Enhanced Service Oriented Software Framework for Embedded Android

P.K.L.S.Wijesekara

158780A

Faculty of Information Technology

University of Moratuwa

May 2018

Enhanced Service Oriented Software Framework for Embedded Android

P.K.L.S.Wijesekara

158780A

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the fulfillment of the requirements of Degree of Master of Science in Information Technology.

May 2018

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student: Mr. P.K.L.S.Wijesekara

Signature of the Student: ____________________

Date:

Supervised by:

Name of Supervisor: Mr. C.P. Wijesiriwardhana

Signature of Supervisor: ____________________

Date:

Dedication

I would dedicate this thesis to my beloved family members who have never failed to give me a tremendous support, for giving all not only throughout my project but also throughout my life as well. As well they teach me that even the largest task can be accomplished if it is done one step at a time.

Acknowledgement

I wish to express my eternally grateful and gratitude to my project supervisor Mr. C.P. Wijeseriwardhana, Senior Lecturer, Faculty of Information Technology, University of Moratuwawho untiringly shared his knowledge, precious guidance, encouragement, advices and help given to complete the project successfully. Also, I thank to Prof A.S Karunananda who gave me guidance and advices to complete the project successfully.

I am also grateful to previous researchers who have contributed to the domain of embedded application developments, academic and non academic staff of Faculty of Information Technology, University of Moratuwa and also batch mates who have given me valuable feedbacks to improve the project.

Importantly I would like to thank my wife and my family members for the encouragement and undying support not only throughout my project but also throughout my life.

Finally, the support of University of Moratuwa is also sincerely acknowledged.

Abstract

In early days, embedded systems were limited to apply in scientific researches and to the devices which supported military requirements. Nevertheless the embedded systems are much more popular in consumer electronic, cooking, industrial, automotive, medical and commercial applications today. Current approaches to embedded system related software development, is having lack of rich frameworks with service orientation to provide freedom to access embedded system services. There for it is much more important to develop service oriented software framework that has enhanced features to operate embedded systems. Out of the embedded systems domain, the research has navigated to select embedded android and its Linux kernel as the preferred system due to its popularity and the availability of related resources. The approach to develop EmSOSwas based on service orientation principles to deliver a software development framework.EmSOS is an acronym for - Enhanced **S**ervice **O**riented **S**oftware Framework for **Em**bedded Android

We hypothesize that issue of having lack of rich frameworks with service orientation to access embedded system services, can be addressed by introducing a rich framework. This hypothesis been inspired by the power of service oriented architecture.

Several categories of users can be identified for EmSOS. They are software architects, software engineers, System analysts, embedded system developers, and also researchers of embedded systems domain.

Table of Contents

List of Figures

List of Tables

Chapter 1

1. Prolegomena

1.1 Prolegomena

Enterprise software always focuses on enhanced features to be added in to previously released versions [3]. Over the last decade, most of the software applications have grown to a level where they are difficult to be maintained and the complexity has increased to integrate with other applications [4]. Therefore it's time to migrate legacy applications in to distribute and SOA based solutions. The same concern is valid for embedded software which has covered a considerable area of software domain through mobile devices such as smart phones, Printers, Tablet PC and GPS navigators [2]. It is evident that there is quiet lot of identified problems to be sorted in the domain of embedded software. One of the identified problems is power constrain in embedded system [1]. There have been number of researches that have conducted to include power consumption feature in to embedded software and still the researches are going on to enhance the feature. That is not only but also there is a problem of fulfilling dynamic requirements that are coming from the industry to embedded systems [1]. The other important problem to be addressed is there is lack of developer support to access the functionalities of peripherals to access through networks and process scheduling and controlling remotely [2]. It has identified that there are plenty of requirements to remotely executable system calls of embedded systems [12, 13, 14].

The growth of embedded software has significantly increased during the last decade. Today it is evident to see that embedded software are distributed and designed in a manner to support dynamic requirements [7]. Embedded software can be considered as one of the major components of large mechanical and electrical systems and the heart of the mobile devices like GPS navigators, tablet PCs, mobile phones. Together with the hardware of the device, embedded software provides controlling capability with greater device functionality. In recent years a movement from distributed systems to automatic, autonomous, and self-configuring systems is noticeable [3]. SOA approach is capable of addressing issues of service addressing, announcement, service discovery and registering and looking up for a service registry.

1.2 Aim and objectives

Aim

The aim of the research project is to design and implement a service oriented software framework that has enhanced features to develop applications for embedded systems based embedded android and embedded Linux distributions.

Objectives

1. Critical review of literature in the approaches to develop embedded systems. (Problem related)

2. In depth study of technology used in approaches to develop embedded systems.

3. Design and develop a prototype Framework for embedded systems development.

4. Evaluate the use of the application of Framework for embedded systems development.

5. Prepare the documentation and provide complete guide line for the framework of the project.

1.3 **Background and motivation**

Widely used kernals to develop embedded systems were Linux and contra in the 90 s But due to the factors like robustness, customizability and availability of resources, the Linux have become more popular among all other kernals[3].Embedded Linux have come to a key point of success after releasing android official versions by goggle in 2008.This could make a rapid change in the market of mobile phone industry after an year of initial release of the OS. It was evident to see that the Nokia had stop releases of the Symbian OS due to the popularity of the android in the market at 2012.All of the mobile phone vendors have come to a conclusion of going with customized embedded android instead of other systems. Embedded android were powerful to work in an ARM processor environment even rather than windows mobile [2].

It is important to consider the resource availability of embedded devices hence they are unlike personal computers that are capable of upgrading resources as required. Technologies such as OpenService Gateway Initiative (OSGi), Home Audio/Video Interoperability (HAVi), Java Intelligent Network Infrastructure (JINI) and Universal Plug and Play (UPnP) have tried out different approaches to introduce SOA based solution with lot of difficulties [4].

1.4 **Problem in brief**

It is evident to see that identified problem of issues in platform dependencies, high complexity to the developer, additional learning curve of a novel embedded system to the developer and also issues of scalability due to resource constrains are highlighted as the problem to be addressed in the research.

1.5 **Solution in brief**

1.5.1 Users

Several categories of users can be identified for EmSOS. They are software architects, software engineers, System analysts, embedded system developers, and also researchers of embedded systems domain.

1.5.2 Input

Inputs for the approach, initializes with the architectural design of the intended framework. The architectural design defines components of the framework and the way they should be integrated together. Itneeds to provide JDK,android development studio and operating system. Planned operating systems for testing were .Type of embedded devices targeted is mobile phones, printers, tablet PCs and smart TVs.

1.5.3 Output

Output of the research consists of EmSOS framework, developer guide and also a test environment for framework validation. Also a documentation to access end points.

1.5.4 Process

Development process of EmSOS is based on a Java development. This process will be converting the architectural design of the framework in to a code base which provides web services. This development consists of number of java micro services to manage as a combination of smaller services that works cohesively together for larger, application-wide functionality. Those micro services will be deployed in a third party server likeNanohttpdor tomcat web server.

The process of the framework development includes J2EE framework development andframework configurations.

1.5.5 Technology

Technologies to be developed the framework is based on Java 8.The framework consists of micro services which structure loosely coupled services for consumers. The project refers to an application building frameworks of gradle and spring java micro services. These building frameworks are necessaries to develop most of the application and other frameworks in current generation. The selected embedded operating systems to develop and test are Linux kernel 2.6, embedded android 6.0, 7.0, 8.0 and Windows 7, windows 10.Entire development process is a test driven development using jUnit to ensure maximum code coverage. Even though the development minimizes possible security issues by adhering to OWASP version 1.2.1 released in 2017. The coding standards have adhered to software engineering best practices. The coding stage was done with continues integration continues deployment, Standardizing the code to security by scanning the code using sonar framework. Version controlling of the software has done by referring to a git repository with a master branch for releases and three other branches as development, bug fixes and security fixes. Once an issue has identified, it has fixed under issue fix branch or under security fixes branch. Then the code has merged in to development branch. Final release has done by merging development branch to master branch of the git repository. The entire development was a test driven development with unit testing which is done by referring to Junit 4.

1.5.6 Features

This system has many encouraging features as follows.

