall the operating system remotely, is unable to fix the problems that prevent booting of the system, or doesn’t provide control mechanisms in situation when server is powered off.

Out of band management on the other hand has a dedicated channel that allows management or monitoring of system/server irrespective of whether system is powered on or if an operating system is installed [12]. Because of this, out of band management is also called lights out management. The remote management allow multiple functions such as shutdown, rebooting, power on, troubleshooting, sensor monitoring, and operating system reinstallation. Integrated solutions which provide in band management as well as out of band management are critical for the modern data centers to allow continuous monitoring of the servers and systems. Figure 5 below displays the overview of the in-band and out-of-band management.

![Figure 5. In-band and out-of-band management](image-url)
2.3 Redfish Key Technologies

DMTF released Redfish, which is a modern RESTful interface that manages compute, storage, network platforms and services. Claimed as an open industry standard specification and schema, Redfish is expected to meet the expectations of end users for modern, simple, and secure management of scalable platform hardware. It also brings enhanced functionality, security and reliability to Baseboard Management Controller Management (BMC). Redfish uses REST API architecture over HTTPS in JSON format based on OData v4 and can be accessed by clients, scripts, and browser based GUIs. These technologies are used because of their widespread use, and the administrators too find it easy to use.

2.3.1 Baseboard Management Controller (BMC)

BMC is a specialized processor that helps in monitoring the physical state of a computer, network server or other hardware device by using sensors and communicating with the system administrator through an independent connection. The BMC is usually contained in the motherboard or main circuit board of the device to be monitored as shown in Fig 6 [13] below.
The internal physical variables such as temperature, humidity, fan speeds, power-supply voltage, communications parameters and operating system (OS) functions are measured by sensors of BMC. The administrator is notified if any of the mentioned variables happens to stray outside certain specified limits. The device being monitored, then can be either power cycled or rebooted as necessary. Thus, a single administrator can remotely manage large numbers of servers and other devices simultaneously. This saves an overall operating cost of the network and helps in ensuring its reliability. The functionalities of BMC include: blade inventory, monitor sensors (blade voltages, temp, fans etc.), log events for failure analysis, power management and provide remote management capabilities (IPMI, KVM, vMedia (using USB) and SOL (Serial over LAN))
2.3.2 **REST API**

Representational State Transfer (REST) architectural style has been a predominant choice for distributed resources [14]. The term was introduced by Roy Fielding as part of his PhD thesis in 2000 [15]. REST is a client-server architecture, where REST defines the set of rules using which client and server communicate with each other through requests and responses. The REST is based on the use of nouns and verbs and doesn’t require envelope and header message formats. Each data is handled as a resource and each resource is handled an atomic data unit [16]. REST is designed on the following principles – addressability, uniform interface and statelessness [17].

The request and responses are built around the representation transfer of resources. Something that is identified with by using Uniform Resource Identification (URI) is a resource, which are hierarchically structured such that each resource is linked to at least one URI [18]. A document which captures the expected state of the resource is representation of a resource. REST improves addressability by modelling the datasets to operate on as resources. REST uses only a uniform interface, which is fixed set of HTTP commands to access the resources, making it simple and light weight. In addition, with frequent changes and updates in the services, the REST API changes accordingly since it is designed to facilitate hypertext-driven navigation.

REST is stateless; each method executes on its own, which is a constraint to the client-server interaction, where each request from the client to the server must contain all the necessary information to understand the request. This constraint promotes the properties of visibility, reliability and scalability. Visibility is improved as the full nature
of the request is determined without requiring the monitoring system to look beyond a single request. The task of recovering from the partial failures is eased that leads to the improvement of reliability. Due to the nature of REST, server does not have to manage resource usage across requests, which means there is no need to store state between requests which in turn allows to quickly free the resources, and hereby improving scalability. However, the stateless constraint may have some disadvantage where it may decrease the network performance when redundant data is sent in a series of requests. Here comes the advantage of adding a cache constraint to it. This can improve efficiency, scalability and performance by reducing the average latency of a series of interactions. However, having a cache could lead to a response with stale data rather than data received from real server. This decreases reliability. Some of the benefits of using RESTful API is that creating a standard, a client and a server are easy. In addition, it is based on TLS (HTTPS) making it more secure and using latest upgrades.

2.3.3 HTTPS

Hyper Text Transfer Protocol Secure (HTTPS) encrypts the data which is sent between the browser and the website, making it more secure than HTTP [19]. HTTPS is mainly used in the situation which require confidential information to be transmitted, for e.g. in online banking and online shopping systems. HTTP uses ‘plain text’ for sending all communications and it has high risk as it can be read by hacker in case the hacker manages to break into the connection. HTTPS is basically the standard HTTP protocol, with an added security layer for encryption.
Usually, one of the two secure protocols are used to encrypt the communication, which are: SSL (Secure Sockets Layer) or TLS (Transport Layer Security), where TLS is basically the successor of SSL [20]. Both the protocols use asymmetric Public Key Infrastructure (PKI) system where two ‘keys’ - a ‘public’ key and a ‘private’ key are used to encrypt the communications. The private key is kept strictly protected and is accessible only by the owner of the private key whereas public key is distributed to everybody intending to decrypt the information encrypted by the private key.

After a TCP connection is established, client initiates an SSL handshake [21]. Client sends number of specifications which include SSL/TLS version, ciphersuites to use as well as compression methods to use. The server then checks the highest version of SSL/TLS supported by both and after finalizing on other factors, sends its certificate to the client [22]. This certificate needs to be trusted by either the client or a third party which client trusts. Once the certificate is verified, a key is exchanged depending on the ciphersuite (public key). Both the client and the server can now use the key for encryption. The client then sends an authenticated message to the server saying that all communication will now be encrypted. The server receives this message, verifies that the MAC is correct and if the message can be decrypted. It returns then a message to the client which the client verifies and this marks completion of the handshake. The two hosts can now communicate securely on HTTPS. An ‘alert’ is used to terminate a connection. This would help both the hosts to know that the connection was interrupted if the attacker tries to complete the connection by terminating the TCP connection.
Redfish uses HTTPS for all its transmissions – request and response, making it more secure owing to the encryption property of HTTPS.

### 2.3.4 JSON

From being markup and display-oriented, data interchange formats have evolved according to the requirements of the information system and have accommodated encoding of meta-data, which describes the structural attributes of the information. However, standard data interchange formats were developed to support data interchange of java applications [23]. JSON and XML are two popular data interchange formats used for encoding data to transfer between a server and an Ajax application, support communication between two servers connected through web services, and in many such scenarios.

The eXtensible Markup Language (XML) was derived from the Standard Generalized Markup language (SGML), the international standard meta-language for text mark-up systems [24] as a result of the SGML complexity. XML was designed not only to improve the web functionality by providing adaptable information identification, but also to consider ease-of-use and human-legibility [25]. However, with the current scale of web systems, construction of modern information system was required which was in line with the current needs [26] of user-experience.

JavaScript Object Notification (JSON) is a light-weight data-interchange format, which is “self-describing”, human readable and easy for machines to parse and use. JSON is based on a subset of JavaScript (Standard ECMA-262 3rd Edition, General purpose programming language - December 1999), and uses a text format which is
independent of languages. However, JSON represents the same data structures as those used by other programming languages such as C, Java, Python, C++ and many others which made JSON very attractive to the programmers. JSON is built on two forms – objects and arrays [27]. An object is a disordered set of name/value pairs where as an array is an ordered list of values. Arrays in JSON start with “[“ and end with “]”, while the object starts with “{“ and ends with “}”. Values in arrays are separated by “,”, while in object each name is followed by “:”, and each name/value pair is followed by “,”.

Redfish uses JSON format which provides it multiple advantages over XML. JSON is estimated to parse almost a hundred times faster than XML [28]. Moreover, JSON uses lesser resources and performs faster as compared to XML [29]. JSON also has a higher data transfer efficiency as compared to that of XML, and the performance gap increases with the size of the data [26]. Using XML for communication between a server and a client needs files in both server and client to be parsed complexly using SAX, DOM or other technologies, which is catastrophic when using large amount of data and affects user experience to a great extent [26]. In addition, client efficiency is much higher in JSON than XML. The deserialization time in JSON has no significant increase with the increase in the amount of data, but deserialization time increases significantly with Data in case of XML.

