TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

BALAJI GOPALAKRISHNAN

A METHODOLOGY FOR THE DEVELOPMENT OF MANUFACTURING AND MONITORING INDEXES: OIL LUBRICATION SYSTEM CASE STUDY

Master of Science thesis

Examiners: Prof. Jose L. Martinez Lastra and
Dr. Andrei Lobov
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on 4th Nov 2015

ABSTRACT

BALAJI GOPALAKRISHNAN: A METHODOLOGY FOR THE DEVELOPMENT OF MANUFACTURING AND MONITORING INDEXES: OIL LUBRICATION SYSTEM CASE STUDY

Tampere University of technology
Master of Science Thesis, 84 pages, 46 Appendix pages
August 2017
Master's Degree Programme in Automation Engineering
Major: Factory Automation and Industrial Informatics
Examiners: Prof. Jose L. Martinez Lastra and Dr. Andrei Lobov
Supervisor: Dr. Andrei Lobov

Keywords: Manufacturing Execution System, Open Knowledge-Driven system, ANSI/ANSI/ISA-95, Key Performance Indicator, Service Oriented Architecture, Ontology, RESTful web services.

The arrival of computers into manufacturing brought a huge revolution in all parts of the manufacturing industries, from the shop-floor level to the enterprise level. Initially it all started at the shop-floor level with the need for highly accurate and flexible systems and then later it moved to the management level. Once the systems in these two levels have been organized, there came MES and monitoring system to increase efficiency and streamline the manufacturing process. Now there are various researches being made to organize these systems, standardize it and develop a suitable software system, which can not only be configurable, but also be flexible and be adapted to any type of manufacturing industry.

This thesis deals with one such methodology, which helps in obtaining indexes (KPI) and functionalities with respect to monitoring and manufacturing, via a knowledge driven framework developed based on the methodology. The methodology revolves around the concept that a framework driven by knowledge, which when implemented with Open Knowledge-Driven Manufacturing Execution System (OKD-MES) concept, will be suitable for obtaining both manufacturing and monitoring system functionalities. Ontology plays a crucial part in the methodology, as it holds the knowledge about functions to be used and the configurations to be made as part of providing the functionalities to the functions. These initial configurations and the functionalities in ontology will be provided by the user and can be later modified by the user as per the manufacturing requirements. Hence system developed with this methodology will have knowledge driven functions that are configurable and extendable to the maximum extent. The methodology is verified with implementation in two use cases: one is an Oil Lubrication System, in which the monitoring functionality of the framework is tested; and the second is a discrete manufacturing industry which manufactures Oil Lubrication System, where the MES functionality is tested.

Preface

"*Learning gives creativity,*
creativity leads to thinking,
thinking provides knowledge,
knowledge makes you great" – Dr. A. P. J. Abdul Kalam

The thesis work is an accomplishment of my one such learning and the completion of a small era in my life. It comes from a time which is so far, the best part of my life.

First and foremost, I would like to thank my parents for providing me with immense love and support to achieve my dream. I also owe a special thanks to my brother Manoj Kumar, who has been helping and guiding me from the day I came to this earth. Without you brother this would have not been possible.

I would like to extend my gratitude towards Prof. Jose L. Martinez Lastra for giving me the opportunity to work in FAST lab. FAST lab provided me with a wonderfull platform to develop my thesis work.

I would also like to thank my supervisor Dr. Andrei Lobov who have provided me with priceless support to complete this thesis. Thanks for listening to all my ideas, being patient and mentoring me towards right direction.

Next, a special thanks to Sergii who has been a mentor and a teacher whom I can approach at any time and get things done properly. I would also like to thank Otto, Roberto and Manu from Fluidhouse for their advice and support.

Finally, thanks to all my friends and FAST colegues who were supportive all the way through. Special thanks to Borja, Luis, Johanna, Wael, Ananda, Amir, Vivek and Naveen.

Balaji Gopalakrishnan
Tampere, August 2017

CONTENTS

ABSTRACT I
PREFACE II
LIST OF FIGURES VI
LIST OF TABLES VIII
LIST OF SYMBOLS AND ABBREVIATIONS IX
1. INTRODUCTION 1
1.1 Background 1
1.2 Problem Definition 2
1.2.1 Justification for work 3
1.2.2 Problem Statement 4
1.3 Work Description 5
1.3.1 Objectives 5
1.3.2 Hypothesis 5
1.3.3 Assumptions and Limitations 5
1.4 Thesis outline 6
2. STATE OF THE ART 7
2.1 Manufacturing Execution System Standards 7
2.1.1 ANSI/ISA-95 7
2.1.2 MESA 10
2.2 MES in Process and Discrete Industry 13
2.2.1 Process Industry MES 13
2.2.2 Discrete Industry MES 14
2.3 Monitoring Methods 14
2.3.1 Process Monitoring 15
2.3.2 Quality Monitoring 15
2.3.3 Performance Monitoring 15
2.3.4 Condition Monitoring 15
2.4 Key Performance Indicators 16
2.4.1 KPI in Manufacturing 16
2.4.2 KPI in Monitoring 17
2.5 Open Knowledge-Driven System 17
2.5.1 Physical Layer 18
2.5.2 Representation Layer 18
2.5.3 Orchestration Layer 18
2.5.4 Visualization Layer 19
2.6 SOA and Web Services 19
2.6.1 Arbitrary WS 19
2.6.2 REST WS 20
2.7 Knowledge-Driven Approach 20
2.7.1 Knowledge Representation 20
2.7.2 OWL 21

2.8 Information Technology in Manufacturing 21
2.8.1 Web Application: 21
2.8.2 JSON Message format 22
2.9 Summary 23
3. METHODOLOGY 24
3.1 Overview 24
3.2 Model 25
3.2.1 Fuseki Server 26
3.2.2 Framework Module 33
3.3 Technique 38
3.3.1 User Interaction 38
3.3.2 Initial Configuration 39
3.3.3 Framework Rule Triggering 43
3.3.4 Framework Rule Execution 44
3.4 Tools 47
3.4.1 OWL development tool 47
3.4.2 Framework development tools 47
4. USE CASE DEFINITION 49
4.1 Oil Lubrication System Simulator 49
4.2 Oil Lubrication System Production Industry 51
5. IMPLEMENTATION 53
5.1 ESCOP and Framework Interactions 53
5.2 Use Case 1: Oil Lubrication System Simulator 55
5.2.1 Data Acquisition 55
5.2.2 Condition Monitoring 56
5.2.3 Quality Monitoring 58
5.2.4 Process Monitoring 59
5.2.5 Immediate Maintenance 60
5.2.6 System Controller 60
5.2.7 Predictive Maintenance 61
5.2.8 HMI Coordinator 62
5.2.9 Key Performance Indicators 62
5.3 Use Case 2: Oil Lubrication System Production Industry 62
5.3.1 Resource Allocation and Status 62
5.3.2 Data Collection and Acquisition 63
5.3.3 Labour Management 63
5.3.4 Product Tracking and Genealogy 64
5.3.5 Key Performance Indicators 67
6. RESULTS 68
6.1 OLS Simulator 68
6.1.1 Condition Monitoring 68
6.1.2 Quality Monitoring 69

6.1.3 Process Monitoring 71
6.1.4 Immediate Maintenance 72
6.1.5 Predictive Maintenance 73
6.2 OLS Production Industry 74
6.2.1 Resource Tracking and Genealogy 74
6.2.2 Labour Management 76
6.2.3 Product Tracking and Genealogy 77
7. CONCLUSIONS AND FUTURE WORK 79
7.1 Implementation and Result Conclusion 79
7.2 Lesson Learned 80
7.3 Future Work 80
REFERENCES 81
APPENDIX A – MMO (CONFIGURATION ONTOLOGY) 85
APPENDIX B – MONITORING FUNCTIONS 88
APPENDIX C – MANUFACTURING FUNCTIONS 110

List of Figures

Figure 1: Data flow in Enterprise [5] 2
Figure 2 ANSI/ISA-95 Functional Hierarchy model[24] 8
Figure 3 Generic Activity Model of Manufacturing Operation Management[24] 8
Figure 4: ERP and MES in Functional Hierarchy Model[23] 9
Figure 5 Functional Enterprise Control model[22] 10
Figure 6 MES Evolution Process[26] 10
Figure 7 MES Functionality Model[10] 11
Figure 8 eScop OKD-MES Architecture[52] 17
Figure 9 REST Architecture[56] 20
Figure 10 Array format in JSON[62] 22
Figure 11 Object format in JSON[62] 23
Figure 12 Block diagram for system integration 24
Figure 13 Manufacturing / Monitoring Framework 25
Figure 14 Manufacturing and Monitoring Ontology 26
Figure 15 Data Ontology 32
Figure 16 Framework Initial Configurator 33
Figure 17 Framework Registry 34
Figure 18 Framework Rule Processor 36
Figure 19 Framework Ontology Administrator 37
Figure 20 Framework RTU 37
Figure 21 User Interaction with Framework 39
Figure 22 Framework configuration (1) 40
Figure 23 Framework Configuration (2) 41
Figure 24 Framework Configuration (3) 42
Figure 25 Event Based Triggering 43
Figure 26 Service Invocation Based Triggering 44
Figure 27 Event Generation 45
Figure 28 Service Invocation 46
Figure 29 Save to Ontology 46
Figure 30 Olingvo tool 47
Figure 31 Oil Lubrication System Simulator 49
Figure 32 OLS Simulator Monitoring Screen 50
Figure 33 OLS Simulator HMI 50
Figure 34 OLS Production Industry Architecture 51
Figure 35 Swagger REST Translator Interface 52
Figure 36 RPL Discovery Operation 53
Figure 37 Meta key value pairs in PHL events 54
Figure 38 RPL Meta search 54
Figure 39 Data Acquisition Sequence 56
Figure 40 Condition Monitoring function sequence 57

Figure 41 Quality Monitoring function sequence 58
Figure 42 Process Monitoring function sequence 59
Figure 43 Immediate maintenance function sequence 60
Figure 44 Predictive Maintenance function sequence 61
Figure 45 Resource Allocation and Status sequence 63
Figure 46 Labour Management Function sequence 64
Figure 47 Product Tracking and Genealogy Function Sequence (All sub project) 65
Figure 48 Product Tracking and Genealogy Function Sequence (specific sub project) 66
Figure 49 Simulator simulation time modifying controls 68
Figure 50 HMI (Condition Monitoring Scenario) 69
Figure 51 Oil Quality Decay Rate value 70
Figure 52 Particle Count and Filter Capacity Values 70
Figure 53 HMI (Quality Monitoring Scenario) 70
Figure 54Oil Consumption Ratio of Flow meter 71
Figure 55 Tank Level value in monitoring Screen 71
Figure 56 HMI (Process Monitoring Scenario) 72
Figure 57 Postman leakage service invocation 72
Figure 58 HMI (Immediate Maintenance scenario) 73
Figure 59 HMI (Predictive Maintenance scenario) 73
Figure 60 Resource allocation screen in Production Organizer 74
Figure 61 RAS function access icon 74
Figure 62 Resource suggestion displayed in Production Organizer 75
Figure 63 RAS service request via Postman tool 75
Figure 64 Time Tracker App RFID scanning screen 76
Figure 65 Time Tracker current sub projects display screen 76
Figure 66 Time Tracker completed sub projects display screen 77
Figure 67 RFID scanning screen (Product Tracker App) 77
Figure 68 Project Info screen (Product Tracker App) 78
Figure 69 Resource Details analysis screen (Product Tracker App) 78
Figure 70 Project Details Analysis screen (Product Tracker App) 78

List of TABLES

Table 1: MES Functions from MESA[27] *11*
Table 2 Ontology Device Elements *27*
Table 3 Ontology Function Elements *27*
Table 4 Ontology Rule Elements *28*
Table 5 Ontology Input Elements *29*
Table 6 Output Value Format *29*
Table 7 Ontology Output Elements *30*
Table 8 Ontology Meta Elements *30*
Table 9 Ontology Service Elements *30*
Table 10 OLS Simulator configuration specifications *55*
Table 11 Ontology Input Individuals (OLS Simulator Monitoring) *88*
Table 12 Ontology Output Individuals (OLS Simulator Monitoring) *93*
Table 13 Ontology Meta Individuals (OLS Simulator Monitoring) *93*
Table 14 Ontology Service Individuals (OLS Simulator Monitoring) *95*
Table 15 Ontology Rule Individuals for Condition Monitoring *95*
Table 16 Ontology Rule Individuals for Data Acquisition *96*
Table 17 Ontology Rule Individuals for HMI Coordinator *98*
Table 18 Ontology Rule Individuals for Immediate Maintenance *98*
Table 19 Ontology Rule Individuals for Predictive Maintenance *100*
Table 20 Ontology Rule Individuals for Process Monitoring *104*
Table 21 Ontology Rule Individuals for Quality Monitoring *106*
Table 22 Ontology Rule Individuals for System Controller *108*
Table 23 Ontology Function Individuals (OLS Simulator Ontology) *108*
Table 24 Ontology Input Individuals (OLS Production MES) *110*
Table 25 Ontology Output Individuals (OLS Production MES) *112*
Table 26 Ontology Meta Individuals (OLS Production MES) *112*
Table 27 Ontology Rule Individuals for Resource Allocation and Status *113*
Table 28 Ontology Rule Individuals for Data Collection and Acquisition *116*
Table 29 Ontology Rule Individuals for Labour Management *117*
Table 30 Ontology Rule Individuals for Product Tracking And Genealogy *122*
Table 31Ontology Function Individuals (OLS Production MES) *130*

List of Symbols and abbreviations

ANSI	American National Standards Institute
BOM	Bill Of Materials
DCS	Distributed Control System
EPS	Evolvable Production System
ERP	Enterprise Resource Planning
eScop	Embedded system Service-based Control for Open manufacturing and Process automation
INT	Interface Layer
ISA	International Society of Automation
JSON	JavaScript Object Notation
LIMS	Laboratory Information Management System
MES	Manufacturing Execution System
MESA	Manufacturing Execution System Association
MRP	Material Requirements Planning
MRP II	Manufacturing Resource Planning
OKD	Open Knowledge-Driven
OKD-MES	Open Knowledge-Driven Manufacturing Execution System
ORL	Orchestartion Layer
OWL	Web Ontology Language
PHL	Physical Layer
PLC	Programmable Logic Controller
REST	Representational State Transfer
RPL	Representation Layer
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
VIS	Visualization Layer
WSDL	Web Service Description Language

1. INTRODUCTION

1.1 Background

From the last century manufacturing industries, have seen an enormous growth. It all started with the development in the shop-floor level. Initially there were just mechanical machines then there came relays to control those machines. Later in 1968, the PLC (Programmable Logic Controller) [1] came to play in-order to overthrow the complex relay based machine control systems. Again, the problem with PLC's was that there was large amount of data and the components were distributed. Then in 1980s, DCS (Distributed Control System) emerged and helped to solve the complex problems of distributed control and handling large amount of data. On the other hand, the enterprise levelwas also improving simultaneously with the evolution of computer systems. In 1965, a team led by IBM developed the Production Information and Control System (PICS) which is considered as mother of all MRP (Material Requirements Planning)[2]. In the 1980's, MRP evolved into MRP II (Manufacturing Resource Planning). The inclusion of Engineering, Finance, Human Resources, and Project Management into MRP II led to ERP (Enterprise Resource Planning). The term ERP was coined in 1990's and it was the one of the systems in the management level, which has been so successful till now. ERP consists of those systems that provide management of finance, order, production, material and related functions [3]. Currently the manufacturing industries ERP system focuses on global planning, business processes and its executions across the whole enterprise (intra-enterprise systems), with an accrued importance to supply chain planning and management. While at the shop-floor, control systems have become hybrid hardware/software such as DCS, SCADA, and other systems designed to automate the way in which the product is being manufactured [4].

Now the challenge was to link these integrated software systems from the management and the shop-floor level [5]. In order to link these layers, most of the manufacturers started using some of manufacturing systems like PAC (Production Automation Control) [6], PRISM (Productivity Improvement Systems for Manufacturing) [7] and SCORE [8]. Later in 1990 [9], the term Manufacturing Execution System (MES) was first used to refer to all the manufacturing systems which communicate between the two layers. The emergence of MES (Manufacturing Execution Systems) provided some major functions to integrate the shop-floor with the management layer. According to the Manufacturing Enterprise Solutions Association (MESA),

"*Manufacturing Execution Systems (MES) deliver information that enables the optimization of production activities from order launch to finished goods. Using current and*

accurate data, MES guides, initiates, responds to, and reports on plant activities as they occur. The resulting rapid response to changing conditions, coupled with a focus on reducing non value-added activities, drives effective plant operations and processes." [10]

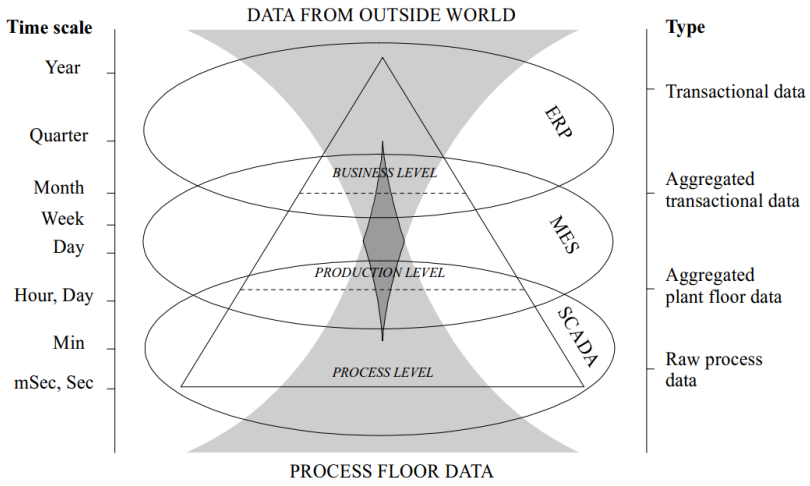

Figure 1: Data flow in Enterprise [5]

MES which provides link between planning and control systems also combines both global and local rules into a plant wide view of not only what is happening but also what should be happening to meet the objectives [10]. Currently, there are software like Simantic IT, Factory Talk, PAS-X MES, Shop-floor online MES and much more based on the above standards, which are used in discrete and process industries.

On the other dimension, due to the manufacturing industries evolution to adapt flexible and intelligent machinery, the need for monitoring them also increased. The need paved the way for monitoring systems. Though there were SCADA based systems in the shop-floor, whose major function is data acquisition and remote monitoring [11]; there was not enough monitoring being done at this end. SCADA was lacking aspects like predictive monitoring, preventive monitoring and overall process monitoring [12]. Hence, there was a necessity for external monitoring systems. The monitoring system should be capable of monitoring both machinery and the complete process. In some cases, as explained in the article [13], monitoring systems are also used in monitoring industrial IT environment.

1.2 Problem Definition

Manufacturing industries must have qualities like rapid and effective decision making, response to customer needs, real time monitoring, high quality products and services. To stay competitive and be sustainable, manufacturing sectors must deal with complex scenarios. But usually the production systems are built based on the known functionalities and predicted operational scenarios. Hence for the manufacturing industries to be innovative and efficient, it needs to be flexible and adoptable.

The MES which acts as a bridge between plan floor and control floor is the most prominent layer which can be modelled / developed to accomplish the goals of reusability, configurability and flexibility in a manufacturing environment. On the other hand, monitoring systems also helps the manufacturing industries to become efficient and competitive by taking care of proper functioning of machinery and process. Hence MES and monitoring system are some of the main components in an industry which must be improvised to achieve innovation and efficiency in manufacturing.

MES takes care of the production management and the monitoring systems take care of the device monitoring in most of the cases. The integration of both is a much necessary thing to the manufacturing industry. Also, the integration will result in an efficient system which will have many benefits as explained in the article [14]. Hence, a software system with the below features became essential.

1. A single solution for both manufacturing and monitoring needs.
2. The system must be configurable, extendable and it must can adapt to any type industry with less amount of work.

1.2.1 Justification for work

The MES should collect a large number of real-time data in the production process; process the real-time events quickly; as well as maintain the two-way communication capabilities between the enterprise level and the shop-floor (control) level. The enterprise level mostly reacts to the market and sets the production goal/plan accordingly, while the control level responds to the goal set by the enterprise level. To be efficient and compete, manufacturing industry has to adapt itself to the market's ups and downs for which, the control layer may be upgraded or degraded based on the demand [15]. Hence MES must react to the changes in the shop-floor, process the information based on the new changes and update it to the ERP. Also at some points the MES must itself be upgraded or degraded depending on the requirements or changes in the other two layers. For these reasons, the MES itself has to be flexible, reusable, extendable and adaptable.

The monitoring systems are used to monitor different aspects in the manufacturing industry like condition and performance of manufacturing systems; quality of product; and manufacturing process. Either directly or indirectly, all the monitoring methods are connected to the control level. Hence, any change in the control level will also affect the monitoring system. So, in order to achieve a flexible manufacturing environment, the monitoring systems must also be made flexible and configurable.

With the development of IT in the manufacturing industries, Service Oriented Architecture (SOA), which was originally created for the high-level communications, has now started to find its ways through industrial sector. This architecture provides a paradigm that allows system integration through publish, search and invoke based services.

Today with the capability of web services available from the shop-floor level, the web-service based SOA has become a reality in the manufacturing industries [16]. Ontology, one of the important technologies to solve the shared knowledge understandings, has already been used in many areas of the manufacturing industry to make knowledge communication and management easier[17]. There has also been numerous research towards ontology based MES [18], [19] and monitoring systems [20]. Hence, web service based SOA and ontology when collectively implemented in manufacturing industry, it paves the way for a knowledge-based web services[21]. Thus, a solution designed with the knowledge-based web services can potentially be flexible, reusable, extendable and adaptable.

There numerous concepts currently in research with respect to knowledge-driven manufacturing. One such research is the OKD-MES (Open Knowledge-Driven Manufacturing Execution System) concept developed as part of eScop project. OKD-MES combines the modularity of the system with the knowledge driven approach in a manufacturing industry. A knowledge-driven framework with web services when implemented as part of the OKD-MES will be the most suitable one to derive the flexible and configurable systems in the manufacturing industry.

1.2.2 Problem Statement

MES and the monitoring system need to be flexible and configurable in order to respond to flexible manufacturing environment. If the system needs to be configurable, then the components and the logics through which the system is built must be global, separated and configured as required. The globalization can be achieved through following global standards for communication like web services, which can be easily used by devices and software systems in the factory. Then the components (I/O, logics, etc.) separation can be performed by having them in ontologies and configuring as required. On the whole, having a knowledge driven web service based application will satisfy all the needs discussed in section 1.2.1. Also, this system can be made to perform as both MES and monitoring system. Though it looks simple, there are many questions to be resolved:

Can the suggested methodology be developed as a framework? Can the developed framework be used as a MES? Can it be used as a Monitoring system? Can it be also used as a hybrid system? Can it be used to compute manufacturing and monitoring KPI's? What are the requirements for the framework to work as a MES or as a Monitoring system? Can it be implemented with less or minimal configurations in any type of industry as MES or in any type of Machine for monitoring?

1.3 Work Description

This part outlines the objectives of this work, the methodology that is used, the assumptions made for evaluating the objectives and the limitations with respect to methodology and the implementations.

1.3.1 Objectives

The main thesis Objectives are,

- Outline the methodology for developing a knowledge-based framework with web services, which can be used as an MES and monitoring medium.
- Exploring the capability and functionality of the developed framework after its integration with Knowledge Driven Systems.
- Identifying key MES functions among the available function in MESA for discrete industry which manufactures oil lubrication System.
- Developing rules to monitor the manufactured Oil Lubrication System while at operation.
- Analysing the Key Performance Indicators associated with manufacturing and monitoring Oil Lubrication System.
- Investigating the developed MES function and Monitoring rules by incorporating them in developed framework and testing it in Oil Lubrication System use case.

1.3.2 Hypothesis

The major hypothesis of the thesis are,

H1: A system having its command in knowledge, which when integrated with OKD (Open Knowledge-Driven) elements, can be used to develope and customize software components.

H2: A framework developed based on the hypothesis (H1) can attain the state of being a configurable, flexible and adaptable tool for developing industrial softwares.

H3: If the famework is formulated with standards like MES or monitoring, it can be built to perform as a MES or Monitoring system or both, which will also pave the way for obtaining indexes (KPI) with respect to monitoring or manufacturing.

1.3.3 Assumptions and Limitations

The current study applies for the design of a methodology for manufacturing and monitoring purpose; and to develop a knowledge-based framework built from the defined

methodology. The Framework, when used as a MES medium, can support plug and produce capability existing in the shop-floor level of the manufacturing industries and also be used in any type of process and discrete industries with simple integration methods. On the other hand, when used as a monitoring medium it can help in proper functioning and long life of the monitored machine. The following assumptions and limitations were taken into account while making this Thesis work:

- There are not many industrial shop-floor devices especially controllers that can provide RESTful web services. Hence it is assumed that the shop-floor devices and the other software in the use case are assumed to be RESTful web service based devices.
- In the Oil Lubrication System manufacturing use case, the framework will be used as a MES medium just to support the already existing manufacturing tools in the ways of suggesting project schedules, analysing labour performance and production status.
- In the Oil Lubrication System monitoring use case, the framework will be tested on Oil Lubrication System, which will be a stand-alone one and simulating the use in a pulp and paper machine.
- The framework is configurable and easy to implement; but it is not easy to setup since the use of ontology to define the MES and monitoring functionality.
- The formulas used in the framework can only be coded in JavaScript or Java or MVEL language.

1.4 Thesis outline

The thesis work is organized as follows: Chapter 2 is a review of the previous work done in the very field and cites related methodologies. Chapter 3 presents the methodology of work. Chapter 4 presents the Use case is presented with the required details and in Chapter 5 the actual implementation and control procedures of the concept with respect to use case are presented. Chapter 6 shows the result and analysis for the implementation. Chapter 7 presents conclusion and outlines future work.

2. STATE OF THE ART

2.1 Manufacturing Execution System Standards

2.1.1 ANSI/ISA-95

ISA, International Society of Automation is an organization for setting standards and educating industry professionals in automation. ANSI/ISA-95 is one such standard developed by ISA to integrate enterprise layer with the shop-floor layer and also to standardize the information flow between them. In short, ANSI/ISA-95 is not an automation system but a method, a way of working, thinking and communicating inside a manufacturing industry[22]. ANSI/ISA-95 is originally a U.S standard, later it was adopted as an industrial standard under IEC/ISO 62246. The ANSI/ISA-95 is divided into 5 parts[23]:

- ANSI/ISA 95.00.01 “Part 1: Models and Terminology” (International Standard ISO/IEC 62264-1)
- ANSI/ISA 95.00.02 “Part 2: Object Attributes” (International Standard ISO/IEC 62264-1)
- ANSI/ISA 95.00.03 “Part 3: Activity Models of Manufacturing Operations Management”
- ANSI/ISA 95.00.04 “Part 4: Object Models and Attributes of Manufacturing Operations Management”
- ANSI/ISA 95.00.05 “Part 5: Business to Manufacturing Transactions”.

In part 1, ANSI/ISA-95 presents various models and terminologies which can be used in preparing and executing the automation of information exchange between ERP and MES[22]. There are many models in part 1 like the functional hierarchy model, the equipment hierarchy model, the Functional Enterprise Control Model, object model and the categories of Information Exchange model. The most important and the most suitable ones which deal with MES are the functional Hierarchy model and the Functional Enterprise Control Model. ANSI/ISA-95 defines a functional hierarchy model in which it consists of 5 levels. Each level provides specialized functions and has characteristic response times as shown in Figure 2.

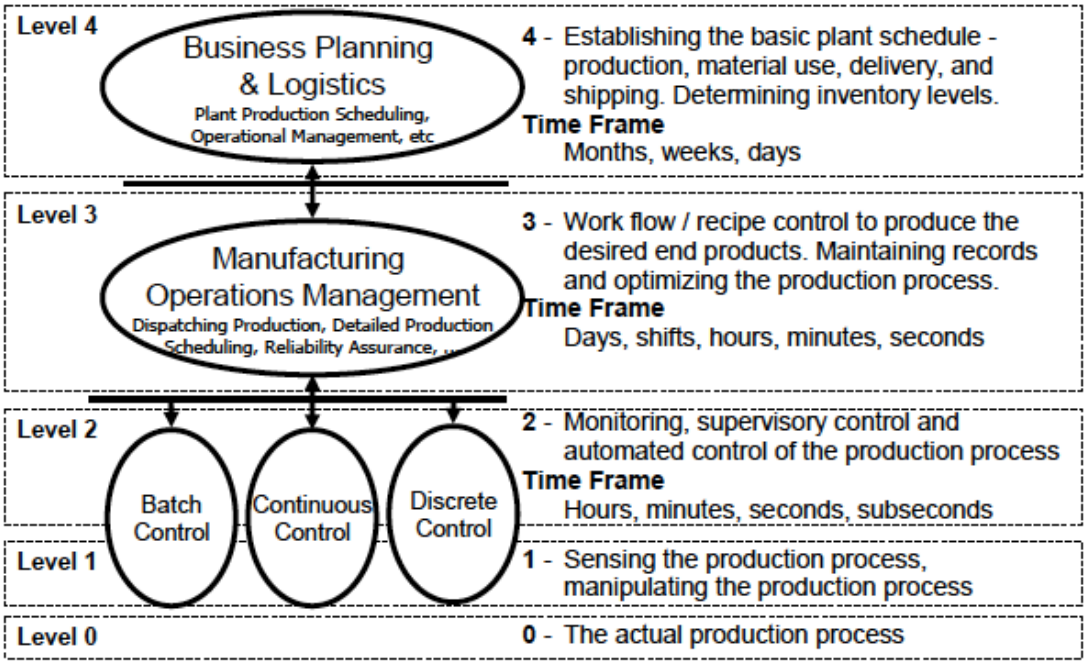

Figure 2 ANSI/ISA-95 Functional Hierarchy model[24]

The Functional Enterprise model shown in Figure 4 symbolizes the twelve functions carried out in a manufacturing industry. It has a dotted line in between the functions which represents the boundary of the enterprise control interface. This model from ANSI/ISA-95 is one of the features which can be closely related to the MES functions from MESA.

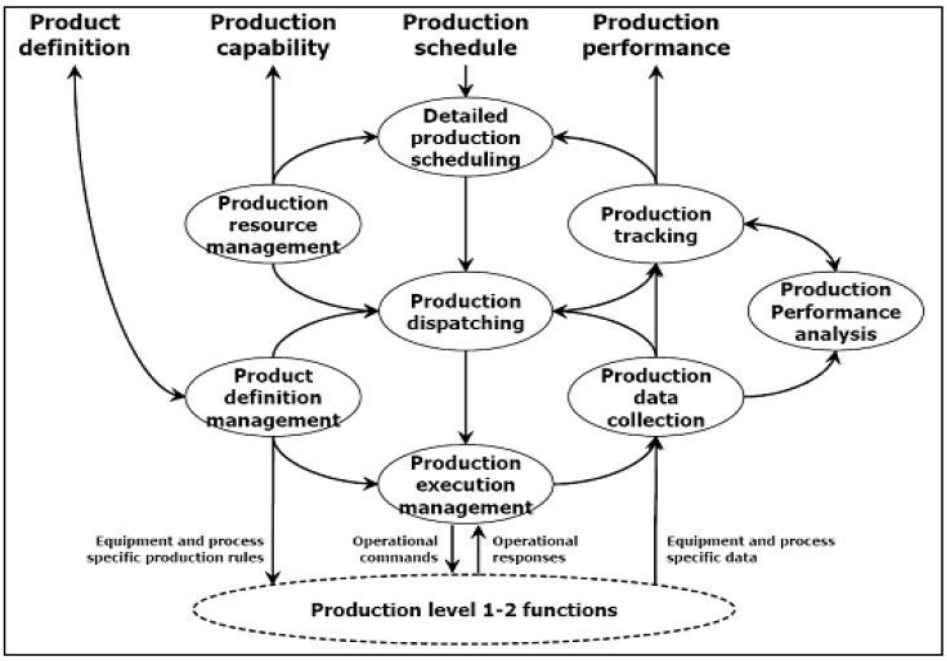

Figure 3 Generic Activity Model of Manufacturing Operation Management[24]

Further in part 3: Activity Models of Manufacturing Operations Management there is Generic Activity Model of Manufacturing Operation Management shown in Figure 3, which extends the 4 sub models (Production Operation, Maintenance operation, Quality Test Operations and Inventory Operations) and also it explains the connection between each sub modules. As said earlier though the term MES and ERP is not mentioned in any part of the ANSI/ISA-95, it does explain the difference between the two in the Functional Hierarchy Model in Figure 2. It distinguishes the Hierarchy into two different domains Enterprise Domain (Level 4) and Control Domain (level 3 and lower), where the Enterprise Domain is the stage where ERP is present and the Control Domain is a combination of MES (Level 3) , PCS layer (level 2 and 1), and level 0 represents the process itself[22] shown in Figure 4.

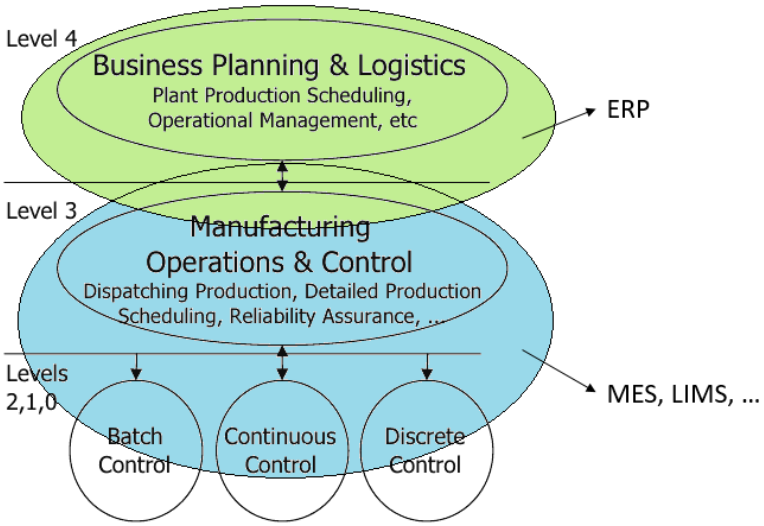

Figure 4: ERP and MES in Functional Hierarchy Model[23]

The Functional Enterprise Control model has 12 functions which are said to be carried out in a manufacturing industry. The model in Figure 5 Figure 5 Functional Enterprise Control model[22]represents the standards from ANSI/ISA-95 that are closely related to the 11 MES functions from MESA and how there should be a proper communication between shop floor and ERP, by in turn having a proper information flow between them. Moreover, ANSI/ISA 95 standardizes the information to be exchanged which has led to ERP and MES software having an ISA-95 interface definition.

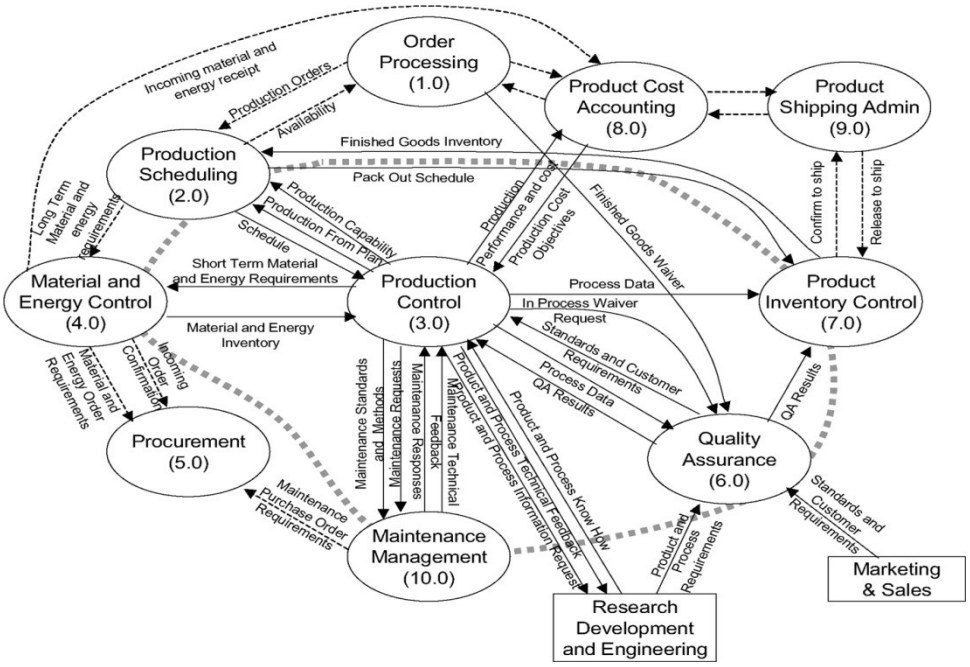

Figure 5 Functional Enterprise Control model[22]

2.1.2 MESA

Manufacturing Enterprise Solutions Association (MESA) International is a worldwide not-for-profit organization to improve business results and production operations through optimized application and implementation of information technology and best management practices[25]. There are large numbers of definitions available for MES but the one given by the MESA is mostly stated everywhere. In MESA, the MES found its evolution from Traditional MES to next generation collaborative MES as shown in Figure 6.