- Easy integration
- Service oriented
- Micro services
- Web-based
- Easy accessible
- Low cost

- Secure connectivity
- Accessibility to system calls

1.6 System requirements

Hardware Profile 1

RAM : 2048 MB (minimum)

Supported device status: Portrait, Landscape

Sensors: Accelerometer, Gyroscope, GPS, Proximity sensor

CPU : ARM 64 bits

Hardware Profile 2

RAM: 2048 MB (minimum)

Supported device status: Landscape

CPU: ARM 64 bits

Software

Embedded android 6.0, 7.0, 7.1 and 8.0

Android development studio 3.0 or higher

Eclipse 4.7.2 or higher

Tomcat web server 8.1 or higher

1.7 Structure of the Thesis

This thesis has been structured with 8 chapters. Chapter 1 presents the overall picture of the thesis. Chapter 2 is on critical review of literature in approaches to embedded system development by defining the research problem and identification of technology to be used to address the research problem. Chapter 3 gives an in depth study of technology used for frameworks for embedded systems development with a particular emphasis on service oriented architecture. Chapter 4 describes the novel approach to embedded system development framework using the service oriented architecture. Chapter 5 discusses the top level architecture of the novel framework. Chapter 6 talks of implementation of the framework presented in the design chapter. Chapter 7 is on evaluation. Chapter 8 concludes the thesis with the note on limitations and further work for this research.

Chapter 2

2 Current issues of embedded systems

2.1 Introduction

The embedded software is capable of controlling a full set of electrical components such as microprocessors, signal processors, RAM, graphic processors, input output devices and many more. This software can be seen as embedded software operating systems and they have been highly specialized to perform relevant operations of the device. The operating systems like embedded Linux have played a big role in the domain of embedded software for years [2]. Embedded systems in consumer electronics have used Linux kernel to operate smart TV s, Networking equipments, industrial automation, medical instruments, personal video recorders and devices like printers in general. However the brand name android has recently became more popular within the family of embedded software. Android was already an embedded operating system since its root stemming from embedded Linux [6].

Even though embedded software has passed years in the industry, still there seems to be no proper software framework to access complete set of functionalities of the device through web services. It is evident to see that there have been no proper researches to cover the identified problem of accessing resources like CPU, network interfaces, display, USB devices and memory through web services[10]. There is a valid problem to provide a language independent solution for developers to minimize the difficulties of integrations. Embedded systems are having constrains of sharing resources of one device to be accessed by others due to reasons like difficulties of providing security [11]. The risk is higher to expose resources without having a proper security mechanism due to situations like hacking attacks.

When considers the resource limitations of hardwire, it additionally requires scalability. The existing mechanisms to connect embedded systems such as embedded networks are still having issues in error detection and recovery in a node failure [5]. Embedded systems needs to be customized for every single installation based on the hardware it resides. The customization is too costly and time consuming. Even though the embedded networks built for automation purposes contains nodes with a broad range of different capabilities. This heterogeneity requires tools that are capable of developing applications without having prior knowledge of the hardware configurations. Below are the identified issues in embedded network application development.

• Heterogeneous hardware from various vendors
• Run-time adaptability
• Life cycle management
• Distributed execution of applications
• Resource efficient data processing
• Scalability to capabilities of the underlying hardware
• Error detection and recovery
• End-user programmability
• Automation support for installation and configuration
• Web service based interface for communication with external services
• Semantic support for enterprise integration

The solution to address the problem is a service oriented software framework that has enhanced features to operate embedded android and Linux systems [7]. The proposed framework is capable of providing endpoints where the services of embedded android to be accessed by consumer applications[9]. These endpoints can be integrated to another application to access the devices and access resources of the service provider. The framework provides freedom to application developers to build applications that are capable of controlling hardware resources in their favor [12, 13].

2.2 Gestation of software frameworks

Birth of research in software framework researches date back to early 1990.[4]In the recent three decades embedded systems has shown multifaceted developments (growing interest) in many disciplines [15, 16]. Undisputedly According to the literature, perhaps people have produced the first reported embedded system applications in web services [2]. This embedded system application was developed to run on Mobile devices. Subsequently, numerous applications in embedded systems were developed by many researches [3]. For instance people have applied embedded systems for the domain of sensor network. This application used a light weight java platform which specially designed for systems with limited resources [14]. That's not only but also people have done a similar research for robotics. However, this work has used slightly powerful java platform. It should be noted that these applications are rather industry based and complex in nature.

In contrast, embedded systems in consumer electronics have also been introduced by many researches [3, 4, 8]. These applications are primarily targeted for automation of home appliances. The rapid growth of IOT [7] has made it required for home appliance to communicate through web services.

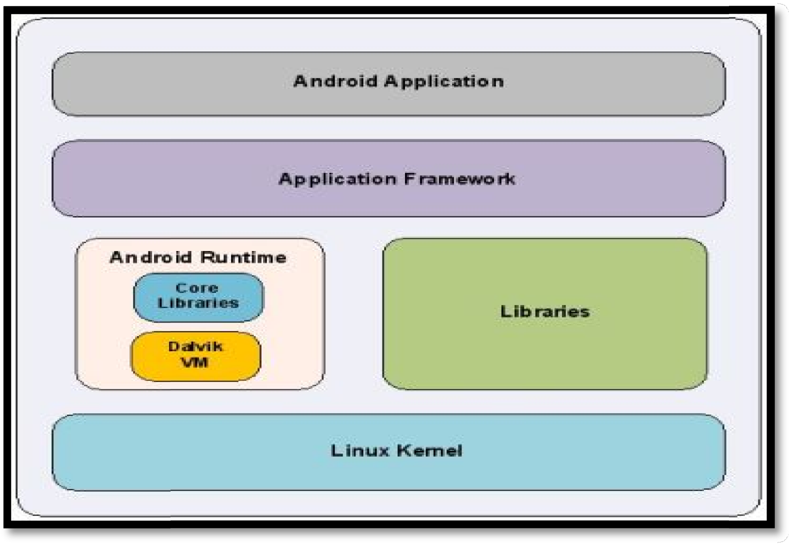


Figure 2-1Structure of an android system

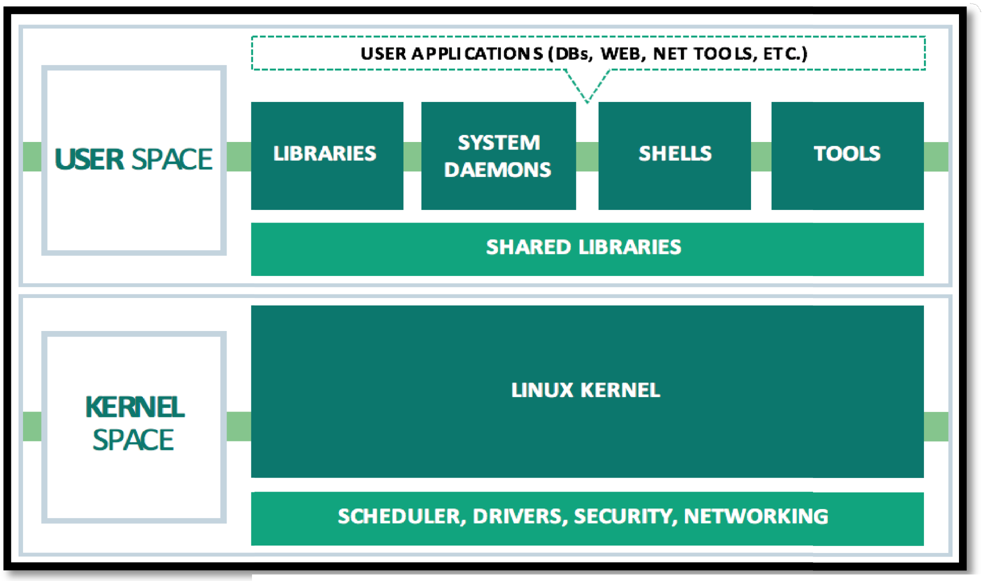

USER APPLICATIONS (DBs, WEB, NET TOOLS, ETC.)
USER SPACE
LIBRARIES
SYSTEM DAEMONS
SHELLS
TOOLS
SHARED LIBRARIES
KERNEL SPACE
LINUX KERNEL
SCHEDULER, DRIVERS, SECURITY, NETWORKING

Figure 2-2Structure of an embedded Linux system

2.3 Summary

Through a large enough literature survey which has conducted, it has identified that there are plenty of issues in embedded application development and in embedded systems domain. In recent years, the functions demanded for embedded systems have grown so numerous and complex that development time is increasingly difficult to predict and control. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically and agile implementations. They can change rapidly. For this reason, and also because of the hardware-manufacturing cycles are more expensive and time-consuming. They have been summarized as below.