JavaScript Object Notation (JSON) is a standard format that transmits the data objects in a human-readable text in the form of attribute-value pairs. JSON is commonly used for asynchronous browser/server communication and has widely replaced XML.
2.3.5 OData

The Open Data Protocol (OData) is considered to be the best practice for building and consuming RESTful APIs [30]. OData is a data access protocol built on widely used technologies such as AtomPub. Rather than using SOAP, OData is based on REST and provides support for CRUD (Create, Read, Update and Delete) operations to create and consume data APIs. Because of this, OData is also called “ODBC for the Web” [31]. OData consists of the following four main parts [32]:

- **OData Protocol**, which allows the client to send requests and receive responses to and from an OData Service;
- **OData Data Model**, which defines the standard way to describe and organize data;
- **OData Service**, which exposes a callable endpoint allowing access to the data or calling functions;
- **OData Client Libraries**, which simplifies the process to create software that accesses data using the OData protocol.

**OData Protocol** – OData is an extension to the Atom Publishing and Atom Syndication standards [33] developed by Microsoft. Using these standards, which are based on XML and HTTP, allows OData to be extensible. OData is able to supplement the data types with additional, domain specific information. Atom consists of a pair of standards – Atom Syndication Format -XML based and Atom Publishing Format – HTTP based. The Atom fails to be fully RESTful because its messages are not self-describing. OData extends Atom by providing metadata description of the message.
OData Data Model – OData helps provide a common view for distinct data sources to clients and uses Entity Data Model (EDM) to fulfil this requirement. The main concepts in EDM are entities, entity sets, relationships, action and functions [34]. Each entity is part of an entity set and each set in-turn is part of an entity container. EDM also defines a variety of data types such as Boolean, String, Binary, Int16, Int32 and DateTime to describe data in properties. Declared properties are the ones, which are declared as part of the structured type’s definition whereas navigation properties are relationships from one entity to another.

OData Service – The service is defined with a generic data model. In order to allow the generic clients to interact with the service in a well-defined manner, the service presents its data model in a machine-readable form [34]. An OData service provides two resources, which describe its data model – a service document and a metadata document. The service document describes the functions, singletons and entity sets which can be retrieved and is used by client to navigate model in a hypermedia-driven way. The metadata document lists the sets, types, functions and types which are understood by the OData service and is used by client to understand method to query and interact with entities.

OData Client Libraries – Client libraries are provided by Microsoft and other providers [35]. The current libraries include .NET, Java, Javascript, C++ among many others [30]. Microsoft, for example provides toll support to create OData clients in visual studio. For example, when a reference to OData service is added, Visual Studio is able to read that service's CSDL definition with the metadata option. Visual studio
will then generate C# or VB classes to work with the data provided by the service, once it has the information.

2.4 Redfish Architecture

Redfish has RESTful API employed in it, that provides a web-based access using URIs, which can be typed in the browser. The Redfish API starts with the Redfish root at “/redfish/”. For eg. URI “https://<ip_address>/redfish/v1/” accesses the root for the server that has IP Address as “ip_address”. The “v1” denotes the version of the API [36]. Every resource is associated with a unique identifier. Redfish URIs consist of three parts as described in RFC3986: “Part one defines the scheme and authority of the URI, part two specifies the root service and version and part three defines a unique resource identifier.” For eg. in the URI: https://mgmt.vendor.com/redfish/v1/Managers/MngrID

a. https://mgmr.vendor.com is the scheme and authority

b. /redfish/v1 is the root and version

c. /Managers/MngrID is the resource identifier

2.4.1 Redfish Tree Structure

Starting from the top-level root, other collections are branched out, which in turn include multiple sub-items and forms a tree-like structure as shown in Figure 7 [37].
As shown in the figure above there are three main collections that includes: Systems, Chassis and Managers.

*Systems* is a collection of properties that are expected from Operating System console. It includes items that are needed to run the computer. It can be considered roughly as a **logical view** of a computer system as seen from the OS.

*Chassis* presents roughly a **physical view** of a computer system as seen by human, i.e. items that are needed to identify, install or service the computer.

*Managers* is collection of properties that are needed to perform administrative functions for e.g. The systems management subsystem (**BMC functionality**).
2.4.2 Redfish Operations

Redfish interface supports different HTTP methods. Operations that are required to be supported includes: GET, POST, PATCH, PUT, DELETE. Redfish resources are accessed using URI (Uniform Resource Identifier), further explained in Chapter IV.

GET: The GET method returns the data along with the header information, wherein the header uses an ETag or entity tag - a part of HTTP, to identify the version of the data. It assists in maintaining control concurrency in addition to preventing updated resources from overwriting each other [38]. If the resource URI successfully returns the data, the server sends a 200 (Success) response.

POST: The HTTP POST method in the Redfish are used in two ways: Object creation, Object Action. Object Creation is a request method, that is used to create a resource. The request body contains the information about the resource to be created. The response that is returned on a successful creation is 201 (Created) and may return a response body containing the information about the new resource created.

Object Action method is used to initiate any operations on the object (such as Actions).

PATCH: This method is used to perform updates on the already existing resources. The request body in this case contains all the information about the changes to be done on the resources. Once the property of the resource is changed, a status code of 200 is returned.

PUT: This method completely replaces a resource. If the request body does not specify the property of the resource, it is reset to the default value.
DELETE: This method remove a resource. Redfish services may return a response body representing the information about the deleted resource. If a resource successful gets deleted, a response code 200 is returned.

2.4.3 Redfish Operational Model

All the Redfish operations are initiated via Client using HTTPS. The Client can interpret the JSON responses received from the server. The responses contain all the information that was requested along with the status code that indicates the success or the failure of the requested operation. Figure 8 [13] below describes the operational model in brief.

![Redfish Operational Model](Image)

Figure 8. Redfish Operational Model

2.5 Importance of Validation to a Streamlined User’s Experience

Validation of any tool or model develops the confidence in the users to use the tool. Verification and validation must be performed for each feature that is specified in
the specification in order to build the credibility of the product (in this case Redfish) among the customer base. To perform verification and validation, a simulation model or a validation tool is usually built. Each and every assertion mentioned in the specification must be considered while performing validation. If a specification of any standard comprises of a set of assertions, it is important that the validity of the specification is determined with respect to each assertion. All the stakeholders in the Redfish ecosystem (like the different OEMs) make decisions using the information obtained from the results of verification and validation of these assertions.

At the end of any software development, the software features are first verified and then they are validated. In this case, each assertion is understood (to be tested), the test functions for each assertion is written and validated. Validation is usually performed during the testing phase. Validating the software helps us in answering the question: ‘Am I building the right product?’ It also helps us in understanding if the data we are using is the right data to satisfy the requirements.

According to [39] there are three basic decision-making approaches to validate. The frequently used approach is, the development team themselves conducting the validation process of the specifications. Here, the decision will be made by using the results of the tests and evaluation procedures conducted as part of the development process. In general, this approach is considered. However, the next two mentioned approaches are used based on applicability.

Another approach is to have the user(s) of the software decide the validity of the specification. In this kind of approach, clearly the users’ involvement in the validation
will be primary. When the development team releases the specification, the users check if the specification is in compliance with the requirements. In this approach, the users decide the validity of the specification. When the development team is not so large, this approach is usually implemented. Also, this approach helps in credibility of the specification.

The third approach is called "independent verification and validation" (IV&V). This approach, a third party is used to decide whether the specification is valid or not. This third party is referred to as ‘the IV&V team'. This team is independent of both the specification developers and users/sponsors to the specification. This kind of approach is generally used when the specification is large and the team building it is considerably large. It is essential that the IV&V team has a thorough understanding of the specifications and the overall system. There are two common ways this third party does verification and validation: (a) IV&V is conducted concurrently, along with the development of specification and (b) IV&V is conducted after the specification is completely built.