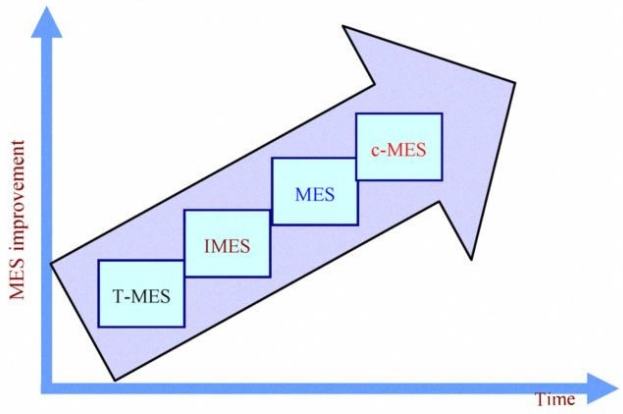

Figure 6 MES Evolution Process[26]

Unlike the previous MES from MESA, the collaborative MES, introduced in was first introduced in 2004, focuses on how core operations activities interact with business operations in a model. The MES functional Model from the next generation collaborative MES is shown in Figure 7 and the respective 11 functions explained in Table 1 as per the definitions from Freedom Technologies[27].

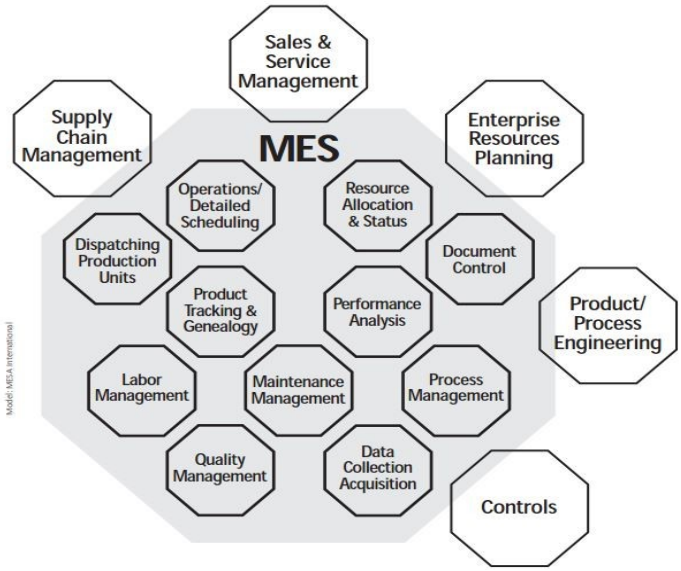

Figure 7 MES Functionality Model[10]

Table 1: MES Functions from MESA[27]

Function	Description
Resource Allocation and Status	Manages resources including machines, tools labour skills, materials, other equipment, and other entities such as documents that must be available in order for work to start at the operation. It provides detailed history of resources and insures that equipment is properly set up for processing and provides status real time. The management of these resources includes reservation and dispatching to meet operation scheduling objectives.
Operations/ Detail Scheduling	Provides sequencing based on priorities, attributes, characteristics, and/or recipes associated with specific production units at an operation such as shape of colour sequencing or other characteristics which, when scheduled in sequence properly, minimize setup. It is finite and it recognizes alternative and overlapping/ parallel operations in order to calculate in detail exact time or equipment loading and adjust to shift patterns.
Dispatching Production Units	Manages flow of production units in the form of jobs, orders, batches, lots, and work orders. Dispatch information is presented in sequence in which the work needs to be done and changes in real time as events occur on the factory floor. It has the ability to alter prescribed schedule on the factory floor. Rework and salvage processes are available, as well as the ability to control the amount of work in process at any point with buffer management.

Function	Description
Document Control	Controls records/forms that must be maintained with the production unit, including work instructions, recipes, drawings, standard operation procedures, part programs, batch records, engineering change notices, shift-to-shift communication, as well as the ability to edit "as planned" and "as built" information. It sends instructions down to the operations, including providing data to operators or recipes to device controls. It would also include the control and integrity of environmental, health and safety regulations, and ISO information such as Corrective Action procedures. Storage of historical data.
Data Collection/ Acquisition	This function provides an interface link to obtain the intra-operational production and parametric data which populate the forms and records which were attached to the production unit. The data may be collected from the factory floor either manually or automatically from equipment in an up-to-the-minute time frame.
Labour Management	Provides status of personnel in and up-to-the-minute time frame. Includes time and attendance reporting, certification tracking, as well as the ability to track indirect activities such as material preparation or tool room work as a basis for activity based costing. It may interact with resource allocation to determine optimal assignments.
Quality Management	Provides real time analysis of measurements collected from manufacturing to assure proper product quality control and to identify problems requiring attention. It may recommend action to correct the problem, including correlating the symptom, actions and results to determine the cause. May include SPC/SQC tracking and management of off-line inspection operations and analysis in laboratory information management system (LIMS) could also be included.
Process Management	Monitors production and either automatically corrects or provides decision support to operators for correcting and improving in-process activities. These activities may be intra-operational and focus specifically on machines or equipment being monitored and controlled as well as inter-operational, which is tracking the process from one operation to the next. It may include alarm management to make sure factory person(s) are aware of process changes which are outside acceptable tolerances. It provides interfaces between intelligent equipment and MES possible through Data Collection/Acquisition.
Maintenance Management	Tracks and directs the activities to maintain the equipment and tools to insure their availability for manufacturing and insure scheduling for periodic or preventive maintenance as well as the response (alarms) to immediate problems. It maintains a history of past events or problems to aide in diagnosing problems.
Product Tracking and Genealogy	Provides the visibility to where work is at all times and its disposition. Status information may include who is working on it; components materials by supplier, lot, serial number, current production conditions, and any alarms, rework, or other exceptions related to the product. The on-line tracking function creates a historical record, as well. This record allows traceability of components and usage of each end product.
Performance Analysis	Provides up-to-the-minute reporting of actual manufacturing operations results along with the comparison to past history and expected business result. Performance results include such measurements as resource utilization, resource availability, product unit cycle time, conformance to schedule and performance to standards. May include SPC/SQL. Draws on information gathered from different functions that measure operating parameters. These results may be prepared as a report or presented online as current evaluation of performance.

2.2 MES in Process and Discrete Industry

2.2.1 Process Industry MES

Process manufacturing industry is the one that produce products by mechanical, heat and/or chemical treatment. According to the IIEs (Institute of Industrial Engineers)[28], "The process industries are those industries where the primary production processes are either continuous, or occur on a batch of materials that is indistinguishable". The classification of process industries according to their end products can be given as food and beverage, oil and gas, pharmaceuticals, paper and pulp industry, chemicals, metal mining and other processes industries.

Process industry is usually attributed centric, it mainly concentrates on ingredients and formulas while manufacturing. Process industries are usually attribute centric since the material/raw material it deals with can only be determined and controlled by various attributes unlike discrete industry where there is a concrete product that doesn't have any inner characteristic changes while being manufactured. While considering MES for a process industry, it should be capable of satisfying these main conditions.

- Define material recipe for the product being processed.
- Determine and control the material attribute at each stage.
- Capable of Attribute based routing.
- Know the manufacturing and equipment constraints.
- Mainly it must be capable of trace and track material based on attributes.

Other than these, MES for process industry must also consider some special conditions based on the process industry differentiation. Depending on the object (physical attributes of the product) and flow type, process industries can be differentiated into continuous and intermittent (batch)[29].

Batch Process Industry

Process industries in which the process can be broken down into simple discrete modular units and which can be manufactured in batches are known as batch process industry. In batch processing, a fault or maintenance in one phase does not affect the processing in other phases. Pharmaceutical, fertilizer and food and process industries are some of the common batch process industries. While considering about MES in batch process,

- Must know when a batch process begins and when it's done.
- Where the process/materials enters in a phase and where it exits.

In a batch process industry, the MES mainly concentrates on track and trace of materials, genealogy, production scheduling and optimization, record keeping and process management.

Continuous Process Industry

Process industries in which the process is a continuous flow of material where other materials are added and the whole mixture is processed to give a final product is known as continuous process industry. In continuous processing, a fault or maintenance in one phase affects all the other phases in the process and also the material cannot be removed at an interval and modified or processed to bring back to the current conditions in the process. Mining and metal, chemical and oil and gas industries are some of the common continuous process industries. While considering about MES in continuous process,

- Must be capable of combining ingredients and treating materials in a precise way at precise points of the process.
- Must be capable of controlling the process conditions to maintain quality and safety operations.

2.2.2 Discrete Industry MES

Discrete manufacturing industry deals with discrete parts manufacturing, where individual parts are processed and assembled to yield the final product[30]. Discrete industry which is product centric usually deals with component, sub-assemblies and finished unit. Unlike the process industry, in discrete industry MES are usually event based with concentration mainly towards parts, BOM (Bill OF Materials) and discrete units. MES software developed for the discrete industry must provide management and synchronization of production tasks with material flow, labour and machine resources management and event based routing.

2.3 Monitoring Methods

Manufacturing systems are subjected to several kinds of risks and disruptions which may interrupt their proper functioning while performing manufacturing activities. These events may occur at any time and may lead to any consequences both internally and externally. The internal consequences will be the ones which affect the performance of the machines and causes production down time. While the external consequences will directly affect the product it manufactures. Hence for these reasons and much more[31], there needs to be a monitoring system for all resources and equipment. A monitoring system is basically built based on some method for a purpose. There are various types of monitoring in a manufacturing environment, some of them are discussed in this section.

2.3.1 Process Monitoring

In a manufacturing industry, process monitoring may refer to the operation of monitoring a process from top to bottom. The process may be either be a whole manufacturing activity or an individual machinery process or a process that a resource does in particular. Usually, condition monitoring is the aspect which takes care of all the monitoring activities considering a machinery. But in some cases like lubrication system and machining system [32], the process of the machinery also needs to be monitored. There has also been plenty of research[33], [34], [35] going with respect to the methods of monitoring a manufacturing process.

2.3.2 Quality Monitoring

In manufacturing industries, quality monitoring refers to the activity of minimizing the occurrence of quality related issues by streamlining the manufacturing process[14]. Particularly, quality refers to the quality of the product which can be achieved in turn by improving the process quality. Usually, the monitoring for the quality is done at the manufacturing process phase; while the control for quality is done at the machinery end. There has been numerous research in the methods [36], [37], [38] and systems [14], [39] for quality monitoring.

2.3.3 Performance Monitoring

Performance monitoring does the operation of monitoring the performance of manufacturing entities in an industry. The entities may vary from a single machinery to a complete production line. In order to attain is objective, performance monitoring first identifies the key factors for the performance; then collects the information related to performance from the key factors; analyses the information and provide reports on the overall operational performance of the system. Usually, the proper functioning of a system to the expected standard is gauged and evaluated by a performance monitoring system. The method of performance monitoring varies depending on the type of process, which has been explained in the article “Monitoring Process Manufacturing Performance”[40].

2.3.4 Condition Monitoring

The process of monitoring the condition of the components (machines and equipment) used as part of manufacturing process is known as condition monitoring. This type of monitoring is so critical because doing this will assist in more activities like increasing quality of the product and the process efficiency. The article by O.J. Bakker[41], it explains how the process of condition based maintenance helps in process monitoring and product quality management.

2.4 Key Performance Indicators

To determine the effective performance and success rate of a process or a system, metrics are used. The metric which evaluate the crucial factors of a system or process is known as Key Performcance Indicator (KPI). There are many definintions available in case of KPI. But the most common ones with respect to manufacturing process is given in the article by Jovan [42] as,

'A variable that quantitatively expresses the effectiveness or efficiency, or both, of a part of or a whole process, or system, against a given norm or target' '

The other definition with respect to monitoring systems is given in the book written by JAN Smith [43] as,

'A performance indicator defines the measurement of a piece of important and useful information about the performance of a program expressed as a percentage, index, rate or other comparison which is monitored at regular intervals and is compared to one or more criterion'

Usually the KPI has four key properties as determined in the article by [44] as,

- Unit of measurement (watt, litres, etc.)
- Type of Measurement (absolute or adjusted)
- Boundries (entire life cycle, production line, etc.)
- Period of measurement (daily, weekly, etc.)

The two most common top-level metrics thet evalualte the manufacturing operations per the efficient use of time, materials and facilities are Total Effective Equipment Perfomance (TEEP) and Overall Equipment Effictiveness (OEE). In addition to that, there are four more metrics which fill the gap between the above two metrics, they are: availability, loading, quality and performance as explained in the whitepaper [45] by Capstone Metrics.

Some of the KPI related to the manufacturing and monitoring system are explained in the section below.

2.4.1 KPI in Manufacturing

In manufacturing industries, the KPI are used for many purposes like tracking, evaluating and analysing the different operations and there by determining the success in the process. Considering the manufacturing levels, the KPI has an interalted connection with them. For example, a KPI calculated in the business level will depend on some metric values from the shopfloor and vice versa. KPI's for manufacturing are particularly reported in

the ISO 22400 report. The report states about 35 indexes associated with the manufacturing process as listed in the article [46] by Fukuda,Y. Some the metrics related to manufacturing are listed in the articles [47] and [48].

2.4.2 KPI in Monitoring

KPI in monitoring has been used in many levels from monitoring individual machinery to the level of monitoring thr whole process. Similarly, the end results also vary depending on the level which it monitors. Apart from the TEEP and OEE metrics, there also some more metrics related to the supply chain monitoring are given in the article [49] by Z.Chorfi. As reported in the article [50] by Dr.Marco, an overall idea of KPI's involved in monitoring and fault diagonosis can be related to the one of the use case of the thesis.

2.5 Open Knowledge-Driven System

Open, Knowledge-Driven Manufacturing Execution System (OKD-MES) aim to provide Open solution, which employs Knowledge Driven approach to implement MES in a manufacturing industry. OKD-MES is a concept which is being in eScop project[51]. The knowledge-driven Service Oriented Architecture of OKD-MES works with the embedded system based factory floor which provides a platform of loosely coupled interoperable and reusable services. Thus the knowledge-driven approach applied to OKD-MES services enables the dynamic reconfiguration of system, making system adaptable to changes such as introduction of new tasks, changes in equipment and other. The OKD-MES consists of five layers: Physical (PHL), Representation (RPL), Orchestration (ORL), Visualization (VIS) and Interface (INT) as shown in the Figure 8.

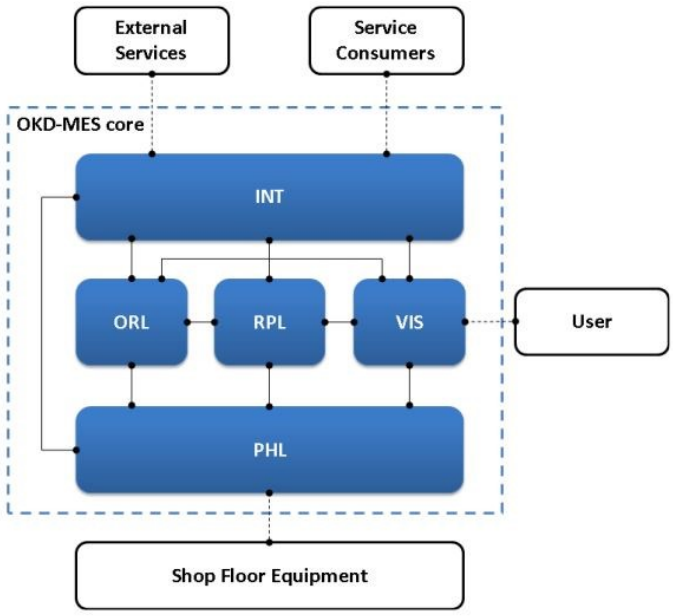

Figure 8 eScop OKD-MES Architecture[52]

The three layers which provides the core functionality in the OKD-MES system are, PHL which is closely related to the equipment and embedded devices in the shop floor, RPL which is a knowledge repository which contains the model of real world, and provides access to it on demand by other layers in OKD-MES and ORL which provides the actual MES functionality in OKD-MES. Two other layers are dedicated to make the system interoperable with humans and external systems. The interactions between these layers have been with the help of services which are to be exposed and consumed by each layers.

This thesis deals with the implementation of a configurable framework which co-exist with eScop OKD-MES architecture. The layers of the OKD-MES system developed as part of the eScop project is explained in this section.

2.5.1 Physical Layer

Physical Layer (PHL) in OKD-MES concept constitutes of service-enabled RTUs which holds the list of sensors and actuators in the shop floor. Each RTU's holds the details of the service and event end points of their respective components (sensors and actuators). The PHL then provides these details to RPL as part of discovery. These are the details which are stored in the RPL as knowledge of the system.

2.5.2 Representation Layer

Representation Layer (RPL) is one of the main component in the eScop architecture which holds the complete knowledge about the PHL layer. It serves the Knowledge to other layers in their suitable format as required by them. In the RPL, the knowledge is stored inside the manufacturing system ontology (MSO). MSO holds the system configuration, such as knowledge about component functionality, its relations and the endpoints for accessing the components.

2.5.3 Orchestration Layer

Orchestration layer (ORL) as the name suggests, orchestrates the sequences of operations provided by the system components based on the defined processes. ORL receives its recipe and the knowledge about the system from the RPL layer and it orchestrates the PHL layer. During the process of orchestration, the ORL works with its own inbuilt logics. It automatically checks if the station is free and allocates the job to it. If the station is not free, then it checks which stations free and allocates to them without disturbing the other process in the recipe.

2.5.4 Visualization Layer

Visualization layer (VIS) is the main layer which mostly interacts with the user as part of the OKD-MES approach with the user interfaces. Though HMI and external systems in the PHL do interact with user, they are just fixed components for each particular use case type of PHL layer. But VIS layer is something more than that. It configures the user interfaces available in it depending on the system configuration provided by the RPL and has the ability to visualize them in a web browser. Hence this is one global visualization solution which can be used in any type of industry.

2.6 SOA and Web Services

Service Oriented Architecture (SOA) in a manufacturing industry is considered to be the state-of-the-art standard used to facilitate the integration of different software components providing services in the shop-floor equipment's. The definition for SOA from OASIS (Organization for the Advancement of Structured Information Standards) is given by,

"*A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations*".[53]

The SOA is widely considered to be implemented by Web Services[54]. There are two main class of web services in the web service architecture as per the W3C(World Wide Web Consortium), they are Arbitrary WS and Representational State Transfer (REST)-compliant WS[21].

2.6.1 Arbitrary WS

Arbitrary WS or SOAP WS are the services that expose an arbitrary set of operations as often designed when adhering to the variety of specifications and languages[55]. The services include more specified standards to address discovery, security and functionality of WS. Communication in the devices with the Arbitrary WS happens with the help of SOAP with one of its main standards WSDL (Web Service Description Language). WSDL contains information about the binding to so-called service endpoints, which allows for the automatic construction of code that provides an interface to the operations. WSDL which are XML based documents contains the information of operations which a web services must provide. SOAP WS are sometimes realized by Device Profile for Web Services (DPWS) and OPC-UA which are applied by real devices[16].

2.6.2 REST WS

Resource-oriented services with a uniform set of stateless operations, that complies with the constraints of Representational State Transfer (REST) architecture[55]. In order to communicate between machines, rather than using complex mechanisms such as CORBA or SOAP, REST architecture uses simple HTTP to make calls between machines. RESTful WS can be interacted with minimal additional software via web browser with the help of HTTP protocol, which simplifies development of user interfaces for the system. REST has a synchronous communication based on server and client request-response pattern shown in Figure 9. In contrast to SOAP, the communications consist of constrained set of operations of HTTP verbs (e.g., GET, PUT, POST and DELETE). Hence REST services are widely considered to be a lightweight and simple solution for SOA implementation[21].

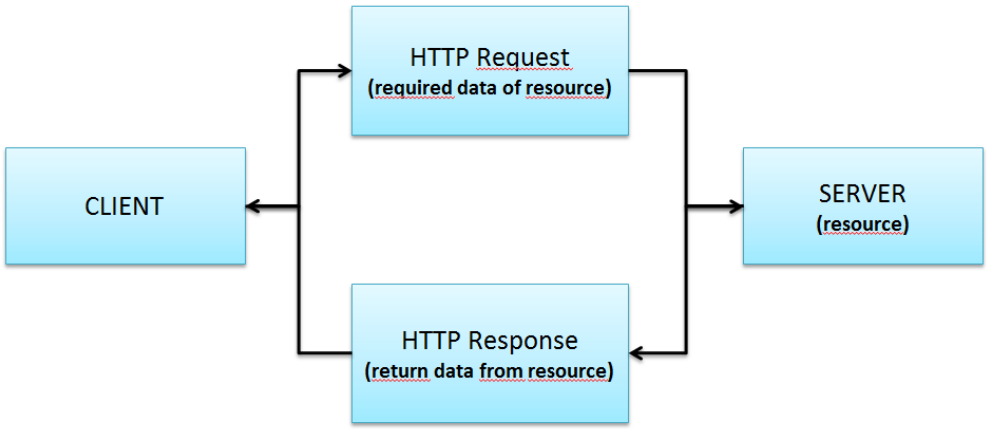

Figure 9 REST Architecture[56]

2.7 Knowledge-Driven Approach

Knowledge Driven approach has been one of the key concepts towards manufacturing industry. With the use of Ontology, knowledge management has been key area of research with respect to manufacturing[57]. The utilization of Knowledge Representation and its combination with SOA implementation facilitates the management of manufacturing system information[21]. The knowledge driven approach has been presented by the eScop Project as part of the OKD-MES system[51].

2.7.1 Knowledge Representation

Knowledge Representation (KR) and reasoning, a portion of the artificial intelligence field describes the world information in certain formalisms which can be interpreted and used by computer systems for solving complex tasks. In manufacturing industry, Knowledge Representation helps in creating of an enterprise system model that incorpo-

rates all the required information that is generated and consumed by manufacturing systems in both in human and machine readable form. The system model is created with the help on ontologies which offer a hierarchical and well organized method for describing the system model. Out of the many languages available for implementing ontology, RDF (Resource Definition Framework) is the most used one[58].

2.7.2 OWL

Web Ontology is a semantic mark-up language based on XML, XML Schema, RDF and RDF Schema (RDFS). OWL is used for ontology definition and is a recommendation of the World Wide Web Consortium (W3C). It also provides the level of abstraction required for the knowledge Representation. OWL also has three different sub Languages or layers: OWL Full, OWL DL, and OWL Lite. Like RDF, OWL which is the extension of RDF can also be queried/ updated by the RDF-based SPARQL Protocol and RDF Query Language (SPARQL)[59]. OWL with the capability of query and update through SPARQL can surely help in the Knowledge Representation and reasoning in a manufacturing industry.

2.8 Information Technology in Manufacturing

In manufacturing industries, information technology plays a major role in all sections. From PLC, SCADA, DCS, etc. in shop floor level to complex software systems like ERP, MES, LIMS, etc. computer and IT plays a major role. Be it controlling the machines or programming the machines or diagnosing those machine IT is one of the important factors for the easy and proper functioning of systems inside the manufacturing industry.

2.8.1 Web Application:

Initially when the IT came into existence inside the manufacturing industries, they were server client models, in which the load for an application was shared between the servers and the application was installed separately in each client machines and connected to a network where it can integrate with other similar applications in other clients. From the evolution of internet, web application came into play. Web application also known as web app is a similar to client-server software applications, the difference being that the client machine can access the information and stuffs through the web browser while the main application can be hosted in the server. Usually, the front end or the web browser is built with HTML5, JavaScript, AngularJS, PHP, etc. and the backend with JAVA, C#, Python, etc.

2.8.1.1 Java

Java, an object-oriented computer programming language came into existence as a language project in 1991 and it was officially released by SUN Micro systems in 1995[60].

Being a platform independent language, it is concurrent, class-based and specifically designed to have as few implementation dependencies as possible. Since Java is an open platform and allows its application to be executed in maximum number of hardware and operating system, it quickly got attention in conventional enterprise application field and also in manufacturing research areas. In manufacturing industries, Java has its root in all levels; starting from the shop-floor level (DCS, PLC, etc.), to the Factory Management Level (MES, PIMS, LIMS, etc.) and the Enterprise Management Level (SCM, ERP, etc.).

2.8.1.2 Spring MVC

Spring Web model-view-controller (MVC) framework is a request driven frame work designed around a central Servlet. It dispatches requests to controllers and offers other functionality that facilitates the development of web applications[61]. Easy and flexible web applications can be developed with the help of ready components in Spring MVC architecture. Different aspects of the application like input logic, business logic and UI logic were separated and provided loose coupling between the elements by the MVC architecture.

2.8.1.3 HTML 5

Hyper Text Mark-up Language or HTML which is used to format the web documents was essentially a tag-based language for building web applications. Globally acknowledged user friendly language structure, rapid emergence of new features and introduction of novel tools are the main reason for HTML to be more common in web development. HTML5 is the latest of the HTML family, which offers more powerful and unique features for web designers. HTML5 allows easy and efficient integration of other web tools like JavaScript, PHP, CSS, etc. Currently HTML5 with CSS and JavaScript is the most suitable technology for developing powerful web applications.

2.8.2 JSON Message format

JavaScript Object Notation (JSON) is a human-readable and format friendly data interchange format which is easier for generating and parsing. It is easy to describe and encode information more lightweight in JSON. It has mainly two data object: Array and Object. An array in JSON is an ordered set of values, while Object is a disorder set of name/value pairs; its form is similar to the object map in Java.

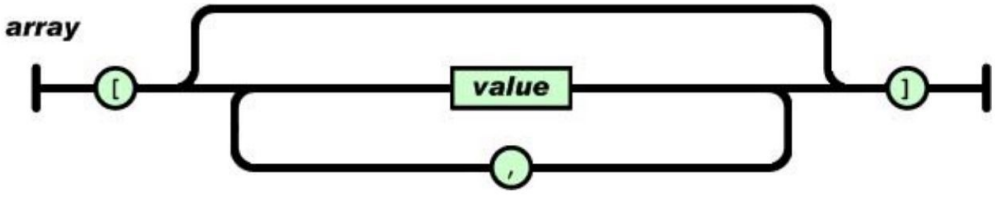

Figure 10 Array format in JSON[62]

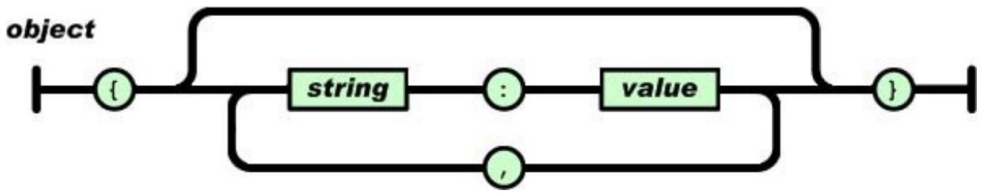

Figure 11 Object format in JSON[62]

2.9 Summary

From the standards to the existing softwares and researches conducted in MES/Monitoring systems, it can be identified that there is still plenty of room for further research to have a single system which perform the activities of both MES and Monitoring. Next chapter provides one such methodology which suggests a framework, which will perform as either MES or Monitoring system or both based on how it is configures.

3. METHODOLOGY

The main objective of this approach is to present a configurable framework that can perform certain functions, which can be a MES system or a monitoring application for devices or any other information system in an enterprise. This section of the thesis starts with the overview of the methodology suggested in the above objective. Then it elobrates the model consisting of the framework to realize rhe methodology and then continues with the techniques to be followed in order to design and develop the model. Later the tools to be used for developing the complete model have been explained in detail.

3.1 Overview

The methodology overview in the Figure 12 of the thesis states a system which can be configured with any standards to act as a software system of that particular standard.

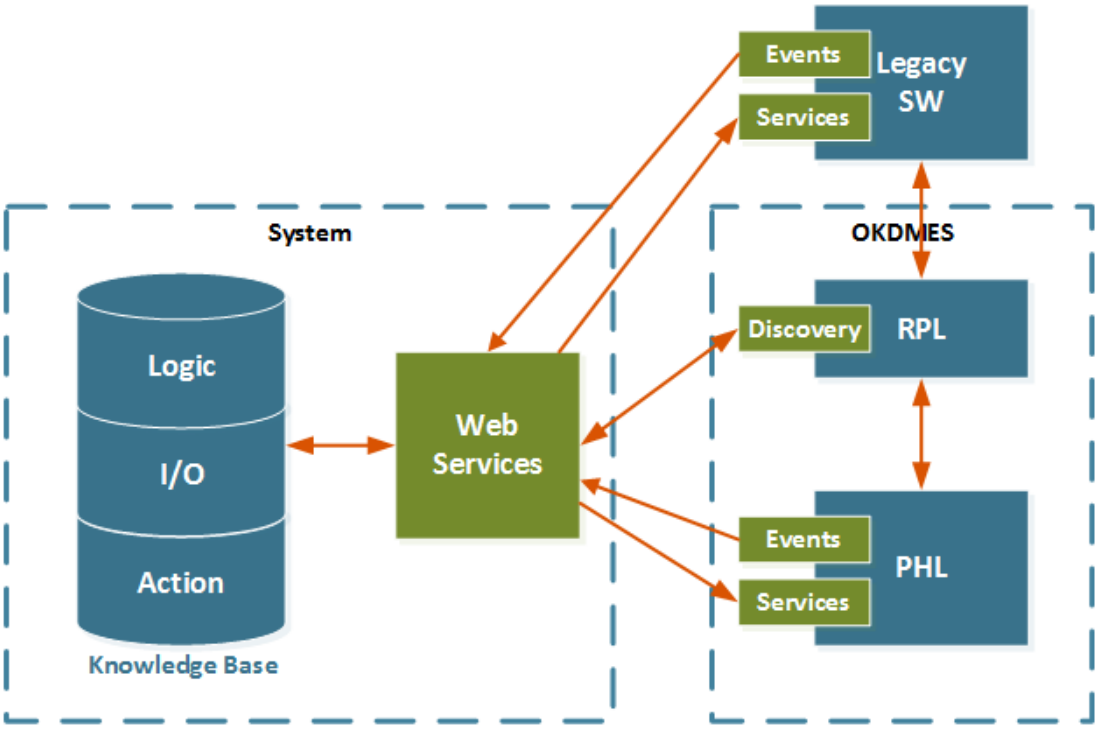

Figure 12 Block diagram for system integration

In order to achieve this, the system must basically be a black box with global components which can be configured from a knowledge base. The knowledge base will basically have the configuration details like,

- Logic - The main math that will be performed to acieve a specific task.
- Input - values on which the logic operates.
- Output - value generated by the logic.

- Action – operation to be carried out once the Logic has been executed.

Basically, these configrations drive the system via exposure as web services. The inputs are the events (web service) from other systems; Outputs are the events to other systems; and the Actions are the Invokation of services in the other systems. The system access the events and the services from the PHL layer devices or the other legacy systems via the integration with the OKDMES system. This way the system can be a complete black box with just primary programs to how to integrate to the OKD system and perform basic tasks. The main logic/ tasks to be executed by the software can be created and configured dynamically with less effort.

3.2 Model

The Model presents a modular framework, which contains control logics in the form of functions that are rendered as events and services, which can be consumed as RESTful web services. The functions in the framework are configured and driven by ontology, which makes the whole concept configurable and flexible. The framework while working along with OKD approach can help in obtaining different kind of information systems in the factory.

The model shown in Figure 13 gives an overall representation of the components in the framework and the interaction between the components. It also shows the sub component in each components which are detailed in section 3.2.1 and 3.2.2. Basically, the model consists of two parts: one is the Fuseki Server which holds the ontologies and the other is the Framework module which is the main controller of the whole methodology.

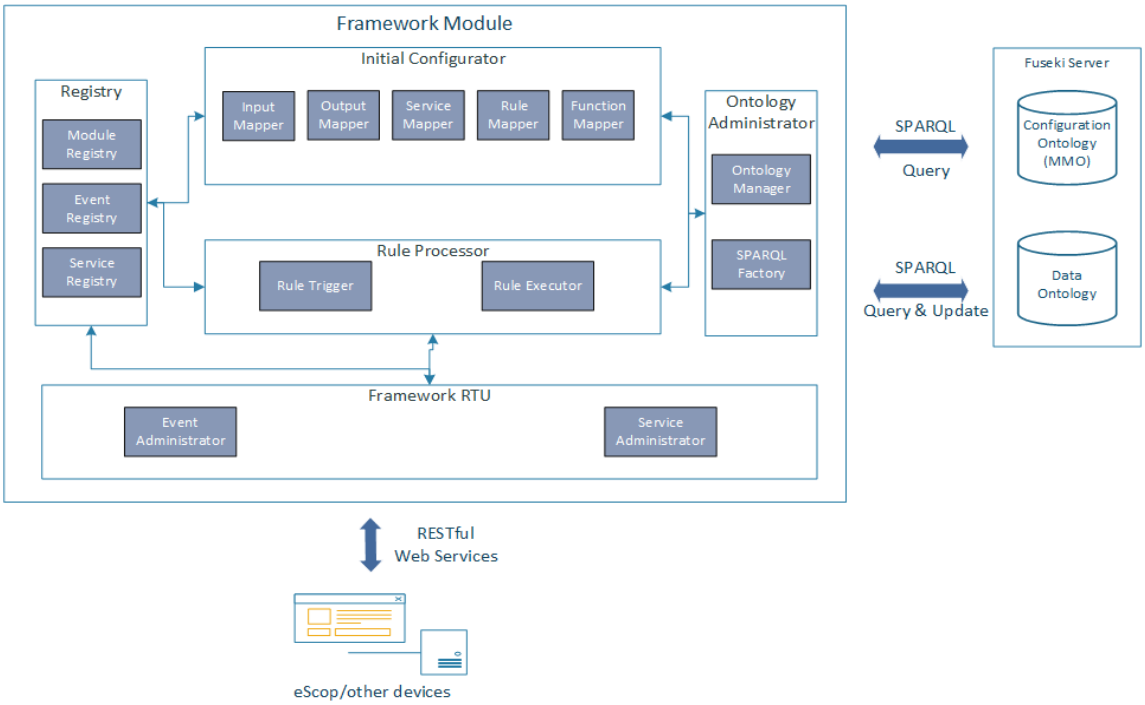

Figure 13 Manufacturing / Monitoring Framework

3.2.1 Fuseki Server

Fuseki server generally holds the ontologies of the framework. Other than hosting the ontologies, the Fuseki server also helps to query and update them by means of SPARQL queries. The SPARQL query/update that are handled by the Fuseki server are usually from the Framework Module. The framework has two separate ontologies that are hosted by Fuseki server: MMO and Data ontology.

3.2.1.1 MMO

MMO (Manufacturing and Monitoring Ontology) is the main ontology for the framework, which holds all the configuration details. MMO is built in such a way that it can be easily configured to suit any type of application the framework will be designed for. It consists of list of functions and also the rules for each function. The rules are nothing but the logic to be performed in order to make the framework behave as per the roles of the function. The components that exist as part of MMO ontology for configuration purpose are represented in Figure 14 and detailed below.

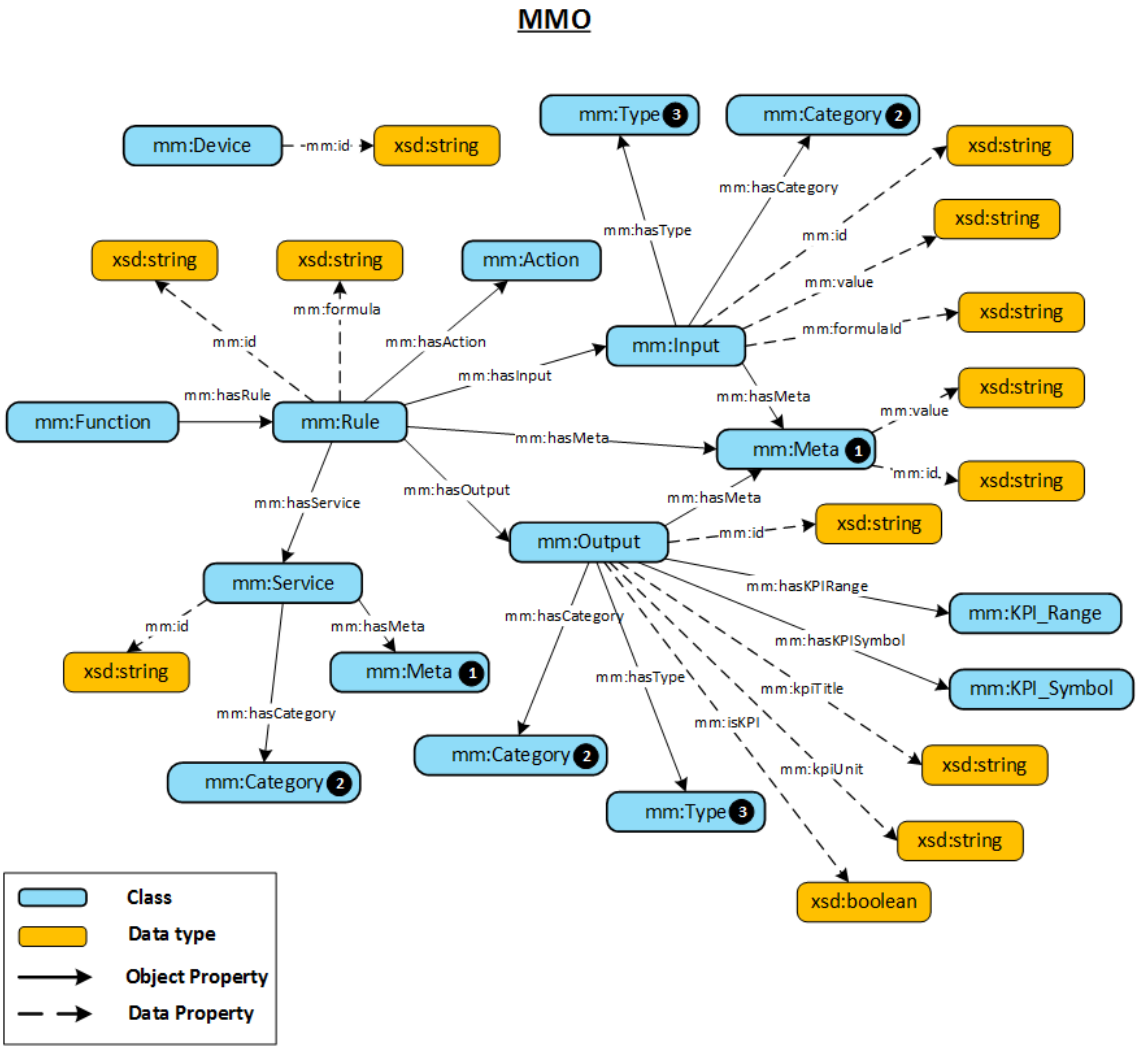

Figure 14 Manufacturing and Monitoring Ontology

Device

Device is the ontology class which holds the list of devices of the shop floor. These are the devices that the Framework should be operating upon. The individuals stored in it can either be individual device id or id (type) for a collection of devices. In case of individual device id, the Framework discovers and maps output of a rule to that specific device. In case of id for a collection of devices, the Framework discovers all the devices and maps outputs of a rule to each device. The outputs of the rule which are mapped will also have the same device ids. The elements as part of the device ontology class are described in the Table 2.