Product Project	**Technologies used**	**Positive Features**	**Negative features**
A.Scholz[4]	Service Oriented Architecture middleware	Efficient in application execution	Not applied in data stream manageme nt system
Birrer, V. Cechticky, A. Pasetti, and O. Rohlik a tool called XWeaver	A new synchronization Mechanism, An optimized algorithm	reducing cost of embedded software development	Tested only for selected embedded systems.
DPWS toolkit implementation based on C and gSOAP[5]	Java service platform	offers plug-and-play as well as Quality-of-service QoS capabilities	semantics of a service are not clearly specified
functions demanded for embedded systems[6]	New platform to merge embedded systems	Increased flexibility	Hardware manufactu ring is more expensive
Daquan Feng, Chenzi Jiang, Gubong Lim [4]	a universal standard for embedded systems	Efficiency	Costly
Service oriented architecture based connectivity of automotive ECU	Ethernet SOME/IP	Higher bandwidth,scalability,re usability	Costly
Integrated development of embedded software	HDSP platform	Powerful than existing	Costly
A runtime resource architecture for embedded systems	JVM LINUX OS /API	Easy access, less maintenance cost	Implement ation is costly

Table 2.1 summary of identified issues

Chapter 3

3 A framework beyond android APIs

3.1 Introduction

Typical Android provides a rich application framework that allows a developer to build innovative apps and games for mobile devices in a Java language environment. Nevertheless it does not provide enhanced API s to access services of the android OS to be accessed through a network. E.g. through a WLAN. This research reveals that it's possible to provide service orientation with security and with platform independency. We adhere to java micro services to minimize the complexity of the application through modularity. This approach will support anyone who wants to develop a web based solution without hosting the application inside the android application. It simply requires http or https connectivity to access the device. That's not only but also the framework provides facility to override libraries or include additional API s to cater any specific requirement which has not covered in the scope. There the developer has to refer to the provided documentation and deploy a new micro service by including the given Core micro services of the framework.

It is evident from the literature that the embedded systems have moved from scientific and medical equipments to household appliances today [7]. They have been designed to run as single task environments with aimed to perform one operation at a time .Embedded systems which were available in medical equipments were not having capabilities like network communications [4]. Later on the embedded systems have innovated to perform multi tasking at run time [17]. Embedded symbian was one of the famous systems which have been developed by nokia with an aim of reaching higher number of customer base around the world [8]. In early days, development of embedded systems happened to be a programming exercise on C language oriented kernels [18]. Since then, the development of embedded systems, have gone through multiple phases [6].

3.2 What is a software framework

Software framework is software written to support developers. It is a skeleton, a complete set of tools that was built with the purpose of allowing developers to focus on one or more specific tasks. Developers can take that skeleton and build their application on top of it [16].

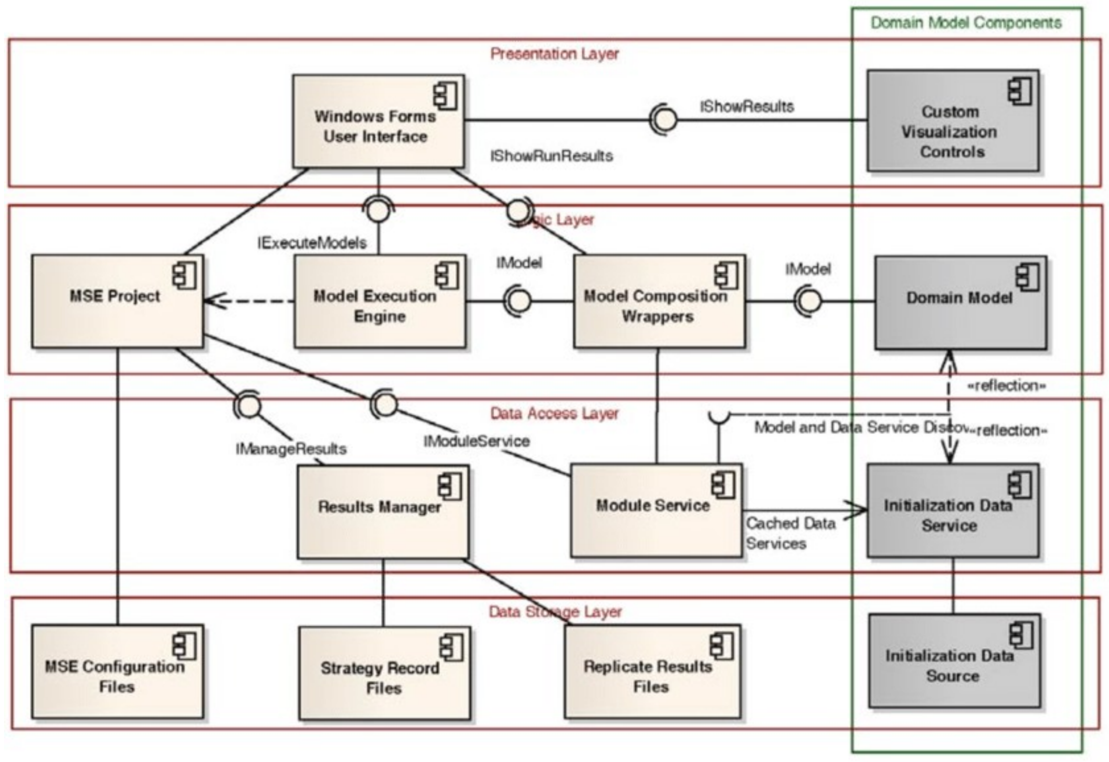


Figure 3-1 a sample diagram of a software framework

3.3 Service oriented architecture

Service-oriented architecture (SOA) is a software development model for distributed application components that incorporates discovery, access control, data mapping and security features [19, 20].

3.4 Micro services

Micro services refer to an architectural approach that independent teams use to prioritize the continuous delivery for single-purpose services. The micro services model is the opposite of traditional monolithic software which consists of tightly coupled modules that ship infrequently and have to scale as a single unit. Although the monolithic approach works fine for some organizations and some applications, micro services is becoming popular with companies that need greater agility and scalability [14].

Java

Java is a robust programming language which is capable of providing solid solutions in software development industry. The entire development was based on java version 1.8.0.

3.5 Summary

This chapter describes the required technologies to learn and implement the proposed framework. These technologies become tools to develop the entire solution.

Chapter 4

4 Service oriented software approach to embedded systems

4.1 Introduction

This chapter describes our approach to embedded systems development, EmSOS, with the use of service oriented software technologies. EmSOS is an acronym for **Em**bedded systems with **S**ervice **O**riented **S**oftware. We present our approach by highlighting the hypothesis, input, process, users, and features of the EmSOS.

4.2 Hypothesis

We hypothesize that issue of unavailability of a robust software framework for embedded system development can be overcome by introducing enhanced service oriented software architecture. This hypothesis been inspired by the power of service oriented architecture as a paradigm for development of software framework.

4.3 Input

Architectural design of the intended embedded system is the major input for the research. It defines modules and their connections. Also needs to provide JDK framework, operating system. The operating system has been limited to embedded android at the moment. Type of embedded devices targeted (mobile phones, printers, tablet PCs etc).

4.4 Output

Major output of the research is Enhanced Service Oriented framework .The framework supports the development of embedded system application for any embedded device that is running Linux and embedded android. The embedded system framework, EmSOS can generate a jar file to run on any ARM processor of an embedded device. User guide for EmSOS is also built in to the framework. There is also test bed to evaluate or validate an embedded system developed before deploying to the target embedded system device. In summary, output of EmSOS includes the following components.

- EmSOS framework
- Developer guide

4.5 Process

Development process of EmSOS is based on a Java development. This process will be converting the architectural design of the framework in to a code base which provides web services. This development consists of number of java micro services to manage as a combination of smaller services that works cohesively together for larger, application-wide functionality. Those micros services will be deployed in a third party server like Nanohttpd Embedded web server.

The architectural design for the proposed framework has designed by taking decisions on the results of technical analysis. There the existing frameworks and also the layers of embedded Linux have taken to the consideration. This analysis could identify that the kernel is the core component which we have to deal with instead of dealing with customized libraries like android run time. The analysis deeply went through the kernel components and system calls. Finally it has decided to develop a core module to deal with the system kernel to cater consumer requests and manipulate the embedded system logic. Although the outcome service component has included to the design to receive the consumer requests and convert them for core service. That is not only but also the outcome service is capable of providing a developer friendly output for every service call.

This has designed to provide a well structures JSON response with required headers.

Then the design has taken to the next step of selecting suitable technologies and preparing an implementation plan. The selected technologies were Java as the programming language, spring framework for build support and micro services for every component.

The implementation was a test driven development to make sure each unit has tested and released with minimum number of errors.