In the concurrent way, the team building the specification gives the details about the requirements to the third party team as they implement. The IV&V team evaluates these results in concurrent and it also provides feedback to the dev team with information like: whether the specification is valid and when not, what the issues are occurring. When this approach is used for validation, the development of a specification should not progress to the next stage, until all the assertions are satisfied for the current
stage. In the paper, the author’s opinion that this concurrent approach is the better of the two mentioned ways and our approach falls into this category.

Validation is usually done by the testers during the testing phase of the software development process. Validation helps in catching the defects as failed features or bugs. Sometimes, due to improper communications between requirement specification and development, some specifications are misunderstood. In the validation phase, by comparing this specification as ‘actual and expected’ outputs for a certain feature, we can identify the difference and report the functionality of the feature to fix it. By performing validation, it helps us in building the right product, per the customer’s requirement. This helps them in accomplishing their tasks as they desire.

Hence, validation helps in confirming that the exact functionality of the features has been implemented in the software product. It also helps the stakeholders to understand the design and product in much better way, without any discrepancies in their perspectives. It helps in making the product as close as possible to meet the user’s specifications and in turn making it user friendly.
CHAPTER III

CONFORMANCE TESTING

3.1 What is conformance testing

“Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test” [40]. Testing provides valuable information to allow the stakeholder to understand and appreciate the risks of software implementation. However, in software testing, there is a difference between functional testing and structural testing.

Structural testing, as the name implies considers the internal structure of the program. In structural testing, test cases are created based on the program structure. The program code is tested rigorously by executing each statement of the code at least once and considering all the possibilities across loops, decisions, branches etc. Structural testing is also called white-box testing. Functional testing on the other hand considers the program as the black box and considers the functionality of the program to generate the test cases. The idea is to make sure whether the product or service has been built according to the initial specification. As evident, here one of the most important requirement is to have a clear and precise specification. The internal structure of the program is hidden and this type of testing is also called black-box testing.

Conformance testing is the type of functional testing where implementation of the protocol entities is tested against the specification or standard [41]. The goal is to ensure that the functioning of the implementation adheres to the specification, and...
improve the probability of protocol implementation success [42]. As discussed earlier for functional testing, conformance testing would require that the specification is correct and valid. Protocol validation is the process of checking the correctness of the specification. For conformance testing, the assumption is that the specification is validated. To ensure standardization and avoid re-work by multiple test centers [43], International Organization for Standardization (ISO) in collaboration with CCITT (now ITU-T), developed a standard – ISO IS-9646 for conformance testing of Open Systems.

3.2 Conformance testing process

Conformation testing is differentiated primarily by three phases [44]. Figure 9 below details the conformance testing process with the three phases highlighted in dark grey. The first phase includes the development of the test suite for a particular protocol. It is referred to as test generation or test derivation.

During test generation phase, protocol specification is used to systematically derive multiple test cases. Here, the idea is to develop an abstract test suite, i.e. develop
tests which are independent of any implementation. However, the test suite is specified in a well-defined test notification language, suitable for standardization and tests all aspects of the protocol in considerable detail. Also, conformance requirements fall into static conformance requirements and dynamic conformance requirements [45]. As static conformance requirements, can be verified using Protocol Implementation Conformance Statement (PICS), test generation uses dynamic conformance requirements as the starting point. Now a three-step procedure is followed to derive test cases from the dynamic conformance requirements. The first step is to derive the test purpose, which is identifying the exact criterion for success, for each conformance requirement. Following this, a generic test case is developed for each test purpose, where actions to achieve the test purpose are detailed. The last step is to develop an abstract test case, where test method is ascertained considering the restrictions.

In the test implementation phase, the standardized abstract test suite specified in test generation is translated into an executable test suite. An executable test suite can be run on a specific system or a device with a Implementation Under Test (IUT). To start the implementation, test selection is made where the tests are selected from the abstract test suite considering the relevance to IUT and PICS. Here, the information related to the IUT and the environment is also considered and it is made sure that the executable test suite can be implemented correctly.

Test Execution phase involves testing of the executable test suite on a specific IUT to conclude about the conformance test. The first part is the static conformance test, where the static conformance requirements of the standard are compared to the PICS
and checked for internal consistency. The second part is the actual execution of the test cases on a real test environment. For each case, a verdict – pass, fail or inconclusive is assigned. Pass indicates that the goal of the specific test purpose was achieved successfully. Fail indicates that the test purpose is not met and the implementation doesn’t meet the specification. Inconclusive indicates that there was no evidence of the conformance or non-conformance, but the test purpose was not met.

The above-mentioned process is a very generic process to perform conformance testing on the system. The conformance testing in this research was followed as shown in the below figure 10.

**Figure 10. Conformance Test Processes – Redfish**

The protocol specification here is the Redfish specification, which is the basis of the conformance testing. Test cases are generated in our testing which are the assertions, extracted based on the terms of “shall”, “shall not”, “should”, “should not”, “may”, “need not”, “can” and “cannot”. RCTT focusses on “shall” implementation for now and all the assertions are developed in Python language. The test implementation phase is developing the assertions. The test execution phase is running the code and
generating the result in an excel file. In this case the testing is not just performed against the mockup data, but the tool was tested with real server as well. The real server was a redfish-enabled Dell server with the firmware version of “2.40.40.40”. In case of Dell vendor, all the firmware versions that are greater or equal to “2.30.30.30” are redfish-service enabled servers. The results were compared against the Redfish Specification. This conformance testing helps in validating Redfish specification and helps in bringing out conclusion on why Redfish is a better choice for the industry. The more detailed explanation on how the conformance test tool is developed and how it works is explained in the Chapter 5.
CHAPTER IV

PLATFORM MANAGEMENT STANDARDS

The firms hold their most critical and sensitive data on the data centers. These data centers are the dedicated spaces for the firms, and thus safety and accessibility of these data centers play a crucial role. It is necessary that these data centers follow practices which will fetch best results and to ensure the same the data centers follow the best practices, i.e. follow a set of rules (protocols) which follow standards.

The aim of the standards is to help improve performance as well as safety of the data centers. Without standards, we will end up having separate tools for each problem, i.e. each vendor will use a different implementation of a solution to manage the data center. To switch vendors, clients would need to have a thorough understanding of the complete implementation of each vendor, which will also add to the training and switching costs.

If there are standards defined, this reduces the number of implementations as discussed above. The administrator can focus on other important tasks and can spend less time in dealing with the management tools, thus improving administrator efficiency. Following standards reduces training requirements and training curve for each proprietary solution as well as helps to manage data centers efficiently. This also lessens the risk of admin error and downtime.

A standard increases choice and flexibility as it avoids vendor lock-in and improves product interoperability that leads to increased stable environment. It drives towards simplification and lowers training requirement too. With the standards defined,
there is a decrease in the cost of management as it reduces training and eliminates inefficiencies. Figure 11 shows the merits of introducing a management standard [46].

![Diagram](image)

*Figure 11. Standards Benefits*

### 4.1 Intelligent Platform Management Interface

Intelligent Platform Management Interface (IPMI) is a hardware health monitoring and management system that manages the system remotely. It monitors the physical health of servers, such as temperature, voltage, fans, power supplies and chassis. The first IPMI version was announced in 1998. Since then, there has been many versions released with many features added. IPMI runs on the Baseboard Management Controller (BMC) and provides access to the BIOS, disks and other hardware. BMC provides out-of-band (OOB) capabilities for a typical motherboards-system like a PC server. It is linked to the outside world via an Ethernet connection as shown in Figure 12 [47]. A range of statistics can be tracked for e.g. temperature, processor status, cooling fan speeds and power supply status. Operating system (OS) status can be
checked via watchdog timers. To communicate with sensor and control devices, IPMI uses Intelligent Platform Management Bus/Bridge (IPMB) interface, which is also known as System Management Bus (SMBus), Inter-Integrated Circuit, and the SMBus System Interface (SSIF) [47].

There were derived versions of IPMI to overcome some of the limitations. One of which was DCMI (Data Center Manageability Interface). DCMI was designed expressly for the platform manageability. This focuses on meeting the needs of High Density Data Center (HDDC) servers. HDDC customers were looking for some set of common characteristics that included management capabilities to be [48]:

- Concise without unnecessary options or unneeded features
- Interoperable that provides the same capabilities in the same way across multiple platforms from multiple vendors.
- Open that allows specifications to be available to all platform vendors and HDDC for deployment without restrictive licensing.
- Reliable; all the management capabilities must be based on proven technologies.