Table 2 Ontology Device Elements

Property	Type	Presence	Description
ID	String	Required	Id of the device

Function

Function is the ontology class that holds the list of functions. The functions are specific functionality (i.e. condition monitoring, preventive maintenance) that a framework will possess. Each function has an id and contains the list of rules. The function will be added by the user as per the requirements. The elements as part of the function ontology class are described in the Table 3.

Table 3 Ontology Function Elements

Property	Type	Presence	Description
ID	String	Required	Id of the function. This Id will be used as the name for the Framework RTU and hence it must be unique.
Rule	Class	Required	Rules which have the logics for the function and is itself another ontology class.

Rule

Rule class of the ontology holds the list of rules. The rules are the basic units, which holds the logics for a function. In a rule, the entire logic is placed in form of a formula written in MVEL language[63]. The elements as part of the rule ontology class are described in the Table 4.

Table 4 Ontology Rule Elements

Property	Type	Presence	Description
ID	String	Required	Id of the rule which will be used by Framework. This must be unique since this will be used in the Framework RTU.
Input	Class	Required	Input which is the necessary to execute the formula of the rule.
Formula	String	Required	Formula which has the math and upon execution computes the result of the logic.
Output	Class	Required	Output of the formula will be associated with this element.
Meta	Class	Required	Tags used by the framework to associate the rule to the device.
Action	Class	Required	The resulted action which the rule should perform once the formula is executed.
Service	Class	Optional	If the action is 'Invoke Service', then there must be a service associated with the rule to be invoked.

Input

Input class in the ontology holds the list of Inputs which are later used in the rule class. The Inputs are the ones used in the formula of each rule. The input may be any one of the 4 types,

1. A constant value which will be added by the user in the value property.
2. Value from data ontology.
3. Value from the event payload.
4. Value provided while the service is invoked.

The framework decides the input type based on the category assigned. It also increments the input based on the Meta which holds the device id / device type. The elements as part of the input ontology class are described in the Table 5.

Table 5 Ontology Input Elements

Property	Type	Presence	Description
ID	String	Required	Id of the Input which will be used by Framework. This must be unique.
Formula Id	String	Optional	Id that is used in the formula of the rule for the input. If it is the same as the ID, then it is not required.
Category	Class	Required	Category of the input which will determine the type of the input.
meta	Class	Required	Tags used by the framework to associate the inputs with device.
Type	Class	Required	The data type of the input value.
Value	String	Optional	Default value for the input.

Output

Output class in the ontology holds the list of outputs which are later used in the rule class. The output may be a value or a message. If the output is a value, then it will be emitted as an event or as reply to the invoked service based on the rule action. If it is a message, then it a different case, where the output follows a specified format as mentioned below in the Table 6.

Table 6 Output Value Format

```
{
        “trigger”: false [boolean],
        “message”: {“payload”: {“value”:” sample”}}
}
```

Based on the status of the trigger, message will be sent as the Request Body while invoking the service. The message type output will be used only when the rule has to invoke any service. The elements as part of the output ontology class are described in the Table 7.

Table 7 Ontology Output Elements

Property	Type	Presence	Description
ID	String	Required	Id of the Output which will be used by framework. This must be unique. In some cases, this ID must be the same as the output variable in formula of the rule.
Category	Class	Required	Category of the output.
meta	Class	Required	Tags used by the framework to associate the outputs with devices.
Type	Class	Required	The data type of the output value.

Meta

Meta class holds the list of Meta (key-value pairs), which are used in many components of the configuration ontology. Meta are one of the main components which help the framework in discovery. The use of Meta has been much explained in the section. The elements as part of the meta ontology class are described in the Table 8.

Table 8 Ontology Meta Elements

Property	Type	Presence	Description
ID	String	Required	Id of the meta which will be used by framework. This must be unique. This is the key for meta.
value	String	Required	The value of the meta.

Service

Service class in the MMO holds the list of services. These are the services that will be invoked by the rules while their action is service invocation. The elements as part of the service ontology class are described in the Table 9.

Table 9 Ontology Service Elements

Property	Type	Presence	Description
ID	String	Required	Id of the Service which will be used by framework to relate with the service from device. This must be unique.

Property	Type	Presence	Description
Category	Class	Required	Category of the Service.
meta	Class	Required	Tags used by the framework to associate the service with device.

Type

Type class in the MMO contains the list of data types. They are the data type of the elements that are used in ontology. The type is a fixed number of individuals. The framework only supports the types which are predefined as part of MMO. The Date as a type has not been added since ISO date format has been used which can be given in long format. There are 7 data types that predefined. They are,

1. Integer
2. Boolean
3. String
4. Array
5. Double
6. Map
7. Long

Action

Action class of the MMO consists of list of actions. Action defines the operation to be performed by the rule once it has been executed. There are four actions and these are predefined. The actions are,

1. **Event** – The rule emits the output as event.
2. **OntologySave** - The rule saves the output to data ontology for future use.
3. **ServiceInvoke** – The rule invokes a service in the external device.
4. **ServiceReply**- This is a special case where the rule will be activated by a service request. Then the rule executes the formula and sends the output as reply to the service invocation

Category

Category class of the MMO holds the list of category used by the elements of MMO ontology. The category contains an ID with it. Like the type and action, the category is also fixed. There are 10 categories that are predefined. They are,

1. **Event** – This category is used in input, which will make the framework realizes that the input will be an event.
2. **Message** – This category is used in output, which will make the framework realize that the output must be sent as a request body while invoking the service.

3. **Ontology** – This category is used in input, which will make the framework realizes that the input must be queried from ontology.
4. **Operation** – This category is used in service, which will make the framework realizes that the service will be of type operation.
5. **Process** – This category is used in service, which will make the framework realizes that the service will be of type process.
6. **Query** – This category is used in input, which will make the framework realizes that the input value must be accessed via GET REST services from the device which provides it as service query.
7. **Value** – This category is used in output and input. In case of output, the framework realizes that the output must be emitted as an event. While, in case of input, the framework realizes that the input will be a default value.
8. **Time** – This category is used in input, which will make the framework realizes that the input will the current time value.
9. **ServiceInput** – This category is used in input, which will make the framework realizes that the input will be provided as a service request body while the rule is being invoked.

3.2.1.2 Data Ontology

Unlike the MMO ontology, the data ontology will only act as a database and hold the values of rule output in run-time. The values are stored in the ontology for historical purposes to be used by both the function in the framework and also by any other external systems. The values are stored with the time when it is generated in order to assist in sorting. The Ontology model is given in the Figure 15.

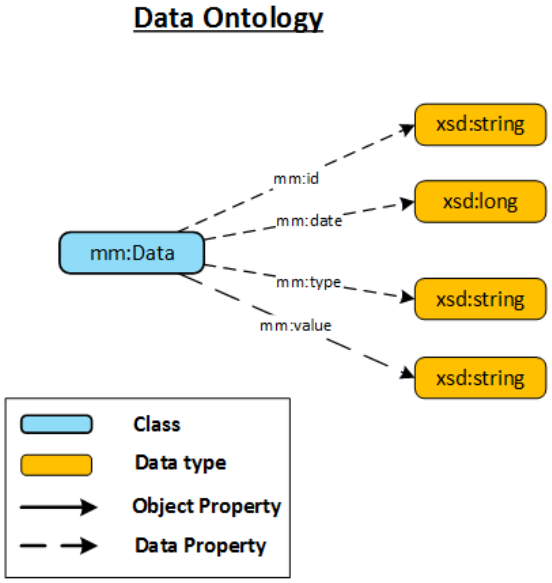

Figure 15 Data Ontology

3.2.2 Framework Module

Framework module is the main controller for the whole framework. The module gets its recipe for the functions it should perform from ontology; it communicates with the Ontology administrator through SPARQL queries and updates. The Module has many sub modules which will perform individual tasks in order to attain the full functionality for a modular framework. The sub modules and its functional goals are discussed below.

3.2.2.1 Initial Configurator

Initial Configurator is the main sub module which executes the tasks of configuring the complete framework. It comes in to action only when the Framework is started and later at runtime this will not do any tasks. The tasks include,

- Constructing the base for the functions and rules as per the ontology model and requesting the Service administrator in RTU to generate the services for the same.
- Initializing the outputs for each rule and requesting the Event Administrator in RTU to generate events for the same.
- Mapping the input with events discovered by the Event Administrator and also requesting the Event Administrator to subscribe to those events.
- Mapping the available service from the Service administrator to the rules which perform service invocation.
- Storing all the inputs, outputs, services, rules and functions to the registry.
- Generating SPARQL queries for the rules which needs its outputs to be saved to ontology

In order to perform these tasks, the Initial Configurators has the following components as shown in Figure 16, to do the respective jobs as per their functionality.

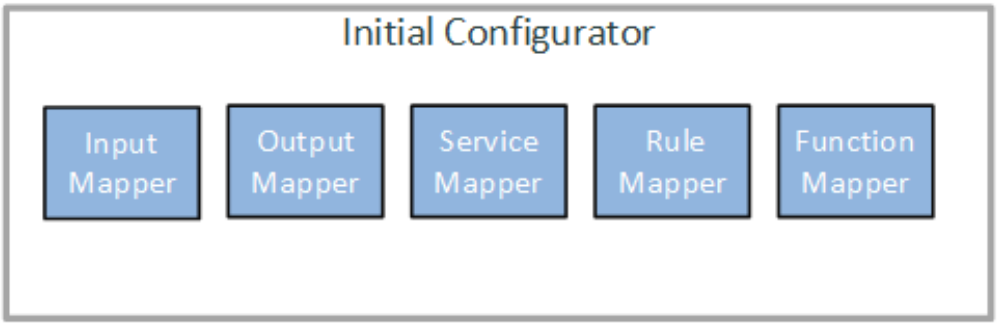

Figure 16 Framework Initial Configurator

Input Mapper

Input mapper creates objects for each input form the ontology. The values for the inputs are either assigned directly or mentioned with the endpoint to access the value. The value accessing depends upon their type which can be known form the Category. Once it maps each input, it also stores them in the registry.

Output Mapper

Output mapper creates objects for each output form the ontology. Then it increments the number of outputs depending upon the devices for the rule. This way there is separate inputs for each device which can either be emitted as an event or sent as a service payload. Once it maps each output, it also stores them in the registry.

Service Mapper

Service mapper, initially queries the list of services from the Ontology. Then it enquires the OKD system and creates object for each service with heir details. Once it maps each service, it also stores them in the registry.

Rule Mapper

Rule Mapper performs the function of creating objects for each rules and associating components (inputs, output and services) with it.

Function Mapper

Function Mapper performs the operation of creating separate objects for each function. Then it maps the rule objects as per the list from the ontology. Once the function is mapped it requests the Framework RTU to create a separate RTU point for each function.

3.2.2.2 Registry

Registry in the framework is responsible for all the tasks with respect to storing the information of each thing in runtime. The components in registry are shown in Figure 17. The tasks include,

- Store entire objects (services, events, inputs, output, rules and functions) with their information.
- Provide the objects in the registry to rule processor at runtime for accessing the information.
- Other than storing the element objects it will also store the information of subscribers to the output events.

Figure 17 Framework Registry

Event Registry

Registry holds the list of events and the information for subscribing to each event. The information includes the device and sensor for which the event is associated, URI and the JSON message format for subscribing to the event. It also holds the list of subscribers to output events of the framework.

Service Registry

Registry holds the list of services and the information for invoking each service. The information includes the device for which the service is associated, URI and the JSON message format for invoking the service.

Module Registry

Module Registry holds the list of inputs, outputs, rules and functions. It includes information which were generated while creation of these objects during the framework initialization. During the framework runtime, the above information for the objects will be modified by Rule processor and later it will also be stored back to the registry with updated information.

3.2.2.3 Rule Processor

Rule Processor in the framework is responsible for all the tasks with respect to executing the rule in runtime. Rule processor is the main brain of the whole framework. The components in rule processor are shown in Figure 18. The tasks include,

- Triggering the rules based upon the scenario they are associated with. The scenario is,
  - I. If any one of the input is event based, then the rule will be triggered once the respective event is generated.
  - II. If the rule doesn't have any event based inputs, then it triggers the rule for every second.
  - III. Other than these, the rule will also be triggered, if the service for the rule is invoked.
  - IV. In case if both case I and case III exists in a rule, then the rule is triggered only when the Service for the rule is invoked.
- While the rule is triggered, the Rule processor creates separate instance of each rule. So that there is no concurrency occurs in the rule executor.
- Processing the formulas of the rules with the inputs and generating respective outputs.
- Requesting the Event Administrator to post events once the respective rule generates outputs.
- Requesting the Service Administrator to invoke service once the respective rule is executed.

- Updating the Ontology Administrator with value if the rule has the action of saving to Ontology.

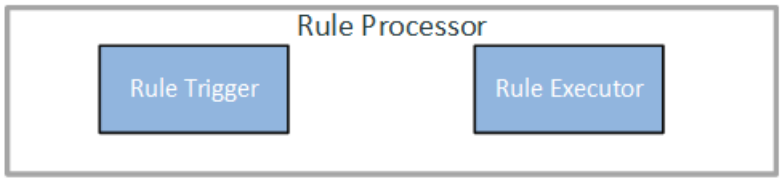

Figure 18 Framework Rule Processor

Rule Executor

The rule executor performs the execution operation in the rule processor. Once the rule is triggered, it initially gets the inputs from the registry/Data ontology and substitutes it to the formula. Then it executes the formula and generates the output. At last based upon the rule action, the output is processed. The action and its respective operation are already explained under Action in section 3.2.1.

Rule Trigger

Rule Trigger's task is to identify the rule which is to be triggered based on the request. The request may be,

1. An event which was subscribed by the rule's input has been generated.
2. The services for the particular rule has been invoked.
3. The rule has no options to be triggered by the above conditions, then it is triggered for every second.

Then it creates an instance of the particular rule and passes it to the rule executor in order to execute the particular rule.

3.2.2.4 Ontology Administrator

Ontology Administrator in the framework is responsible for all the tasks related to ontology. The components in ontology administrator are shown in Figure 19. The tasks related to ontology administrator include,

- Generate individuals in Ontology for each output which is to be saved to ontology.
- Query individuals to assign to input of a rule.

Figure 19 Framework Ontology Administrator

Ontology Manager

Ontology manager performs the function of communication the Fuseki Server through SPARQL queries in order to perform the respected operations for the framework.

SPARQL Factory

SPARQL Factory is the one which generates and holds all the SPARQL query and updates.

3.2.2.5 Framework RTU

Framework RTU in the framework is responsible for all the tasks related to communication with the external systems (OKD systems and Devices). The components in Framework RTU are shown in Figure 20. The tasks related to Framework RTU include,

- Provide Event endpoints as RESTful web services for every outputs which are events.
- Provide Service endpoints as RESTful web services for rules which are to be invoked.
- Registers the event subscribers with their information to registry.
- Register to events and services as per the request from the initial configurator.
- Modify the inputs and update the registry while the subscribed event or service pushes new value.
- Indicate the rule trigger when it updates an input.
- Posting the output value to the subscribers or the devices which invoked the rule service.

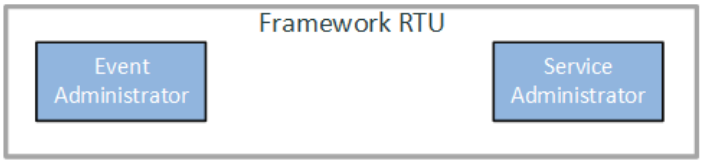

Figure 20 Framework RTU

Event Administrator

Event Administrator takes care of all the activities with respect to events in the Framework. The tasks include generating event endpoints; registering the subscribers for the events; posting the generated events to the subscribers; and indicating Rule Trigger when a subscribed event generates a value and posts it.

Service Administrator

Service Administrator takes care of all the activities with respect to services in the Framework. The tasks include generating service endpoints; indicating Rule Trigger when a Service is invoked; posting the generated output to the device which invoked the service as part of Service Invoke reply.

3.3 Technique

The framework suggested in the above model in section 3.1 is a completely configurable one. It consists of various segments like the Fuseki Server, Framework module and the components inside it. This section of the thesis gives an overview of the communications between the segments made during the initial configurations and also during the actual functioning. It also gives an overview of how the human interacts with the framework.

3.3.1 User Interaction

User Interactions refers to the activities performed by the user/operator while setting up the whole framework. Initially the MMO is configured by th user/operator as per the requirements. The configurations include functions with its rules and device details. Then user configures the devices / application for interaction with the framework. The device/application configurations can either be performed as part of OKD systems or as a separate entity. Then the framework performs the initial configurations based on the MMO and interacts with the respective devices and provide the required results. The different actors and their use cases while the framework is acting as a MES/monitoring system are represented in Figure 21.

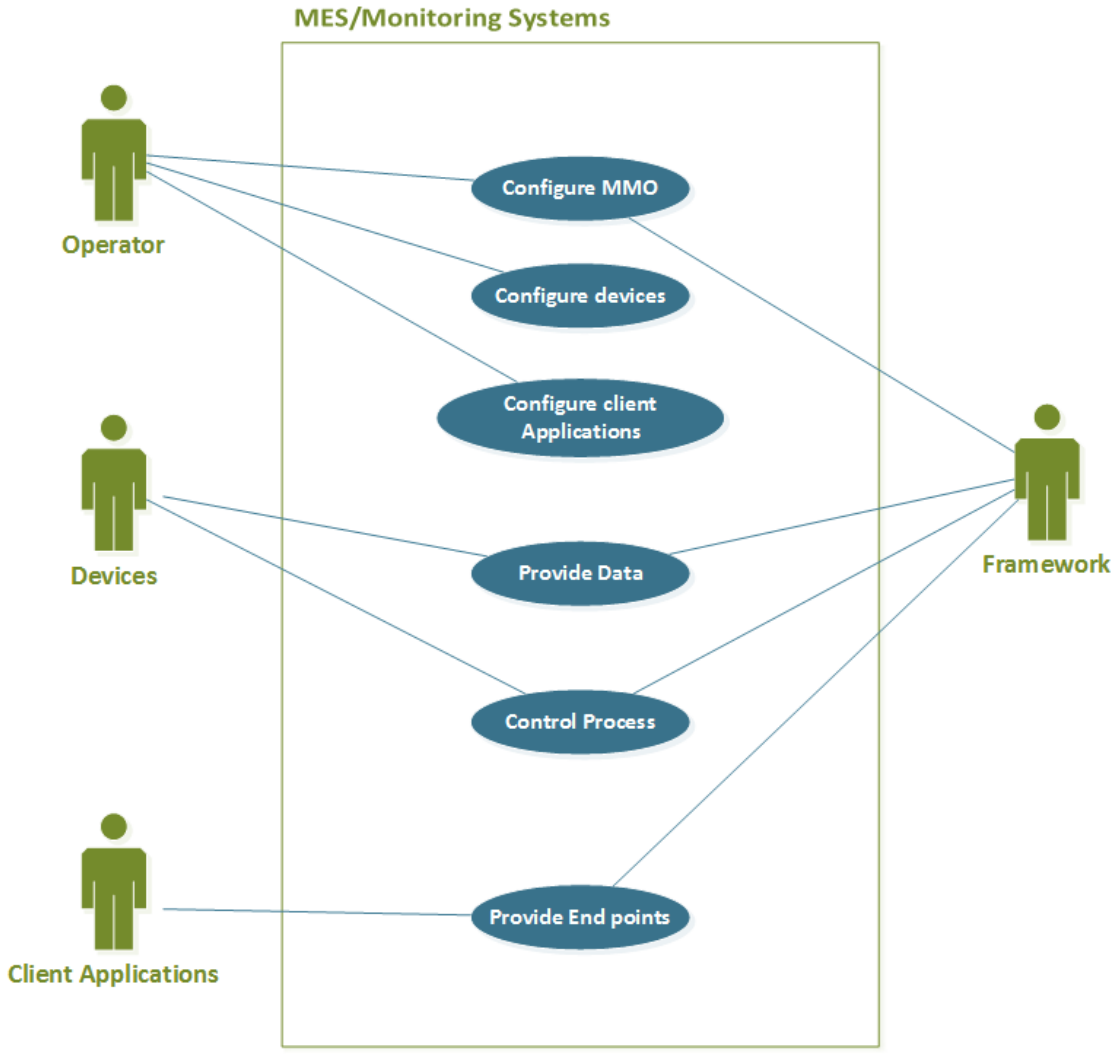

Figure 21 User Interaction with Framework

3.3.2 Initial Configuration

As suggested above, Ontology plays a major part in configuring the framework. The activity model shown in the Figure 22, Figure 23 and Figure 24 gives a complete outline of the communications made in the framework during the configuration. This activity is done only once when the system is started.

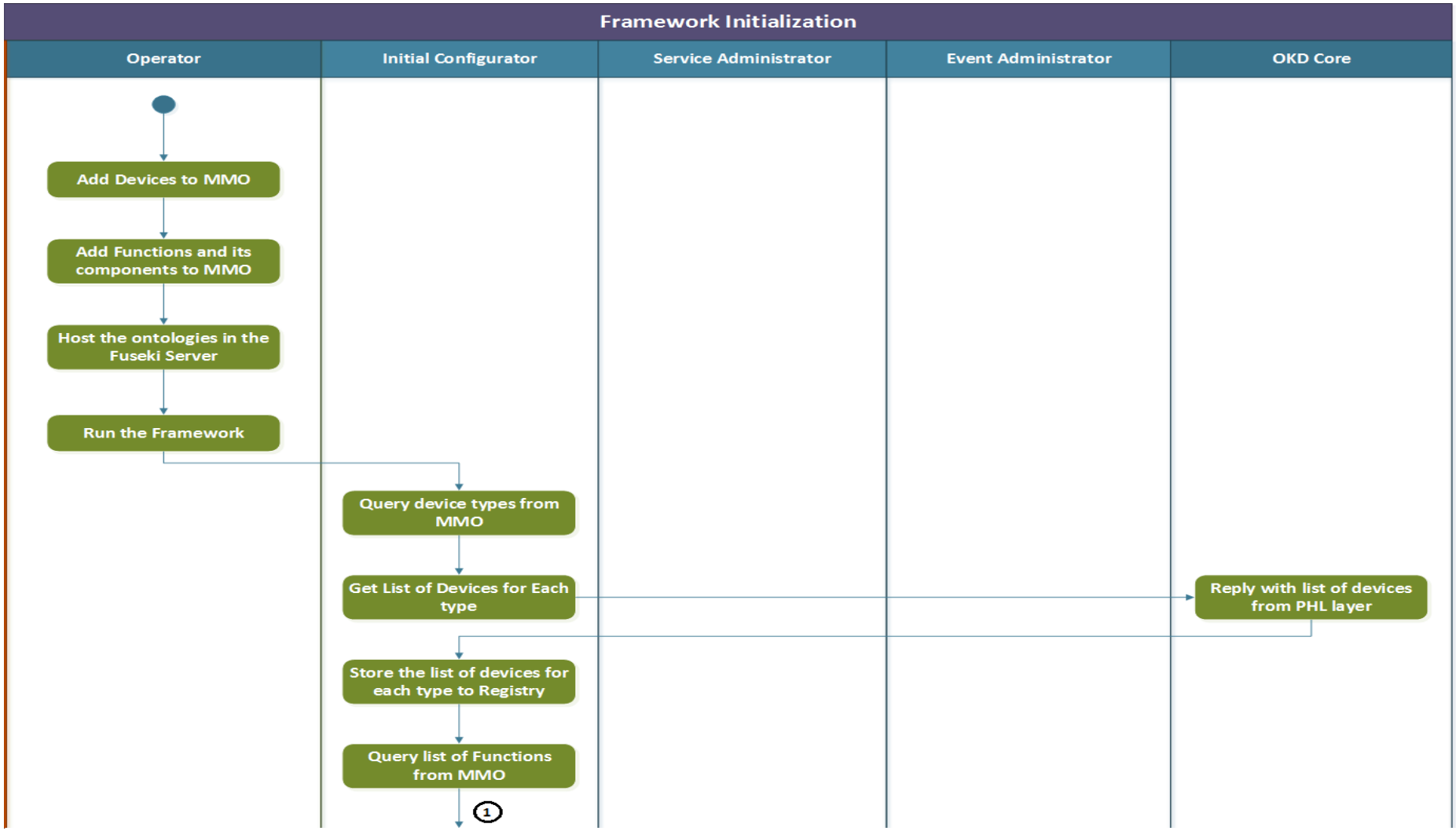

Figure 22 Framework configuration (1)

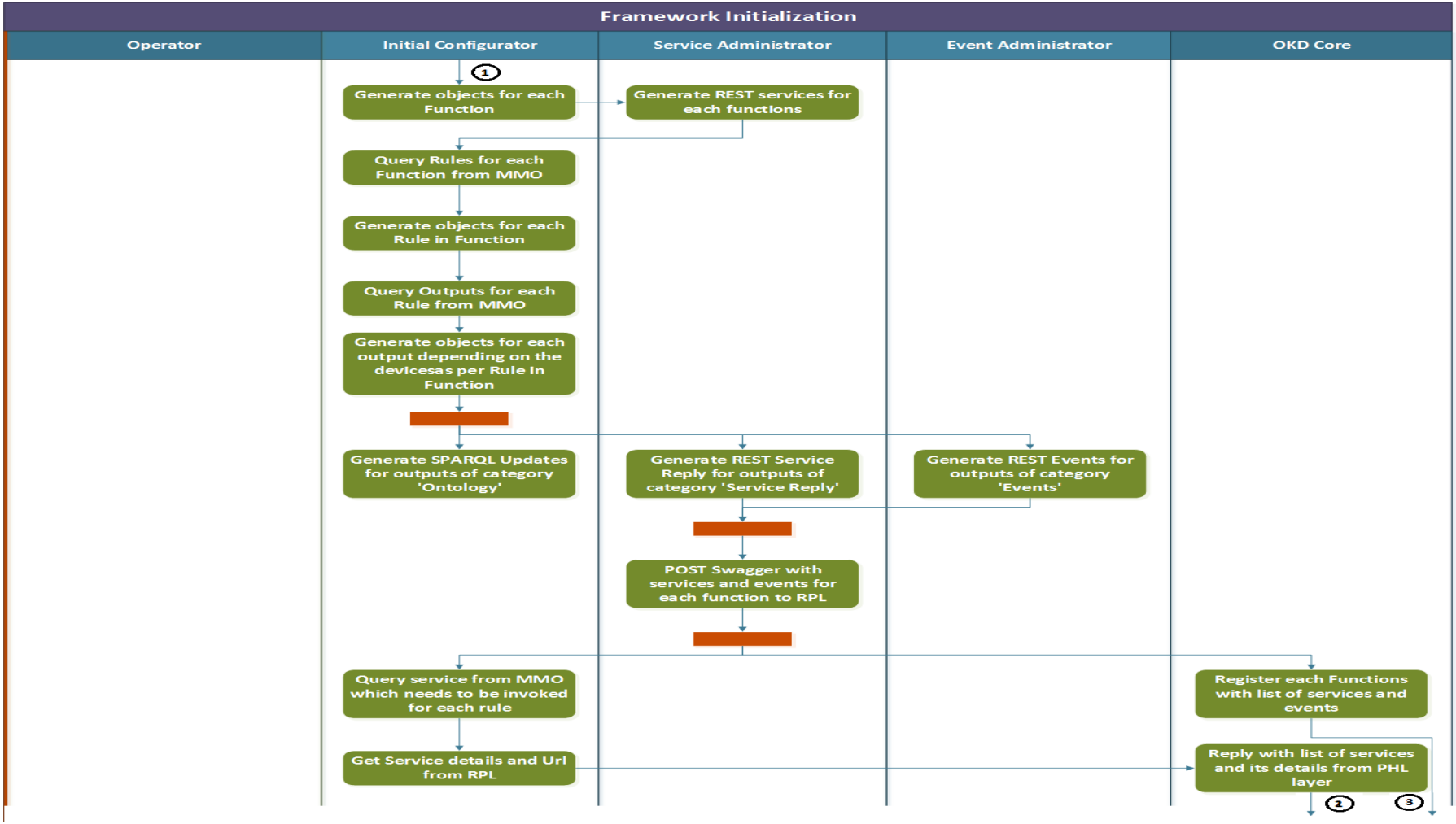

Figure 23 Framework Configuration (2)

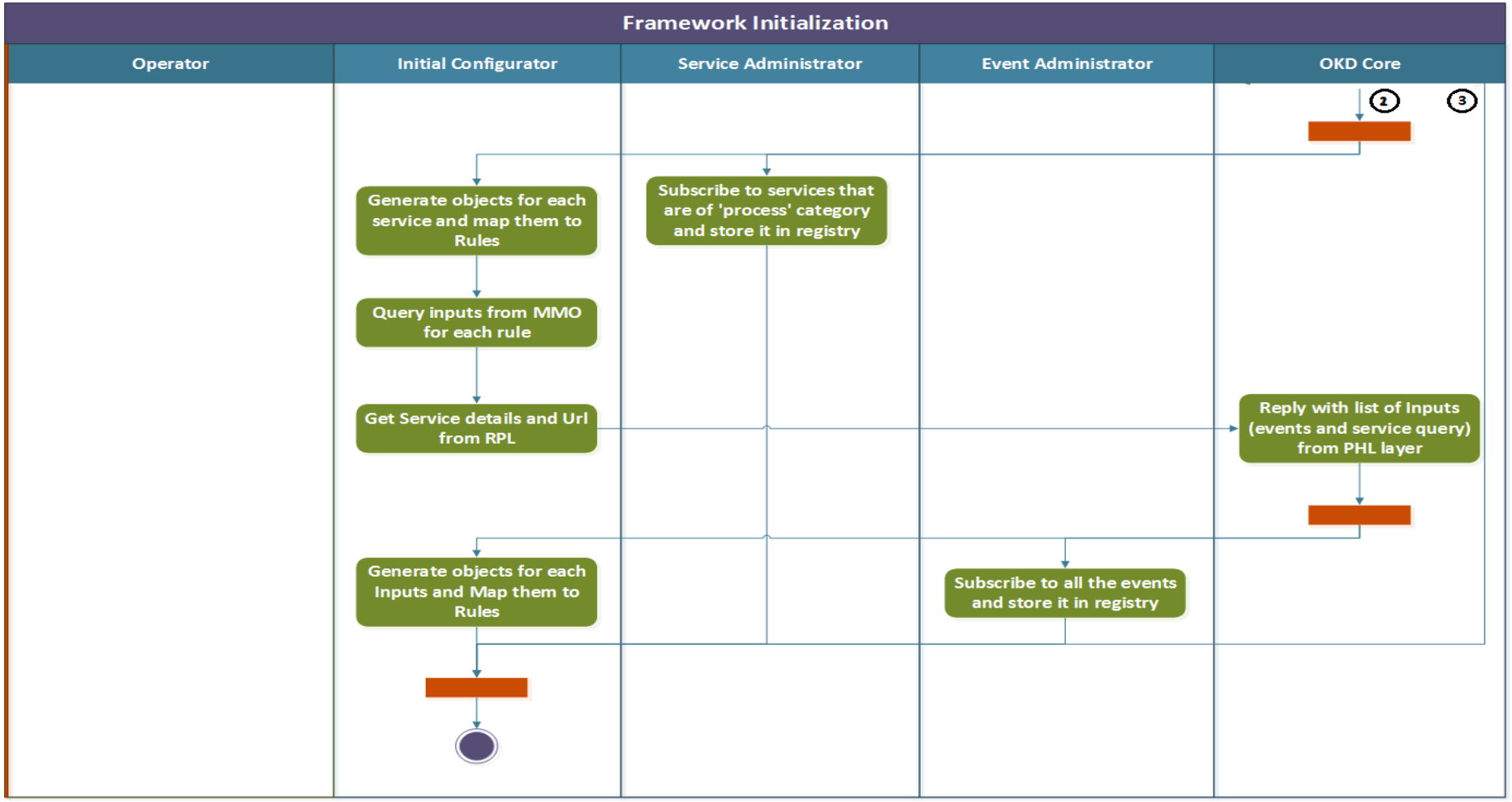

Figure 24 Framework Configuration (3)

3.3.3 Framework Rule Triggering

Once the configuration is done, the Framework starts triggering the rules. The rules are triggered based on three methods which are explained in this section. Framework detects these three methods based on the inputs for each rule.

3.3.3.1 Event Based Triggering

Event based rule triggering occurs when an event is generated at the device level. Once the event is generated at the device level, the framework is notified as it has been subscribed to that event (already as part of initial configuration). Then the framework does the following operations as explained in the sequence diagram Figure 25.

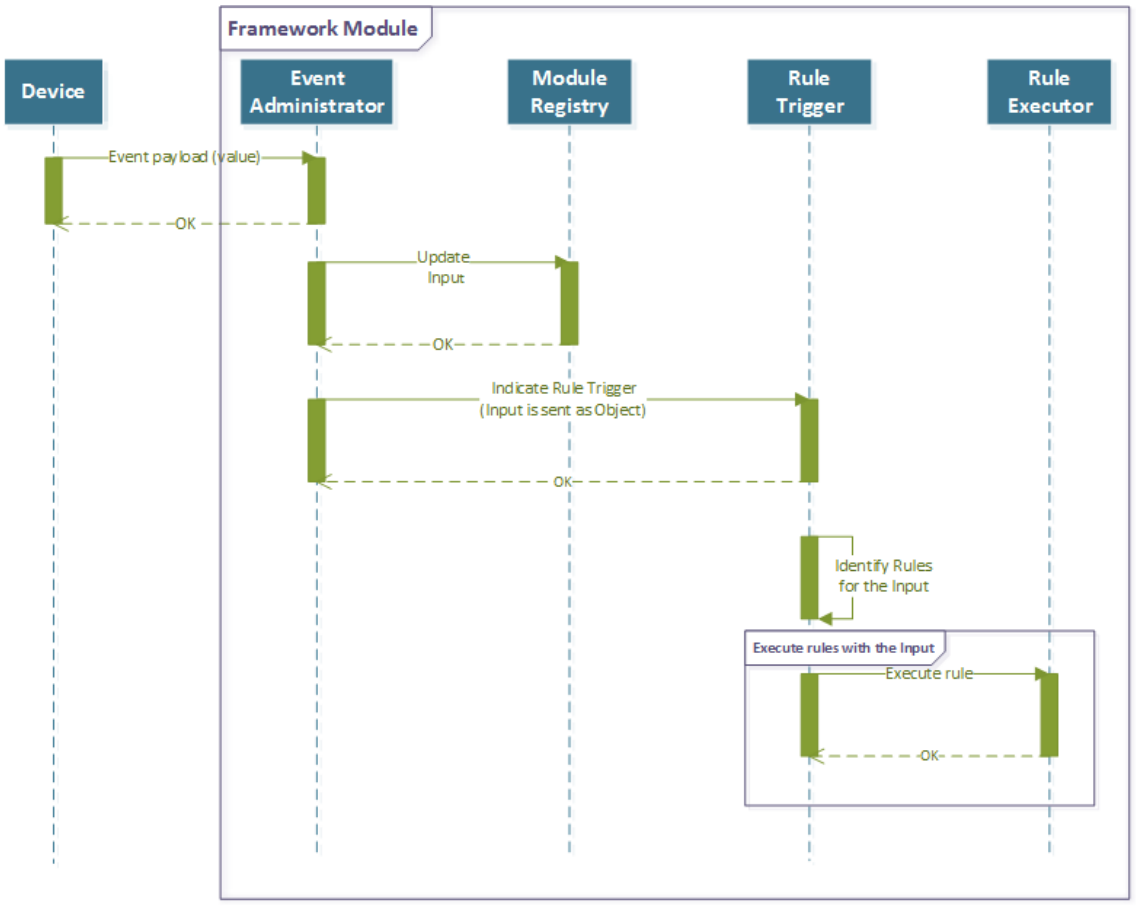

Figure 25 Event Based Triggering

3.3.3.2 Service Invocation based Triggering

Service invocation based rule triggering occurs when an external device invokes a rule. The Invoke request is actually a RESTful HTTP POST method, where it can also have a value in the body as input. Once the rule has been invoked, the rule will be triggered. Then after the execution of the rule, the result will be sent as the response for the invoked service. This method is particularly useful in cases of client application which doesn't have the option of providing RESTful end points. The operations performed by the

Framework as part of Service Invocation based triggering is explained in the sequence diagram Figure 26.