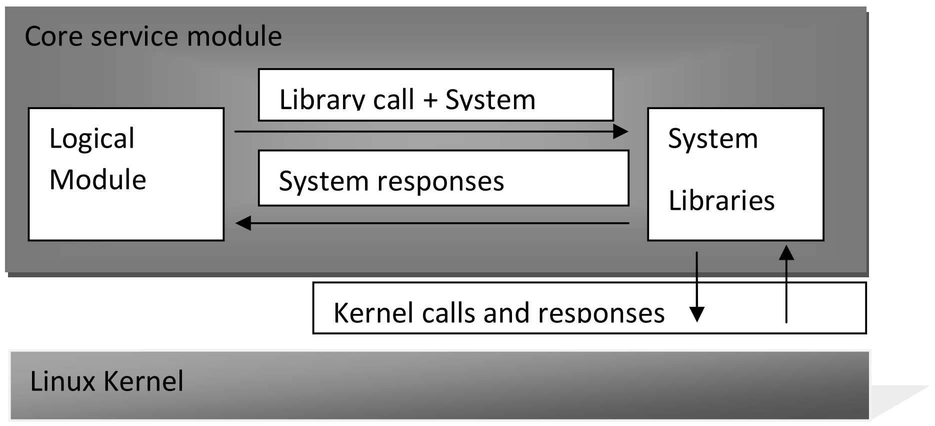


Figure 2.1 Finalized design of Core service module

4.6 Summary

This chapter includes the entire approach of the proposed framework to be completed at the implementation. It has defined that approach is continues process to input, process and output.

Chapter 5

5 Design and Implementation

5.1 Introduction

This chapter presents the design to develop EmSOS. For this purpose, we present high level design with the components of the EmSOS. The framework consist three major components, namely core service, outcomes service module and configuration and linking service module. In this chapter we describe role of each module, connection among there modules within the top level architecture of EmSOS. Among other modules, EmSOS API and Core service are specific in this design.

EmSOS top level architecture has been designed with three modules namely core service, outcomes service module and configuration and linking service module.

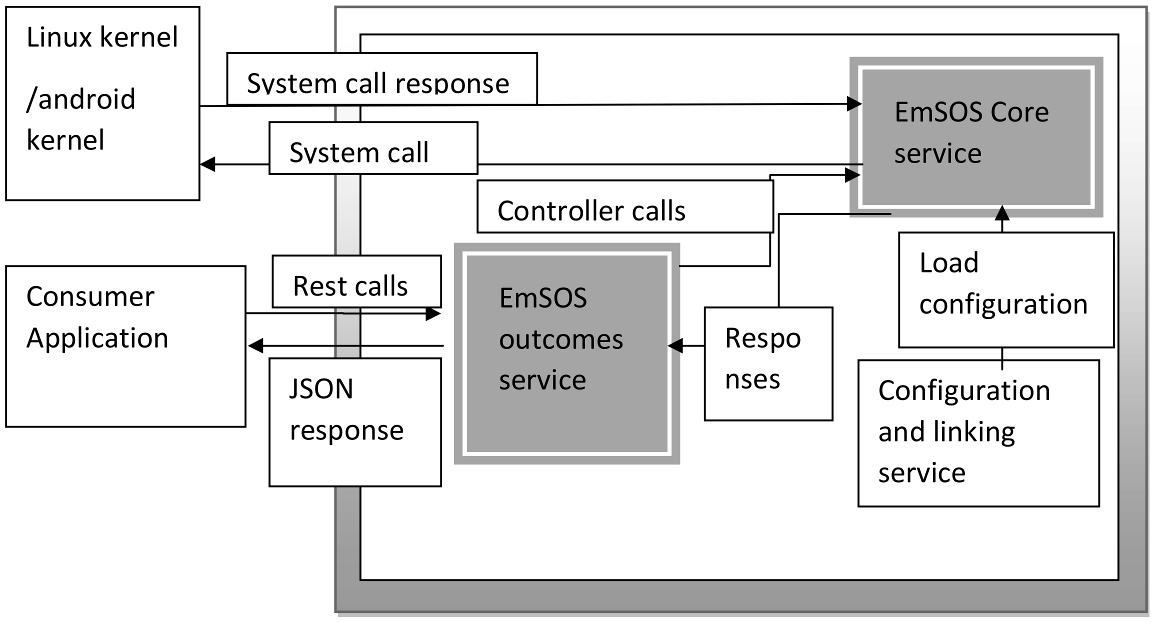


Figure 5-1-Top level architecture of EmSOS

5.2 Core service

Core service receives information and executes required system calls of operating system kernel. This service has designed to communicate with outcomes core service to receive client requests. These requests may include memory details, CPU, sensor data or any other resource related data for processing. Having received that information, the core service initializes the libraries that are required to connect with base operating system. Core service offers a set of services which are commonly used in major embedded software system calls. The output of the core services is guided by the different libraries and system logics. Design of the core service is having java interfaces that are having abstract methods for implementations of system logic. The interfaces are designed to implement classes for set of identified hardware and also for every operating system kernel that targets to be run the framework. This design is capable of including new interfaces for new hardware components that may arrive in future.

5.3 Outcomes service

This service has specifically designed to reduce the complexity that has to be managed by the embedded system developer in working with different system resources. Importance of the outcomes service is the simplicity of the request calls and output. Any programmer with little knowledge in embedded systems can proceed to develop the code by referring to the API documentation. This service has designed to provide simple REST calls with minimum number of headers and return JSON formatted text to the user. The output of the outcomes service would be a kind of template for programming for the intended embedded system application. Programmer can extract required data from JSON objects for processing. This service includes an authentication logic which has designed to provide security for service calls that are reaching the service from consumers.

This is a module consists of a service logic to be carried out for every REST service call. The module will process the request that has received from client and validate it at the beginning. Once it has validated, the requested service will be executed. There it refers to the above core service to communicate with operating system layer. Then the service logic will return the received outcome from core service to the user.

5.4 Configuration and linking service

This module consists of major configurations for EmSOS core service which needs to be load at the run time at the entire API. The structure of the configuration can be modified as it required to be loaded for new libraries that can add in future. Entire configuration is saved in config.xml file if the service.

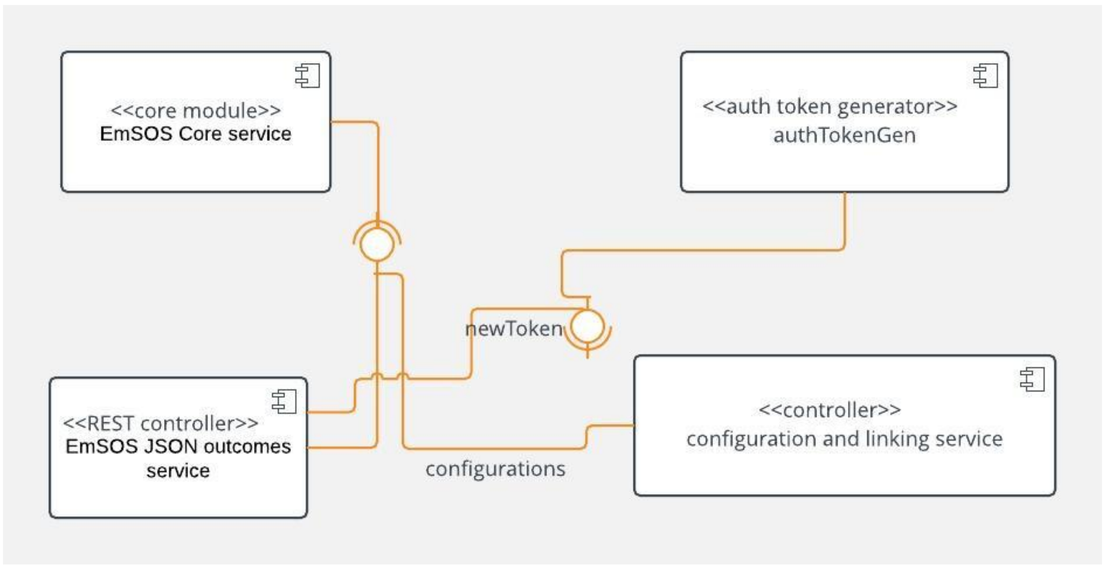


Figure 5-2Component diagram of EmSOS Software framework

5.5 Implementation

Implementation of EmSOShas based on the design explained in the previous chapter. The framework consists of major two modules as and one minor module. They are as per below

Major modules: Core service, JSON outcomes service

Minor module: Configuration and linking service

Implementation of the major modules have done as micro services which provide capability to include future requirements that might arrive based on the developers requirement. JSON outcomes micro service has referred to core micro service to access system logic and system calls of operating system kernel. The framework provides an API to developers. Every core module has implemented in java and JSON outcomes service module specially referring to the spring framework for java development. API has implemented to be published as .jar library which can be integrated to embedded systems development. The .jar files can be included in to an embedded project in IDE s like eclipse. The API runs as a web service which runs on a web application server.

5.6 Core service

The core service has implemented as below java packages to execute the system logic and communicate with libraries.