Figure 12. IPMI 2.0 Architecture
• Simple; the interfaces, data models and the capabilities should be easily understandable to the developer having basic networking and software knowledge.

• Stable; the interfaces and capabilities remain stable and compatible across platform generations. It should minimize the configuration and implementation options.

IPMI 2.0 covered many server segments, but did not provide a set of mandatory requirements that fit the functions identified by HDDC. IPMI 2.0 therefore had many options that were not applicable or valued for HDDC management. Thus, the mandatory requirements of IPMI did not meet the HDDC needs, and the full definition of IPMI 2.0 capabilities exceeded them. DCMI specification explicitly addresses the unique requirements of server platform management and other HDDC where large numbers of servers are deployed. DCMI take advantage of the existing industry infrastructure of IPMI management controllers, firmware and expertise. In most of the cases, to support DCMI, only the management controller’s firmware needs to be changed, server platform hardware support is already present. The capabilities included in DCMI are available to both local and remote management software. They are: Standardized Local and Remote Access Interfaces, Platform Power on/off/reset control, Baseline sensors, Platform identification, Improved sensor access, Authentication, integrity, and confidentiality and Platform Event Logging as mentioned in [48]. DCMI is simple, less expensive, robust and interoperable and is used for in-band and out-band Data Center applications.
Some of the risks that were identified by Dan Farmer [49] with the use of IPMI that is applicable to DCMI as well since DCMI is an extension of IPMI includes gaining the physical-level access to the server by attackers using IPMI. Once accessed, attacker can reboot the system, install a new operating system, bypass any operating system control or compromise data. These are possible due to various reasons:

- The passwords used for the authentication of IPMI are saved in a clear text.
- The knowledge of one IPMI password gives the password for all the computers in the IPMI managed group.
- Once the root access for the system is gained, the complete control over hardware, software, firmware on the system is gained.
- It has been observed that BMCs run excess and older network services that may be vulnerable.
- Remote control access to the system is gained if access to IPMI is gained, which results in access to the BIOS.
- Certain types of traffic to and from the BMC are not encrypted.

4.2 Alert Standard Format (ASF)

ASF is a Distributed Management Task Force (DMTF) standard introduced in 2001. ASF was used for remote monitoring and control of the computer systems in both operating system-present (OS-present) and operating system-absent (OS-absent) environments. DMTF had taken several initiatives towards OS-present environments also considered as in-band management such as Web-Based Enterprise Management (WBEM) and Desktop Management Interface (DMI) standards. ASF addressed the OS-
absent environment and defined the mechanisms that had a remote control and alerting interfaces for computers. ASF was introduced when there were multiple solutions existing in the industry which were causing a loss of interoperability in system alert and multiple corrective-action offerings.

An Alerting System consists of a client and a management console that both monitors and controls the clients. The interfaces are provided by an ASF-aware client to allow interoperability between the client and its management console. These interfaces provide [50]:

- “the alert messages transmitted by the client system”
- “the remote maintenance requests sent to the client system and the associated responses”
- “the data description of the client’s system-specific capabilities and characteristics”
- “the software used to configure or control the client system in an OS-present state”

There are two network protocols that ASF uses [51]: In order to send alerts from a client to the management console, the ASF protocol uses Platform Event Trap (PET) and to have remote control of the system, it uses Remote Management and Control Protocol (RMCP).

The PET protocol, in turn, uses a Simple Network Management Protocol (SNMP) Trap Protocol Data Unit (PDU), which is IETF’s SNMP Trap PDU in order to carry system management information. The alerts that are covered are various low-level
system activities that includes: environmental events such as temperature, voltage and fan problems, system firmware error such as system memory and hard disk problems, system firmware progress events that provides information on the progress of the boot operation, chassis intrusion that alerts when any system has been tampered with/or opened, system heartbeat event that ensures that a system is still present in the managed environment, CPU error and system boot failure events that indicate that the host processor has been removed or is not functioning.

RMCP (Remote Management and Control Protocol) is a UDP (User Datagram Protocol) based protocol, which is used for system control when a managed client is in an OS-absent state. In such environment, RMCP packets are exchanged between a managed client and a management console. The client gets control over functions such as Power-up, Reset and power-down, reboot to multiple paths.

RMCP methods are used by a management console to manage client systems as a part of two-tiered approach. Primarily, the console uses the OS-present methods to power down or reset a managed client while console employs RMCP methods only when OS-present methods fail to be employed by managed client. The reason for using RMCP methods secondarily is the possibility of a data loss on the client system from hardware-based RMCP methods. RMCP Security-Extensions Protocol (RSP) enhances security in RMCP by providing integrity and authentication while sending RMCP messages. RSP header and trailer encapsulates an entire RMCP message [50].

Some of the advantages of ASF technology are as follows: An IT administrator can send a command form the management console to power on a series of desktops. A
software patch can be sent to these systems utilizing deployment software. Once the upgrade is completed, a notification is received and the IT admin then invokes ASF from the console to power down the systems.

Some of the limitations in an ASF-enabled system include: The need for at least one good boot to the system’s OS-present environment, if there is a change to the system’s hardware configuration. For e.g. Adding or removing a card. With the OS-absent environment it is difficult to determine the information that is needed to configure ASF alert-sending device, and so OS-present environment helps in doing that for e.g. management console and local system TCP/IP addresses. The other limitation is observed in case of a non-ACPI-aware operating system. In this scenario, the OS-present control of system-specific ASF feature is reduced, since ACPI provides current-generation “standard” methods in order to establish a communication between the OS-present environment and system firmware. For e.g. Plug-and-play calling interfaces; not easily supported in current-generation operating systems [50].

ACPI (Advanced Configuration and power Interface) is an industry specification to handle power consumption in desktop and mobile computers efficiently. ACPI defines a set of rules for computer's operating system, basic input/output system and peripheral devices to communicate with each other for efficient power usage. ACPI must be supported by the computer motherboard, basic input/output system (BIOS), and the operating system [52]. In an ACPI-aware environment, the client’s configuration data is interrogated to retrieve necessary information by the alerting device’s configuration software. They are Manufacturer ID and System ID, system GUID,
TCP/IP address assigned to the ASF-alerting device, waiting time of the alerting device before issuing a system boot failure alert, etc. This information is collected and placed into the device’s non-volatile storage for use in the OS-absent environment [51].

4.3 Active Management Technology

Intel Active Management Technology (Intel AMT) is a remote hardware management (out-of-band management) technology for personal computers introduced in year 2006. AMT is built into PCs via Intel vPro technology on Intel Core Processors and into workstations via Intel Xeon Processors [53] [54]. Typically used along with a software application, AMT provides remote access to a networked system even with power or OS failure. AMT also has features not available in traditional remote management systems, such as secure and wireless drive wiping, remote access to boot or power processes, access to BIOS settings, system re-installation. These features are also available even when the PC is powered off, given that the PC is plugged into a power source (i.e. power cord attached). In such scenarios, traditional remote management technologies would usually need an administrator’s presence. AMT is not intended to be used by a software management application, a management application is given a better access to the PC, to remotely and securely do tasks that are difficult or sometimes impossible when working on a PC that does not have remote functionalities built into it. Intel also allows developers to build applications using APIs for AMT. AMT provides similar functionality to IPMI, although AMT is designed for client computing systems as compared with the typically server-based IPMI.
AMT is designed into a secondary processor which is located on the motherboard, and it uses TLS-secured communication and strong encryption to provide additional security [55]. One of the major concerns about Intel’s AMT has been about security. Since it allows access remote access to OS, it may be misused and is vulnerable to rootkits. Also, since AMT is a software application by itself, compromising AMT would allow unauthorized and illegitimate access the host computer. Such exploits have been evaluated in [56] [57]. Note that system chipset or non-vPro processors can limit the manageability features to Intel Standards Manageability [58]. Apart from these weaknesses, AMT also does not provide a complete view of a network of PCs and management over them, which is the focus of Redfish’s platform management.