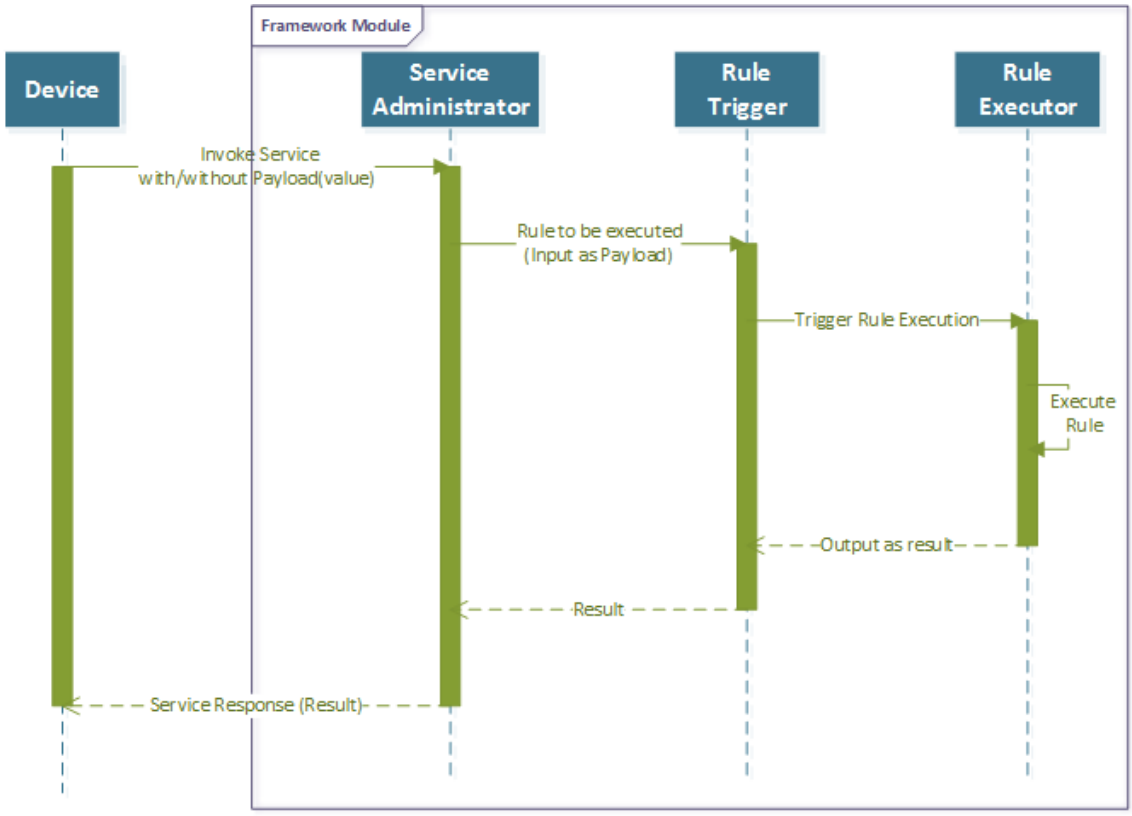

Figure 26 Service Invocation Based Triggering

3.3.3.3 Time Based Triggering

Time based Rule triggering happens only if the rule does not have the above two triggering methods. In time based triggering, the rule is trigged for every second. Once the rule has been triggered and executed, the output is processed as per the rule action. The action may be emitting event or invoking service or saving to ontology.

3.3.4 Framework Rule Execution

Once the Rule has been triggered by any one of the methods as explained in Section 3.3.3, the rule will be executed by the framework and the result will be processed as per the rule's action. The actions associated with the rule are user defined and are gathered from the Ontology by the framework. The means how the framework performs the action are explained in this section.

3.3.4.1 Event Generation

Event generation is performed by the framework when the action for the rule is 'Event'. Once the rule is executed, the rule executor informs the event administrator that an event has been generated. Then the event administrator enquires the subscription list for the output from the module registry and posts the vent to the respective subscribers. The

posting is done via RESTful HTTP POST method, where the event value is sent as the body of the method. The sequence of steps are explained in the Figure 27.

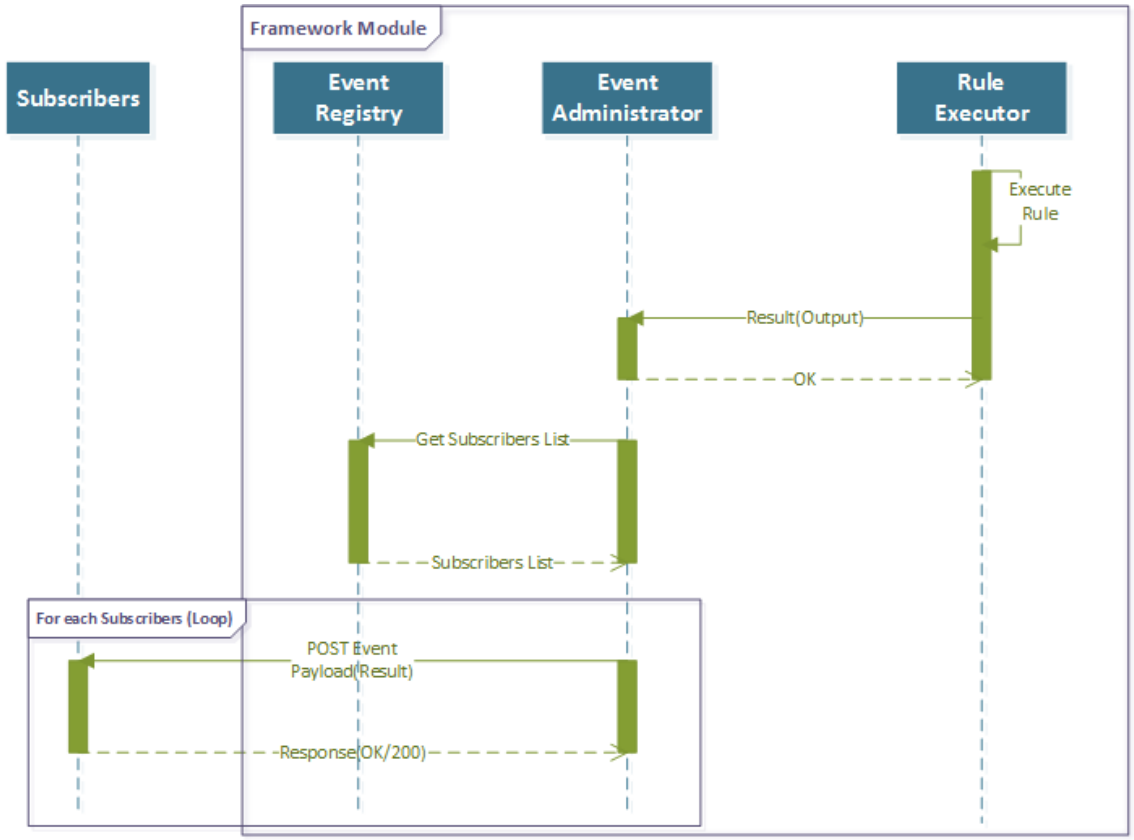

Figure 27 Event Generation

3.3.4.2 Service Reply

Service Reply is performed by the framework when the action for the rule is 'ServiceReply'. Initially, the rule will be triggered as explained in Section 3.3.3.2. Once the rule is executed, the rule executor informs the service administrator with result. Then the service administrator sends the result as part of response the web service which invoked the rule. The invoking is done via RESTful HTTP POST method, where it might have an input as body of the method. Once the result is generated, the result is sent as response body to the same method. The sequence of steps are explained in the Figure 26.

3.3.4.3 Service Invocation

Service Invocation is performed by the framework when the action for the rule is 'ServiceInvocation'. Initially, the rule will be triggered by any one of the method as explained in section 3.3.3. Once the rule is executed, the rule executor informs the service administrator with result in the format mentioned in Table 6 Output Value Format. Then the service administrator verifies if the 'trigger' value in result is true. If true, it invokes the service and sends the 'message' value as body of the service invocation. The invoking is done via RESTful HTTP POST method. The sequence of steps are explained in the Figure 28.

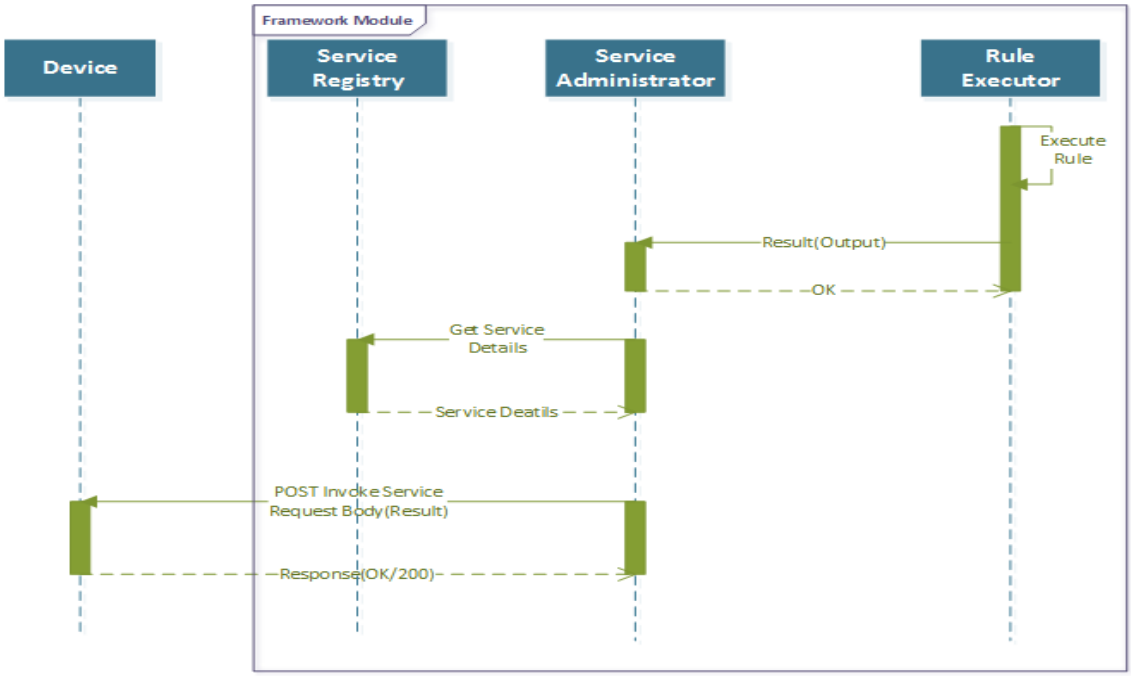

Figure 28 Service Invocation

3.3.4.4 Saving to Ontology

Saving to Ontology is performed by the framework when the action for the rule is 'OntologySave'. Initially, the rule will be triggered by any one of the method as explained in section 3.3.3. Once the rule is executed, the rule executor informs the ontology administrator with result. Then the ontology administrator asks the SPARQL Factory to generate an SPARQL expression for the result. Once the SPARQL Factory supplies the expression, the ontology administrator adds a new entry in the fuseki server with SPARQL Update. The sequence of steps are explained in the Figure 29.

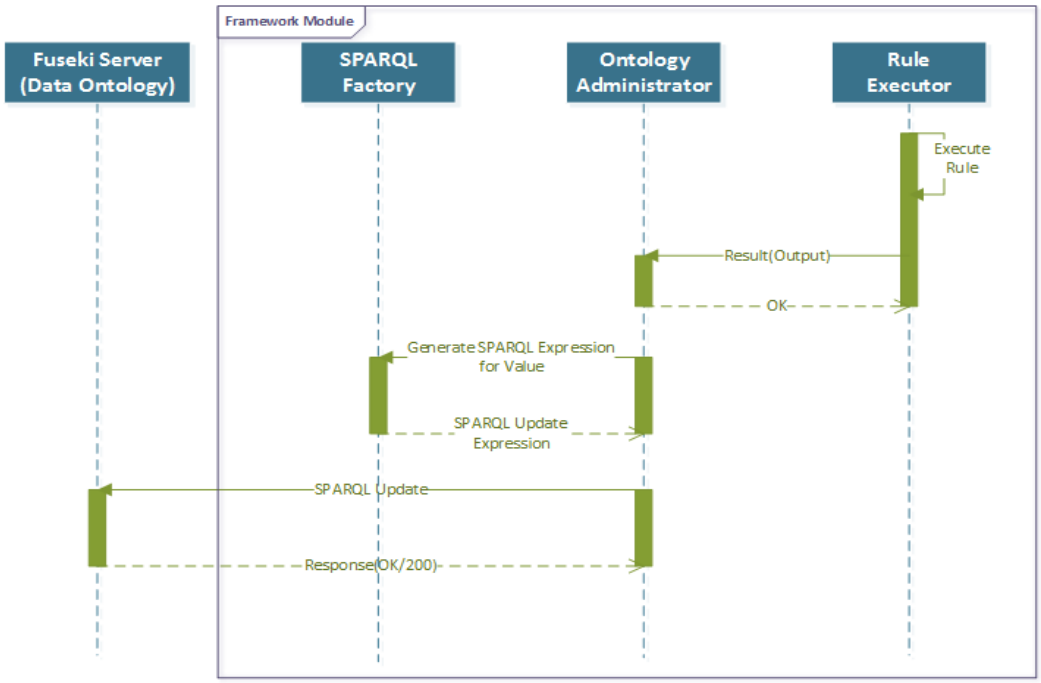

Figure 29 Save to Ontology

3.4 Tools

3.4.1 OWL development tool

OWL is one of the most formal languages for ontology. Hence the ontology part in this methodology was built with OWL. Olingvo [64], an ontology editor tool shown in Figure 30 Olingvo was used to develop the OWL files. Olingvo tool is developed in FAST laboratory of Tampere University of Technology. It is a modular tool with easy to use interfaces for editing and forming the ontologies. As stated before, the ontology reasoning has been planned to be executed by means of SPARQL queries.

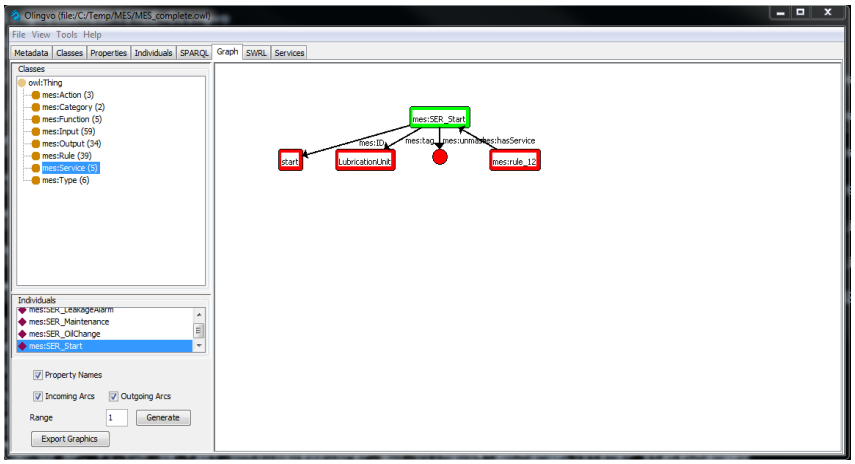

Figure 30 Olingvo tool

3.4.2 Framework development tools

From the development point of view, the framework is divided into two. One is the development of the framework module and the other is the Fuseki Server. Fuseki Server needs no programming; since it is the Jena Fuseki server which hosts the Owl files developed using the Olingvo Tool. The reason for choosing Jena Fuseki is it being a SPARQL server which is capable of providing REST style SPARQL HTTP Update and query[65]. Version 1.1.2 of Jena Fuseki was used in this thesis. Initially the Fuseki was installed in a server and then the ontologies were loaded into the server through the web interface provided by Jena Fuseki.

On the other hand, the Framework module was developed as a RESTful web service based application. The development was done using technologies like Java, Maven and Spring MVC. Java is a general purpose object oriented programming language which can be used to build modular software tools. The biggest advantage of Java is its platform independency. Maven on the other hand is a project management medium based on the Project object model (POM) developed by Apache. It holds a central piece of information in a POM file through which it can manage the whole project from build to reporting[66].

Spring MVC (Model View Controller) is a modular web development framework which helps in building RESTful websites and applications easily. Combining all three, gives the possibility to develop and implement this methodology for the current trend.

4. USE CASE DEFINITION

4.1 Oil Lubrication System Simulator

In Manufacturing industries, fluid automation plays an important role in many sections. Of all the sections, one important section is lubrication of heavy machinery. Heavy machinery, which constitute of large number of mechanical components, are tend to wear and become rusty depending on the usage. Hence in order to overcome these, all the mechanical moving parts in the machinery are lubricated with liquids.

Circulating oil lubrication method is one such method, which is most commonly used in lubricating heavy machinery. In this method, the oil is initially pumped from the tank into the machinery for lubrication purpose. Once the points are lubricated, the oil returns to tank where it is filtered and cooled down. Then the process repeats again. Since, this is an important piece of machinery which secures long and smooth running of other heavy machinery, monitoring this becomes an important aspect. Hence this has been chosen as a medium for testing the framework.

Usually, the lubrication is a long run process; hence if this system is to be monitored, it takes a longer duration. This is where the simulator's come into play. Simulator is a device/software, which does the simulation of a real world process over time. In here too, a simulator has been used for the Oil Lubrication System in-order to simulate the real system. The Oil Lubrication System (OLS) simulator has been developed as part of eScop project in Tampere University of Technology and it is available as an open source for study and research purpose[57].

Figure 31 Oil Lubrication System Simulator

The simulator has many special features like,

- **Customization:** The simulator allows to create any number of sub components (Measuring Station, Flow Meter) with the required specs.
- **Time based Simulation:** The simulator has the option to fast forward the simulation to a particular period of time.
- **Monitoring and HMI:** The simulator has a monitoring and HMI. The monitoring window displays the sensor vales and also has the options for time related controls. While the HMI displays the alarms and has the options to simulate scenarios like changing oil, filter and doing maintenance. The monitoring interface with the fast forwarding option is shown in Figure 32 and the HMI is shown in Figure 33.

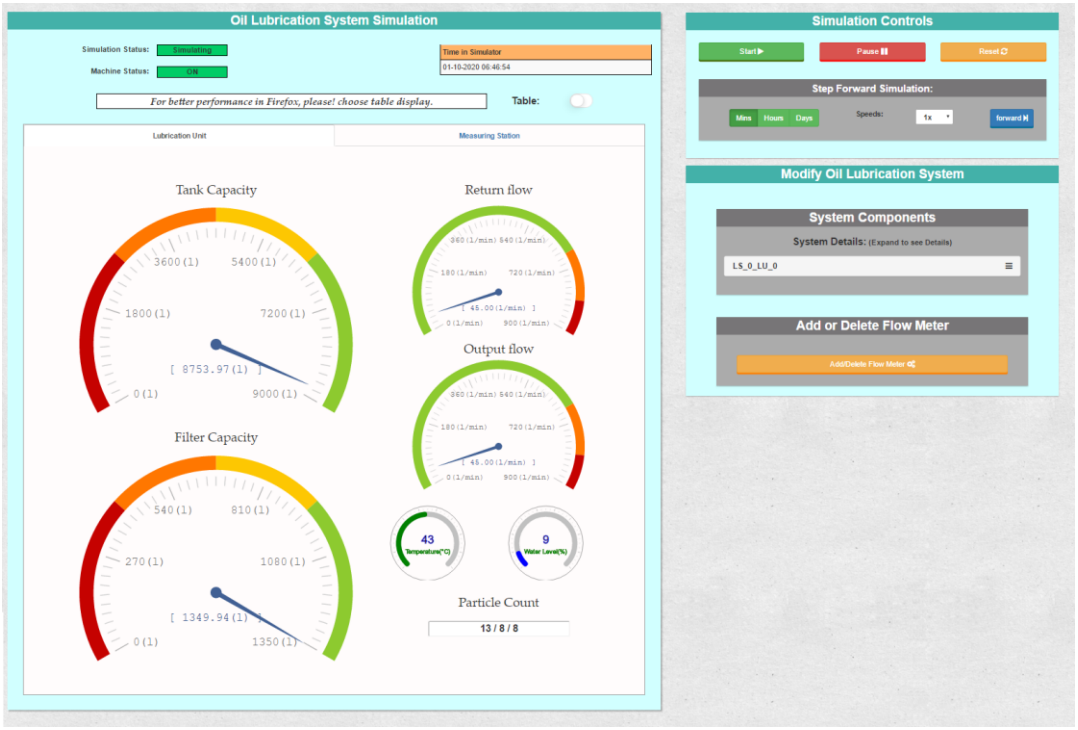

Figure 32 OLS Simulator Monitoring Screen

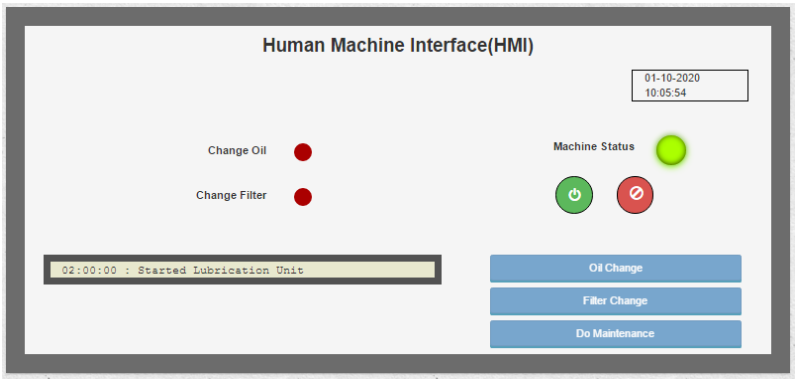

Figure 33 OLS Simulator HMI

OLS Simulator is available online[68] and the source files are available in gitlab[69] as open source. For the purpose of this thesis, the simulator was run in the local computer with the source from gitlab.

4.2 Oil Lubrication System Production Industry

Oil Lubrication System production industry is the place where the lubrication systems are produced. Fluidhouse is one such company which produces Oil Lubrication Systems[70]. This use case is particularly inspired from the Fluidhouse production environment. The framework was tested for its capabilities as Manufacturing Execution System by deploying in this scenario.

Circulating Oil Lubrication System varies in size depending upon the application. Usually, they are available with tank capacity of more than 100 litres. Due to their size and complex assembly, they are generally manufactured by human labour. The production environment which consists of human labour has many legacy systems in the shop floor level and the enterprise level. The complete architecture of the production environment is given in the Figure 34.

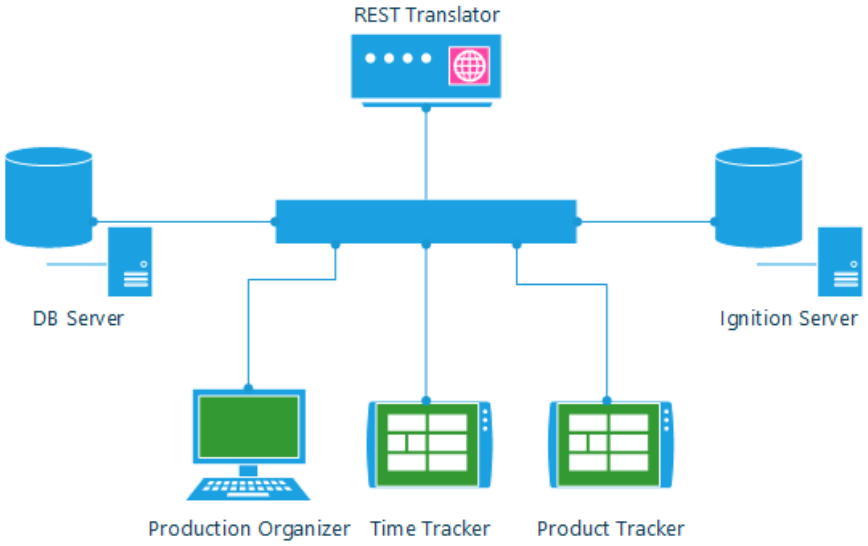

Figure 34 OLS Production Industry Architecture

In the production environment, while an order is received for Lubrication System, it is received in quantities more than one. Hence each order is referred to a project and subprojects are allocated for each individual elements in that quantity. The production environment consists of legacy client applications which are developed with Ignition SCADA and Java. The legacy systems are discussed below.

DB Server

DB (Data Base) Server is an MS SQL server which is hold all the data with respect to the production environment. Production Organizer, Time Tracker and Product Tracker are

the systems that store and retrieve the data. While the REST Translator just retrieves the data and provides them to the external systems.

Ignition Server

Ignition Server is a SCADA based system from Inductive Automation. This is a server based system which enables the user to create the multiple client applications for enterprise applications.

Production Organizer

Production Organizer represents the user-end of the Ignition Server in the form of Ignition Clients. The main function of the production organizer is to register a project, allocate resources, design the project, place order for the project, etc.

Time Tracker

Time Tracker is the application which are hosted in Android tablets with NFC support. An application that uses employee's RFID cards and tags to register and login users. Once the user logs into the application, the user then can manage his times with respect to each project.

Product Tracker

Product Tracker is a Java based client app available in the Android tablet's. It is used to register the NFC tag of the product to the respective project. Later, when the registered tag is viewed at the shop floor, it gives the complete details of the project.

REST Translator

The REST translator is a web-service gateway of sorts. It is a REST-based system that translates between the Database server and other external systems. The RESTful Services available from the translator are shown in Figure 35.

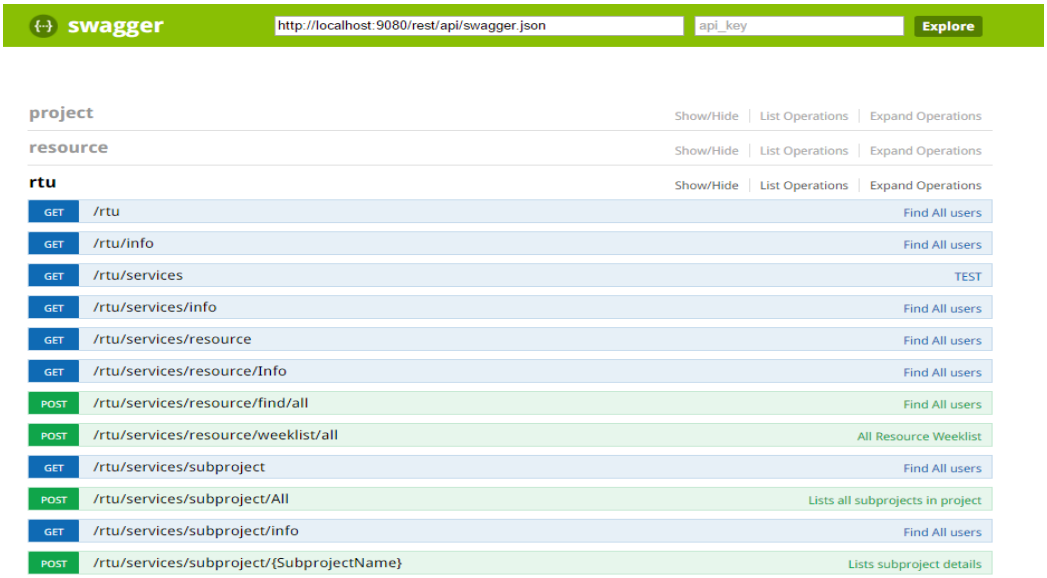

Figure 35 Swagger REST Translator Interface

5. IMPLEMENTATION

5.1 ESCOP and Framework Interactions

The Framework has to work with an OKD system in order to attain its functionality. In this thesis, the OKD-MES components from the eScop project has been chosen as that system. The OKD-MES layers has already been discussed in section 2.4. Out of the layers, only PHL and RPL has been used by the framework to attain its functionality. In which, the PHL layer is either the OLS simulator or the OLS production environment. while the RPL helps in discovery and searching of events and services in both Framework and PHL layer. The sequence of operation performed as part of discovering events and services is explained in sequence diagram shown in Figure 36.

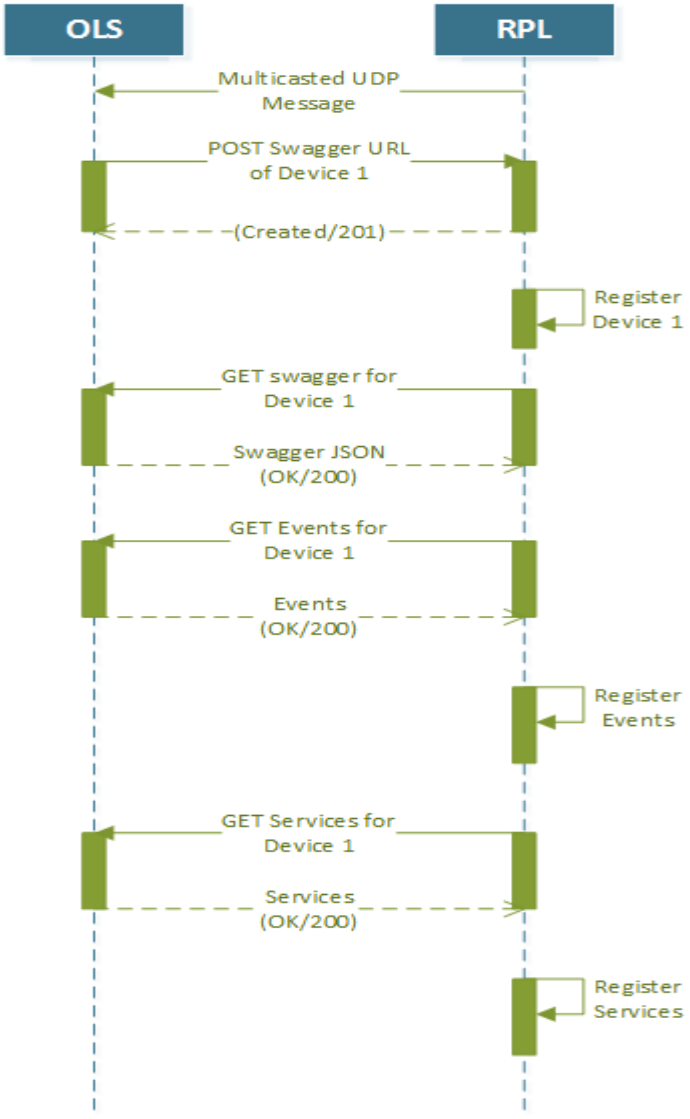

Figure 36 RPL Discovery Operation

Once all the events, services, devices have been discovered by the RPL, it registers them and provides their information based on request. The event/service/device can be searched in the RPL with the help of tags called ‘Meta’. The ‘Meta’ tags will be available as part of each event, service and device as shown in the Figure 37. This Meta key value pairs will be used by the Framework while searching for a particular event/service/device in the RPL. A manual search for a particular event in the RPL with Meta key value pairs and its respective result are shown in the Figure 38.

Figure 37 Meta key value pairs in PHL events

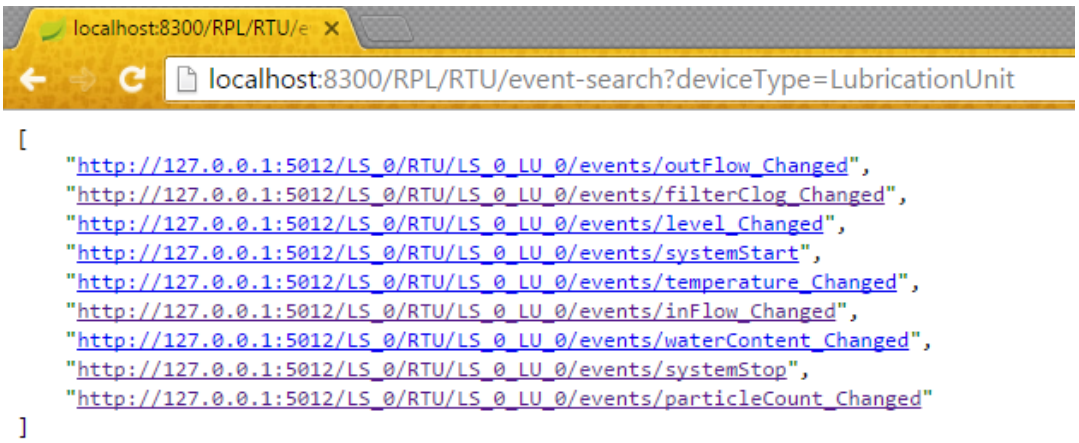

Figure 38 RPL Meta search

5.2 Use Case 1: Oil Lubrication System Simulator

Oil lubrication system simulator has to be initially configured with Lubrication Unit specs and components (Measuring Station and Flow Meter) for each lubrication unit. Once the configurations are done, the simulation can then be started. Hence, as a first step towards this use case implementation, the OLS Simulator was configured with the following setup as in Table 10.

Table 10 OLS Simulator configuration specifications

Lubrication Unit:		
• **Nos**	-	1
• **Tank Capacity**	-	4000 l
• **Maximum Flow**	-	400 l/min
• **Filter Capacity**	-	550 l
Measuring Station:		
• **Nos**	-	2
Flow Meter:		
• **Nos**	-	20
• **FM in each MS**	-	10
• **Maximum Flow**	-	20 l/min
• **Nominal Flow**	-	15 l/min
• **Minimum Flow**	-	5 l/min

Once the OLS simulator is configured, next the Framework should be configured for the implementation. The framework is implemented with 8 functions, as discussed in this section.

5.2.1 Data Acquisition

Data Acquisition (DA) function performs the role of collecting the data and saving them to ontology for future use. They also process the data to the required format before storing them to ontology. This is the major function which updates the data ontology. It will be triggered based on the events that are generated from the OLS simulator. Once triggered they process the event value to the required format and store them to ontology. The rules which assist the function are given in Appendix B.

The Sequence of steps in which it performs the operation is as below,

- Initially the OLS simulator emits the generated event. The events which the framework listen in this part are,
    - Outflow of LU - Output Oil Flow of Lubrication Unit.

- Oil Changed - when operator inputs about the action of oil change via HMI.
- Filter changed - when operator inputs about the action of filter change via HMI.
- Maintenance done - when operator inputs about the action of maintenance done via HMI.

- Then the corresponding DA rule for the event is triggered.
- Once triggered, the DA executes the rules and contacts Ontology manager to update the details to Ontology.
- The ontology Manager then generates the SPARQL Update query and updates the details to data ontology in Fuseki Server.

A sample of how the process is performed in sequence can be found in the Figure 39.

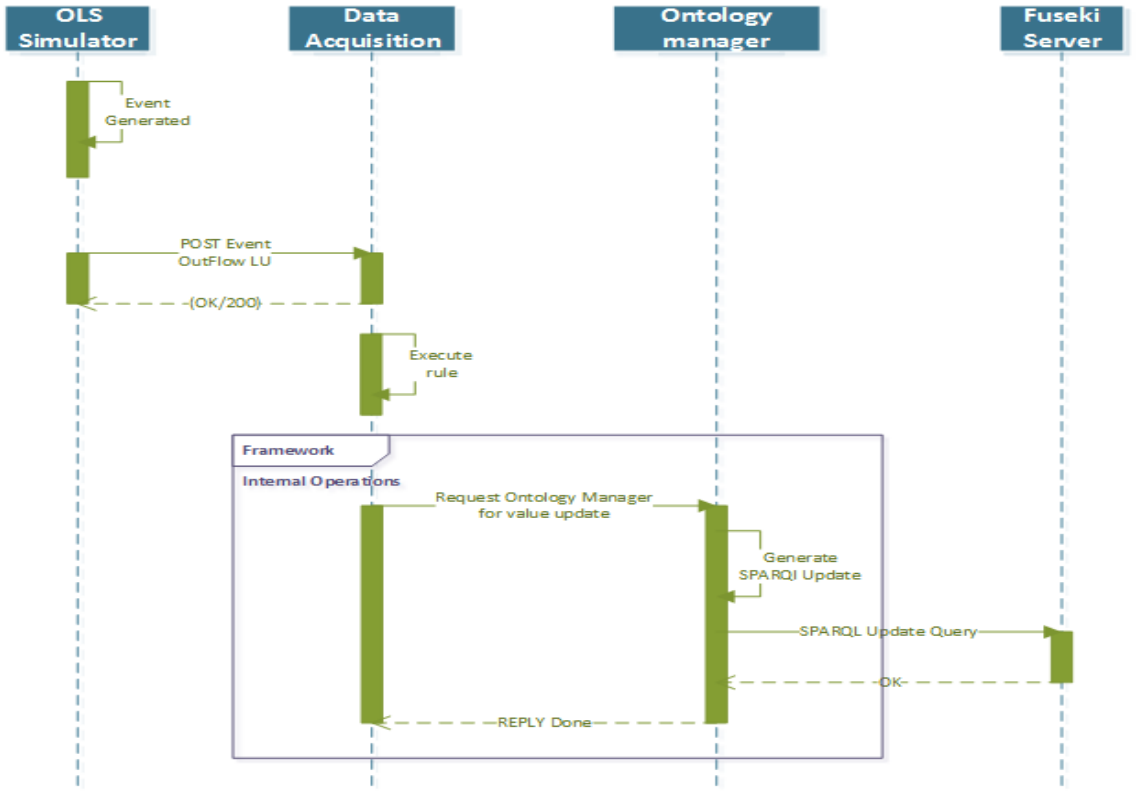

Figure 39 Data Acquisition Sequence

5.2.2 Condition Monitoring

Condition Monitoring (CM) function performs the operation of pre-screening the details of LU and checking if all the condition exists for the proper functioning of LU. If condition doesn't exist, then the function stops the lubrication unit. It checks for the condition, only while the system is being started. The basic conditions that must exist to start a lubrication system is that,

- There must not be any alarm trigged in the system.
- The Level of oil in the system must be at least 95% of tank capacity.
- The oil allocated for sub components must be less the maximum allowed oil flow.