Package	**Activities**
emsos	Provides accessibility to OS information and Hardware profile **OS related :** OS type, Build version, **Hardware profile :** Manufacturer , model , description , version , serial number **Firmware Related :** manufacturer , description, version, name , release date
emsos.hd.hardware	Provide accessibility to CPU, Display, Memory Disks, NIC (interfaces), Sensors, Power and USB Devices
Emsos.com	Abstract layer to work with other classes
emsos.hd.plt.linux	Provide accessibility to power, memory and CPU which contains Linux systems. E.g android, ubuntu core
emsos.hd.plt.unix.solaris	Provide accessibility to power, memory and CPU which contains solaris
emsos.hd.plt.unix.windows	Provide accessibility to power, memory and CPU which contains windows. E.g : windows mobile
emsos.hd.plt.unix.freebsd	Provide accessibility to power, memory and CPU which contains BSD Unix
emsos.hd.plt.unix.macos	Provide accessibility to power, memory and CPU which contains macos
emsos.jna.*	Provides Java Native Access libraries
emsos.json.*	Provide JSON formatted text for REST calls to get above OS access
emsos.sw.os	Provide cross platform accessibility for retrieve process, file systems, OS
emsos.util	Access utilities for parsing and formatting
emsos.jna.plt.win.com	An special utility to access windows COM objects
emsos.util.plt.*	Provide utility access for different platforms.

Table 5.1 Packages of EmSOS

5.7 JSON outcomes service

This micro service has specifically implemented to receive service calls from consumer and return a json output. AbstractJsonObject class has implemented as the initial inheritance to child classes which contains service logic to format strings to json. HardwareInfoController class contains all of the rest controllers that are required to execute. This controller receives GET requests including a header which is user-agent. Based on the header value controller executes the required logic which is available in Core service module through an intermediate layer. This layer has a series of java classes to access different services as below.

emsos.json.util	Provide special utility methods for json
emsos.json	Json formatter package
emsos.json.hardware	Format hardware oriented outputs to json format
emsos.json.hardware.impl	Implementation package for hardware package
emsos.json.os	format operating system related output to json format
emsos.json.os.impl	Implementation package for OS package

Table 5.2 Java classes of services

5.8 Controller Logic

HardwareInfoController class contains four methods to cater the service logic of the API. There the method getOsInfoInJSON() method maps to the request that may arrive as web service calls. This method capable of accepting GET requests with get headers of User-agent, auth-key, req-type. User-agent header has defined to pass different parameters to access hardware and software resources of the embedded system. Auth-key parameter carries authentication key that is required to access API. User-agent parameter can access different logics which has implemented to access resources and process system calls. If the requirement is to access CPU resource details of the system, the consumer has to pass parameters as per below

Below is a sample request to get CPU relation data.

Request-type= GET

User-agent=cpu

Auth-key=abnbddjxxiuiuo%676598iuoi788787ssjhjhxxhjshkjhfe78erhjkxhjxhdjds (sample key of 256 character length)

Req-type=null

5.9 Configuration and linking service

This module is capable of the required configurations for development environment. This contains the module wise configurations to be loaded at the run time. The implementation of this module can be changes according to the future requirements of a developer. The current configurations are available in config.xml and assembly.xml files to be loaded at the runtime of the API. This module has the adoptability for future changes that can arise.

5.10 Summary

The design of the framework has done based on the analysis done to cater identified issues to be addressed in the proposed solution. The modularity has introduced to reduce the complexity and the micro services have included managing scalability of the system.

Chapter 6

6 Evaluation

6.1 Introduction

The chapter 6 discussed the detailed implementation of each module mentioned in the proposed solution. This chapter justifies the method of evaluation to entire EmSOS framework. The classification of testing has done in to four categories as developer testing, functionality testing, security testing and a design and a code review by industry experts.

6.2 Functionality testing

Functionality test includes selected use cases of API calls that has done as web service calls. The Postman tool has used to test service calls with selected parameters and expected results. One of the defined test cases is as per below.

Project name		EmSOS		
Test case		1		
Test Name		CPU resources		
Description		This test has conducted to retrieve CPU related information for application development.		
Input data	Expected Results	Actual Results		Notes
url: http://localhost:8080/outcomes-json/v1/ user-agent: cpu auth-key : 787aajdkjakjl7\7dfd\7\f @@ fdfd4fsfjksjkjds5459405	Detailed CPU information	"name": "Octa-core (4x2.0 GHz Cortex-A53 & 4x1.0 GHz Cortex-A53) ", "physicalProcessorCount": 4,		Result was providing developer data

09ueoi		"logicalProcessorCount": 4, "vendor": "GenuineIntel", "vendorFreq": 2900000000, "processorID": "BFEBFBFF000906E9", "identifier": "Intel64 Family 6 Model 158 Stepping 9", "cpu64bit": false, "family": "6", "model": "158", "stepping": "9", "systemCpuLoadBetweenTicks": 0.11179365849633661, "systemCpuLoadTicks": [250965906, 0, 185362429, 3478292781, 0, 936181, 530562, 0],		

Table 6.1 a functionality test case

Selected test cases are as per below.

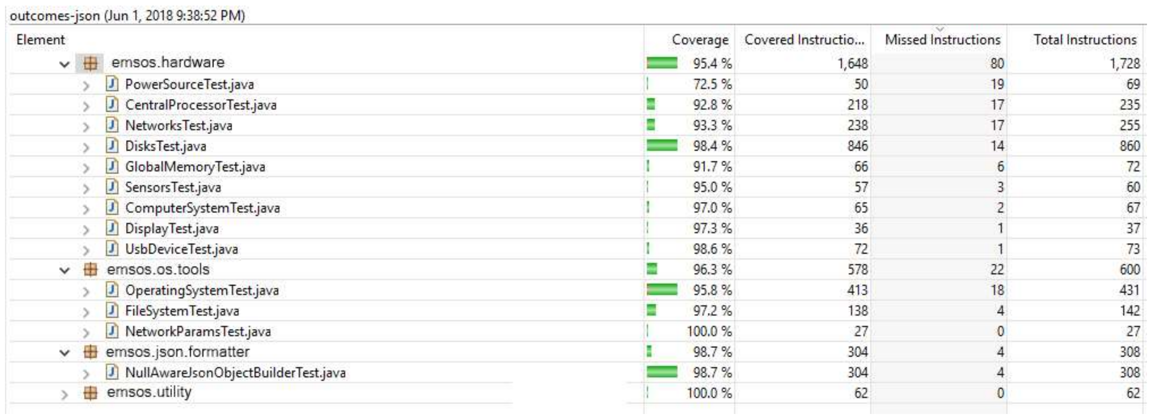
Test case name	**Functionality / Description**
CPU resources	To test CPU resources like cache, clock speed, cores test
System status	Entire embedded OS related parameter test
Running processes	Access running processes with process ID
Access power resources	Access Battery or other power related information test
Sensor	Access available sensor test
Display	Access display parameters test
Network parameters	Access network interfaces and related ports test
File system	Access file system test
Disk access	Access internal storage and resources
USB port	USB plugged devices.
Power sources	Access battery related information

Table 6.2 Entire set of functionality tests

6.3 Developer testing

Developer testing has done as a unit test and an integration test. Each unit has tested for expected outcome with inserted input values for parameters. There the method of each implementation class has been thoroughly tested. The referred unit tests have been done by jUnit testing framework.

Below are some of the jUnit tests that have written to test.



outcomes-json (Jun 1, 2018 9:38:52 PM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
emsos.hardware	95.4 %	1,648	80	1,728
PowerSourceTest.java	72.5 %	50	19	69
CentralProcessorTest.java	92.8 %	218	17	235
NetworksTest.java	93.3 %	238	17	255
DisksTest.java	98.4 %	846	14	860
GlobalMemoryTest.java	91.7 %	66	6	72
SensorsTest.java	95.0 %	57	3	60
ComputerSystemTest.java	97.0 %	65	2	67
DisplayTest.java	97.3 %	36	1	37
UsbDeviceTest.java	98.6 %	72	1	73
emsos.os.tools	96.3 %	578	22	600
OperatingSystemTest.java	95.8 %	413	18	431
FileSystemTest.java	97.2 %	138	4	142
NetworkParamsTest.java	100.0 %	27	0	27
emsos.json.formatter	98.7 %	304	4	308
NullAwareJsonObjectBuilderTest.java	98.7 %	304	4	308
emsos.utility	100.0 %	62	0	62

Figure 6-1 jUnit tests

Above unit tests are capable of testing the application logic of each and every method of implemented java classes. That's not only but also written jUnit tests, able to measure the coverage of the application logic of java classes. The coverage was as below.

Metric	**Indication**	**Reached Level (Result)**
Unit Test Coverage	For new projects : value should be greater than 82 % For already implemented codes (Legacy applications) : value should be greater than 60 %	92.3 %
Code complexity coverage	Depends on the nature of the application	67.25 %

Table 6.3 coverage

Detailed test results of Controller class

Package	**Coverage**	**Covered Instructions**	**Missed Instructions**	**Total Instructions**
emsos Controller	94.3%	492	63	555

Table 6.4 Controller test results

6.4 Security test

Entire code base has been tested for security issues to make sure that code is up to the standards. This was a security scan which is done by tool “sonar lint”. This tool aligns with OWASP standards for application security. The source code has continuously improved by referring to the results provided by the security scanning tool. Final results of the security scan are as below.