4.4 Desktop and Mobile Architecture for System Hardware (DASH)

DASH [59], an acronym for Desktop and mobile Architecture for System Hardware introduced in year 2007, is one that delivers web service management for desktop and mobile client systems, adhering to DMTF’s “Web Services for Management Specification.” The DASH management data and operations are defined by DMTF CIM schema and thus DASH uses DMTF’s WS-Management protocol for communication of CIM objects and services. It allows DMTF to securely and remotely manage desktop and mobile systems. Examples of such tasks would be safely and remotely starting up a system, analyzing asset information for a system that’s currently switched off or tallying system health information under the unavailability of the OS.
It is an industry open standards initiative. It defines the protocols and the processes for over-the-wire management of desktops, workstations, laptops, and converged devices [58]. It can be considered as an alternative to Intel AMT as it shares the same features and goals as Intel AMT. In its very entelechy, DASH extends support for the re-routing of various peripherals like text consoles, keyboards, and mice, apart from providing the necessary support towards backing the management of software updates, Network Interface Card(NIC), BIOS (Basic Input Output System). It supplements this by concerning itself with the addressing of issues primarily governing opaque data management and operating system status, among others. DASH is compatible with any chipset providing certain requirements are met, unlike AMT that can only be used with high-end Intel desktop and mobile chipsets. The DASH later versions have additional features like USB media redirection or text console redirection.

When the system is plugged into an AC outlet, the presence of auxiliary power is used; the Management Controller is independent from the system’s state and the DASH requests can be monitored through LAN traffic and OOB tasks like power control, inventory, reporting systems state, etc. are performed. Additional services are offered such as alternate boot paths, additional event alerts, remoting the BIOS post screen, etc. once the system is powered on.

4.5 Systems Management Architecture for Server Hardware (SMASH)

With the idea of addressing lack of cross-platform standards that directly manage the servers from multiple vendors, DMTF created the SMASH Initiative.
order to unify the management of data center, the SMASH Initiative comprises of a set of specifications which deliver architectural semantics, DMTF Management Profiles and industry standards protocols. SMASH describes system management interactions independent of the actual implementations, providing a common vocabulary for server management. This enables simple and intuitive management of heterogeneous servers in the data center which are independent of operating system state, access method, machine state, or server system topology. Local and remote management of server hardware is facilitated in both Out-of-Band and Out-of-Service management environments.

SMASH Management Protocols comprises of both web services and command line protocols for server management. SMASH specifies the Web Services for Management Protocol (WS-MAN) for programmatic access by management software. SMASH defines a command line protocol or SMASH CLP or uses command line tools or scripts for management by an administrator. In addition to this, SMASH includes Management Profiles covering stand-alone servers, modular systems (blades), racks, system hardware components etc. addressing both the enterprise and telecommunications server environments.

A standardized data model representation has been defined as profiles in order to use simply the management. Profiles are defined to be able to use management commonly across implementations. Using profiles simplify managing the power, system configuration, storage, system boot, hardware product assets and firmware update.
Management tools might need to support heterogeneous server management. Support in SMASH for a programmatic interface via WS-Man allows this for the management software developers. To manage their multi-vendor environments, the server administrators do not have to juggle a huge set of proprietary tools. Instead, these administrators can manage their data center environments from a single management tool or a single console. This lowers the complexity of the system and reduces the cost. The SM CLP’s interface is human oriented and it provides a standard command set to control the hardware in heterogeneous environments to reduce management complexity and costs in data center. The systems offered by different vendors might seem inconsistent, but with SMASH, they will be represented in standard ways. To perform common operations on a management station or a client (like system power on and off, system log display, boot order configuration and text-based remote console), users can use SM CLP which makes it possible to use the same commands across disparate vendor platforms. SMASH has support in products from the many industry’s leading vendors and it is widely implemented.

4.6 Summary

Table 2 below describes major differences between the above-mentioned standards and how Redfish overcomes their limitations [55] [58].
### Table 2. Standards comparison results

<table>
<thead>
<tr>
<th></th>
<th>IPMI</th>
<th>ASF</th>
<th>AMT</th>
<th>DASH</th>
<th>SMASH</th>
<th>Redfish</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>A hardware health monitoring and management system that manages the system remotely.</td>
<td>A standard defined for OOB PC management (when OS is not running)</td>
<td>Intel initiative delivering enhanced OOB management</td>
<td>Next generation standard for secure OOB remote PC management based on web services (WS management)</td>
<td>A cross-platform standard that enables simple and intuitive management of heterogeneous servers in the data center.</td>
<td>Performs OOB systems management via RESTful interface</td>
</tr>
<tr>
<td><strong>OOB Manageability?</strong></td>
<td>Yes, system can be managed in any state</td>
<td>No, system must first be remotely woken to S0 state.</td>
<td>Yes, system can be managed in any state</td>
<td>Yes, system can be managed in any state</td>
<td>Yes, system can be managed in any state</td>
<td>Yes, system can be managed in any state</td>
</tr>
<tr>
<td><strong>Transport Layer Protocol</strong></td>
<td>UDP</td>
<td>UDP</td>
<td>TCP</td>
<td>TCP</td>
<td>TCP</td>
<td>TCP</td>
</tr>
<tr>
<td><strong>Remote Control?</strong></td>
<td>Yes, Serial Over LAN, KVM</td>
<td>Limited. Remote Reboot only</td>
<td>Yes, Serial Over LAN, Media and Text Redirection</td>
<td>Yes, Serial Over LAN, USB Media Redirection or Text Console</td>
<td>Yes, Serial Over LAN, Text message-based transport protocol</td>
<td>Yes, Serial Over LAN, Serial Console Command Shell, KVM-IP and Virtual Media</td>
</tr>
</tbody>
</table>
Table 2. Continued

<table>
<thead>
<tr>
<th></th>
<th>IPMI</th>
<th>ASF</th>
<th>AMT</th>
<th>DASH</th>
<th>SMASH</th>
<th>Redfish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Communication via</td>
<td>RMCP+</td>
<td>Simple Authentication, Pre-Shared Keys (PSK)</td>
<td>HTTP Digest/TLS</td>
<td>HTTPS</td>
<td>HTTPS</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Remote BIOS Update?</td>
<td>In some servers</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3 below does the comparison between REST and SOAP protocols and Table 4 does the comparison between TCP and UDP.
### Table 3. REST vs SOAP comparison

<table>
<thead>
<tr>
<th>Factor</th>
<th>REST</th>
<th>SOAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What is it?</strong></td>
<td>REST is not a protocol but a technique to implement web services using HTTP with its standard operations (PUT, GET, POST, DELETE) as well as XML documents</td>
<td>IT is an XML based protocol, which was originally developed to be used for distributed applications and uses HTTP for communication</td>
</tr>
<tr>
<td><strong>Technologies</strong></td>
<td>JSON, XML, HTTP, ATOM, MIME, SSL/TLS</td>
<td>Can use different protocol - SMTP, uses XML, supports WSDL</td>
</tr>
<tr>
<td><strong>Type Handling</strong></td>
<td>For REST, the return values need to be de-serialized from XML in HTTP based APIs. However, this is minimal effort for dynamic languages as traversing an object is very similar to traversing an XML tree, so there is no specific advantage of having standard data types</td>
<td>SOAP provides a stronger typing as there are fixed set of data types which are supported. This ensures that the return type will be available in a specific native type for a particular platform</td>
</tr>
<tr>
<td><strong>API Flexibility and simplicity</strong></td>
<td>REST uses URI to write Web Services. Exposing a service could be very simple and include making a script accessible to Web Server via URI. Then any Client or server application could call the service using HTTP GET command and get the response as a header, text string or XML document</td>
<td>SOAP is comparatively complex here and requires specific knowledge of the XML specification. Most of the developers would also require a SOAP toolkit to create requests and parse results</td>
</tr>
<tr>
<td><strong>Bandwidth Usage</strong></td>
<td>Requests and responses can be short here as REST is flexible. Developers can write WSDL documents for their services or provide a human readable declaration. This makes REST much lighter</td>
<td>As SOAP requires an XML wrapper around each request and response, a SOAP response could need 10 times more number of bytes than similar REST response. As the application as well as the service know the</td>
</tr>
<tr>
<td>Security</td>
<td>With REST using HTTPS, each message can be analyzed and its intent can be identified by administrator/ firewall by analyzing the HTTP command that was used. REST methodology uses industry-standard certificates and identity management systems for authorization and authentication.</td>
<td>SOAP uses an envelope structure and without looking into SOAP envelope - a task which is not built into most firewalls and is highly resource-consuming, one cannot know the intent of the request and this compromises network safety. SOAP puts the responsibility of authorization and authentication in the hands of the application developer and causes huge burden on developer and non-standardizations.</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Client Side Complexity</td>
<td>Calling HTTP APIs is relatively easier than calling a SOAP API. Its native to all programming languages and requires constructing an HTTP request with appropriate parameters.</td>
<td>SOAP requires a client library, a stub as well as a learning curve making it more complex.</td>
</tr>
<tr>
<td>Caching</td>
<td>Proxy servers and reverse proxies can cache their responses easily as RESTful APIs can be consumed using simple GET requests.</td>
<td>SOAP requests use POST and need complex XML request to be created that makes response-caching difficult.</td>
</tr>
</tbody>
</table>
Table 4. TCP vs UDP Comparison