The Rule for this is given in Appendix B. The Condition Monitoring does this in a series of steps,

- The LU start event is raised, when the system is started. The event then triggers CM function.
- CM gets the necessary details from the OLS Simulator.
- Then the rule is processed which checks the system if the necessary conditions exist. If the necessary condition does not exit it will raise the LU stop alarm event.
- This alarm then stops the LU via System Controller function and passes the Message to HMI via HMI Controller function.

The steps have been explained in the sequence diagram shown in Figure 40.

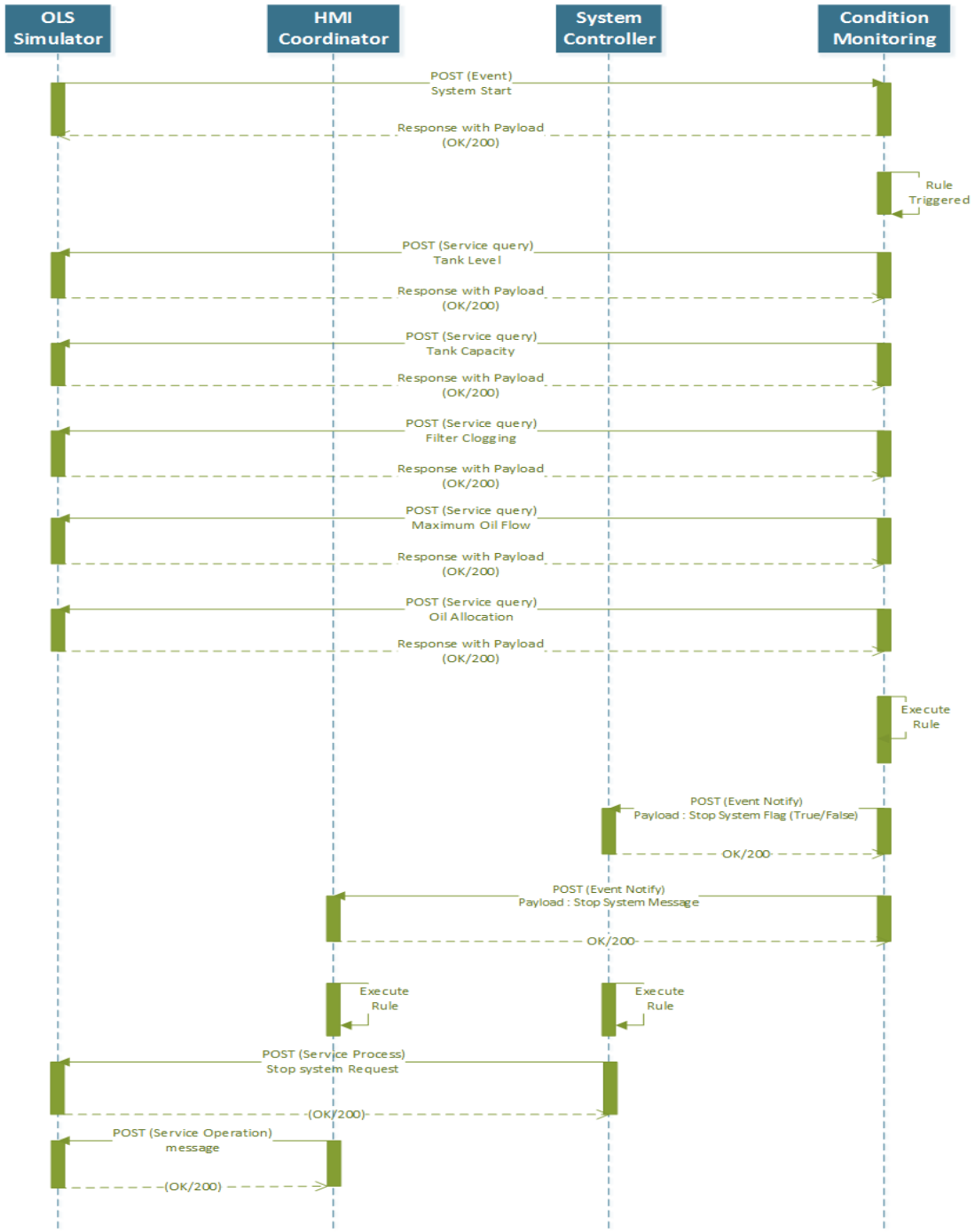

Figure 40 Condition Monitoring function sequence

5.2.3 Quality Monitoring

Quality Monitoring (QM) function performs the operation of monitoring the quality of oil and raising the alarm to change oil, if the quality is low. The complete logic depends upon the filter changing time frequency. If the frequency (no of days between a filter change) becomes low and the oil particle count reaches a specified limit, the flag is raised by QM. Once the flag is raised, the HMI Coordinator invokes the alarm and sends message to HMI. The rules which does the above functionality are specified in Appendix B. The sequence of operation performed by QM are explained in the Figure 41.

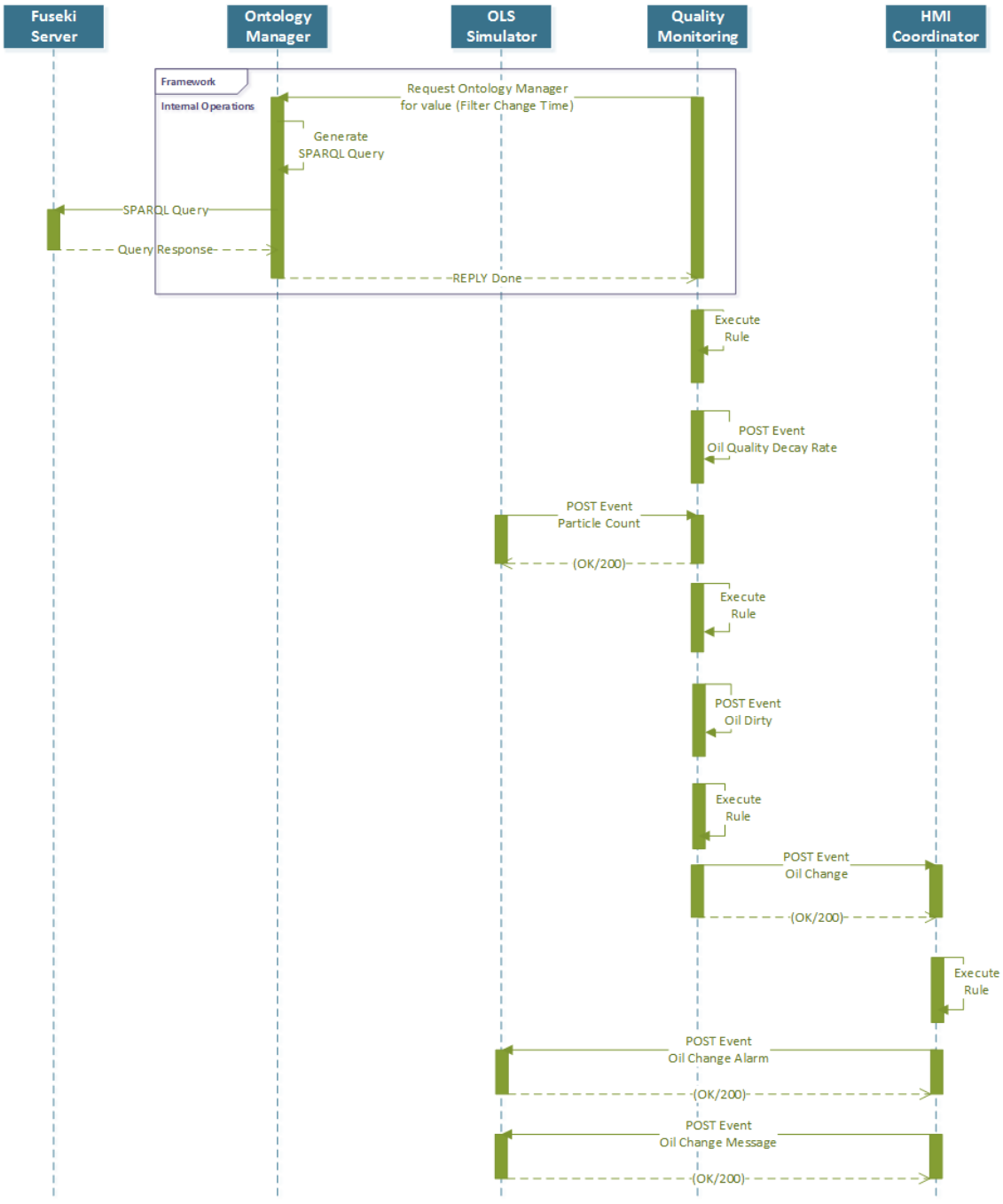

Figure 41 Quality Monitoring function sequence

5.2.4 Process Monitoring

Process Monitoring (PM) function performs the operation of monitoring the whole process. It monitors the oil level and also generates the oil consumption ratio for each component. When the oil level becomes lower than 50%, the oil change flag is raised by PM. Once the flag is raised, the HMI Coordinator invokes the alarm and sends message to HMI. It also stops the LU by invoking service via System controller. The rules which does the above functionality are specified in Appendix B. The sequence of operation performed by PM are explained in the Figure 42 and Figure 41.

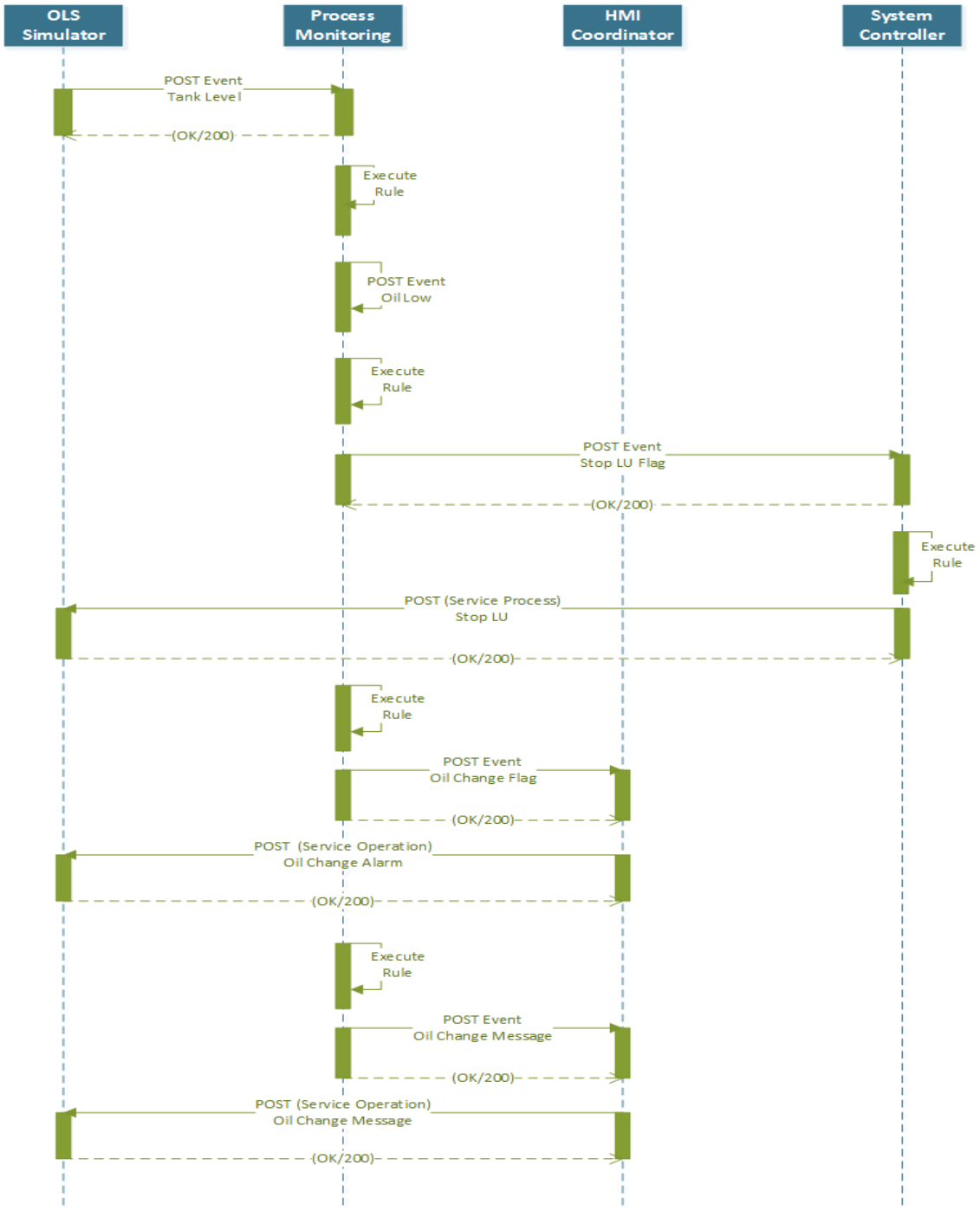

Figure 42 Process Monitoring function sequence

5.2.5 Immediate Maintenance

Immediate maintenance function performs the operation of monitoring the leakage in the components and raising the alarm for the same if there is leakage. It monitors the components based on the oil Consumption ratio (OCR) values generated by the Process monitoring function. If the OCR is more than 5%, then it will generate the leakage alarm. Once the leakage alarm is raised, the HMI Coordinator passes the message to the HMI screen mentioning the leakage in specific component. The rules which does the above functionality are specified in Appendix B. The sequence of operation performed by Immediate maintenance are explained in the Figure 43 and Figure 41.

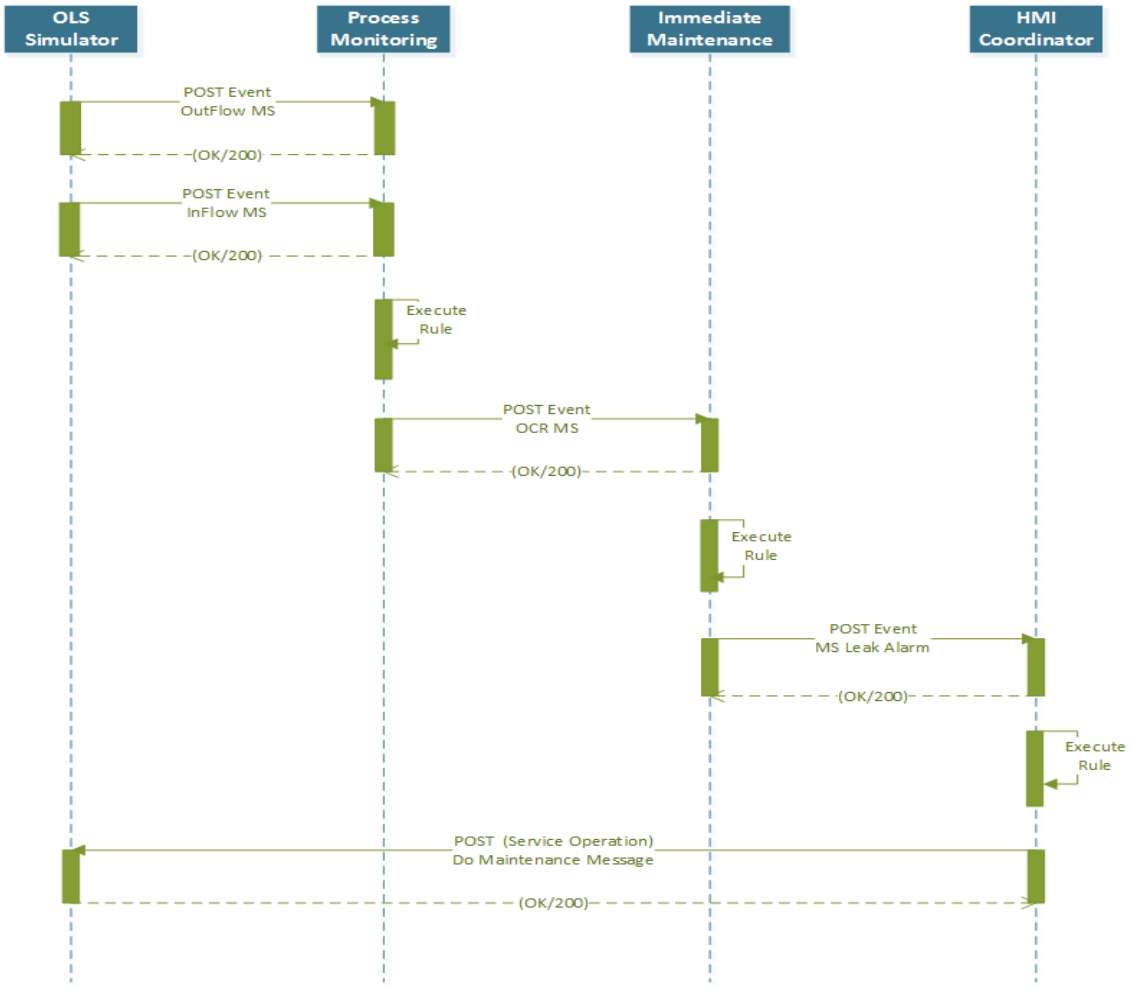

Figure 43 Immediate maintenance function sequence

5.2.6 System Controller

System Controller performs the function of invoking the stop operations in the OLS simulator. Based on the stop flag raised by other functions, the System controller invokes the service in the simulator to stop LU. The rules for this function are specified in Appendix B. Its usage and sequence of operation can be seen in Figure 40 and Figure 42.

5.2.7 Predictive Maintenance

Predictive Maintenance function performs the operation of scheduling periodic maintenance and also updating the schedules or the maintenance. It performs both the operation of preventive and predictive maintenance. During the initial stages, the function schedules the maintenance based on the inputs provided initially. Later, when the oil and filter are changed periodically, it recalculates the scheduled dates and assumes the new date from there on. The rules for this function are specified in Appendix B. The sequence of operation performed by this function are mentioned in Figure 44.

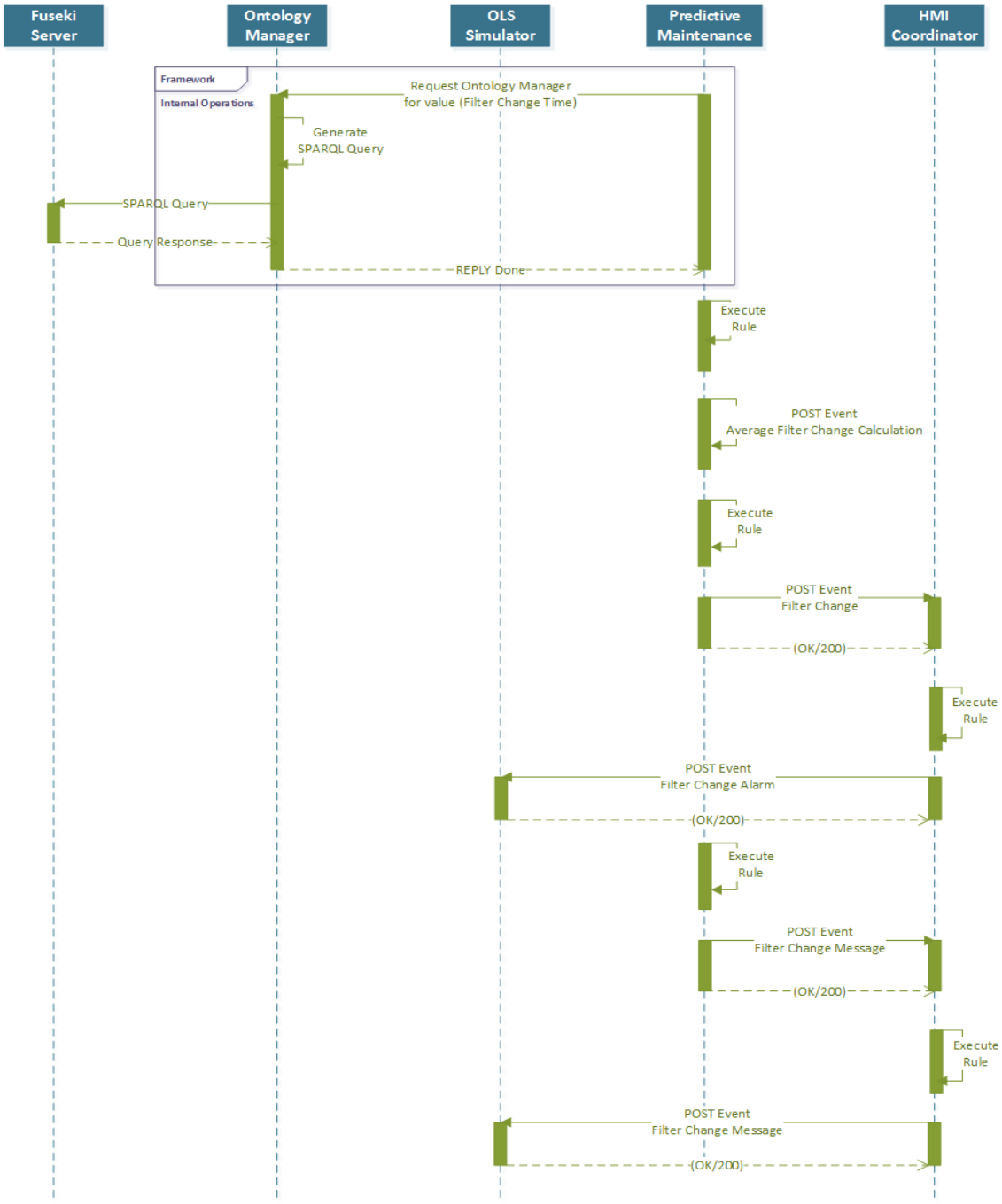

Figure 44 Predictive Maintenance function sequence

5.2.8 HMI Coordinator

HMI Coordinator performs the function of invoking the operations related to HMI in the OLS simulator. It Invokes operations like oil change, filter change and also passes any message to be displayed in HMI screen. The rules which does the above functionality are specified in Appendix B. Its usage and sequence of operation can be seen in Figure 40, Figure 41, Figure 42, Figure 43 and Figure 44.

5.2.9 Key Performance Indicators

In this part of the usecase, the metrics performed as part of KPI are related much towards monitoring the condition of machine. The availability metrics starts with condition monitoring of the device in the Section 5.2.2, where it secures the condition of the machine buy gathering the required informations about the OLS and determines its availability. Similarly, the performace evaluation metrics is performed by the quality monitoring of the device as described in Section 5.2.3, where the quality of the oil is governed to determine the efficiency of the Paper machine. It does this in a way that if the oil quality gets degraded faster, then it can be assumed that the gears in the papermachine are having a heavy wear and it will decrease the efficieny of the device and the paper outcome quality. Initially, the KPI's related to OEE like availability, quality and performance are calculated to monitor the machine. Later, the metrics quality and performance are measured with respect to time as part of TEEP and used for maintenance purposes.

5.3 Use Case 2: Oil Lubrication System Production Industry

In this implementation, the framework is made to attain the functionalities of a MES system. It follows the MESA standards for implementing the MES functionality. The framework, which acts as MES is used in here to support the production related activities; by connecting the legacy client applications in the shop floor level to the enterprise level. The framework has 4 functions as part of this implementation which are explained in this section.

5.3.1 Resource Allocation and Status

The Major function of the Resource allocation and status (RAS) function is to generate a suggested schedule for a project and suggest it to the production organizer application. Initially, the production organizer system sends the request for suggestion to RAS. Once RAS gets the service request, it triggers the rule and then gets the necessary details form the REST translator. Then it executes the rule and sends the suggestion as a reply to the same service which invoked the function. The rule which does this is explained in Appendix C. The sequence of Operation by which the above functionality is achieved is shown in Figure 45.

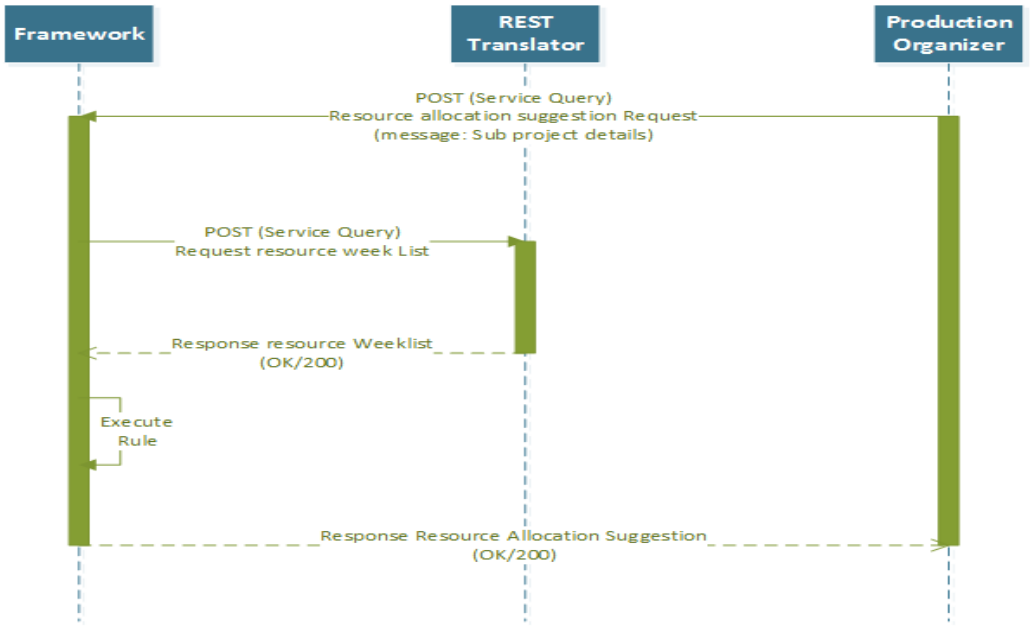

Figure 45 Resource Allocation and Status sequence

5.3.2 Data Collection and Acquisition

Data Collection and Acquisition (DCA) function is used to save the value to the ontology for future use. The sequence of steps is already mentioned in the section 5.2.1. In here the DCA function stores the tagId details of the subproject to ontology. These will be later used by the Product Tracking and Genealogy function for analysing. The rules for this are given in Appendix C.

5.3.3 Labour Management

The Labour management function helps to generate the report for no of hours of work performed by a labour in total and on each sub project. The function mainly operates with the time tracker app. When a user scans his RFID, the time tracker app requests the LM for the employee related details. Then the LM function gets some more information for analysis from REST translator. Once the rule is executed, it sends the analysis as a reply to the same service which invoked the function. The rule which does this is given in Appendix C.

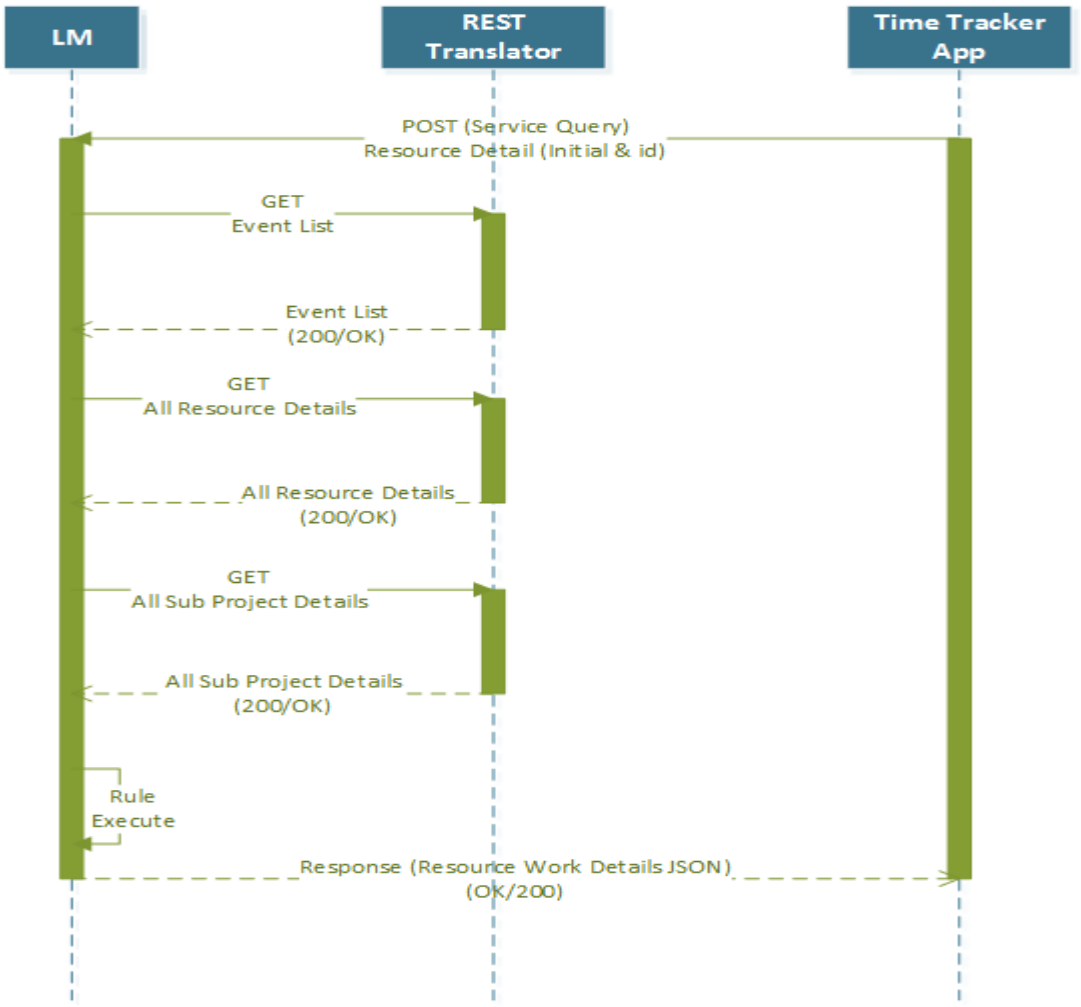

Figure 46 Labour Management Function sequence

5.3.4 Product Tracking and Genealogy

Product Tracking and Genealogy mainly works for identifying the total budgeted time ratio for the sub project at the current time. It does this with the help of the values for event list stored in the ontology. The rule for this is mentioned in Appendix C. The series of steps how this performs the operation is given by,

- The Product Tracker App requests for the current budgeted ratio of a sub project with its tag Id by invoking the service for the same.
- The Product tracking and Genealogy (PTG) rule is triggered.
- It queries the event list and the tag details form the ontology.
- Then it executes the rule and emits the total budgeted sub project details for the current tag as response to the same service which invoked it.
- There may also be other case where all the sub project details may be requested. In that case, PTG function calculates the budgeted ratio for each sub project and send them all as a collection.

The sequence of operation steps is given in Figure 47 and Figure 48.

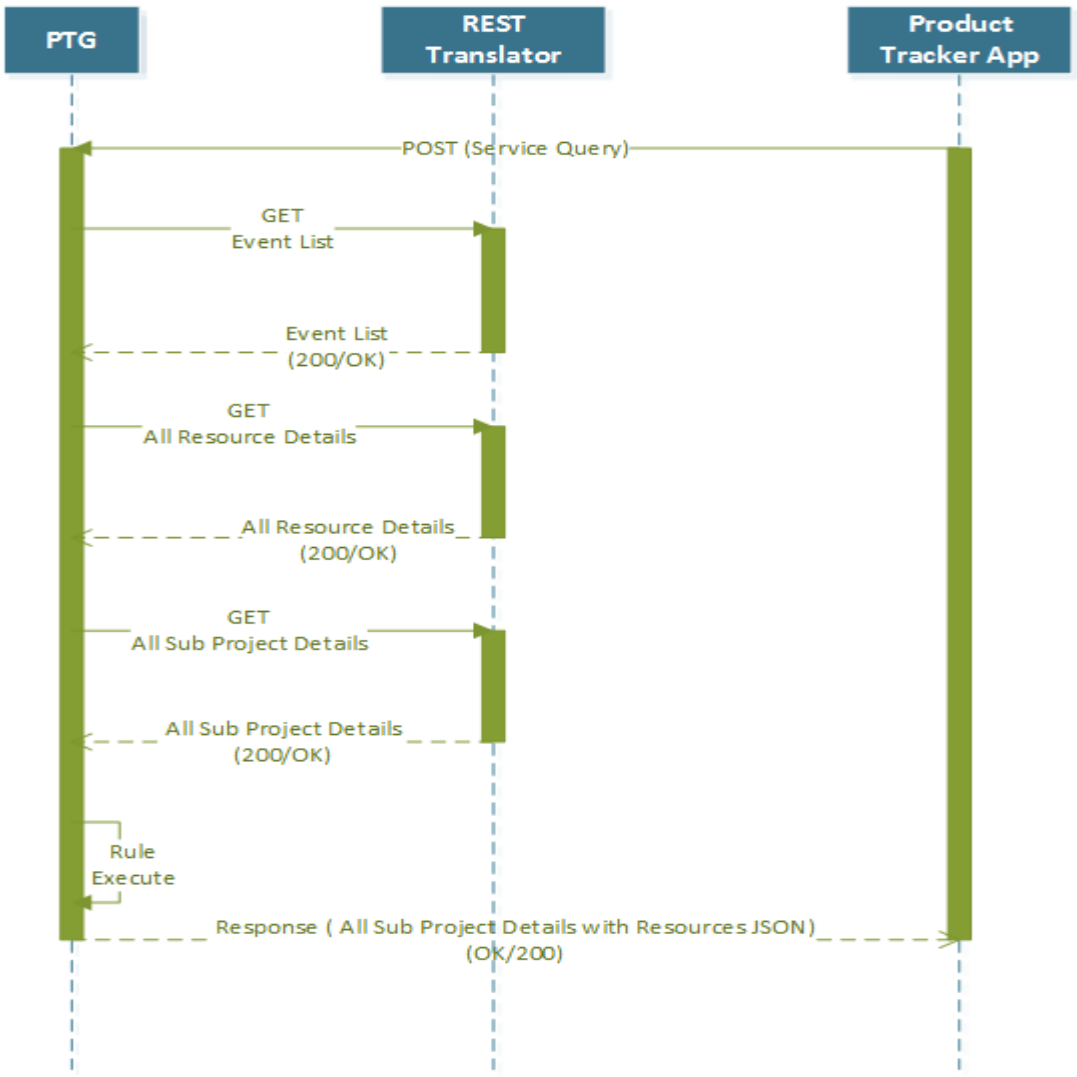

Figure 47 Product Tracking and Genealogy Function Sequence (All sub project)

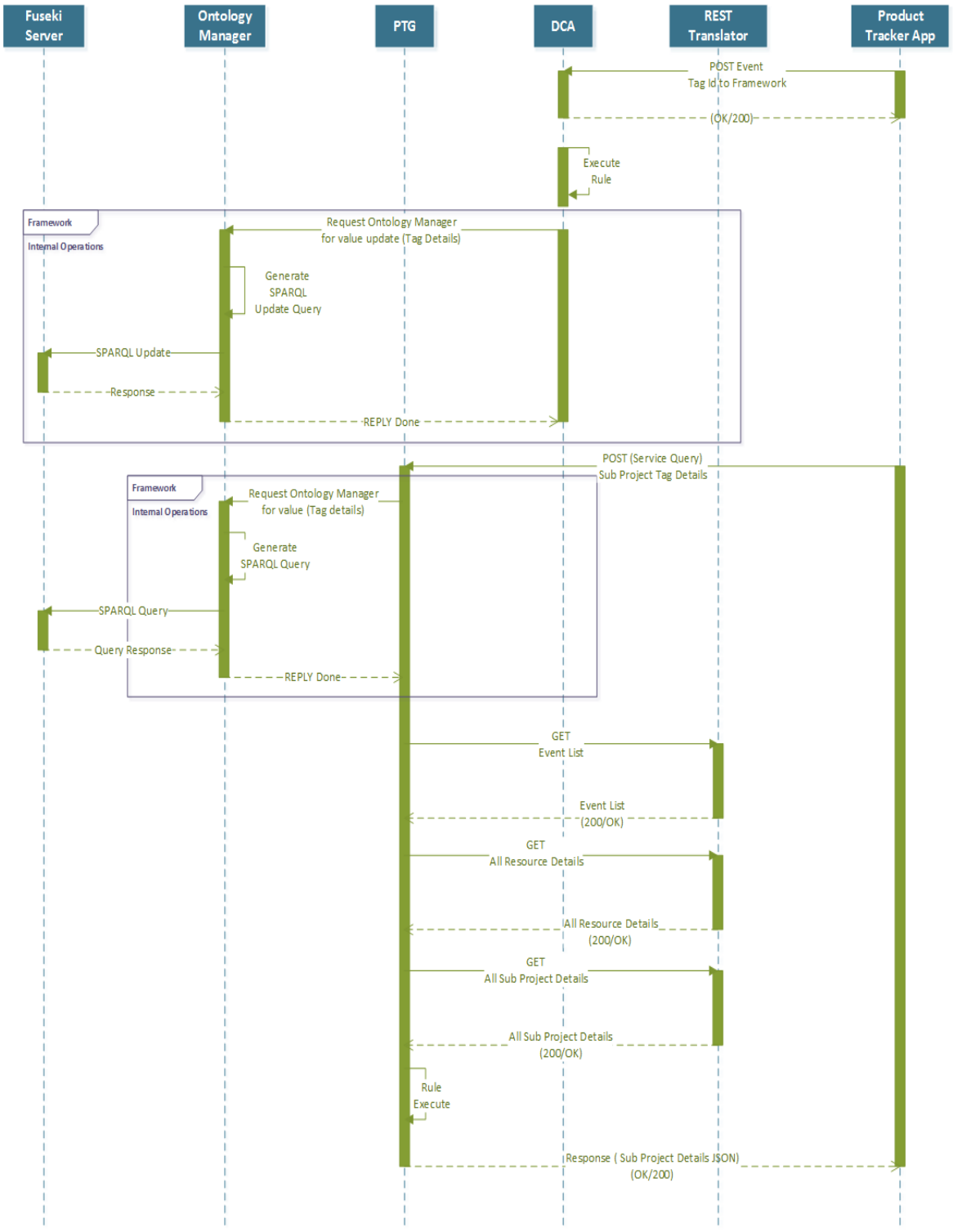

Figure 48 Product Tracking and Genealogy Function Sequence (specific sub project)

5.3.5 Key Performance Indicators

In this part of the usecase, the metrics performed as part of KPI are related much towards the manufacturing of the OLS system. This usecase mainly covers the indexes stated in ISO 22400 report which are listed in the article [46] by Fukuda. The effective allocation degree starts with RAS MES function in the Section 5.3.1, where depending upon the availability and the experience of the operator with the particular production order, they are allocated. This inturn helps in faster, good quality and efficient delivery of the product satisfying the allocation efficiency metrics. Similarly, the workers prodoctivity metrics is performed by the LM functions as described in Sections 5.3.3, in which it determines the performance of the operator with respect to the projects he has been working and also the projects he is working.