Security Analysis	Results
Coverage	87.50%
Line Coverage	87.20%
Condition Coverage	90.00%
Bugs	1.3 %

Table 6.5 Security scan results

6.5 Summary

Evaluation of the entire implementation against objectives has covered in this chapter. Each module has tested with development test, functionality test and a security scan.

Chapter 7

7 Conclusion and future work

7.1 Introduction

Previous chapter briefly explained the method of evaluation with performed tests. This chapter provides the conclusion to the entire research which has conducted in providing the enhanced service oriented framework for embedded android.

7.2 Limitations

This research does not cover the devices that are not capable of running embedded Linux or embedded android kernels. That's not only but also the framework doesn't support embedded devices that are not capable of running a web application server. Hence the web server is a major component in service orientation, it's mandatory to run a web server to deploy the EmSOS API to run. Also framework does not cover capabilities such as accessing UI components; accessing shell commands in Linux systems and any capability which doesn't aligning with addressed problem.

7.3 Future developments

Future developments can be implemented to provide solutions to other problems that has identified in literature survey. The research can be further develop to track application development life cycle management and also for resource efficient data processing. It is identified that embedded networks are having issues in error detection and recovery in small network communications. Therefore the research can be improved to cover those embedded networks related issues. Today frameworks are having capabilities of providing automation support for installation and configuration in applications. This also becomes a great feature to be implemented to minimize deployment effort of embedded applications.

7.4 Summary

This chapter concludes the thesis by describing the solution which has given to the identified problem of issues in platform independency , reduce complexity of the developer, reduce the learning curve of a novel embedded system to the developer and also sort out the scalability issues due to resource constrains. Therefore the outcome API and documentation of the entire research becomes an enhanced framework.

Reference

[1] Cheju halla University, S. H. Moon, C. sick Lee, and Cheju halla University, “Dynamic Management Software Design in Embedded System using Middle,” 2014, pp. 186–191.

[2] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski, “Service-oriented architectures for embedded systems using devices profile for web services,” in *Advanced Information Networking and Applications Workshops, 2007, AINAW’07. 21st International Conference on*, 2007, vol. 1, pp. 956–963.

[3] A. Scholz*et al.*, “ SOA-Service Oriented Architectures adapted for embedded networks,” in *2009 7th IEEE International Conference on Industrial Informatics*, 2009, pp. 599–605.

[4] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, “A network-centric approach to embedded software for tiny devices,” in *International Workshop on Embedded Software*, 2001, pp. 114–130.

[5] W.-T. Tsai, X. Sun, and J. Balasooriya, “Service-Oriented Cloud Computing Architecture,” 2010, pp. 684–689.

[6] G. L. Gopu, K. V. Kavitha, and J. Joy, “Service Oriented Architecture based connectivity of automotive ECUs,” in *Circuit, Power and Computing Technologies (ICCPCT), 2016 International Conference on*, 2016, pp. 1–4.

[7] D. Guinard, V. Trifa, and E. Wilde, “A resource oriented architecture for the web of things,” in *Internet of Things (IOT), 2010*, 2010, pp. 1–8.

[8] X. Cai, G. Ouyang, and X. Zhang, “Design of Fan Performance Detection System Based on ARM Embedded System,” *Int. J. Smart Home*, vol. 8, no. 1, pp. 311–316, Jan. 2014.

[9] S. H. Moon, “Designing and Embodiment of Software that Creates Middle Ware for Resource Management in Embedded System,” *Int. J. Softw. Eng. Its Appl.*, p. 12, 2014.

[10] S. Arulselvi, “Advanced Internet Access System Using Embedded Linux,” p. 4, 2014.

[11] P. Pannuto *et al.*, “A networked embedded system platform for the post-mote era,” 2014, pp. 354–355.

[12] S. Thilagavathi, J. Sathyapriya, T. M. Minipriya, T. Mohanapriya, and C. Subashini, “Modern Coding Theory Based On Fountain Code Implementation In Embedded System,” vol. 3, no. 2, p. 4, 2014.

[13] R. K. Tiwari and S. K. Agrahari, “Arduino Compatible World Wide Web Controlled Embedded System,” vol. 3, no. 9, p. 4, 2014.

[14] Cheju halla University, S. H. Moon, C. sick Lee, and Cheju halla University, “Dynamic Management Software Design in Embedded System using Middle,” 2014, pp. 186–191.

[15] PG. Student of, Sinhagad college of engineering,Pune, P. Kumbhar, and D. S. . Lokhande, “Embedded Based Compact Fuzzy System to Speed Control of Single Phase Induction Motor,” *IOSR J. Electr. Electron. Eng.*, vol. 9, no. 5, pp. 56–59, 2014.

[16] S. Salgaonkar, M. J. D. Bhosale, and M. P. Bangde, “Embedded Web Server based on ARM Cortex for DAC System,” vol. 5, no. 7, p. 6, 2014.

[17] J. K. Arthur, T. Robinson, and R. Latha, “Implementation aspects of Bio-Metric system in Electronic Voting Machine by using embedded security and big data approach,” vol. 1, no. 3, p. 6.
[18] P. G. Pachpande, “Internet Based Embedded Data Acquisition System,” vol. 5, p. 4, 2014.
[19] S. V. Shinde and M. P. Zaware, “Wi-Fi based Monitoring and Controlling of Embedded System,” vol. 5, p. 7, 2014.
[20] “International Journal of Combined Research & Development (IJCRD) eISSN:2321-225X; pISSN:2321-2241 Volume: 2; Issue: 5; May -2014,” vol. 2, no. 5, p. 4.

Appendix A -Acronyms

Endpoint - This is an URL where a service can be accessed by a client application

Peripheral device -This is generally defined as any auxiliary device such as a computer mouse or keyboard that connects to the computer

EmSOS - Enhanced **S**ervice **O**riented **S**oftware Framework for **Em**bedded Android