<table>
<thead>
<tr>
<th>Factor</th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transmission Control Protocol</strong></td>
<td><strong>User Datagram Protocol or Universal Datagram Protocol</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Connection-oriented protocol</td>
<td>Connectionless protocol</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Suited for applications that require high reliable and less time critical</td>
<td>Suitable for time critical applications that need fast and efficient transmission such as games</td>
</tr>
<tr>
<td><strong>Data Packets reliability</strong></td>
<td>The data packets are arranged in an orderly fashion and is guaranteed to remain intact and arrives in the same order in which it was sent, making it reliable</td>
<td>The data packets are independent of each other, and thus do not follow any order. If ordering is needed, it must be managed by the application layer. There is no guarantee of the messages or the packets to reach at all, making it less reliable</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>Slower than UDP</td>
<td>Faster as error recovery is not attempted</td>
</tr>
<tr>
<td><strong>Header Size</strong></td>
<td>20 bytes</td>
<td>8 bytes</td>
</tr>
<tr>
<td><strong>Acknowledgement</strong></td>
<td>Acknowledgement segments</td>
<td>No Acknowledgement</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>Heavyweight, requires three packets to set up a socket connection, before sending the data.</td>
<td>Lightweight, as no ordering of messages and no tracking connections required.</td>
</tr>
<tr>
<td><strong>Transfer Method</strong></td>
<td>TCP data is read as byte stream and there is no method to distinguish between beginning and end of packets. TCP this has multiple packets per read call</td>
<td>AS UDP is designed to send packets individually and have defined boundaries. UDP this has one packet per read call</td>
</tr>
<tr>
<td><strong>Error Detection</strong></td>
<td>TCP uses error detection as well as error recovery. TCP uses checksum to detect errors and also supports Positive Acknowledgement and Retransmission (PAR) mechanism, where a packet is retransmitted in case it is not acknowledged by the receiver</td>
<td>UDP uses checksum but instead of retransmitting packet, when an error is detected EDP drops the packet</td>
</tr>
</tbody>
</table>
CHAPTER V

IMPLEMENTATION AND RESULTS

The Redfish Conformance Test Tool (RCTT) validates the current Redfish specification (version – 1.0.5). The specification document is the document that provides description about all data and requirements (including both functional and behavioral) of the software that is being produced or developed [60].

5.1 Assertions and its Types

Assertions are the tests that have been extracted from the Redfish Scalable Platforms Management API Specification (RSPMS) version -1.0.5. The extractions are performed based on the terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not - required"), "can" and "cannot" [61]. The Redfish Specification includes total of 242 assertions. They are divided into: Protocol Testing (141 assertions), Data Model Testing (35 assertions), Service Details Testing (29 assertions) and Security Testing (37 assertions). Figure 13 describes these categories.

![Figure 13. Assertion categories]
Figure 14 below displays all the dependencies of the RCTT.

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Python 2.7 or 3.4</td>
<td>Python environment setup on Client</td>
</tr>
<tr>
<td>2</td>
<td>openpyxl 2.3.5</td>
<td>To read/write Excel 2010 xlsx/xlsm/xlt/xltm files</td>
</tr>
<tr>
<td>3</td>
<td>Pillow 2.5.3</td>
<td>Python Imaging Library</td>
</tr>
<tr>
<td>4</td>
<td>xld 1.0.0</td>
<td>Extract data from Excel spreadsheets</td>
</tr>
<tr>
<td>5</td>
<td>XlsxWriter 0.9.3</td>
<td>Python module for writing files in the Excel 2007+ XLSX file format.</td>
</tr>
<tr>
<td>6</td>
<td>xlwt 1.1.2</td>
<td>Library to create spreadsheet files</td>
</tr>
<tr>
<td>7</td>
<td>tkinter</td>
<td>Standard Python interface to the Tk GUI toolkit</td>
</tr>
<tr>
<td>8</td>
<td>assertions folder</td>
<td>To run the tool as-is, assertions folder should be present in the same</td>
</tr>
<tr>
<td></td>
<td></td>
<td>directory as rf_client.py with rf-assertions-run.xlsx file in the folder</td>
</tr>
<tr>
<td>9</td>
<td>properties json file</td>
<td>To run the tool as-is, properties.json should be present in the same</td>
</tr>
<tr>
<td></td>
<td></td>
<td>directory as rf_client.py with proper settings</td>
</tr>
<tr>
<td>10</td>
<td>Schemas</td>
<td>Schema files should be present in a separate folder in the same directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>as rf_client.py, with folder name correctly set in properties.json.</td>
</tr>
</tbody>
</table>

*Figure 14: RCTT Dependencies*

**5.2 High Level Design**

The tool is implemented in programming language Python. The tool provides a graphical interface to the user/administrator to enter the server information. The entries from the UI serve as SUT properties in a json file (properties.json), further explained below. These are then used to set up a connection with the server as shown in Figure 15. Once these details are submitted, it directs to the second page where the user is provided with a choice of “Run Conformance Test Tool” as shown in Figure 16.

*Figure 15: RCTT User Interface I*
The “Run Conformance Test” button calls the run() function in Conformance Test Suite to run the assertions. In the end the user must click on “Clear Details” button to clear out all the login information of the server, for the security purpose. To explore the operational Redfish Service, this tool provides an implementation. It also verifies the service’s conformance to the normative statements present in the RSPMS. The overview of the design is shown in the below Figure17:

![Redfish Conformance Test Tool](image)

Figure 16: RCTT User Interface II
The tool starts with the GUI as explained above where user enters the details, these details gets updated into the properties.json SUT values. SUT (System Under Test) refers to a system, in this case Redfish services, to be tested for the correct operations. Always defined from the perspective of the test, it can be regarded as an abbreviation of "whatever thing we are testing". For example, in case of writing a unit tests, the SUT could be a class, object or methods; whereas in case of customer tests, the SUT is probably the entire application [62]. In our case the SUT contains all the server information on which the Redfish services are being tested. The properties.json file has two configurations: SUT configuration and RCTT_SchemaFiles configuration.

Each SUT is represented as a dictionary within the SUTs list. The required properties for each SUT are as follows: DisplayName, DnsName, LoginName and Password. In case of RCTT_SchemaFiles, it is required to provide the folder name where all the schema files would be residing. This folder should be placed in the same directory.
as rf_client.py. The folders are expected within this parent Schema folder, one for json schemas named “json-schema” and csdl schemas folder named “metadata” [61]. If schema is required to be downloaded from DMTF Repository hosted on DMTF’ site, set “RetrieveDMTFSchemas” to “yes”, else change it to “no”. If proxy is required to bypass firewall, set appropriate “https_proxy” or ‘http_proxy’ according to the requirement of URL. If no proxy is required, set both “https_proxy” and http_proxy to “none”.