6. RESULTS

This chapter discusses about the results that are generated while implementing the methodology in the use cases. The section 6.1 details the results obtained while implementing the framework functions on OLS simulator. While the section 6.2 details the results obtained while implementing the framework as MES on OLS Production Industry. At the later part in section **Error! Reference source not found.** the discussion with respect to the thesis is made.

6.1 OLS Simulator

Initially, the implementation was made as specified in the section 5.2. Later all the components were made to run and the results were taken. In the OLS simulator's, HMI and Monitoring screen were used as the main source for the results representation. The scenario for obtaining the results and the screenshots for results are available in this section.

6.1.1 Condition Monitoring

In the condition monitoring phase, the condition of the Lubrication unit is monitored every time while the system is being started. The conditions and the expected results are already mentioned in section 5.2.2. The functionality was tested by having the following scenario.

A new lubrication Unit was created in the OLS simulator and started. Then in order to test the condition monitoring functionality, the filter is made to be clogged by forwarding the simulation to 30 days using the controls in Monitoring screen as shown in Figure 49.

Figure 49 Simulator simulation time modifying controls

Then the system is stopped and started again. But the Condition Monitoring function stops the system and mentions that filter must be changed to start the system. Only

after the filter has been changed, the condition monitoring function allows to start the Lubrication unit. The complete scenario can be seen from the message log display in the HMI as shown in Figure 50.

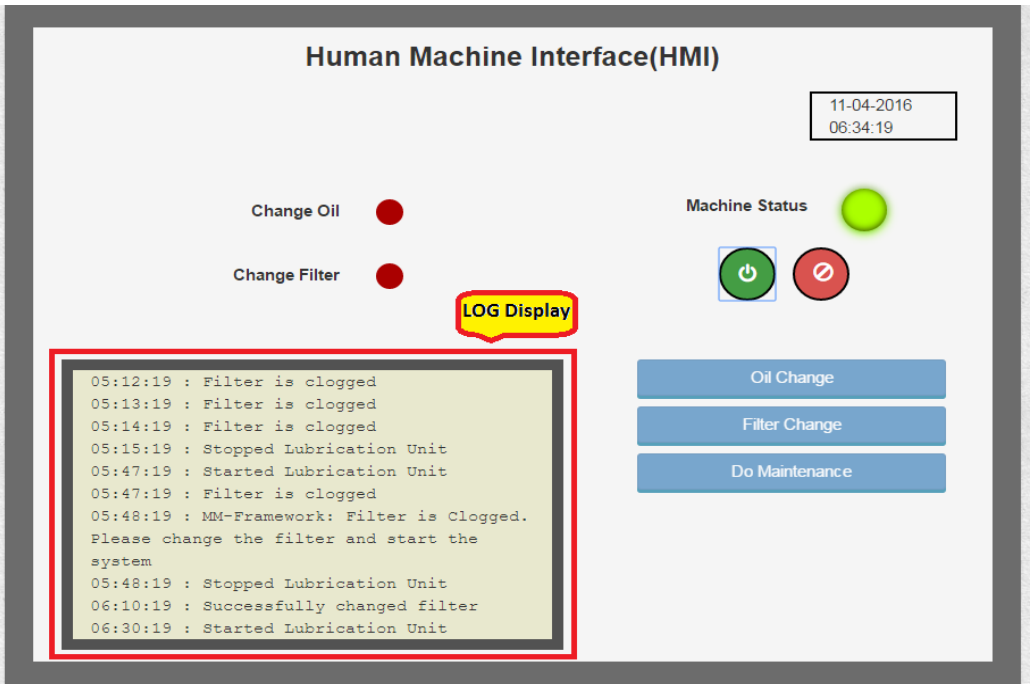

Figure 50 HMI (Condition Monitoring Scenario)

6.1.2 Quality Monitoring

In the quality monitoring (QM) phase, the quality of the oil is monitored. The concept has already been explained in section 5.2.3. The following scenario was used to test the functionality.

A new lubrication Unit was created in the OLS simulator and started. Then in order to test the quality monitoring functionality, Initially the filter is changed for three times leaving some time gaps. This is done for decreasing the frequency of days between each filter changing. Once this is decreased, the QM function will be knowing that something is wrong with the oil by calculating oil quality decay rate. Now it looks at the particle count and when it is more than the specified limit (25/20/18), it raises the oil change alarm and sends the message to the HMI, that the oil is dirty and needs to be changed. The Oil Quality Decay rate calculated by the QM is shown in Figure 51 and the particle count value and the filter capacity during the alarm raised is shown in Figure 52. The complete scenario can be seen from the message log display in the HMI as shown in Figure 53.

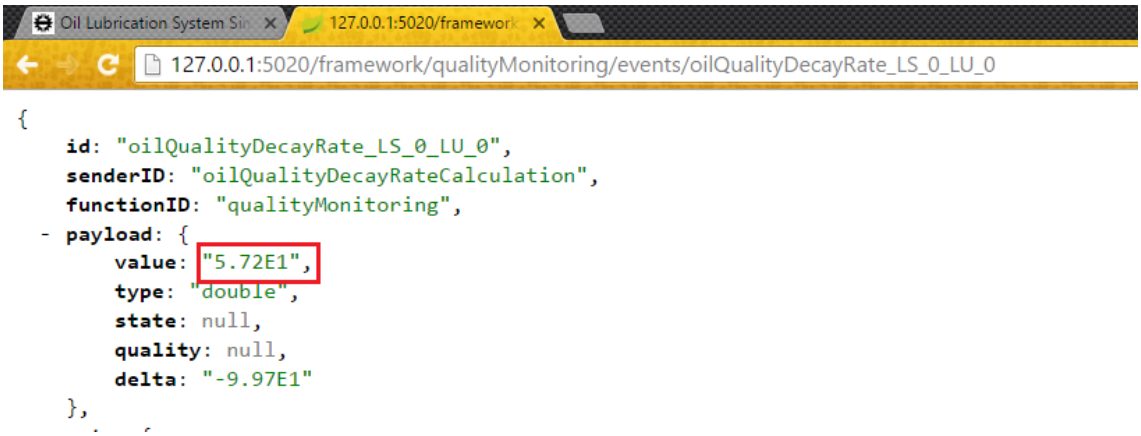

Figure 51 Oil Quality Decay Rate value

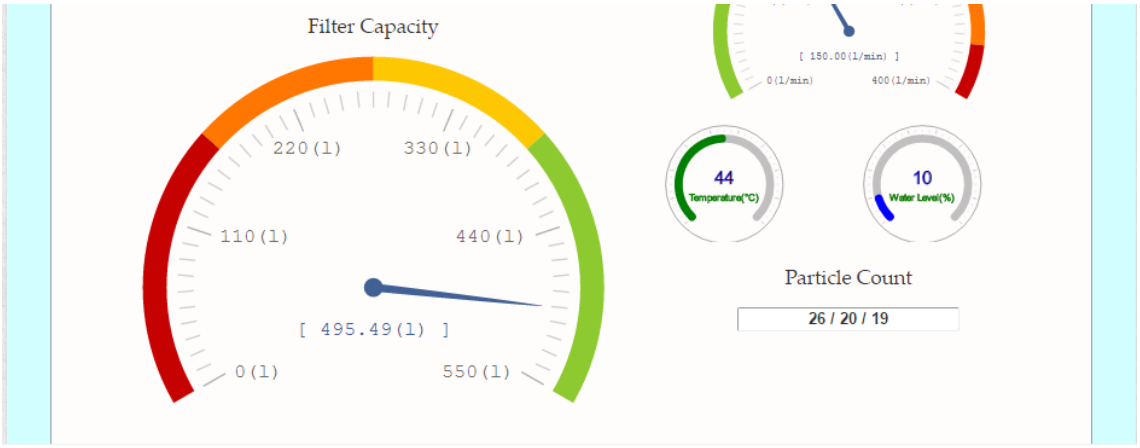

Figure 52 Particle Count and Filter Capacity Values

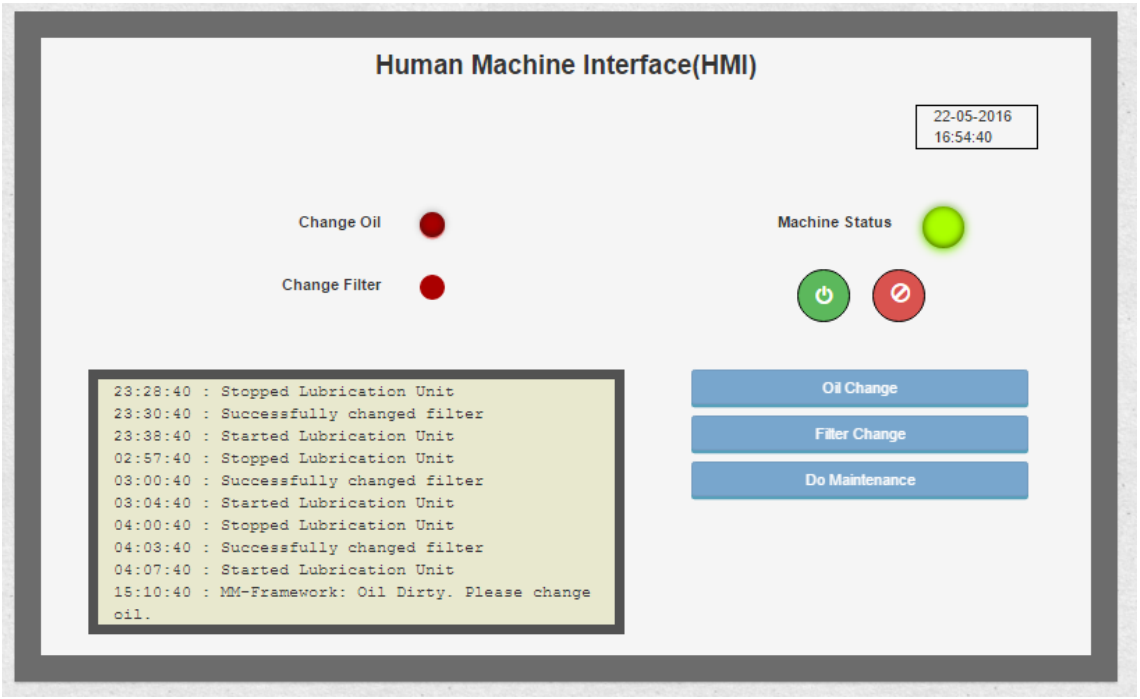

Figure 53 HMI (Quality Monitoring Scenario)

6.1.3 Process Monitoring

In the Process monitoring (PM) phase, the level of oil is monitored. The function also provides the Oil Consumption Ration (OCR) values for each component, which will be used by the Immediate maintenance function. The concept has already been explained in section 5.2.3. The following scenario was used to test the functionality.

A new lubrication Unit was created in the OLS simulator and started. Then in order to test the process monitoring functionality, the simulation is fast forwarded till the oil level becomes 50 % of tank capacity. Once the level is low, PM raises the oil change alarm, stops the system and sends the message to the HMI, that the oil is Low and needs to be changed.

The Oil Consumption Ratio calculated by the PM is shown in Figure 54 and the Oil Level value is shown in Figure 55. The complete scenario can be seen from the message log display in the HMI as shown in Figure 56.

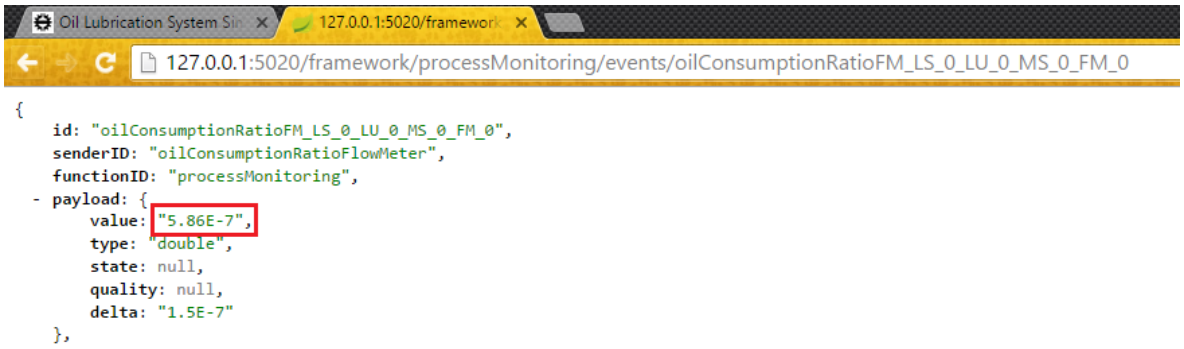

Figure 54Oil Consumption Ratio of Flow meter

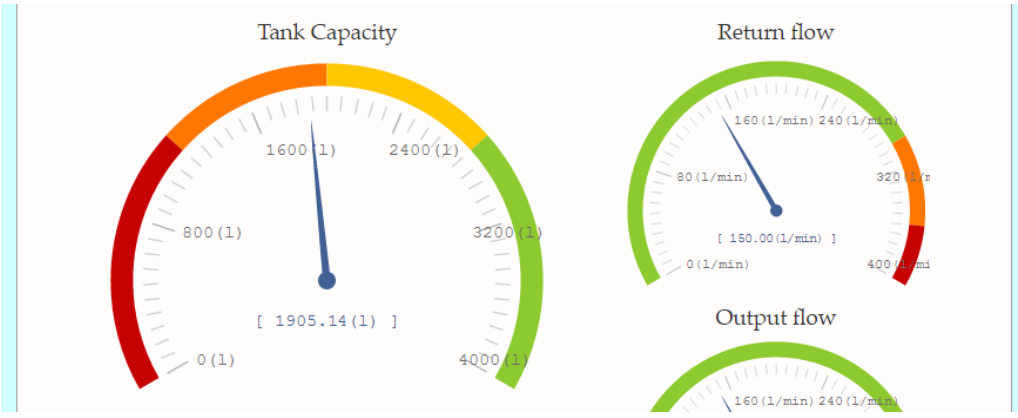

Figure 55 Tank Level value in monitoring Screen

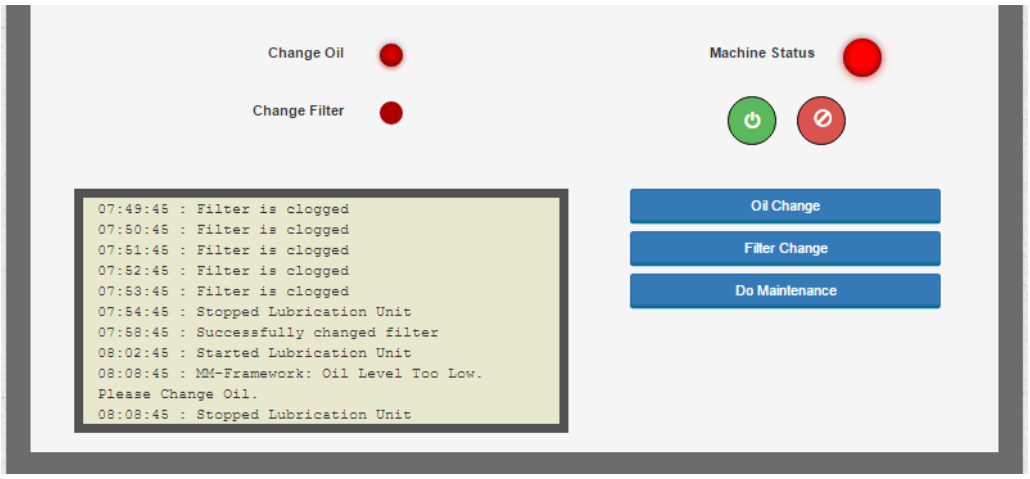

Figure 56 HMI (Process Monitoring Scenario)

6.1.4 Immediate Maintenance

In the Immediate Maintenance (IM) phase, the leakage of component is monitored and necessary alarm are raised if leakage is detected. The concept has already been explained in section 5.2.5. The following scenario was used to test the functionality.

A new lubrication Unit was created in the OLS simulator and started. Then in order to test the functionality, the leakage is generated by invoking a service. The immediate maintenance function then detects the leakage in components based on the OCR values for components, which are provided by the Process Monitoring function. Once the leakage is detected, IM raises the do maintenance alarm and sends the message to the HMI, that there is leakage in component.

The leakage service invocation via postman tool is shown in Figure 57. The complete scenario can be seen from the message log display in the HMI as shown in Figure 58.

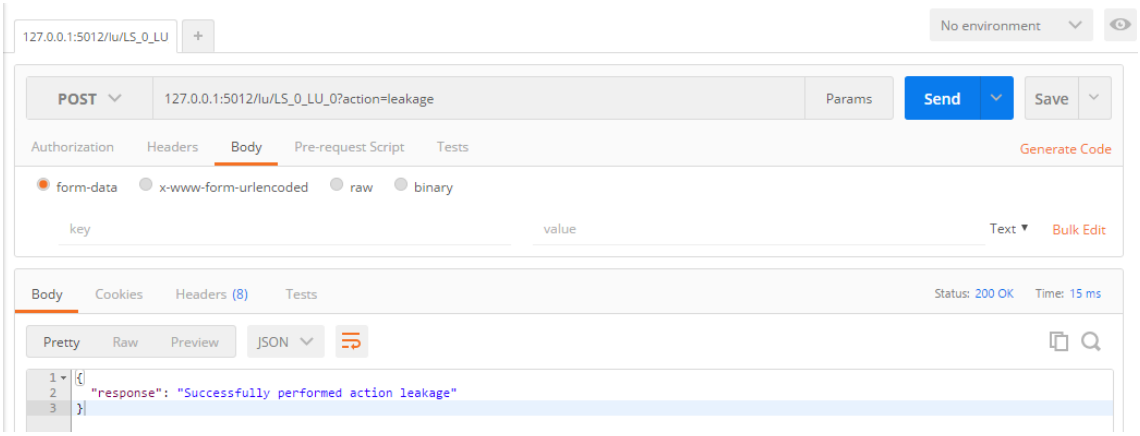

Figure 57 Postman leakage service invocation

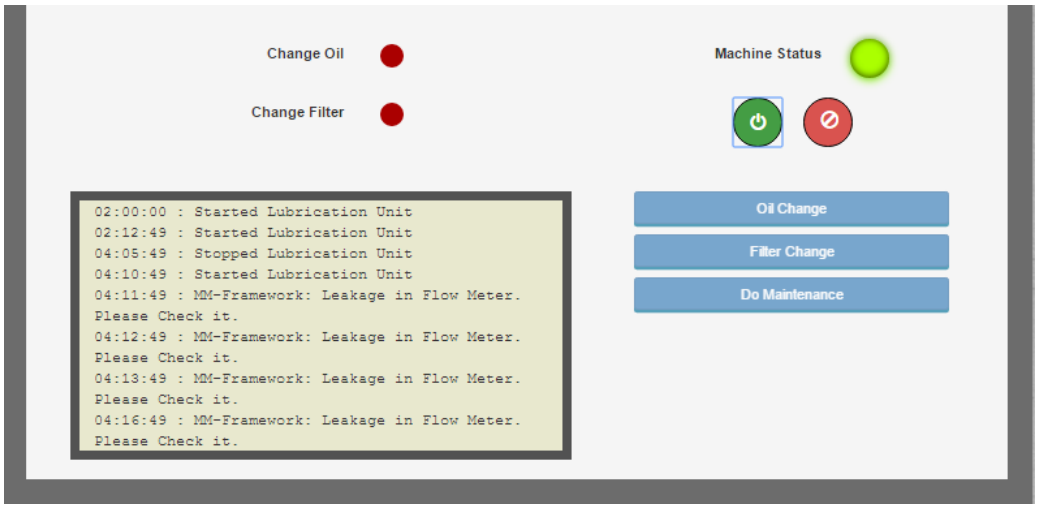

Figure 58 HMI (Immediate Maintenance scenario)

6.1.5 Predictive Maintenance

In the Predictive Maintenance phase, the scheduled maintenance is taken care. Initially, it schedules the maintenance based on the provided values (days). Later, the values are calculated and modified based on the system maintenance. The concept has already been explained in section 5.2.7. The following scenario was used to test the functionality.

A new lubrication Unit was created in the OLS simulator and started. Then in order to test the functionality, the filter is changed for three times leaving some time gaps. This is done for decreasing the frequency of days between each filter changing. Hence this will change the schedule date for filter change. Now the Predictive Maintenance starts to monitor the days according to new scheduled date. Once the time arrives, predictive maintenance raises the do maintenance alarm and sends the message to the HMI, that there should be a scheduled filter change.

The complete scenario can be seen from the message log display in the HMI as shown in Figure 59. Note: Since the filter was changed in short span of time, the days were calculated to zero and the Predictive Maintenance raised the alarm immediately.

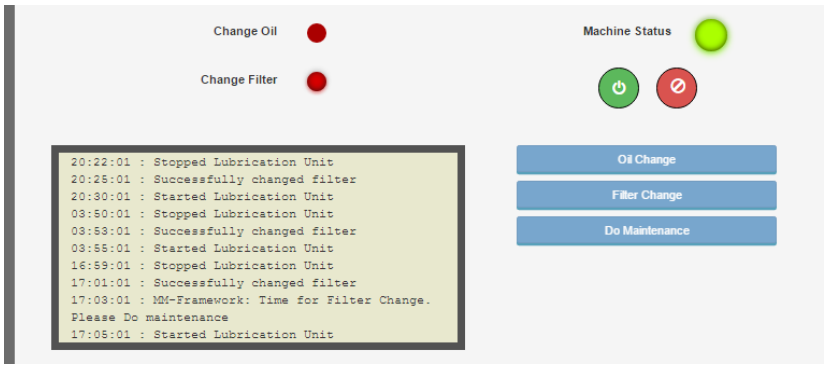

Figure 59 HMI (Predictive Maintenance scenario)

6.2 OLS Production Industry

Initially, the implementation was made as specified in the section 5.3. Later all the components were made to run and the results were taken. The scenario for obtaining the results and the screenshots for results are available in this section. Note: In this implementation all the company related details are blacked.

6.2.1 Resource Tracking and Genealogy

Resource Tracking and Genealogy (RAS) function is used to suggest the resource list for a subproject. Once the order is placed, the subproject will be created and organized through the Production Organizer tool. During the creation process, the resources are allocated manually through the Production Organizer tool. But in order to allocate the resource, the office personnel first ask for resource list suggestion form the RAS function. Then the RAS performs the analysis as mentioned in the section 5.3.1 and returns the result to the organizer. Then the office personnel, either uses this suggestion or allocates as he wishes.

Figure 60 shows the resource allocation page in the Production organizer tool. The request for resource suggestion is given via the marked option in Figure 61. The result form the RAS section is will be displayed as shown in Figure 62. The REST request and JSON reply for the above operation is shown in Figure 63.

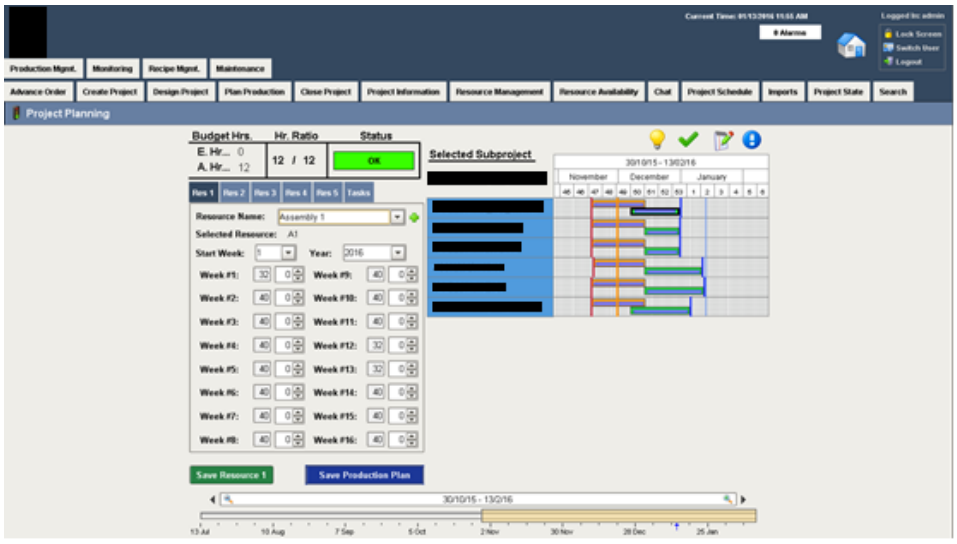

Figure 60 Resource allocation screen in Production Organizer

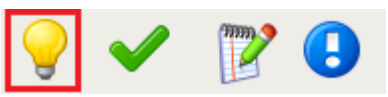

Figure 61 RAS function access icon

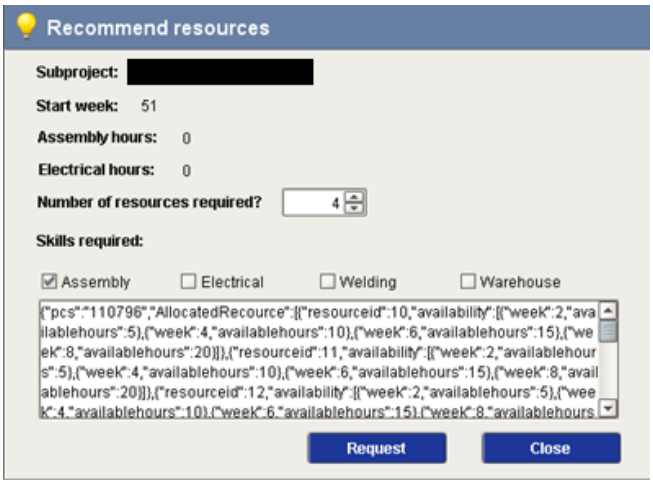

Figure 62 Resource suggestion displayed in Production Organizer

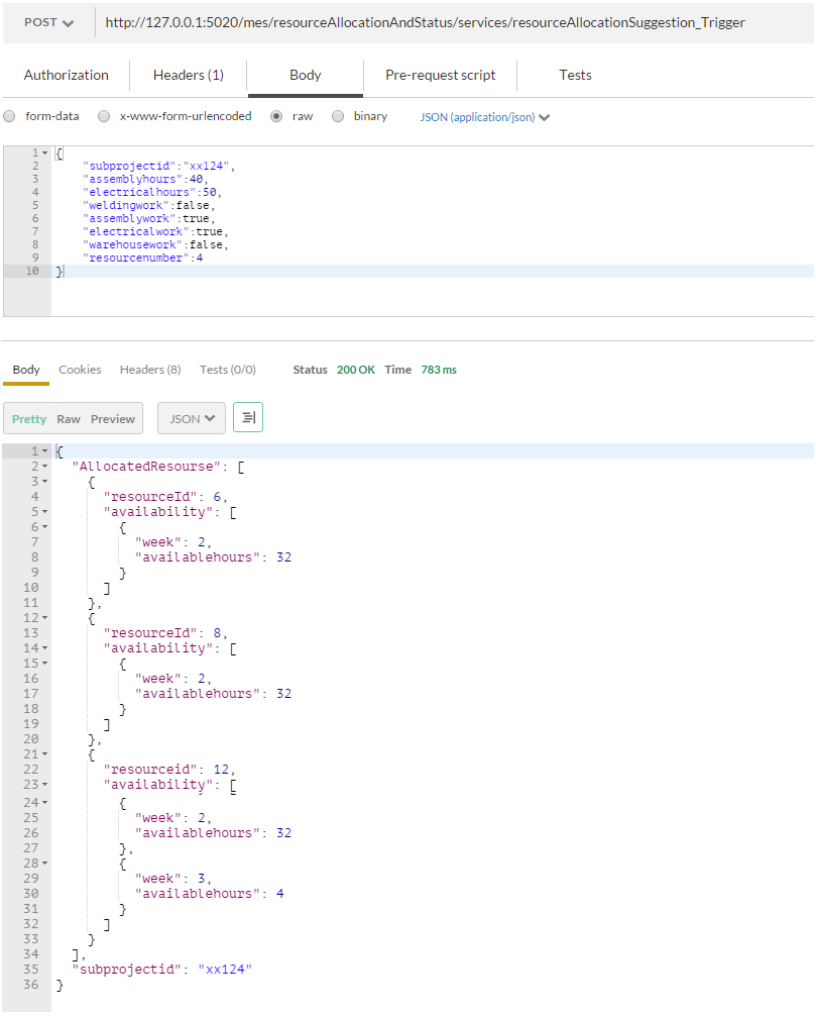

Figure 63 RAS service request via Postman tool

6.2.2 Labour Management

Labour Management (LM) function is used to support the time tracker tool by providing the analysis with respect to resource subproject time. Once the resource scans his RFID tag in the tablet which has the time tracker app, the app displays information about his current subproject, the subprojects he has completed and also the his spent and allocated time with respect to those subprojects. The LM provides the analysis based on the operations mentioned in the section 5.3.3.

Figure 64 shows time tracker app screen where the RFID of the user is scanned. The current and the finished subprojects with its details are displayed in tablet as shown in the Figure 65 and Figure 66.

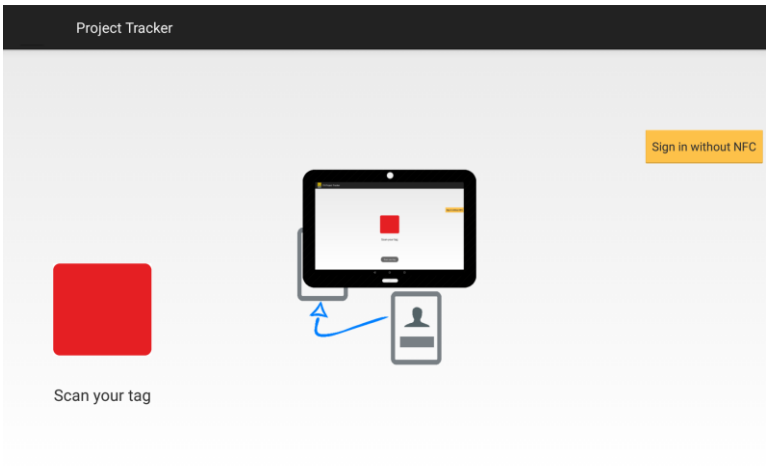

Figure 64 Time Tracker App RFID scanning screen

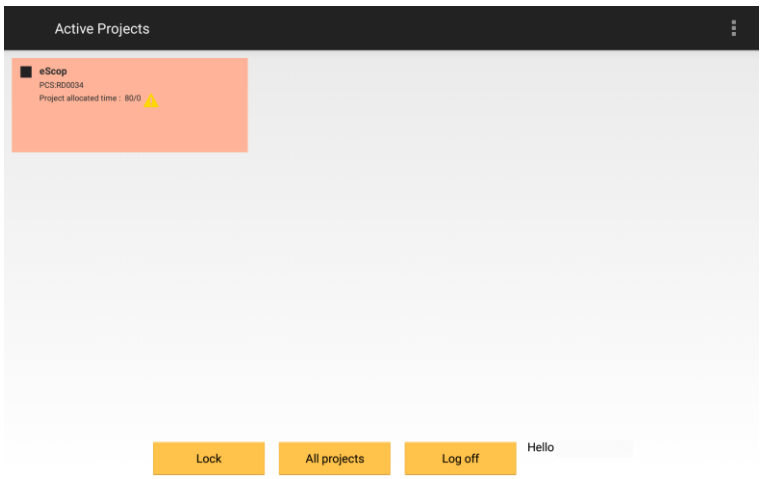

Figure 65 Time Tracker current sub projects display screen

Figure 66 Time Tracker completed sub projects display screen

6.2.3 Product Tracking and Genealogy

Product Tracking and Genealogy (PTG) function is used to support the product tracker tool by providing the analysis with respect to subproject budgeted time. Once the subproject RFID tag is scanned in the tablet, the product tracker app displays information about the subproject and the list of resource who have worked in the sub project. The PTG provides the analysis based on the operations mentioned in the section 5.3.4.

Figure 67 shows product tracker app screen where the RFID of the user is scanned. The sub project details are displayed in tablet in a series of screens as shown in the Figure 68, Figure 69 and Figure 70.

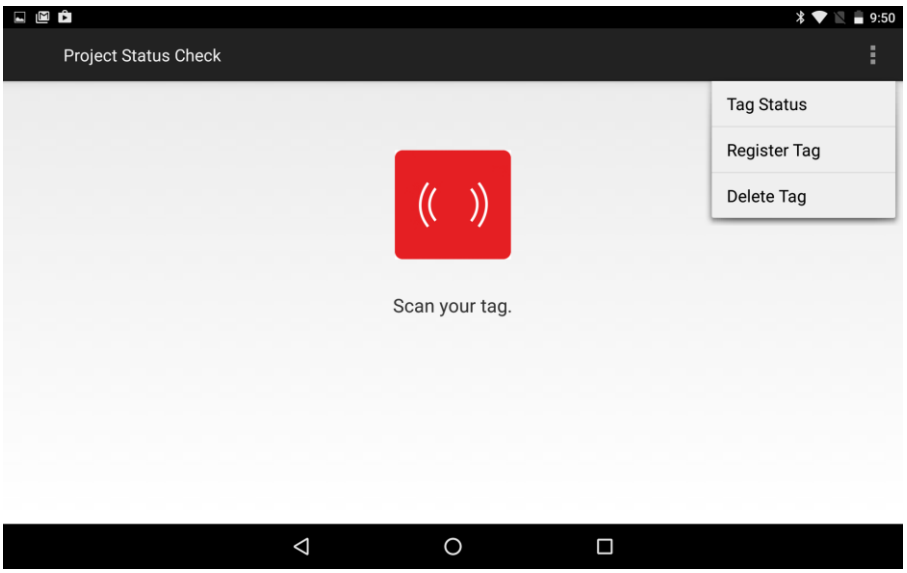

Figure 67 RFID scanning screen (Product Tracker App)

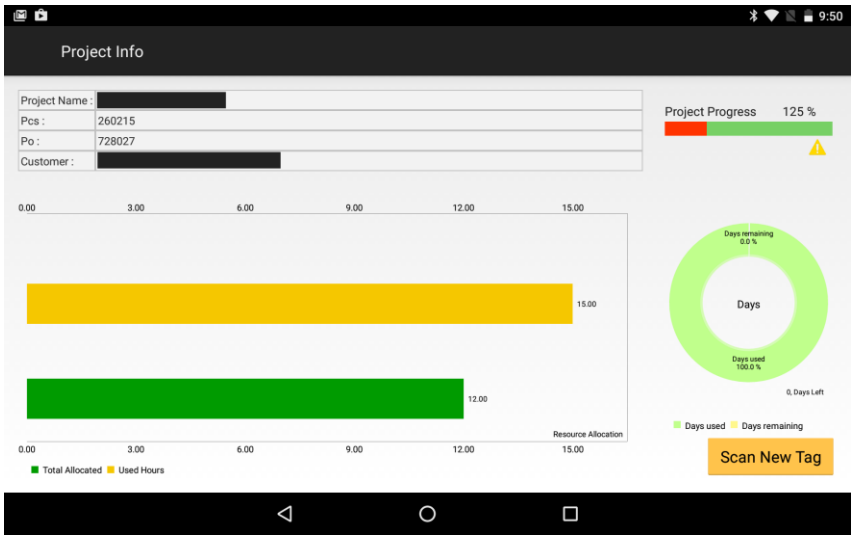

Figure 68 Project Info screen (Product Tracker App)

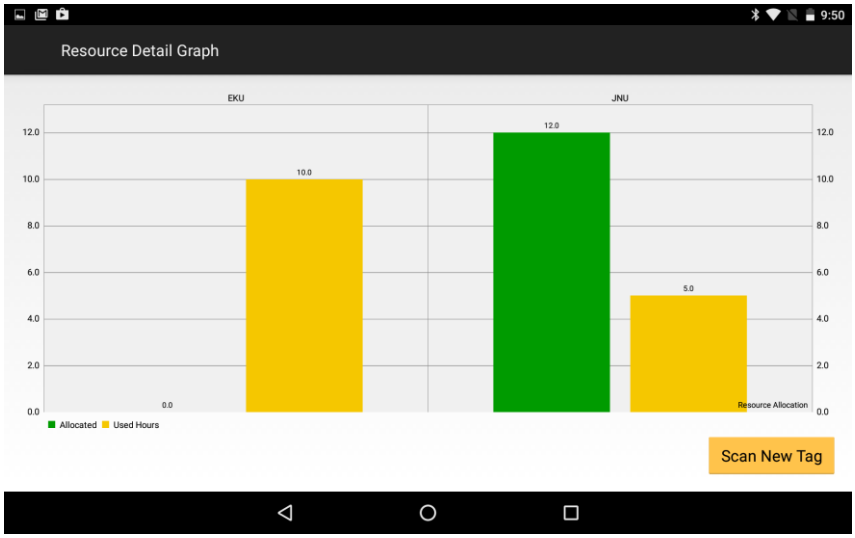

Figure 69 Resource Details analysis screen (Product Tracker App)

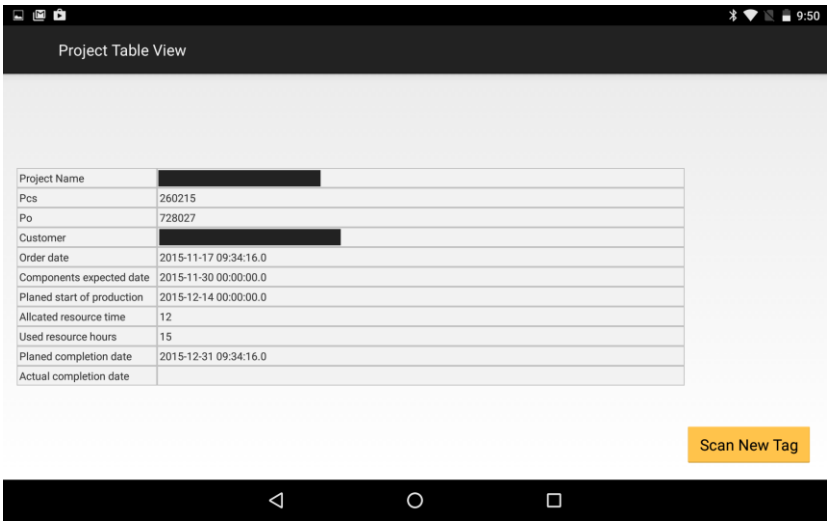

Figure 70 Project Details Analysis screen (Product Tracker App)

7. CONCLUSIONS AND FUTURE WORK

The main aim of this thesis was to generate a methodology (framework) for developing monitoring and manufacturing systems. This section will evaluate the developed framework; its implementation as monitoring and manufacturing system; and also assess the result gained from the use cases. Additionally, this section also gives some views about future work.