API - Application Programming Interface

jUnit – A unit test framework for java programming language

OWASP- Open Web Application Security Project

IDE – Integrated Development Environment

REST – This is an architectural style for designing distributed systems

Appendix B -Test results and source codes

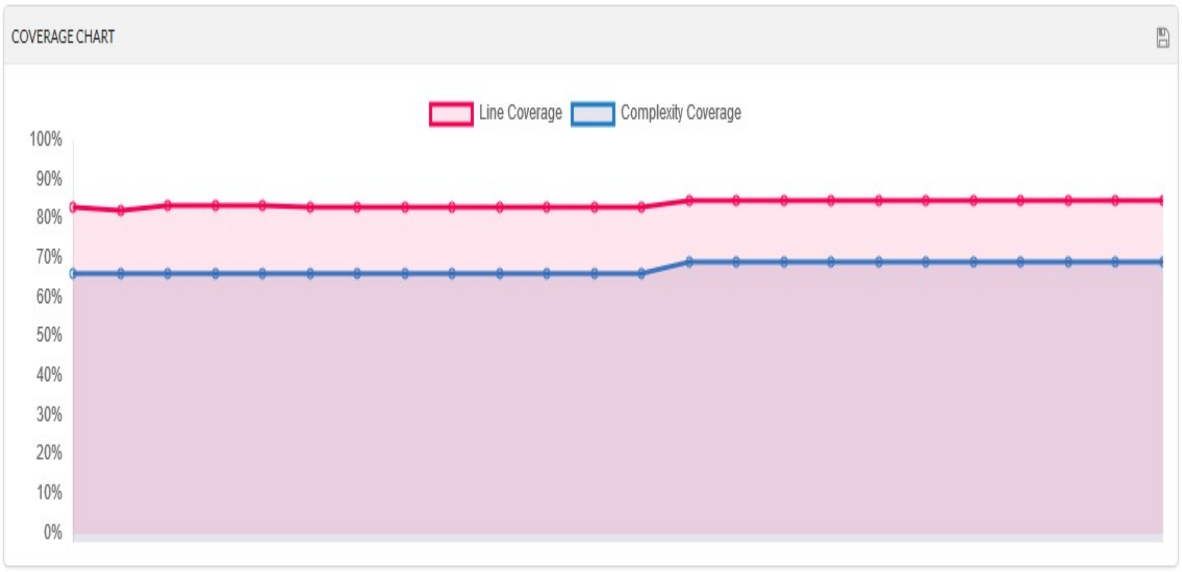
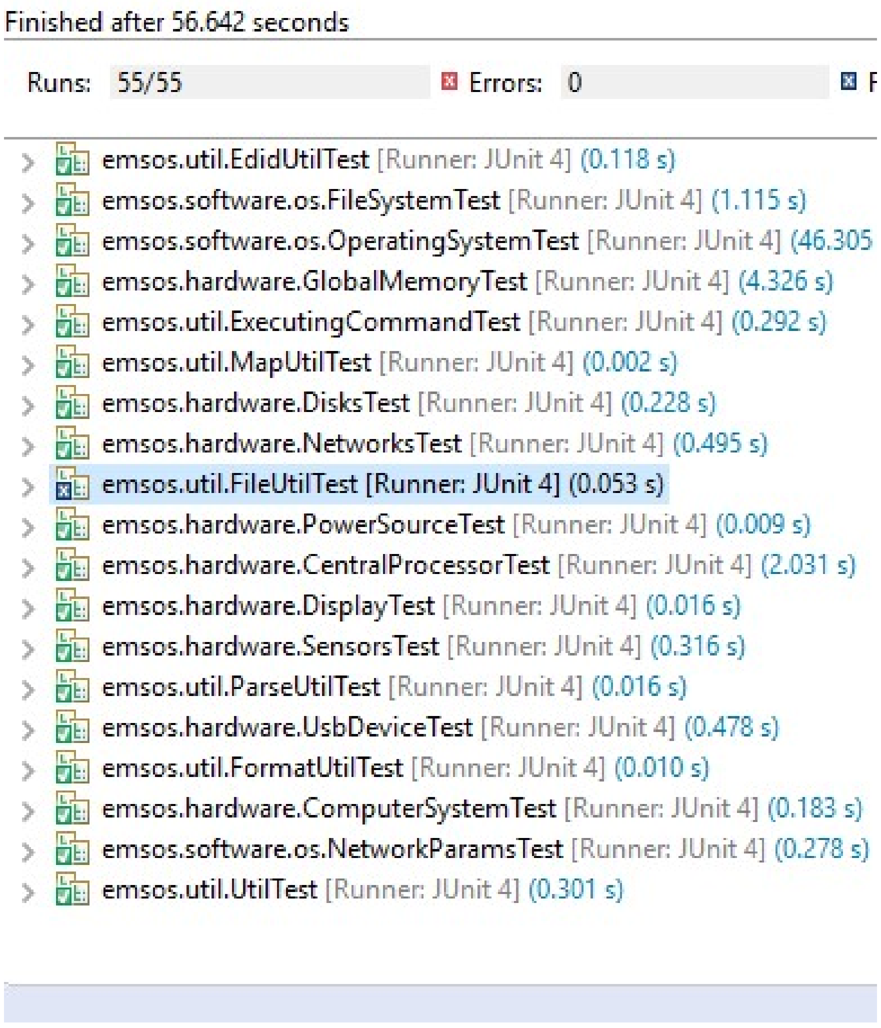


Figure 4 Coverage chart with the development commits which has done with version controlling tool. (Git)

```
publicclassSysInfTest {
     */
publicstaticvoid main(String[] args) {

System.setProperty(org.slf4j.impl.SimpleLogger.DEFAULT_LOG_LEVEL_KEY,
"INFO");
System.setProperty(org.slf4j.impl.SimpleLogger.LOG_FILE_KEY,
"System.err");
        Logger LOG = LoggerFactory.getLogger(SystemInfoTest.class);

LOG.info("Initializing System...");
SystemInfosi = newSystemInfo();

HardwareAbstractionLayerhal = si.getHardware();
OperatingSystemos = si.getOperatingSystem();

System.out.println(os);

LOG.info("Checking computer system...");
printComputerSystem(hal.getComputerSystem());

LOG.info("Checking Processor...");
printProcessor(hal.getProcessor());

LOG.info("Checking Memory...");
printMemory(hal.getMemory());

LOG.info("Checking CPU...");
printCpu(hal.getProcessor());

LOG.info("Checking Processes...");
printProcesses(os, hal.getMemory());

LOG.info("Checking Sensors...");
printSensors(hal.getSensors());

LOG.info("Checking Power sources...");
printPowerSources(hal.getPowerSources());

LOG.info("Checking Disks...");
printDisks(hal.getDiskStores());

LOG.info("Checking File System...");
printFileSystem(os.getFileSystem());

LOG.info("Checking Network interfaces...");
printNetworkInterfaces(hal.getNetworkIFs());
```

Figure 9 Test class of System test in test driven development



```
Finished after 56.642 seconds
Runs: 55/55     Errors: 0     F
> emsos.util.EdidUtilTest [Runner: JUnit 4] (0.118 s)
> emsos.software.os.FileSystemTest [Runner: JUnit 4] (1.115 s)
> emsos.software.os.OperatingSystemTest [Runner: JUnit 4] (46.305
> emsos.hardware.GlobalMemoryTest [Runner: JUnit 4] (4.326 s)
> emsos.util.ExecutingCommandTest [Runner: JUnit 4] (0.292 s)
> emsos.util.MapUtilTest [Runner: JUnit 4] (0.002 s)
> emsos.hardware.DisksTest [Runner: JUnit 4] (0.228 s)
> emsos.hardware.NetworksTest [Runner: JUnit 4] (0.495 s)
> emsos.util.FileUtilTest [Runner: JUnit 4] (0.053 s)
> emsos.hardware.PowerSourceTest [Runner: JUnit 4] (0.009 s)
> emsos.hardware.CentralProcessorTest [Runner: JUnit 4] (2.031 s)
> emsos.hardware.DisplayTest [Runner: JUnit 4] (0.016 s)
> emsos.hardware.SensorsTest [Runner: JUnit 4] (0.316 s)
> emsos.util.ParseUtilTest [Runner: JUnit 4] (0.016 s)
> emsos.hardware.UsbDeviceTest [Runner: JUnit 4] (0.478 s)
> emsos.util.FormatUtilTest [Runner: JUnit 4] (0.010 s)
> emsos.hardware.ComputerSystemTest [Runner: JUnit 4] (0.183 s)
> emsos.software.os.NetworkParamsTest [Runner: JUnit 4] (0.278 s)
> emsos.util.UtilTest [Runner: JUnit 4] (0.301 s)
```

Figure 6 jUnit test results

```
importemsos.json.SystemInfo;
importemsos.json.hardware.CentralProcessor;
importemsos.json.hardware.Display;
importemsos.json.hardware.HardwareAbstractionLayer;
importemsos.json.software.os.NetworkParams;
importemsos.json.software.os.OperatingSystem;
importoutcomes.auth.AuthTokenGenerator;
importemsos.json.hardware.impl.DisplayImpl;
importjava.util.Arrays;

importjavax.json.JsonString;

importorg.springframework.web.bind.annotation.RequestHeader;
importorg.springframework.web.bind.annotation.RequestMapping;
importorg.springframework.web.bind.annotation.RequestMethod;
importorg.springframework.web.bind.annotation.ResponseBody;
importorg.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("v1")
publicclassHardwareInfoController  {
      publicstaticvoid main(String args[]) {
            SystemInfosi = newSystemInfo();
            HardwareAbstractionLayerhal = si.getHardware();
            OperatingSystemos = si.getOperatingSystem();

            System.out.println(os.toJSON());
            System.out.println(hal.toJSON());
      }
```

Figure 11 HardwareController class

```
                System.out.println(os.toJSON());
                System.out.println(hal.toJSON());
        }

public String authToken() {
        returnAuthTokenGenerator.nextToken();
    }
        // Operating System + Hardware related information
        @RequestMapping(headers = "User-Agent", method = RequestMethod.GET
, produces = "application/json")
        public@ResponseBody String getOsInfoInJSON(@RequestHeader("User-
Agent") String userAgent) {
System.out.println(this.authToken());
                SystemInfosi = newSystemInfo();
                OperatingSystemos = si.getOperatingSystem();
                HardwareAbstractionLayerhal = si.getHardware();
                if (userAgent.equals("")) {
                        returnos.toPrettyJSON();
                } elseif  (userAgent.equals("hw")){
                        returnhal.toPrettyJSON();
                }elseif (userAgent.equals("cpu")) {
                        returnhal.getProcessor().toPrettyJSON();

                }elseif(userAgent.equals("cpucount")) {
                        CentralProcessorprocessor=hal.getProcessor();
                        return ("{ \"No of
cpu\":"+processor.getPhysicalProcessorCount()+"}");
                }elseif (userAgent.equals("d")) {
                        returnhal.getComputerSystem().toPrettyJSON();
                }elseif (userAgent.equals("disk store")){
                        returnhal.getDiskStores()+"";
                }elseif (userAgent.equals("display")){
                        returnhal.getDisplays()+"";
                }elseif (userAgent.equals("memory")) {
                        returnhal.getMemory().toPrettyJSON();
                }elseif (userAgent.equals("nic")) {
                        returnprintNetworkParameters(os.getNetworkParams());
                }elseif (userAgent.equals("power")) {
                        returnhal.getPowerSources()+"";
                }elseif (userAgent.equals("sensors")) {
                        returnhal.getSensors().toPrettyJSON();
                }elseif (userAgent.equals("usb")) {
                        returnhal.getUsbDevices(true)+"";
                }elseif (userAgent.equals("l")) {
                        returnhal.getUsbDevices(true)+"";
                }elseif (userAgent.equals("m")) {
                        returnhal.getUsbDevices(true)+"";
```