The Client (rf_client.py) uses the SUT properties from properties.json file. This property object (sut_prop) is passed to the setup_tool() function of the Client (rf_client.py). Once SUT setup is done successfully, the Client calls the run() function of Redfish Conformance Test Suite (rfs_test) to run all the assertions, where all the assertions includes protocol (TEST_protocol_details.py), data model (TEST_datamodel_schema.py), security (TEST_security.py) and service details (TEST_service_details.py) as shown in Figure 18. Redfish Conformance Test Suite uses the Logger (required) and some functions in the HTTP and Restful Utility module.

Figure 18: Redfish Service Conformance Test Suite
The testing of the assertions is based on the interpretation. For example, Figure 19 displays one of the assertion that validates the *Action Representation* and Figure 20 displays the algorithm to validate that assertion. The algorithm is self-explanatory.

### 6.5.4.7.1. Action representation

Actions are represented by a property nested under "Actions" whose name is the unique URI that identifies the action. This URI shall be of the form:

```markdown
#Namespace.ActionName
```

*Figure 19: Assertion – Redfish Specification*

- For all URIs with ‘Actions’ in JSON Payload:
  - Extract Namespace `ns` and ActionName `action` from ‘#ns.action’
  - Go to Namespace `ns` metadata file
  - For every Action `ns.action` in Namespace file:
    - If `ns.action == action`:
      - Found an action `ns.action` within `ns` as mentioned in ‘#ns.action’
      - PASS (proving that URI of the form ‘#ns.action’ is an action)
    - If no such action exists:
      - FAIL

*Figure 20: Assertion Interpretation*

### 5.3 Evaluation

The RCTT was evaluated against Mockup data and Real server provided by Dell specifically for this testing purpose. All the Redfish resources are identified by a URI (Uniform Resource Identifier). This serves as one of the base components of the internet. For easy access to the resources, URI has been encoded in numerous applications. Now-a-days, encoding the URI includes a percentage symbol followed by
hexadecimal code (%hex) or the use of Base64 [63]. For Redfish, URI request is sent via POSTMAN client for the mockup server and via RESTED client for the Real Server.

*POSTMAN* is a tool for testing APIs. The founders came up with the idea of POSTMAN when old testing APIs were not easy to use. They realized that a new and better language was necessary to help developers communicate about the APIs, which led to the creation of POSTMAN. This tool helps in building, testing and documenting APIs much faster [64].

A new take on the REST clients for Firefox is *RESTED*. It is easier to use and it helps to work in the most effective way. It features most of the commonly used HTTP methods such as setting headers, saving the requests to a local storage, and many more [65]. Once responses were received from both the clients, they were evaluated against responses received from the RCTT. Some of the test cases were not valid for mockup server, for e.g. POST operation could not be checked with the mockup server, but has been tested with the Real server. Regression Testing was performed in this case, as the code must be changed when shifting from mockup server to the real server. Figure 21 below describes this in brief.

*Figure 21: Evaluation Flow Diagram*
5.4 Results

5.4.1 Validation of Redfish Result:

When we run the tool, it creates a “logs” folder in the directory which has the rf_client.py. It then copies the rf_assertions-run.xlsx file in this directory. The current time stamp is appended to the filename. The assertions are numbered in the spreadsheet. They are assigned the colors Green, Yellow and Red, which represent the results ‘PASS’, ‘WARN’ and ‘FAIL’ respectively. The assertions also have comments (whenever necessary) right against the assertion. Figure 22 below displays the result with proper color and comments assigned. This excel file will help in identifying if all the assertions work fine as per the specification. The result generated with mockup and real server will be different, as some of the operations are not supported with mock up data for e.g. POST, PATCH.

There were some noticeable findings while performing the testing. For e.g. for every action to be performed on a resource, there is a “target” property beginning with a small ‘t’ provided with a URI. We observed that for some of the resources, target property was beginning with “Target” capital ‘T’. The tool found this error in performing the test where it showed “The target property is not found”. This is a trivial problem, but the user might get confused since Redfish follows standard. This finding was informed to DMTF team, and the issue was on the Dell server when the transition happened from mockup schema into the production. The mockup schema had ‘Target’, and so it got reflected in one of the resources.
<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Section 6.4 HTTP Methods</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.7</td>
<td>URIs, as described in RFC1806, may also contain a query (#query) and a fragment (#frag) component. Queries are addressed in the section Query Parameters. Fragments (#frag) shall be ignored by the server when used as the URI for submitting an operation.</td>
<td>The query includes top and skip. The fragments '#frag' are considered as the comment in the URL. Tested</td>
</tr>
<tr>
<td>6.1.8.2</td>
<td>GET Object or Collection retrieval Yes</td>
<td>Assertion tested</td>
</tr>
<tr>
<td>6.1.8.3</td>
<td>PATCH Object update Yes</td>
<td>Assertion Tested™ note: the header returned from GET /redfish/v1/Managers/iDRAC.Embedded.1/Accounts/1 do not indicate support for PATCH{ &quot;date&quot;: &quot;Sat, 14 Mar 2017 09:02:51 GMT&quot;, &quot;server&quot;: &quot;RedfishMockupHTTP_v0.9 Python/3.4.0&quot; }, &quot;note&quot;: the header returned from GET /redfish/v1/Managers/iDRAC.Embedded.1/Accounts/1 do not indicate support for DELETE{ &quot;date&quot;: &quot;Sat, 14 Mar 2017 09:03:41 GMT&quot;, &quot;server&quot;: &quot;RedfishMockupHTTP_v0.9 Python/3.4.0&quot; }, &quot;&lt;PUT and HEAD may be also be supported&gt; - Other HTTP methods are not allowed and shall receive a 405 response. Tested with TRACE operation and received 405™ TRACE:/redfish/v1/ failed: HTTP status 501:Not Implemented, Expected status 405:Method Not Allowed™ TRACE:/redfish/v1/Chassis failed: HTTP status 501:Not Implemented, Expected status 405:Method Not Allowed™</td>
</tr>
<tr>
<td>6.1.9</td>
<td>All resources shall be made available using the JSON media type &quot;application/json&quot;.</td>
<td>Assertion tested</td>
</tr>
<tr>
<td>6.1.11</td>
<td>resource available in a representation based on JSON, as specified in RFC4627. Receivers shall not reject a message because it is encoded in</td>
<td>Checked all the responses and all the responses are in JSON and it is all accepted by the client.</td>
</tr>
</tbody>
</table>
5.4.2 Comparison Result:

This section summarizes the results from section 4.6 providing reasons for why Redfish is a better standard than the existing standards. For OOB functionality, as shows in table 1, Redfish uses RESTful interface as compared to other standards which use WS-* based management standards or legacy techniques. A sample high level block diagram summarizing the operations of Redfish vs other standards is shown below [66].

![RESTful interface](image1)

*Figure 23 RESTful interface*

![WS-* based Interface](image2)

*Figure 24 WS-* based Interface*

One example of differentiation is the usage of REST/JSON based connection to communicate to the BMC as compared to the RMCP based communication used in IPMI [67]. In IPMI, the messages were encapsulated in RMCP data packets and transferred via LAN connection using UDP. However, Redfish allows interaction with BMC using good data structure, specifically JSON payloads [68]. Redfish provides a reusable API standard across compatible servers. This is a significant development from the old method as developer can now expect to get an HTTP error code in place of an
obscure error message earlier. Moreover, Redfish also provides support for remote
BIOS update which was not provided in IPMI for all servers. RMCP is also limited on
security measures, which is not a problem in Redfish as it uses HTTPS with latest TLS
based security technique.