7.1 Implementation and Result Conclusion

The thesis proposes a methodology to develop monitoring and manufacturing systems on a configurable environment which is based on ontology. The configurable environment mentioned is nothing but a framework developed as part of the thesis. It was developed in such a way that it co-exists with Open Knowledge-Driven Systems in order to achieve the need of being a MES or monitoring software in any type of industry. Especially, the eScop concept of OKD approach was used for it.

The complete methodology with the components of the framework and how it has been created and initialized is presented in Section 3. The framework defined in the methodology was totally designed in such a way that it answers to all the questions discussed in the Section 1.2.2. Later, the developed framework was implemented in the use case defined in Section 4. During the phase of implementation, it was shown that, how simple the framework can be configured especially for the cases where the system has more sensors of the same type. To note, the simplicity of the framework referred above is only with respect to the amount of work an IT engineer must perform to configure the system when he is well worse with the concept of ontology. At the current manufacturing world, it is clear that ontology has not yet been still adapted completely. That why the drawback has already been mentioned in the Section 1.3.3 as part of limitations towards this methodology.

At last, the results obtained as part of the implementation were discussed in the section 6. The results prove that the framework has achieved all the objectives which were made as part of this thesis. Still, the system has some more limitations with respect to the devices and the software systems to which it operates with. The limitation being that the devices and software must have RESTful means of communication. It can be seen clearly from the implementation in Section 5.3, where a REST translator was used to achieve the communication.

7.2 Lesson Learned

The thesis in particular, gave a good understanding about the capabilities of ontology driven models. It is seen that the usage of ontology in industrial software system provided an edge over the traditional software. The thesis also taught lessons about the factors to be considered while creating a monitoring/MES system and implementing them.

7.3 Future Work

The Framework developed as part of this thesis, is still in its growing phase and it has much more areas to be improved as part of future work. Some of those improvements are discussed in this section.

One important area for development would be with ontology reasoning. Since the complete methodology revolves around the framework built by ontology, it will be a good phase for improvement. SWRL is one of the powerful tools, which will enrich the reasoning capability in ontology model. Hence the next phase will be to modify the SPARQL queries to SWRL based rules and logics.

Additionally, the framework being an event based system to some extent, it must be improved to have better performance while handling events. CEP (Complex Event Processing) is one such method to process events and improve the system performance. Hence, the next possible future work will also be towards implementing CEP in the Framework.

REFERENCES

[1] R. D. Sandusky, 'PLC and PC system documentation concepts', in *, IEEE Conference Record of 1989 Forty-First Annual Conference of Electrical Engineering Problems in the Rubber and Plastics Industries, 1989*, 1989, pp. 38–47.

[2] J. Peeters, 'Early MRP Systems at Royal Philips Electronics in the 1960s and 1970s', *IEEE Ann. Hist. Comput.*, vol. 31, no. 2, pp. 56–69, Apr. 2009.

[3] W. Youyuan, 'SCM/ERP/MES/PCS integration for process enterprise', in *Control Conference (CCC), 2010 29th Chinese*, 2010, pp. 5329–5332.

[4] B. Zhou, W. Shi-jin, and X. Li-feng, 'Data model design for manufacturing execution system', *J. Manuf. Technol. Manag.*, vol. 16, no. 7/8, pp. 909–935, 2005.

[5] MESA International, 'MES Functionalities & MRP to MES Data Flow Possibilities'. Pittsburgh Manufacturing ExecutionSystems Assoc., 1997.

[6] M. Dato, 'Control systems for integrated manufacturing - The CAM solution', in *1986 IEEE International Conference on Robotics and Automation. Proceedings*, 1986, vol. 3, pp. 791–795.

[7] G. W. Arnold and K. Lin, 'PRISM (PRoductivity Improvement Systems for Manufacturing)-a CIM integration at AT amp;T', in *, International Conference on Computer Integrated Manufacturing, 1988*, 1988, pp. 2–8.

[8] V. Chiodini, 'SCORE: an integrated system for dynamic scheduling and control of high-volume manufacturing', in *, Fifth Conference on Artificial Intelligence Applications, 1989. Proceedings*, 1989, pp. 271–278.

[9] MESA International, 'The Benefits of MES: A Report from the Field'. Pittsburgh Manufacturing Execution Systems Assoc, 1997.

[10] MESA International, 'MES Explained: A High Level Vision'. Pittsburgh Manufacturing Execution Systems Assoc, 1997.

[11] S. K. Ghosh, 'Changing role of SCADA in manufacturing plant', in *, Conference Record of the 1996 IEEE Industry Applications Conference, 1996. Thirty-First IAS Annual Meeting, IAS '96*, 1996, vol. 3, pp. 1565–1566 vol.3.

[12] J. Zhou *et al.*, 'Intelligent prediction monitoring system for predictive maintenance in manufacturing', in *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005*, 2005, p. 6 pp.-pp.

[13] G. Kotte and K. Kabitzsch, 'Monitoring in semiconductor manufacturing: A contribution to diagnosis in complex computer systems', in *Control Conference (ECC), 1999 European*, 1999, pp. 2154–2159.

[14] S. Yu, J. Fang, S. Shi, and H. Li, 'Development of MES-based Real Time Statistical Process Quality Monitoring System', in *IEEE International Symposium on Knowledge Acquisition and Modeling Workshop, 2008. KAM Workshop 2008*, 2008, pp. 7–10.

[15] Y. Koren *et al.*, 'Reconfigurable Manufacturing Systems', *CIRP Ann. - Manuf. Technol.*, vol. 48, no. 2, pp. 527–540, 1999.

[16] M. J. A. G. Izaguirre, A. Lobov, and J. L. M. Lastra, 'OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing', in *2011 9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011, pp. 205–211.

[17] J. Zhou and R. Dieng-Kuntz, 'Manufacturing ontology analysis and design: towards excellent manufacturing', in *2004 2nd IEEE International Conference on Industrial Informatics, 2004. INDIN '04*, 2004, pp. 39–45.

[18] W. Long, ‘Construct MES Ontology with OWL’, in *ISECS International Colloquium on Computing, Communication, Control, and Management, 2008. CCCM ’08*, 2008, vol. 1, pp. 614–617.
[19] W. Long, ‘Development of MES Based on Component and Driven by Ontology’, in *International Seminar on Future Information Technology and Management Engineering, 2008. FITME ’08*, 2008, pp. 648–651.
[20] P. Papadopoulos and L. Cipcigan, ‘Wind turbines’ condition monitoring: an ontology model’, in *International Conference on Sustainable Power Generation and Supply, 2009. SUPERGEN ’09*, 2009, pp. 1–4.
[21] B. Ramis *et al.*, ‘Knowledge-based web service integration for industrial automation’, in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 733–739.
[22] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing*. ISA, 2007.
[23] ‘ISA-95 Overview’, *ISA 95*. .
[24] ‘Plant to Enterprise Integration: ISA-95 is a Nice Start’. Dassault Systèmes, Jun-2012.
[25] ‘MESA International - About MESA’. [Online]. Available: http://mesa.org/en/aboutus/aboutmesa.asp. [Accessed: 08-Sep-2015].
[26] M. Younus, C. Peiyong, L. Hu, and F. Yuqing, ‘MES development and significant applications in manufacturing -A review’, in *2010 2nd International Conference on Education Technology and Computer (ICETC)*, 2010, vol. 5, pp. V5-97-V5-101.
[27] S. Schmidt, ‘Understanding Manufacturing Execution Systems (MES)’. Freedom Technologies.
[28] ‘Process Industries Division - Institute of Industrial Engineers’. [Online]. Available: https://www.iienet2.org/details.aspx?id=887. [Accessed: 03-Sep-2015].
[29] S. Noroozi and J. Wikner, ‘Sales and operations planning in process industries based on types of object, flow and driver’. .
[30] E. W. Kamen, ‘Chapter 1 - Introduction’, in *Industrial Controls and Manufacturing*, E. W. Kamen, Ed. San Diego: Academic Press, 1999, pp. 1–8.
[31] *Manufacturing Systems: Theory and Practice*. New York: Springer-Verlag, 2006.
[32] P. Stavropoulos, D. Chantzis, C. Doukas, A. Papacharalampopoulos, and G. Chryssolouris, ‘Monitoring and Control of Manufacturing Processes: A Review’, *Procedia CIRP*, vol. 8, pp. 421–425, 2013.
[33] J. Fang, S. Yu, and X. Ding, ‘Development and Application of Networked Manufacturing Process Monitoring System’, in *International Symposium on Computational Intelligence and Design, 2008. ISCID ’08*, 2008, vol. 1, pp. 432–435.
[34] Z. Yongman and H. Zhen, ‘Applications of Cascade Correlation Neural Networks for Manufacturing Process Monitoring’, in *2010 International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII)*, 2010, vol. 2, pp. 47–50.
[35] C.-H. Kuo and H.-P. Huang, ‘Failure modeling and process monitoring for flexible manufacturing systems using colored timed Petri nets’, *IEEE Trans. Robot. Autom.*, vol. 16, no. 3, pp. 301–312, Jun. 2000.
[36] B. Grošelj, R. Zupančič, and A. Sluga, ‘Quality monitoring service for distributed manufacturing systems’, *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 6, pp. 639–649, Jun. 2015.
[37] C. I. Thorsten Wuest, ‘An approach to quality monitoring in manufacturing using supervised machine learning on product state data’, *J. Intell. Manuf.*, vol. 25, no. 5, pp. 1167–1180, 2014.

[38] M. W. Milo, M. Roan, and B. Harris, ‘A new statistical approach to automated quality control in manufacturing processes’, *J. Manuf. Syst.*, vol. 36, pp. 159–167, Jul. 2015.
[39] J. Fang, S. Yu, and X. Ding, ‘Development of Internet-based Long-range Manufacture Quality Monitoring System’, in *2008 International Symposium on Electronic Commerce and Security*, 2008, pp. 685–689.
[40] E. Martin, J. Morris, and S. Lane, ‘Monitoring process manufacturing performance’, *IEEE Control Syst.*, vol. 22, no. 5, pp. 26–39, Oct. 2002.
[41] O. J. Bakker, S. M. Ratchev, and A. A. Popov, ‘Towards a Condition-Monitoring Framework for Quality Assurance in Intelligent Multistage Manufacturing EnvironmentÕ’, *IFAC-Pap.*, vol. 48, no. 3, pp. 2089–2094, 2015.
[42] V. Jovan and S. Zorzut, ‘Use of Key Performance Indicators in Production Management’, in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, 2006, pp. 1–6.
[43] ‘Planning and monitoring your program : first steps in program evaluation / [written by Jan Smith]. - Version details’, *Trove*. [Online]. Available: http://trove.nla.gov.au/version/43313221. [Accessed: 15-Dec-2016].
[44] A. Rakar, S. Zorzut, and V. Jovan, ‘Assesment of Production Performance by Means of KPI’. Control 2004, 2004.
[45] ‘Overall Equipment Effectiveness (OEE) A General Discussion with Calculations Methods’. Capstone Metrics, 2007-2001.
[46] Y. Fukuda and R. Patzke, ‘Standardization of Key Performance Indecator for manufacturing execution system’, in *Proceedings of SICE Annual Conference 2010*, 2010, pp. 263–265.
[47] L. Zheng, J. Xiao, F. Hou, W. Feng, and N. Li, ‘Cycle time reduction in assembly and test manufacturing factories: A KPI driven methodology’, in *2008 IEEE International Conference on Industrial Engineering and Engineering Management*, 2008, pp. 1234–1238.
[48] B. Zhang, C. Postelnicu, and J. L. M. Lastra, ‘Key Performance Indicators for energy efficient asset management in a factory automation testbed’, in *IEEE 10th International Conference on Industrial Informatics*, 2012, pp. 391–396.
[49] Z. Chorfi, A. Berrado, and L. Benabbou, ‘Selection of Key Performance Indicators for Supply Chain monitoring using MCDA’, in *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, 2015, pp. 1–6.
[50] M. Sorrentino, G. Rizzo, and A. Trifiro, ‘A Model-Based Key Performance Index for Monitoring and Diagnosis of Cooling Systems in Telecommunication Rooms and Data-Centers’, in *Intelec 2013; 35th International Telecommunications Energy Conference, SMART POWER AND EFFICIENCY*, 2013, pp. 1–6.
[51] ‘eScop | Artemis project about Embedded systems Service-based Control for Open manufacturing and Process automation’. .
[52] S. Iarovyi, X. Xu, A. Lobov, J. L. M. Lastra, and S. Strzelczak, ‘Architecture for Open, Knowledge-Driven Manufacturing Execution System’, in *Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth*, S. Umeda, M. Nakano, H. Mizuyama, H. Hibino, D. Kiritsis, and G. von Cieminski, Eds. Springer International Publishing, 2015, pp. 519–527.
[53] R. Cover, ‘Service Oriented Architecture (SOA)’. [Online]. Available: http://xml.coverpages.org/soa.html. [Accessed: 13-Sep-2015].
[54] ‘SOA and Web Services’. [Online]. Available: http://www.oracle.com/technetwork/articles/javase/soa-142870.html. [Accessed: 13-Sep-2015].

[55] S. Stadtmuller, S. Speiser, M. Junghans, and A. Harth, 'Comparing Major Web Service Paradigms'. SALAD 2013 Workshop, 2013.
[56] Z. Niu, C. Yang, and Y. Zhang, 'A design of cross-terminal web system based on JSON and REST', in *2014 5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2014, pp. 904–907.
[57] N. Chungoora *et al.*, 'A model-driven ontology approach for manufacturing system interoperability and knowledge sharing', *Comput. Ind.*, vol. 64, no. 4, pp. 392–401, May 2013.
[58] 'Resource Description Framework (RDF): Concepts and Abstract Syntax'. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. [Accessed: 13-Sep-2015].
[59] I. Kollia, B. Glimm, and I. Horrocks, 'SPARQL Query Answering over OWL Ontologies', in *The Semantic Web: Research and Applications*, G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Pan, Eds. Springer Berlin Heidelberg, 2011, pp. 382–396.
[60] 'Java (programming language)', *Wikipedia, the free encyclopedia*. 24-Oct-2015.
[61] '21. Web MVC framework'. [Online]. Available: http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/mvc.html. [Accessed: 11-Oct-2015].
[62] 'JSON'. [Online]. Available: http://www.json.org/. [Accessed: 10-Oct-2015].
[63] 'MVEL Language Guide'.
[64] J. T. Puttonen, *A Semantically Enhanced Approach for Orchestration of Web Services in Factory Automation Systems*. 2014.
[65] 'Apache Jena - Fuseki: serving RDF data over HTTP'. [Online]. Available: https://jena.apache.org/documentation/serving_data/. [Accessed: 15-Nov-2015].
[66] 'Maven – Welcome to Apache Maven'. [Online]. Available: https://maven.apache.org/. [Accessed: 15-Nov-2015].
[67] 'Virtual lubrication system provides an environment for learning, experimenting and innovating'. Tampere University of Technology, FAST-Lab, 04-Nov-2015.
[68] 'Oil Lubrication System Simulator'. [Online]. Available: http://escop.rd.tut.fi:5012/. [Accessed: 04-Apr-2016].
[69] 'eScop-project / OilLubricationSystem', *GitLab*. [Online]. Available: https://gitlab.com/eScop-project/oillubricationsystem. [Accessed: 04-Apr-2016].
[70] 'Fluidhouse - Complete system deliveries'. [Online]. Available: http://www.fluidhouse.fi/en/products-engineering/lubrication-automation/fluidcirc-lu. [Accessed: 04-Apr-2016].

APPENDIX A – MMO (Configuration Ontology)

This section presents the basic MMO (Configuration Ontology) which will be used by the Framework to build the required Monitoring of MES system.

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:mm="http://www.MM-Framework.eu/MM.owl#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
 <owl:Ontology rdf:about="http://www.MM-Framework.eu/MM.owl"/>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Metas">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Rule">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Service">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Category">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Output">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Action">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Input">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Type">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Function">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:Class rdf:about="http://www.MM-Framework.eu/MM.owl#Device">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 </owl:Class>
 <owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasService">
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Rule"/>
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Service"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasCategory">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Category"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasRule">
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Function"/>
```

```
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Rule"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasInput">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Input"/>
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Rule"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasAction">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Action"/>
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Rule"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="http://www.MM-Framework.eu/MM.owl#hasOutput">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Output"/>
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Rule"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:ObjectProperty>
<rdf:Property rdf:about="http://www.MM-Framework.eu/MM.owl#hasType">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Type"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.MM-Framework.eu/MM.owl#hasMeta">
  <rdfs:range rdf:resource="http://www.MM-Framework.eu/MM.owl#Metas"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.MM-Framework.eu/MM.owl#description">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.MM-Framework.eu/MM.owl#id">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.MM-Framework.eu/MM.owl#value">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</rdf:Property>
<owl:DatatypeProperty rdf:about="http://www.MM-Framework.eu/MM.owl#url">
  <rdfs:domain rdf:resource="http://www.MM-Framework.eu/MM.owl#Input"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.MM-Framework.eu/MM.owl#formula">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.MM-Framework.eu/MM.owl#formulaId">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</owl:DatatypeProperty>
<mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#double"/>
<mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Operation">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >operation</mm:id>
</mm:Category>
```

```
<mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Ontology">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >ontology</mm:id>
 </mm:Category>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Services">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >service</mm:id>
 </mm:Category>
 <mm:Action rdf:about="http://www.MM-Framework.eu/MM.owl#ServiceInvoke"/>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#long"/>
 <mm:Action rdf:about="http://www.MM-Framework.eu/MM.owl#OntologySave"/>
 <mm:Action rdf:about="http://www.MM-Framework.eu/MM.owl#ServiceReply"/>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Process">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >process</mm:id>
 </mm:Category>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_ServiceInput">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >serviceInput</mm:id>
 </mm:Category>
 <mm:Action rdf:about="http://www.MM-Framework.eu/MM.owl#Event"/>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#integer"/>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Message">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >message</mm:id>
 </mm:Category>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#array"/>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Value">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >value</mm:id>
 </mm:Category>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#string"/>
 <rdf:Description rdf:about="http://www.MM-Framework.eu/MM.owl#default">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >default</mm:id>
 </rdf:Description>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#map"/>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Events">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >event</mm:id>
 </mm:Category>
 <mm:Type rdf:about="http://www.MM-Framework.eu/MM.owl#boolean"/>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Query">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >query</mm:id>
 </mm:Category>
 <mm:Category rdf:about="http://www.MM-Framework.eu/MM.owl#Cat_Time">
  <mm:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >time</mm:id>
 </mm:Category>
</rdf:RDF>
```

Appendix B – Monitoring Functions

The elements added to the MMO as part of monitoring Oil Lubrication System is discussed in this section. The individuals of the ontology added to each class has been specified in detailed manner.

Input

Inputs are the basic necessary elements for the rules that are executed in the framework. The list of inputs added as part of the Oil Lubrication System Monitoring are listed in Table 11.

Table 11 Ontology Input Individuals (OLS Simulator Monitoring)

ID	Formula ID	Category	Meta	Type	Value	Description
avgFilterChange	-	Event	DeviceType_Framework, SensorType_AvgFilterChange	integer	-	Average filter changing frequency in days event from Framework
avgFilter-ChangeDays	-	Event	DeviceType_Framework	double	50.0	Average filter changing days value directly from Framework
avgMaintenance	-	Event	DeviceType_Framework, SensorType_AvgMaintenance	integer	-	Average filter changing frequency in days event from Framework
avgMaintenance-Days	-	Event	DeviceType_Framework	double	100.0	Average Maintenance days value from Framework
avgOilChange	-	Event	DeviceType_Framework, SensorType_AvgOilChange	integer	-	Average oil changing frequency in days event from Framework

avgOilChangeDays	-	Event	DeviceType_Framework	double	200.0	Average Oil Change days value from Framework
filterChange	-	Event	DeviceType_Framework, SensorType_FilterChange	boolean	-	Filter change alarm event from Framework
filterChangeTime	-	Ontology	DeviceType_Ontology	array	-	Time when the filter has been changed which will be got from ontology.
filterChanged	-	Event	DeviceType_HMI, SensorType_FilterChanged	boolean	-	Filter changed event from HMI
filterClog	-	Event	DeviceType_LU, SensorType_FilterClog	double	-	Filter clog service query from Lubrication Unit
inflow	inflowFM	Event	DeviceType_FM, ParentType_MS, SensorType_Inflow	double	-	Inflow event of Flow Meter from OLS Simulator
inflow	inflowMS	Event	DeviceType_MS, ParentType_LU, SensorType_Inflow	double	-	Inflow event of Measuring Station from OLS Simulator
inflow	inflowLU	Event	DeviceType_LU, SensorType_Inflow	double	-	Inflow event of Lubrication Unit from OLS Simulator
oilAllocation	-	Query	DeviceType_LU, SensorType_OilAllocation	double	-	Oil flow allocated in the lubrication unit accessed as Service Query
maxOilFlow	-	Query	DeviceType_LU, SensorType_MaxOilFlow	double	-	Maximum Oil flow in the lubrication unit accessed as Service Query

maintenanceDone	-	Event	DeviceType_Framework, SensorType_MaintenenceDone	boolean	-	Maintenance alarm raised as Framework event
messageEvent	-	Event	DeviceType_Framework, SensorType_MessageEvent	boolean	-	Event for messages raised by Framework rules
ocrFM	-	Event	DeviceType_Framework, SensorType_OcrFM	integer	-	Oil Consumption Ratio Of Flow Meter emitted as Framework event
ocrMS	-	Event	DeviceType_Framework, SensorType_OcrMS	integer	-	Oil Consumption Ratio Of Measuring Station emitted as Framework event
ocrLU	-	Event	DeviceType_Framework, SensorType_OcrLU	integer	-	Oil Consumption Ratio Of Lubrication Unit emitted as Framework event
oilChange	-	Event	DeviceType_Framework, SensorType_OilChange	boolean	-	Oil Change alarm generated by Framework event
oilChangeTime	-	Ontology	DeviceType_Ontology	array	-	Oil Change alarm time from ontology
oilDirty	.	Event	DeviceType_Framework, SensorType_OilDirty	boolean	-	Oil Dirty alarm generated by Framework event
oilLow	.	Event	DeviceType_Framework, SensorType_OilLow	boolean	-	Oil Low alarm generated by Framework event
oilQualityDecayRate	-	Event	DeviceType_Framework, SensorType_OilQualityDecayRate	integer	-	Oil quality decay rate in % emitted as event in Framework
outflow	outflowFM	Event	DeviceType_FM, SensorType_OutFlow	double	-	Outflow event of Flow Meter from OLS Simulator

outflow	outflowMS	Event	DeviceType_MS, SensorType_OutFlow	double	-	Outflow event of Measuring Station from OLS Simulator
outflow	outflowLU	Event	DeviceType_LU, SensorType_OutFlow	double	-	Outflow event of Lubrication Unit from OLS Simulator
outflow	outflowLU	Ontology	DeviceType_Ontology	array	-	Outflow of Lubrication Unit from ontology
particleCounter	-	Event	DeviceType_LU, SensorType_ParticleCounter	double	-	Particle Counter value as event from OLS Simulator
particleQuality	-	Value	DeviceType_Value	String	“25/20/18”	Particle Quality value for determining the oil quality rate
percentLeak	-	Value	DeviceType_Value	integer	5	% value used to determine leak in the system
percentQualityDecay	-	Value	DeviceType_Value	integer	50	% value used to determine oil quality decay rate in the system
percentTankLevel-Start	-	Value	DeviceType_Value	integer	90	% value used to determine whether the Tank level is ok to start
percentTankLevel-Low	-	Value	DeviceType_Value	integer	50	% value used to determine whether the Tank level is low
simulationTime	-	Time	DeviceType_Framework	Long	-	Time form Framework
start	-	Event	DeviceType_LU, SensorType_Start	boolean	-	Event raised from OLS Simulator when LU is started
stop	-	Event	DeviceType_LU, SensorType_Stop	boolean	-	Event raised from OLS Simulator when LU is stopped

stopUnit	-	Event	DeviceType_Framework, SensorType_StopUnit	boolean	-	Event generated by Framework to raise alarm for stopping LU
tankCapacity	-	Query	DeviceType_LU, SensorType_TankCapacity	double	-	Tank capacity of the lubrication unit accessed as Service Query
level	-	Event	DeviceType_LU, SensorType_TankLevel	double	-	Tank Level of LU event from the OLS Simulator
level	-	Query	DeviceType_LU, SensorType_Stop	double	-	Tank level of the lubrication unit accessed as Service Query

Output

The list of outputs for the rules that are executed in the framework are given below in the Table 12.

Table 12 Ontology Output Individuals (OLS Simulator Monitoring)

ID	Category	Type	Description
avgFilterChange	Value	integer	Average filter changing frequency in days
avgMaintenance	Value	integer	Average filter changing frequency in days
avgOilChange	Value	integer	Average oil changing frequency in days
filterChange	Value	boolean	Filter change alarm
filterChangeTime	Value	long	Time when the filter has been changed which will be stored to ontology.
maintenanceTime	Value	long	Time when the maintenance has been done which will be stored to ontology.
messageEvent	Message	map	Message sent while service invoke
ocrFM	Value	integer	Oil Consumption Ratio Of Flow Meter
ocrMS	Value	integer	Oil Consumption Ratio Of Measuring Station
ocrLU	Value	integer	Oil Consumption Ratio Of Lubrication Unit
oilChange	Value	boolean	Oil Change alarm
oilChangeTime	Value	long	Oil Change alarm time
oilDirty	Value	boolean	Oil Dirty alarm
oilLow	Value	map	Oil Low values
oilQualityDecay-Rate	Value	integer	Oil quality decay rate in %
outflow	Value	double	Outflow of Lubrication Unit
serviceOutput	message	map	Alarm to schedule Oil change
stopUnit	Value	boolean	Alarm for stopping LU

Meta

The list of Meta for the elements in the framework are given below in the Table 13.

Table 13 Ontology Meta Individuals (OLS Simulator Monitoring)

Name	ID	Value
DeviceType_LU	deviceType	LubricationUnit

DeviceType_MS	deviceType	MeasuringStation
DeviceType_FM	deviceType	FlowMeter
DeviceType_Ontology	deviceType	ontology
DeviceType_Value	deviceType	value
DeviceType_HMI	deviceType	HMI
DeviceType_Framework	deviceType	framework
ParentType_LU	parentType	LubricationUnit
ParentType_MS	parentType	MeasuringStation
SensorType_AvgFilterChange	sensorType	avgFilterChange
SensorType_AvgMaintenance	sensorType	avgMaintenance
SensorType_AvgOilChange	sensorType	avgOilChange
SensorType_FilterChange	sensorType	filterChange
SensorType_FilterChanged	sensorType	filterChanged
SensorType_InFlow	sensorType	inFlow
SensorType_Level	sensorType	level
SensorType_MaintenanceDone	sensorType	maintenanceDone
SensorType_MaxOilFlow	sensorType	maxOilFlow
SensorType_MessageEvent	sensorType	messageEvent
SensorType_OcrFM	sensorType	ocrFM
SensorType_OcrMS	sensorType	ocrMS
SensorType_OcrLU	sensorType	ocrLU
SensorType_OilAllocation	sensorType	oilAllocation
SensorType_OilChange	sensorType	oilChange
SensorType_OilChanged	sensorType	oilChanged
SensorType_OilDirty	sensorType	oilDirty
SensorType_OilLow	sensorType	oilLow

SensorType_OilQualityDecayRate	sensorType	oilQualityDecayRate
SensorType_OutFlow	sensorType	outFlow
SensorType_ParticleCount	sensorType	particleCount
SensorType_Start	sensorType	start
SensorType_StopUnit	sensorType	stopUnit
SensorType_TankCapacity	sensorType	tankCapacity

Service

The list of services that are executed in the framework are given below in the Table 14.

Table 14 Ontology Service Individuals (OLS Simulator Monitoring)

ID	Category	Meta	Description
filterChange	Operation	DeviceType_HMI, ParentType_LU	Service to generate alarm for changing filter in HMI
oilChange	Operation	DeviceType_HMI ParentType_LU	Service to generate alarm for changing oil in HMI
Message	Process	DeviceType_HMI ParentType_LU	Service to generating message in HMI
Stop	Process	DeviceType_LU ParentType_LU	Service to start the lubrication unit

Rule

Condition Monitoring

The rules as part of the function condition monitoring are given below in the Table 15.

Table 15 Ontology Rule Individuals for Condition Monitoring

RULE 1	
RuleId	startConditionCheck
Inputs	tankCapacity, tankLevel, filterClog, luOilAllocated, luTotalFlow, percentTankLevelStart

Formula (MVEL Format)	import java.util.*;
	stopUnit = new HashMap();
	stopUnit.put('trigger', false);
	stopUnit.put('message', 'All is Well');

	if (start){
	System.out.println('pinged');
	level = Math.round(level*100)/100;
	if(level < ((percentTankLevelStart/100)* tankCapacity)){
	stopUnit.trigger = true;
	stopUnit.message = 'MM-Framework: Level of oil is lower than the required level to start the system';
	}
	if(filterClog){
	stopUnit.trigger = true;
	stopUnit.message = 'MM-Framework: Filter is Clogged. Please change the filter and start the system';
	}
	if(oilAllocation > maxOilFlow){
	stopUnit.trigger = true;
	stopUnit.message = 'MM-Framework: Oil Flow Allocation is more than maximum Flow. Please make the changes and start the system';
	}
	}
	stopUnit;

Outputs	stopUnit
Meta	Lubrication Unit
Action	Event
RULE 2	
RuleId	conditionMonitoringMessageGeneration
Inputs	stopUnit
Formula (MVEL Format)	import java.util.*; ['trigger':stopUnit.trigger, 'message':{'payload':{'value':stopUnit.message}}];
Outputs	messageEvent
Meta	Lubrication Unit
Action	Event

Data Acquisition

The rules as part of the function data acquisition are given below in the Table 16.

Table 16 Ontology Rule Individuals for Data Acquisition

RULE 1

RuleId	ontologySaveOutFlowLU
Inputs	outFlowLU
Formula (MVEL format)	outFlowLU
Outputs	outFlowLU
Meta	Lubrication Unit
Action	Save to ontology
RULE 2	
RuleId	saveLastOilChangeTime
Inputs	oilChanged, simulationTime

Formula (MVEL format)	If(oilChanged){
	simulationTime;
	}

Outputs	oilChangeTime
Meta	Lubrication Unit
Action	Save to ontology
RULE 3	
RuleId	saveLastFilterChangeTime
Inputs	filterChanged, simulationTime

Formula (MVEL format)	If(filterChanged){
	simulationTime;
	}

Outputs	filterChangeTime
Meta	Lubrication Unit
Action	Save to ontology
RULE 4	
RuleId	saveLastMaintenenceDoneTime
Inputs	maintenanceDone, simulationTime

Formula (MVEL format)	If(maintenanceDone){
	simulationTime;
	}

Outputs	maintenanceTime
Meta	Lubrication Unit
Action	Save to ontology

HMI Coordinator

The rules as part of the function HMI Coordinator are given below in the Table 17.

Table 17 Ontology Rule Individuals for HMI Coordinator

RULE 1	
RuleId	hmiMessageInvoke
Inputs	messageEvent
Formula (MVEL Format)	messageEvent;
Outputs	serviceOutput
Meta	Lubrication Unit
Action	Invoke Service
ServiceId	message
RULE 2	
RuleId	hmiOilChangeInvoke
Inputs	oilChange
Formula (MVEL format)	import java.util.*; ['trigger':oilChange, 'message':{'payload':{'value':oilChange}}];
Outputs	serviceOutput
Meta	Lubrication Unit
Action	Invoke Service
ServiceId	oilChange
RULE 3	
RuleId	hmiFilterChangeInvoke
Inputs	filterChange
Formula (MVEL format)	import java.util.*; ['trigger':filterChange, 'message':{'payload':{'value':filterChange}}];
Outputs	serviceOutput
Meta	Lubrication Unit
Action	Invoke Service
ServiceId	filterChange

Immediate Maintenance

The rules as part of the function immediate maintenance are given below in the Table 18.

Table 18 Ontology Rule Individuals for Immediate Maintenance

RULE 1	
RuleId	lubricationUnitLeakage

Inputs	ocrLU, percentLeak

Formula (MVEL format)	if ((ocrLU *100) > leakagePercent){
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Leakage in Lubrication Unit. Please Check it.'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All is well'}}];
	}

Outputs	messageEvent
Meta	Lubrication Unit
Action	Event
RULE 2	
RuleId	measuringStationLeakage
Inputs	ocrMS, percentLeak

Formula (MVEL format)	if ((ocrMS *100) > leakagePercent){
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Leakage in Measuring Station. Please Check it.'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All is well'}}];
	}

Outputs	messageEvent
Meta	Measuring Station
Action	Event
RULE 3	
RuleId	flowMeterLeakage
Inputs	ocrFM, percentLeak

Formula (MVEL format)	if ((ocrFM *100) > leakagePercent){
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Leakage in Flow Meter. Please Check it.'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All is well'}}];
	}

Outputs	messageEvent
Meta	Flow Meter
Action	Event

Predictive Maintenance

The rules as part of the function predictive maintenance are given below in the Table 19.

Table 19 Ontology Rule Individuals for Predictive Maintenance

RULE 1	
RuleId	averageFilterChangeCalculation
Inputs	[filterChangeTime]

Formula (MVEL format)	import java.util.*;
	result = 0.0;
	if(filterChangeTime.size() >= 3){
	n = filterChangeTime.size() - 1;
	j = 0;
	for (int i =0; i <= n-1; i++) {
	j += (filterChangeTime[n-(i+1)] - filterChangeTime[n-i]);
	}
	result = (j/(n))/(1000*60*60*24);
	}else{
	result = avgFilterChangeDays;
	}
	result;

Outputs	avgFilterChange
Meta	Lubrication Unit
Action	Event
RULE 2	
RuleId	averageOilChangeCalculation
Inputs	[oilChangeTime]

Formula (MVEL format)	import java.util.*;
	result = 0.0;
	if(oilChangeTime.size() >= 3){
	n = oilChangeTime.size() - 1;
	j = 0;
	for (int i =0; i <= n-1; i++) {
	j += (oilChangeTime[n-(i+1)] - oilChangeTime[n-i]);
	}
	result = (j/n)/(1000*60*60*24);
	}else{
	result = avgOilChangeDays;
	}
	result;

Outputs	avgOilChange
Meta	Lubrication Unit

Action	Event
RULE 3	
RuleId	averageMaintenanceCalculation
Inputs	[leakageAlarmTime]

Formula (MVEL format)	import java.util.*;
	result = 0.0;
	if(maintenanceTime.size() >= 3){
	n = maintenanceTime.size() - 1;
	j = 0;
	for (int i =0; i <= n-1; i++) {
	j += (maintenanceTime[n-(i+1)] - maintenanceTime[n-i]);
	}
	result = (j/(n))/(1000*60*60*24);
	}else{
	result = avgMaintenanceDays;
	}
	result;

Outputs	avgMaintenace
Meta	Lubrication Unit
Action	Event
RULE 4	
RuleId	scheduleOilChangeMessage
Inputs	avgOilChange, [oilChangeTime], simulationTime

Formula (MVEL format)	import java.util.*;
	if(oilChangeTime.size() >= 3){
	n = oilChangeTime.size() - 1;
	timeValue = 0l;
	timeValue = simulationTime - oilChangeTime[n];
	timeValue = timeValue / (1000*60*60*24);
	if(timeValue >= avgOilChange) {
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Time for Oil Change. Please Do maintenance'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}];
	}
	}else{
	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}];
	}

Outputs	messageEvent

Meta	Lubrication Unit
Action	Event
RULE 5	
RuleId	scheduleFilterChangeMessage
Inputs	avgFilterChange, [filterChangeTime], simulationTime

Formula (MVEL format)	import java.util.*;
	if(filterChangeTime.size() >= 3){
	n = filterChangeTime.size() - 1;
	timeValue = 0l;
	timeValue = simulationTime - filterChangeTime[n];
	timeValue = timeValue / (1000*60*60*24);
	if(timeValue >= avgFilterChange) {
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Time for Filter Change. Please Do maintenance'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}];
	}
	}else{
	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}];
	}

Outputs	messageEvent
Meta	Lubrication Unit
Action	Event
RULE 6	
RuleId	scheduleMaintenanceMessage
Inputs	avgMaintenace, [leakageAlarmTime], simulationTime

Formula (MVEL format)	import java.util.*;
	if(maintenanceTime.size() >= 3){
	n = maintenanceTime.size() - 1;
	timeValue = 0l;
	timeValue = simulationTime - maintenanceTime[n];
	timeValue = timeValue / (1000*60*60*24);
	if(timeValue >= avgMaintenance) {
	['trigger':true, 'message':{'payload':{'value':'MM-Framework: Time for Maintenance. Please Do maintenance'}}];
	}else{
	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}];
	}
	}else{

	['trigger':false, 'message':{'payload':{'value':'All Is Well'}}]; }
Outputs	messageEvent
Meta	Lubrication Unit
Action	Event
RULE 7	
RuleId	scheduleOilChange
Inputs	avgOilChange, [oilChangeTime], simulationTime

Formula (MVEL Format)	import java.util.*;
	result = false;
	if(oilChangeTime.size() >= 3){
	n = oilChangeTime.size() - 1;
	timeValue = 0l;
	timeValue = simulationTime - oilChangeTime[n];
	timeValue = timeValue / (1000*60*60*24);
	if(timeValue >= avgOilChange) {
	result = true;
	}
	}
	result;

Outputs	oilChange
Meta	Lubrication Unit
Action	Event
RULE 8	
RuleId	scheduleFilterChange
Inputs	avgFilterChange, [filterChangeTime], simulationTime

Formula (MVEL Format)	import java.util.*;
	result = false;
	if(filterChangeTime.size() >= 3){
	n = filterChangeTime.size() - 1;
	timeValue = 0l;
	timeValue = simulationTime - filterChangeTime[n];
	timeValue = timeValue / (1000*60*60*24);
	if(timeValue >= avgFilterChange) {
	result = true;
	}
	}
	result;

Outputs	filterChange
Meta	Lubrication Unit

Action	Event

Process Monitoring

The rules as part of the function process monitoring are given below in the Table 20.

Table 20 Ontology Rule Individuals for Process Monitoring

RULE 1	
RuleId	oilConsumptionRatioLU
Inputs	inFlowLU, [outFlowLU]

Formula (MVEL format)	import java.util.*;
	result = 0.0;
	if(outFlowLU.size() >= 4){
	n = outFlowLU.size() - 1;
	result = (outFlowLU[n-3]-inFlowLU)/outFlowLU[n-3];
	}

Outputs	ocrLU
Meta	Lubrication Unit
Action	Event
RULE 2	
RuleId	oilConsumptionRatioMS
Inputs	inFlowMS, outFlowMS

Formula (MVEL format)	flow = (inFlowMS-outFlowMS)/inFlowMS;
	result = 0.0;
	if(flow > 0){
	result = flow;
	}

Outputs	ocrMS
Meta	Measuring Station
Action	Event
RULE 3	
RuleId	oilConsumptionRatioFM
Inputs	inFlowFM, outFlowFM

Formula (MVEL format)	flow = (inFlowFM-outFlowFM)/inFlowFM;
	result = 0.0;
	if(flow > 0){
	result = flow;
	}

Outputs	ocrFM
Meta	Flow Meter

Action	Event
RULE 4	
RuleId	oilLevelLowDetection
Inputs	tankCapacity, tankLevel, percentTankLow

Formula (MVEL format)	import java.util.*;
	level = Math.round(level * 100)/100;
	tankCap = ((percentTankLow)/100)* tankCapacity;
	tankCap = Math.round(tankCap * 100)/100;
	oilLow = new HashMap();
	oilLow.put('trigger', false);
	oilLow.put('message', 'All is Well');
	if(level <= tankCap){
	oilLow.trigger = true;
	oilLow.message = 'MM-Framework: Oil Level Too Low. Please Change Oil.';
	}
	oilLow;

Outputs	oilLow
Meta	Lubrication Unit
Action	Event
RULE 5	
RuleId	oilLowLevelHMITrigger
Inputs	oilLow
Formula (MVEL Format)	oilLow.trigger;
Outputs	oilChange
Meta	Lubrication Unit
Action	Event
RULE 6	
RuleId	oilLowLevelStopTrigger
Inputs	oilLow
Formula (MVEL Format)	oilLow;
Outputs	stopUnit
Meta	Lubrication Unit
Action	Event
RULE 7	
RuleId	oilLowLevelMessageTrigger

Inputs	oilLow
Formula (MVEL format)	import java.util.*; ['trigger':oilLow.trigger, 'message':{'payload':{'value':oilLow.message}}];
Outputs	messageEvent
Meta	Lubrication Unit
Action	Event

Quality Monitoring

The rules as part of the function quality monitoring are given below in the Table 21.

Table 21 Ontology Rule Individuals for Quality Monitoring

RULE 1	
RuleId	oilQualityDecayRateCalculation
Inputs	[filterChangeTime]

Formula (MVEL format)	import java.util.*;
	result = 0.0;
	if(filterChangeTime.size() >= 3){
	n = (filterChangeTime.size() - 1);
	result = (((filterChangeTime[n-1]-filterChangeTime[n])-(filterChangeTime[0]-filterChangeTime[1]))/(filterChangeTime[n-1]-filterChangeTime[n]))*100;
	}
	result;

Outputs	oilQualityDecayRate
Meta	Lubrication Unit
Action	Event
Rule 2	
RuleId	oilQualityMonitoring
Inputs	oilQualityDecayRate, particleCount, percentQualityDecay, particleQuality

Formula (MVEL format)	import java.util.*;
	result = false;
	if((oilQualityDecayRate >= percentQualityDecay)){
	if(Integer.parseInt(particleCount.substring(0, particleCount.indexOf('/')).replaceAll('\\s+','')) >= Integer.parseInt(particleQuality.substring(0, particleQuality.indexOf('/')))){
	particleCount = particleCount.substring(particleCount.indexOf('/')+1);

	particleQuality = particleQuality.substring(particleQuality.indexOf('/')+1);
	if(Integer.parseInt(particleCount.substring(0, particleCount.indexOf('/')).replaceAll('\\s+','')) >= Integer.parseInt(particleQuality.substring(0, particleQuality.indexOf('/')))){
	particleCount = particleCount.substring(particleCount.indexOf('/')+1);
	particleQuality = particleQuality.substring(particleQuality.indexOf('/')+1);
	if(Integer.parseInt(particleCount.replaceAll('\\s+','')) >= Integer.parseInt(particleQuality)){
	result = true;
	}
	}
	}
	}
	result;

Outputs	oilDirty
Meta	Lubrication Unit
Action	Event
Rule 3	
RuleId	oilQualityMessageGeneration
Inputs	oilDirty

Formula (MVEL format)	import java.util.*;
	if(oilDirty){
	['trigger':oilDirty, 'message':{'payload':{'value':'MM-Framework: Oil Dirty. Please change oil.'}}];
	}else{
	['trigger':oilDirty, 'message':{'payload':{'value':''}}];
	}

Outputs	messageEvent
Meta	Lubrication Unit
Action	Event
RULE 4	
RuleId	oilChangeTrigger
Inputs	oilDirty
Formula (MVEL Format)	oilDirty;
Outputs	oilChange

Meta	Lubrication Unit
Action	Event

System Controller

The rules as part of the function system controller are given below in the Table 22.

Table 22 Ontology Rule Individuals for System Controller

RULE 1	
RuleId	stopServiceInvoke
Inputs	stopUnit
Formula (MVEL Format)	import java.util.*; ['trigger':stopUnit.trigger, 'message': '{}'];
Outputs	serviceOutput
Meta	Lubrication Unit
Action	Invoke Service
ServiceId	stop

Function

The list of functions in the framework are given below in the Table 23.

Table 23 Ontology Function Individuals (OLS Simulator Ontology)

ID	Rules	Description
conditionMonitoring	startConditionCheck, conditionMonitoringMessageGeneration	Function that monitors the condition of Lubrication unit. It monitors by checking whether the LU is suitable for operation at the start of the system.
dataAcquisition	ontologySaveOutFlowLU, saveLastMaintenenceDoneTime, saveLastFilterChangeTime, saveLastOilChangeTime	Function that takes care of saving data to Ontology.
hmiCoordinator	hmiMessageInvoke, hmiOilChangeInvoke, hmiFilterChangeInvoke	Function that invokes the service with respect to HMI of OLS simulator.

immediateMaintenance	lubricationUnitLeakage, measuringStationLeakage, flowMeterLeakage	Function that monitors the leakage in components and raises alarm for maintenance if leakage occurs.
predictiveMaintenance	averageFilterChangeCalculation, averageOilChangeCalculation, averageMaintenanceCalculation, scheduleOilChangeMessage, scheduleFilterChangeMessage, scheduleMaintenanceMessage, scheduleOilChange, scheduleFilterChange	Function that performs both preventive and predictive maintenance for Lubrication System.
processMonitoring	oilConsumptionRatioLU, oilConsumptionRatioMS, oilConsumptionRatioFM, oilLevelLowDetection, oilLowLevelHMITrigger, oilLowLevelStopTrigger, oilLowLevelMessageTrigger	Function that monitors the whole process pf a Lubrication system.
qualityMonitoring	oilQualityDecayRateCalculation, oilQualityMonitoring, oilQualityMessageGeneration, oilChangeTrigger	Function that monitors the Quality of oil in Lubrication unit. If the quality is degraded, it informs the respective Lubrication Unit to change Oil.
systemController	stopServiceInvoke	Function that invokes services in Lubrication Unit of OLS simulator.

Appendix C – Manufacturing Functions

Input

Inputs are the basic necessary elements for the rules that are executed in the framework. The list of inputs added as part of the OLS production MES are listed in Table 24.

Table 24 Ontology Input Individuals (OLS Production MES)

ID	Formula ID	Category	Meta	Type	Value	Description
eventList	-	Query	DeviceId_RestTranslator, ServiceId_EventList	Array	-	List of events registered for each subproject. The details are fetched from Rest translator.
subProjectTagDetails	-	Event	EventId_SubProjectTag	map	-	Details of the subproject and the tag to which it is associated with. This is emitted as an event by Tablet devices in shop floor.
subProjectTagDetails	-	Ontology	DeviceType_Ontology	array	-	Array of details consisting of the subproject and the tag queried from ontology.
subProjectRequirement	-	ServiceInput	DeviceType_MERP	map	-	Details of the subproject to which the framework must suggest the resources to be allocated.

simulationTime	-	Time	DeviceType_Framework	long		Time from framework.
subPojectTag	-	ServiceInput	DeviceType_ProductTracker	map	-	Input as part of REST Service request. The input has the tag details of a subproject for which it needs the details.
resourceAvailabilityDetails	-	Query	DeviceId_RestTranslator, ServiceId_ResourceWeekList	map	-	Details of the resource with type of work he can perform and the no of hours he is available.
subProjectAll	-	Query	DeviceId_RestTranslator, ServiceId_AllSubProjects	array	-	Details of each subproject.
resourceDetails	-	ServiceInput	DeviceType_ProductTracker	map	-	Input as part of REST Service request. The input has the resource details of a particular resource.
resourceAll	-	Query	DeviceId_RestTranslator, ServiceId_AllResource	array	-	Details of each resource which will be used to identify the technician and the other user.

Output

The list of outputs for the rules that are executed in the framework are given below in the Table 25.

Table 25 Ontology Output Individuals (OLS Production MES)

ID	Category	Type	Description
allSubProjectResourceData	Value	array	Details of all subprojects with respect to resources. This will be sent as reply to the service.
resourceSubProjectData	Value	map	Details of a particular resource with respect to subprojects he has worked. This will be sent as reply to the service.
resourceSuggestion	Value	map	Suggested resource allocation details. This will be sent as reply to the service.
subProjectResourceData	Value	map	Details of a particular subproject with respect to resources. This will be sent as reply to the service
subProjectTagDetails	Value	map	Value that will be stored to ontology.

Meta

The list of Meta for the elements in the framework are given below in the Table 26.

Table 26 Ontology Meta Individuals (OLS Production MES)

Name	ID	Value
DeviceType_MES	deviceType	MES
DeviceType_Product-Tracker	deviceType	productTracker
DeviceType_Produc-tionOrganizer	deviceType	productionOrganizer
DeviceType_Ontology	deviceType	ontology
DeviceId_RESTTranslator	deviceId	Rest_Translator
EventId_LogEvent	eventId	logEvent
EventId_SubProjectTag	eventId	subProjectTag
ServiceId_AllResource	serviceId	allResource
ServiceId_AllSubProjects	serviceId	allSubProjects
ServiceId_EventList	serviceId	eventList
ServiceId_Re-sourceWeekList	serviceId	resourceWeekList

Rule

Resource Allocation and Status

The rules as part of the function resource allocation and status are given below in the Table 27.

Table 27 Ontology Rule Individuals for Resource Allocation and Status

RULE 1	
RuleId	resourceAllocationSuggestion
Inputs	resourceAvailabilityDetails, subProjectRequirement

Formula (MVEL format)	import java.text.SimpleDateFormat;
	import java.util.*;
	SimpleDateFormat a = new SimpleDateFormat('yyyy-MM-dd HH:mm:ss.SSS');
	resourceId = resource.id;
	pcsArray = [];
	totalWorkTime = 0;
	totalLogTime = [];
	resourceSubProjectData = new HashMap();
	for (event : eventList){
	Date c = a.parse(event.timestamp);
	tim = c.getTime();
	if((event.resourceid == resourceId) && (!pcsArray.contains(event.pcs))){
	pcsArray.add(event.pcs);
	}
	if((event.resourceid == resourceId) && (!totalLogTime.contains(tim))){
	totalLogTime.add(tim);
	}
	}
	Collections.sort(totalLogTime);
	currentProject = '';
	currentPcs = '';
	currentStatus = '';
	for(event : eventList){
	Date c = a.parse(event.timestamp);
	tim = c.getTime();
	if((event.resourceid == resourceId) && (tim == totalLogTime.get(totalLogTime.size() - 1))){