Figure 12 Configuration and linking settings

```
<?xmlversion="1.0"encoding="UTF-8"?>
<assembly
       xmlns="http://maven.apache.org/plugins/maven-assembly-
plugin/assembly/1.1.3"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-
plugin/assembly/1.1.3 http://maven.apache.org/xsd/assembly-1.1.3.xsd">
       <id>distribution</id>
       <formats>
              <format>tar.gz</format>
              <format>tar.bz2</format>
              <format>zip</format>
       </formats>
       <!-- Root of archive has files -->
       <includeBaseDirectory>false</includeBaseDirectory>
       <!-- Put all dependency jars (except modules) in /lib -->
       <dependencySets>
              <dependencySet>
                     <useProjectArtifact>false</useProjectArtifact>

       <useTransitiveDependencies>true</useTransitiveDependencies>
                     <outputDirectory>lib</outputDirectory>
                     <unpack>false</unpack>
                     <excludes>
                            <exclude>${project.groupId}:*:*</exclude>
                     </excludes>
              </dependencySet>
       </dependencySets>
              <files>


       </files>
       <!-- Include module jars in root level -->
       <moduleSets>
              <moduleSet>
                     <!-- Enable access to all projects in the current
multimodule build! -->

                     </binaries>
              </moduleSet>
       </moduleSets>
</assembly>
```

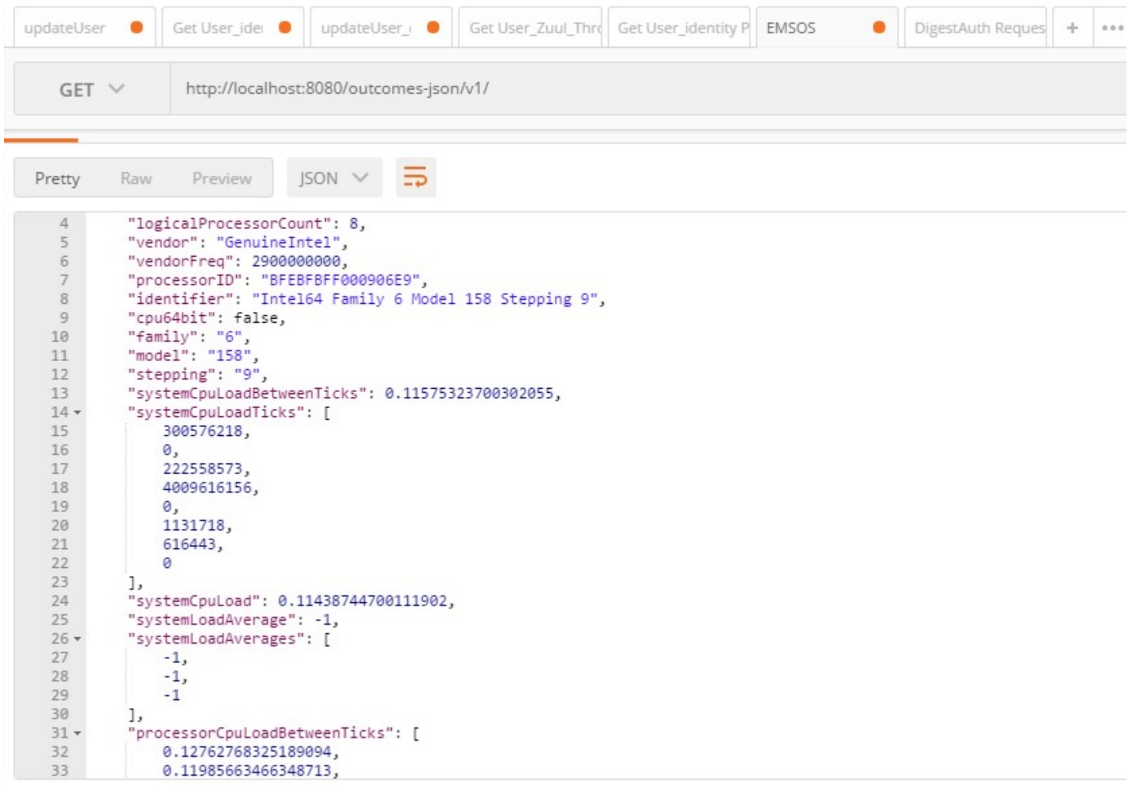
Figure 9 Configuration and linking settings

```
<useAllReactorProjects>true</useAllReactorProjects>
                   <!-- Now, select which projects to include in this
module-set. -->
                   <includes>
                         <include>com.github.emsos:emsos-core</include>
                         <include>com.github.emsos:emsos-json</include>
                   </includes>
                   <binaries>

      <includeDependencies>false</includeDependencies>
                         <outputDirectory>/</outputDirectory>
                         <unpack>false</unpack>
                   </binaries>
             </moduleSet>
      </moduleSets>
</assembly>
```

Security Scan results of code base

Matrix	**Reached Level**
Unit Test Coverage	92.30%
Security Analysis	
Coverage	87.50%
Line Coverage	87.20%
Condition Coverage	90.00%
Bugs	0
Complexity Function	1.2

Table 6 Security Scan results of code base



```
http://localhost:8080/outcomes-json/v1/
```

```
"logicalProcessorCount": 8,
"vendor": "GenuineIntel",
"vendorFreq": 2900000000,
"processorID": "BFEBFBFF000906E9",
"identifier": "Intel64 Family 6 Model 158 Stepping 9",
"cpu64bit": false,
"family": "6",
"model": "158",
"stepping": "9",
"systemCpuLoadBetweenTicks": 0.11575323700302055,
"systemCpuLoadTicks": [
    300576218,
    0,
    222558573,
    4009616156,
    0,
    1131718,
    616443,
    0
],
"systemCpuLoad": 0.11438744700111902,
"systemLoadAverage": -1,
"systemLoadAverages": [
    -1,
    -1,
    -1
],
"processorCpuLoadBetweenTicks": [
    0.12762768325189094,
    0.11985663466348713,
```

Figure 10 a sample JSON result of API

Appendix C –API documentation

Package	**Description**
emsos	Provides accessibility to OS information and Hardware profile **OS related :** OS type, Build version, **Hardware profile :** Manufacturer , model , description , version , serial number **Firmware Related :** manufacturer , description, version, name , release date
emsos.hd.hardware	Provide accessibility to CPU, Display, Memory Disks, NIC (interfaces), Sensors, Power and USB Devices
Emsos.com	Abstract layer to work with other classes
emsos.hd.plt.linux	Provide accessibility to power, memory and CPU which contains Linux systems. E.g. android, ubuntu core
emsos.hd.plt.unix.solaris	Provide accessibility to power, memory and CPU which contains solaris
emsos.hd.plt.unix.windows	Provide accessibility to power, memory and CPU which contains windows. E.g. windows mobile
emsos.hd.plt.unix.freebsd	Provide accessibility to power, memory and CPU which contains BSD Unix
emsos.hd.plt.unix.macos	Provide accessibility to power, memory and CPU which contains macos
emsos.jna.*	Provides Java Native Access libraries
emsos.json.*	Provide JSON formatted text for REST calls to get above OS access
emsos.sw.os	Provide cross platform accessibility for retrieve process, file systems, OS
emsos.util	Access utilities for parsing and formatting
emsos.json.util	Provide special utility methods for JSON
emsos.jna.plt.win.com	An special utility to access windows COM objects
emsos.util.plt.*	Provide utility access for different platforms.

Table 7 API Documentation

REST endpoints

```
GET <host hostip>/cpu
GET <host ip>/v1/

Header parameters : User-agent : (cpu/ram/file/,ded,ems,hw,fm)

GET <host ip>/gpu
GET <host ip>/mem
POST <host ip>/niclist
POST <host ip>/nicaccess/{id}
GET <host ip>/sensors
GET <host ip>/cpus
GET <host ip>/vcpus

GET <host ip>/staccess
GET <host ip>/staccess/{name}
GET <host ip>/systemaccess


GET <host ip>/mbd

GET <host ip>/pwrsources (get power resources)
GET <host ip>/PRO/      (retrieve processes that are currently running)
GET <host ip>/PID/{pid}      (retrieve PID related details)
GET <host ip>/PROALLOCATE/    (allocating processors)
GET <host ip>/cpulgcore    (get cpu logical cores)
```