As compared with other WS-based architectures which are mostly based on
SOAP, REST architecture used by Redfish provides significant advantages as shown in
table 2. As REST provides a very simple way to write web services, as it uses URI as
compared to the SOAP approach which requires to have specific knowledge of XML as
well as SOAP toolkit. REST requests can use 10 times lesser number of bytes as
compared to those of SOAP because of flexibility to use human readable declaration
making it much lighter. As RESTful APIs can be consumed using GET requests, proxy
servers can cache the responses easily as compared to SOAP requests. REST also uses
TCP as compared to UDP used by IPMI which makes it much more reliable, and secure.
Using technologies such as JSON and OData makes Redfish easy-to-use as well as
human readable as compared to other standards. Having a flexible architecture allows
Redfish to be scalable and to be used in modern scalable data centers. Looking at all the
comparison results as well as the technology differences explained in this research for
key Redfish technologies, it can be concluded that Redfish is very well designed to meet
the requirements of the modern scalable data centers while providing good user
experience and simple solution.
CHAPTER VI

RELATED WORK

This chapter discusses different tests developed for varied platform management standards. These tests not only provide information on the topics of conformance testing and validation, but also show case the importance of testing in the industry for any new product or standard.

6.1 IPMI Validating system

IPMI validating system was developed and patented by Chih-Tao Hsieh and Chun-Te Wu to use between a host system having an IPMI and an operating terminal [69]. The validation system presented included an IPMI command engine module, a user interface, a channel management unit and an IPMI command management unit. This specific design can encode a test program into an IPMI command, rather than using ICTS and TCL/TK which was used previously. This method instead encoded the ICTS framework manager and test module directly into IPMI commands. This improved the validating system to provide better readability and performance, thereby improving the validating efficiency.

IPMI primarily consists of a baseboard management controller (BMC) and BMC embedded firmware. IPMI controls the system management software and the interface between hardware and platform. IPMI operates in different hardware and firmware platforms at the same time and provides system status information. However, as multiple manufacturers have their individual IPMI, the efficiency varies across the
vendors for hardware as well as firmware. This required to use the IPMI conformance test suite (ICTS) to determine if the IPMI is compatible across different system environment using a Pass/Fail test.

When ICTS is required to validate IPMI, it needs to be loaded with TCL/TK test module as well as ICTS must be connected to the TCL/TK engine out of ICTS. As TCL/TK language is used to write TCL/TK test module, TCL/TK engine as well as the ICTS framework, the TCL/TK module need to first encode the test module and framework into IPMI commands. Moreover, the configuration files that do not belong to ICTS need to be loaded from user computer and the transport modules are initially written according to the requirement before compiling. All these factors combined make the validation process complicated as well as add limitations on ease of use, simplicity, performance and time required to validate IPMI.

The invention which was presented developed an easy-to-use management structure by using IPMI command engine module, a user interface, a channel management unit and an IPMI command management unit. This design eliminates the need to use prior ICTS and TCL/TK engine. The IPMI command engine translates and encodes a test program directly into an IPMI command. This design also uses a simplified script language to write into a test program. All these improvements avoid the prior need to have multiple connections between ICTS framework manager and TCL/TK engine hence improving the performance efficiency. This patent clearly highlights the limitation of IPMI in ease-of-use and simplicity, which is one of the
primary improvements in Redfish as well as shows importance of validation. This thesis validates all the claims of Redfish and provides a tool to simplify test.

6.2 Data Center Manageability Interface (DCMI) test system

As discussed earlier in section 5.1, DCMI is a modified version of IPMI developed to target high density data centers. Intel developed DCMI conformance test suite (DCTS) which provides a baseline set of tests to validate compliance with DCMI. As with other conformance test tools, DCTS also provides “black box” testing where the tests are generated to check the functionality of the specification based on the input/output combinations.

The test suite was designed with four main objectives [70]. The first objective was to simplify the description for DCMI specification’s interface. Second objective was to increase confidence of the customer on interoperability as there were multiple implementations in the market, which limited customer’s choice to use only a specific type of system from a vendor. The third objective of the test suite was to recognize the parts of the specification which had been implemented. This helped Intel as well the customer to identify any misses in the specification and develop a solution in case of a miss. Finally, the test suite provided a platform to implementers and users to understand working of DCMI interfaces and to understand the way in which interfaces are expected to be called.

Using these four objectives as the based, DCTS was developed as a simple menu driven solution where each test case validated a specific functionality and reported whether the logical test was a pass or a fail. The testing was supported by two modes
depending on the source of communication used – In-band and Out-of-band. In case of in-band testing, the IPMI driver needs to be installed and the monitoring and management tools should be chosen from the provided software installation. For out-of-band testing, it was necessary to configure the server manageability stack. The test would be one-to-one once the controller was configured with an IP address, and there was IP connectivity between the host PC and the UUT. DCMI test system highlights the importance of interoperability and customer satisfaction, which can both be tackled with the conformance test suite.

6.3 DASH and SMASH Conformance Testing

Developing standards help improve and solve multiple problems such as interoperability, simplicity and standardization while providing a cost-effective solution to scalability and integration of systems. However, developing standards is not sufficient to drive the change and improvements. Customers expect a minimum level of quality and support from implementations before adopting the standards. Desktop Management Task Force (DMTF) created System Management Force (SMF) to develop strong conformance programs and test suites for DASH and SMASH [71].

DASH and SMASH standards help describe which parts of the product are necessary for a product to be conformant and is a normative specification for the CIM schema and the DMTF Web based Management specifications. SMF conformance test tools verify the products based on the profile specifications and certify if their implementation is on grounds with the conformance policy. To enhance quality and improve customer satisfaction, SMF members take inputs from vendors and application
developers to develop conformance requirements and then translate those requirements into detailed tests. SMF requires its member companies to self-test implementations and provide results to be validated and certified. This shows the importance of conformance testing, especially related to quality of conformance testing and customer satisfaction.

The conformance test tool presented here provides a simple solution to test the Redfish specification. Related work discussed in section 6.1 describes the initiatives undertaken to simplify IPMI testing. The conformance test tool developed in this thesis is made even more simpler. Only a user interface is required to run the conformance test tool. Even a non-programmer can run the conformance test by providing required information in the UI and pressing the Run button. This conformance test tool also fulfilled all the objectives as discussed in section 6.2. Developing this test helped identify areas of improvement with the specification and thus simplify the description for the specification. Also, the tool would help instill confidence in the customer regarding conformance of the specification as well as allow implementers to learn how different APIs are called. Redfish specification was developed by collaboration of various participants and the conformance requirements were defined by the forum. The test tool was developed based on the specification and DMTF members were kept informed on the findings of this thesis.
CHAPTER VII

CONCLUSION AND FUTURE WORK

In the last few years, the data centers industry has seen a tremendous growth in number of servers for solving big computation problems. This enormous number of servers has shown some unique problems in the areas of server manageability. There has been a variety of platform manageability features and interfaces offered by different server platform vendors to manage many different classes of servers in many application environments. Different server vendor’s solutions tie to vendor-specific management console software, agents, drivers and utilities that are required to provision and utilize the management capabilities. This calls out for the standard, and Redfish Scalable Platforms Management API (“Redfish”) is a specification that uses RESTful interface and provide out-of-band management and is suitable for a wide range of servers. Redfish brings standardization and switching to a different vendor won’t be a painful task as it has been in the past.

In this thesis research, we have validated Redfish Specification through Redfish Conformance Test Tool (RCTT), where each assertion is tested and is validated if the result achieved is what expected in the specification. This also checks if the specification is understandable to the user, who might not be expert in programming as well. If something in the specification is not understandable, the Redfish development team is informed and necessary changes had been done in the newer version of the specification. This research also focuses on comparing the other standards with Redfish and provides
a summary how standards have evolved and why Redfish stands out to be a better choice.

The future work will be to keep the tool up to date with the new version of the specification. Also, the tool now focusses on “shall” implementation which is mandatory. The tool in future will also focus on the “may” implementation. RCTT runs one assertion at a time and thus, takes a lot of time to complete the whole testing. If the testing can be parallelized i.e. more than one assertion running in parallel, the testing runtime can be reduced. Also, the testing does not cover the remote aspects of the BMC configuration, as it’s a new feature in Redfish services. This can also be covered in future.

At present, Redfish implementations are relatively new and offered so far by only some major vendors. The current testing program has been done with only one vendor, Dell, but includes the ability to test mockups drawn from other vendor implementations and has been designed to test the most important basic features common to all Redfish-compliant hardware. Future work can be pursued to extend the techniques described in this thesis to other vendors and other redfish implementations.
BIBLIOGRAPHY


75


77