```
                currentProject = event.project;
                currentPcs = event.pcs;
                currentStatus = event.projectstatus;
        }
}
resourceInitial = '';
for(indRes : resourceAll){
        if(indRes.resourceID == resourceId){
                resourceInitial = indRes.initials;
        }
}
subProjectList=[];
for(pcs : pcsArray){
        subProjectdata = new HashMap();
        resourceandAllocated = [];
        subProjectLogTime = [];
        subProjectdata.put('pcs',pcs);
        for(subProjRT : subProjectAll){
                if(subProjRT.pcs == pcs){
                        subProjectdata.put('project',subProjRT.pro-
ject);
                        for(i=0;i<=20;i++){
                                resName = 'resource'+i;
                                resHour = resName+'Hours';
                                resInitialHash = new HashMap();
                                if((subProjRT.get(resName) != null) &&
(subProjRT.get(resHour) != null)){
                                        resInitialHash.put('initial',subPro-
jRT.get(resName));
                                        resInitialHash.put('hours',subPro-
jRT.get(resHour));
                                        resourceandAllo-
cated.add(resInitialHash);
                                }
                        }
                }
        }
        totAllocTime = 0;
        for(resourceAllo : resourceandAllocated){
                if(resourceAllo.initial == resourceInitial){
```

```
                        totAllocTime = resourceAllo.hours;
                }
        }
        subProjectdata.put('totalallocatedtime',totAllocTime);
        for(event : eventList){
                Date c = a.parse(event.timestamp);
                tim = c.getTime();
                if((event.resourceid == resourceId) && (event.pcs ==
pcs) && (!subProjectLogTime.contains(tim))){
                        subProjectLogTime.add(tim);
                }
        }
        Collections.sort(subProjectLogTime);
        lastStatus = '';
        subProjectLogDetails = [];
        for(d : subProjectLogTime){
                resDetail = new HashMap();
                for(event : eventList){
                        Date c = a.parse(event.timestamp);
                        tim = c.getTime();
                        if((event.resourceid == resourceId) &&
(event.pcs == pcs) && (tim == d)){
                                resDetail.put('time', d);
                                resDetail.put('action', event.projectsta-
tus);
                                if(d == subProjectLogTime.get(subPro-
jectLogTime.size() - 1)){
                                        subProjectdata.put('lastStatus',
event.projectstatus);
                                }
                        }
                }
                subProjectLogDetails.add(resDetail);
        }
        timeHour = 0;
        stTime = 0;
        setFlag = false;
        for(e : subProjectLogDetails){
                if(e.action == 'Start'){
                        stTime = e.time;
                        setFlag = true;
```

	}else if((e.action == 'Stop' || e.action == 'Pause') && (setFlag)){
	timeHour = timeHour + Math.round((e.time - stTime)/(1000*60*60));
	setFlag = false;
	}
	}
	subProjectdata.put('totaltime',timeHour);
	totalWorkTime = totalWorkTime + timeHour;
	subProjectList.add(subProjectdata);
	}
	resourceSubProjectData.put('resourceid', resourceId);
	resourceSubProjectData.put('initial', resourceInitial);
	resourceSubProjectData.put('totaltime', totalWorkTime);
	resourceSubProjectData.put('currentproject', currentProject);
	resourceSubProjectData.put('currentpcs', currentPcs);
	resourceSubProjectData.put('currentstatus', currentStatus);
	resourceSubProjectData.put('subprojectlist', subProjectList);
	resourceSubProjectData;

Outputs	resourceSuggestion
Meta	ProductionOrganizer
Action	ServiceReply

Data Collection and Acquisition

The rules as part of the function data collection and acquisition are given below in the Table 28.

Table 28 Ontology Rule Individuals for Data Collection and Acquisition

RULE 1	
RuleId	saveProjectTagId
Inputs	subProjectTagDetails
Formula (MVEL format)	subProjectTagDetails;
Outputs	subProjectTagDetails
Meta	Ontology
Action	Save to Ontology

Labour Management

The rules as part of the function labour management are given below in the Table 29.

Table 29 Ontology Rule Individuals for Labour Management

RULE 1	
RuleId	resourceSubProjectDetails
Inputs	eventList, resourceAll, resourceDetails, subProjectAll

Formula (MVEL format)	import java.util.HashMap;
	import java.util.ArrayList;
	suggestionHash = new HashMap();
	people = subProjectRequirement.resourcenumber;
	minimumNoPeople = 0;
	assemblyPeople = 0;
	electricalPeople = 0;
	weldingPeople = 0;
	weldingHours = 0l;
	assemblyResourcePeople = 0;
	electricalResourcePeople = 0;
	weldingResourcePeople = 0;
	unavailableResourceNumber = 0;
	criticalCase = false;
	for(b : resourceAvailabilityDetails){
	if ((b.type.toLowerCase() =='assembly') && (b.availability.size() > 0)) {
	assemblyResourcePeople = assemblyResourcePeople + 1;
	} else if (b.type.toLowerCase() =='electrical' && (b.availability.size() > 0)){
	electricalResourcePeople = electricalResourcePeople + 1;
	} else if (b.type.toLowerCase() =='welding' && (b.availability.size() > 0)){
	weldingResourcePeople = weldingResourcePeople + 1;
	}
	}
	if(subProjectRequirement.weldingwork){
	noWeldingRes = 0;
	noWeldingRes = Math.round(people/3);

```
        weldingHours = Math.round(subProjectRequirement.as-
semblyhours / 3);
        subProjectRequirement.assemblyhours = subProjec-
tRequirement.assemblyhours - weldingHours;
        if(noWeldingRes > weldingResourcePeople){
                weldingPeople = weldingResourcePeople;
                people = people - weldingResourcePeople;
        }else{
                weldingPeople = noWeldingRes;
                people = people - noWeldingRes;
        }
}
if((subProjectRequirement.electricalwork) && (subProjec-
tRequirement.assemblywork)){
        noAssemblyRes = 0;
        noElectricalRes = 0;
        noHours = subProjectRequirement.assemblyhours + sub-
ProjectRequirement.electricalhours;
        noAssemblyRes = Math.round(people * (subProjectRequire-
ment.assemblyhours / noHours));
        noElectricalRes = people - noAssemblyRes;
        if(noElectricalRes < electricalResourcePeople){
                electricalPeople = noElectricalRes;
        }else{
                electricalPeople = electricalResourcePeople;
                noAssemblyRes += Integer.valueOf(noElectricalRes) -
Integer.valueOf(electricalResourcePeople);
        }
        if(noAssemblyRes < assemblyResourcePeople){
                assemblyPeople = noAssemblyRes;
        } else {
                assemblyPeople = assemblyResourcePeople;
        }
}else if(subProjectRequirement.assemblywork){
        if(people < assemblyResourcePeople){
                assemblyPeople = people;
        } else {
                assemblyPeople = assemblyResourcePeople;
        }
} else if(subProjectRequirement.electricalwork){
        if(people < electricalResourcePeople){
```

```
                electricalPeople = people;
        } else{
                electricalPeople = electricalResourcePeople;
        }
}
allocatedResource = [];
if((subProjectRequirement.assemblywork) && (assemblyPeople > 0)){
        hoursForEach = Math.round(subProjectRequirement.assemblyhours / assemblyPeople);
        for(b : resourceAvailabilityDetails){
                resourceHash = new HashMap();
                hours = hoursForEach;
                flagAssembly = false;
                if ((b.type.toLowerCase() =='assembly') && (assemblyPeople > 0)){
                        availability = [];
                        for(d: b.availability){
                                if((d.week >= subProjectRequirement.startweek) &&(d.availablehours > 0) && (hours > 0)){
                                        availability.add(d);
                                        if(d.availablehours< hours){
                                                hours = hours - d.availablehours;
                                        }else{
                                                hours = 0;
                                        }
                                        flagAssembly = true;
                                }
                        }
                        if(flagAssembly){
                                assemblyPeople = Integer.valueOf(assemblyPeople) - 1;
                                resourceHash.put('resourceid' , b.resourceid);
                                resourceHash.put('resourcename', b.resourcename);
                                resourceHash.put('availability' , availability);
                                allocatedResource.add(resourceHash);
                        }
```

```
                }
        }
}
if((subProjectRequirement.electricalwork) && (electricalPeople > 0)){
        hoursForEach = Math.round(subProjectRequirement.electricalhours / electricalPeople);
        for(b : resourceAvailabilityDetails){
                resourceHash = new HashMap();
                hours = hoursForEach;
                flagElectrical = false;
                if((b.type.toLowerCase() == 'electrical') && (electricalPeople > 0)){
                        availability = [];
                        for(d: b.availability){
                                if((d.week >= subProjectRequirement.startweek) &&(d.availablehours > 0) && (hours > 0)){
                                        availability.add(d);
                                        if(d.availablehours< hours){
                                                hours = hours - d.availablehours;
                                        }else{
                                                hours = 0
                                        }
                                        flagElectrical = true;
                                }
                        }
                        if(flagElectrical){
                                electricalPeople = Integer.valueOf(electricalPeople) -1;
                                resourceHash.put('resourceid', b.resourceid);
                                resourceHash.put('resourcename', b.resourcename);
                                resourceHash.put('availability', availability);
                                allocatedResource.add(resourceHash);
                        }
                }
        }
}
```

```
if(subProjectRequirement.weldingwork){
        hoursForEach = Math.round(weldingHours / weldingPeo-
ple);
        for(b : resourceAvailabilityDetails){
                resourceHash = new HashMap();
                hours = hoursForEach;
                flagWelding = false;
                if((b.type.toLowerCase() == 'welding') && (weld-
ingPeople > 0)){
                        availability = [];
                        for(d: b.availability){
                                if((d.week >= subProjectRequire-
ment.startweek) &&(d.availablehours > 0) && (hours > 0)){
                                        availability.add(d);
                                        if(d.availablehours< hours){
                                                hours = hours - d.available-
hours;
                                        }else{
                                                hours = 0
                                        }
                                        flagWelding = true;
                                }
                        }
                        if(flagWelding){
                                weldingPeople = Integer.valueOf(weld-
ingPeople) -1;
                                resourceHash.put('resourceid', b.re-
sourceid);
                                resourceHash.put('resourcename', b.re-
sourcename);
                                resourceHash.put('availability', availa-
bility);
                                allocatedResource.add(resourceHash);
                        }
                }
        }
}
suggestionHash.put('subprojectid', subProjectRequirement.sub-
projectid);
suggestionHash.put('AllocatedResource' , allocatedResource);
suggestionHash;
```

Outputs	resourceSubProjectData
Meta	ProductTracker
Action	serviceReply

Product Tracking and Genealogy

The rules as part of the function product tracking and genealogy are given below in the Table 30.

Table 30 Ontology Rule Individuals for Product Tracking And Genealogy

RULE 1	
RuleId	subProjectResourceDetails
Inputs	evenetList, resourcerAll, simulationTime, subProjectAll, subProjectTagDetails, subprojectTag

Formula (MVEL format)	import java.text.SimpleDateFormat;
	import java.util.*;
	SimpleDateFormat a = new SimpleDateFormat('yyyy-MM-dd HH:mm:ss.SSS');
	resourceIds = [];
	subProjectResourceDetails = new HashMap();
	eventTime = [];
	resourceandAllocated = [];
	subTotalTime = 0;
	subTotalAllocatedTime = 0;
	subprojectid = '';
	pcsNumber = '';
	projId = '';
	actualStartDateArray = [];
	for(subProjectEntry : subProjectTagDetails){
	  if(subProjectTag.id == subProjectEntry.tagid){
	  subprojectid = subProjectEntry.subproject;
	  pcsNumber = subProjectEntry.pcs;
	  }
	}
	for(subProj : subProjectAll){
	  if((subProj.subproject == subprojectid) && (subProj.pcs == pcsNumber)){
	  projId = subProj.project;
	  for(i=0;i<=20;i++){
	  resName = 'resource'+i;
	  resHour = resName+'Hours';

```
                        resInitialHash = new HashMap();
                        if((subProj.get(resName) != null) && (sub-
Proj.get(resHour) != null)){
                                resInitialHash.put('initial',sub-
Proj.get(resName));
                                resInitialHash.put('hours',sub-
Proj.get(resHour));
                                resourceandAllo-
cated.add(resInitialHash);
                        }
                }
        }
}
for(b : eventList){
        if((b.project == projId) &&(b.pcs == pcsNumber)){
                if(!resourceIds.contains(b.resourceid)){
                        resourceIds.add(b.resourceid);
                }
        }
}
resourceList = [];
for(resourceId : resourceIds){
        resourceWorkHours = new HashMap();
        for(b : eventList){
                if((b.resourceid == resourceId) && (b.project == pro-
jId) && (b.pcs == pcsNumber)) {
                        Date c = a.parse(b.timestamp);
                        tim = c.getTime();
                        eventTime.add(tim);
                }
        }
        res = [];
        Collections.sort(eventTime);
        actualStartDateArray.add(eventTime[0]);
        for(d : eventTime){
                resDetail = new HashMap();
                for(b : eventList){
                        Date c = a.parse(b.timestamp);
                        tim = c.getTime();
                        if((b.resourceid == resourceId) && (tim == d)){
                                resDetail.put('time',d);
```

```
                                resDetail.put('action',b.projectstatus);
                        }
                }
                if(!resDetail.isEmpty()){
                        res.add(resDetail);
                }
        }
        timeHour = 0;
        stTime = 0;
        setFlag = false;
        for(e :res){
                if(e.action == 'Start'){
                        stTime = e.time;
                        setFlag = true;
                }else if((e.action == 'Stop' || e.action == 'Pause') &&
(setFlag)){
                        timeHour = timeHour + Math.round((e.time -
stTime)/(1000*60*60));
                        setFlag = false;
                }
        }
        resourceInitial = '';
        for(indRes : resourceAll){
                if(indRes.resourceID == resourceId){
                        resourceInitial = indRes.initials;
                }
        }
        allocatedTime = 0;
        for(indResAlloc : resourceandAllocated){
                if(indResAlloc.initial == resourceInitial){
                        allocatedTime = indResAlloc.hours;
                }
        }
        subTotalTime +=  timeHour;
        subTotalAllocatedTime += allocatedTime;
        resourceWorkHours.put('resourceId', resourceId);
        resourceWorkHours.put('totalTime',timeHour);
        resourceWorkHours.put('allocatedTime',allocatedTime);
        resourceWorkHours.put('initials',resourceInitial);
        resourceList.add(resourceWorkHours);
}
```

```
Collections.sort(actualStartDateArray);
dateFormatted = '';
if(!actualStartDateArray.isEmpty()){
        Date date = new Date(actualStartDateArray[0]);
        dateFormatted = a.format(date);
        subProjectResourceDetails.put('actualStartDate', dateFor-
matted);
}
subProjectResourceDetails.put('ptuserTimeList', resourceList);
subProjectResourceDetails.put('totalTime', subTotalTime);
subProjectResourceDetails.put('allocatedTotalTime', subTotalAllo-
catedTime);
for(subProj : subProjectAll){
        if((subProj.subproject == subprojectid) && (subProj.pcs ==
pcsNumber)){
                subProjectResourceDetails.put('project', subProj.pro-
ject + ', ' + subProj.pcs + ', ' + subProj.description);
                subProjectResourceDetails.put('po', subProj.po);
                subProjectResourceDetails.put('orderDate', sub-
Proj.orderDate);
                subProjectResourceDetails.put('customer', sub-
Proj.customer);
                subProjectResourceDetails.put('pcs', subProj.pcs);
                subProjectResourceDetails.put('plannedComple-
tionDate', subProj.plannedCompletionDate);
                subProjectResourceDetails.put('plannedStartOfPro-
duction', subProj.plannedStartOfProduction);
                subProjectResourceDetails.put('componentsExpDate',
subProj.componentsExpDate);
                if(subProj.actualCompletionDate == null){
                        Date date = new Date(simulationTime);
                        dateFormatted = a.format(date);
                        subProjectResourceDetails.put('actualComple-
tionDate', dateFormatted);
                }else{
                        subProjectResourceDetails.put('actualComple-
tionDate', subProj.actualCompletionDate);
                }
        }
}
subProjectResourceDetails;
```

Outputs	subProjectResourceData
Meta	ProductionOrganizer
Action	ServiceReply
RULE 2	
RuleId	allSubProjectResourceDetails
Inputs	evenetList, resourcerAll, simulationTime, subProjectAll, subProjectTagDetails

Formula (MVEL format)	import java.text.SimpleDateFormat;
	import java.util.*;
	SimpleDateFormat a = new SimpleDateFormat('yyyy-MM-dd HH:mm:ss.SSS');
	resourceIds = [];
	eventTime = [];
	resourceandAllocated = [];
	subTotalTime = 0;
	subTotalAllocatedTime = 0;
	actualStartDateArray = [];
	allSubProjectResourceDetails = [];
	for(subProjectEntry : subProjectTagDetails){
	subProjectResourceDetails = new HashMap();
	subprojectid = '';
	pcsNumber = '';
	projId = '';
	subprojectid = subProjectEntry.subproject;
	pcsNumber = subProjectEntry.pcs;
	for(subProj : subProjectAll){
	if((subProj.subproject == subprojectid) && (subProj.pcs == pcsNumber)){
	projId = subProj.project;
	for(i=0;i<=20;i++){
	resName = 'resource'+i;
	resHour = resName+'Hours';
	resInitialHash = new HashMap();
	if((subProj.get(resName) != null) && (subProj.get(resHour) != null)){
	resInitialHash.put('initial',subProj.get(resName));
	resInitialHash.put('hours',subProj.get(resHour));
	resourceandAllocated.add(resInitialHash);
	}
	}
	}

```
}
for(b : eventList){
if((b.project == projId) &&(b.pcs == pcsNumber)){
if(!resourceIds.contains(b.resourceid)){
resourceIds.add(b.resourceid);
}
}
}
resourceList = [];
for(resourceId : resourceIds){
resourceWorkHours = new HashMap();
for(b : eventList){
if((b.resourceid == resourceId) && (b.project == projId) && (b.pcs
== pcsNumber)) {
Date c = a.parse(b.timestamp);
tim = c.getTime();
eventTime.add(tim);
}
}
res = [];
Collections.sort(eventTime);
actualStartDateArray.add(eventTime[0]);
for(d : eventTime){
resDetail = new HashMap();
for(b : eventList){
Date c = a.parse(b.timestamp);
tim = c.getTime();
if((b.resourceid == resourceId) && (tim == d)){
resDetail.put('time',d);
resDetail.put('action',b.projectstatus);
}
}
if(!resDetail.isEmpty()){
res.add(resDetail);
}
}
timeHour = 0;
stTime = 0;
setFlag = false;
for(e :res){
if(e.action == 'Start'){
```

```
stTime = e.time;
setFlag = true;
}else if((e.action == 'Stop' || e.action == 'Pause') && (setFlag)){
timeHour = timeHour + Math.round((e.time -
stTime)/(1000*60*60));
setFlag = false;
}
}
resourceInitial = '';
for(indRes : resourceAll){
if(indRes.resourceID == resourceId){
resourceInitial = indRes.initials;
}
}
allocatedTime = 0;
for(indResAlloc : resourceandAllocated){
if(indResAlloc.initial == resourceInitial){
allocatedTime = indResAlloc.hours;
}
}
subTotalTime +=  timeHour;
subTotalAllocatedTime += allocatedTime;
resourceWorkHours.put('resourceId', resourceId);
resourceWorkHours.put('totalTime',timeHour);
resourceWorkHours.put('allocatedTime',allocatedTime);
resourceWorkHours.put('initials',resourceInitial);
resourceList.add(resourceWorkHours);
}
Collections.sort(actualStartDateArray);
dateFormatted = '';
if(!actualStartDateArray.isEmpty()){
Date date = new Date(actualStartDateArray[0]);
dateFormatted = a.format(date);
subProjectResourceDetails.put('actualStartDate', dateFormatted);
}else{
Date date = new Date(simulationTime);
dateFormatted = a.format(date);
subProjectResourceDetails.put('actualStartDate', dateFormatted);
}
subProjectResourceDetails.put('ptuserTimeList', resourceList);
subProjectResourceDetails.put('totalTime', subTotalTime);
```

	subProjectResourceDetails.put('allocatedTotalTime', subTotalAllocatedTime);
	for(subProj : subProjectAll){
	if((subProj.subproject == subprojectid) && (subProj.pcs == pcsNumber)){
	subProjectResourceDetails.put('project', subProj.project + ', ' + subProj.pcs + ', ' + subProj.description);
	subProjectResourceDetails.put('po', subProj.po);
	subProjectResourceDetails.put('orderDate', subProj.orderDate);
	subProjectResourceDetails.put('customer', subProj.customer);
	subProjectResourceDetails.put('pcs', subProj.pcs);
	subProjectResourceDetails.put('plannedCompletionDate', subProj.plannedCompletionDate);
	subProjectResourceDetails.put('plannedStartOfProduction', subProj.plannedStartOfProduction);
	subProjectResourceDetails.put('componentsExpDate', subProj.componentsExpDate);
	if(subProj.actualCompletionDate == null){
	Date date = new Date(simulationTime);
	dateFormatted = a.format(date);
	subProjectResourceDetails.put('actualCompletionDate', dateFormatted);
	}else{
	subProjectResourceDetails.put('actualCompletionDate', subProj.actualCompletionDate);
	}
	}
	}
	allSubProjectResourceDetails.add(subProjectResourceDetails);
	}
	allSubProjectResourceDetails;

Outputs	allSubProjectResourceData
Meta	ProductionOrganizer
Action	ServiceReply

Function

The list of functions in the framework are given below in the Table 31.

Table 31Ontology Function Individuals (OLS Production MES)

ID	Rules	Description
resourceAllocationAndStatus	resourceAllocationSuggestion	Function that has the functionality of MES function 'Resource Allocation and Status'. In here it does the work of suggesting the suitable resources to be allocated to a project.
dataCollectionAndAcquisition	saveProjectTagId	Function that has the functionality of MES function 'Data Collection and Acquisition'. In here it does the work of collecting the information form other systems and storing them to ontology.
laborManagement	resourceSubProjectDetails	Function that has the functionality of MES function 'Labour Management'. In here it does the work of calculating subproject details for each resource.
productTrackingAndGenealogy	subProjectResourceDetails, allSubProjectResourceDetails	Function that has the functionality of MES function 'Labour Management'. In here it does the work of calculating resource details for each and all sub projects.