TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

WAEL MOHAMMED

ENCAPSULATION OF MES FUNCTIONALITIES AS RESTFUL WEB SERVICES FOR KNOWLEDGE-DRIVEN MANUFACTURING SYSTEMS

Master of Science thesis

Examiners: Dr. Andrei Lobov and Prof. Jose L. Martinez Lastra
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences
on 7th of October, 2015

ABSTRACT

WAEL MMOHAMMED: ENCAPSULATION OF MES FUNCTIONAL-ITIES AS RESTFUL WEB SERVICES FOR KNOWLEDGE DRIVEN MANUFACTURING SYSTEMS

Tampere University of technology
Master of Science Thesis, 47 pages, 5 Appendix pages
January 2017
Master's Degree Program in Automation Engineering
Major: FACTORY AUTOMATION AND INDUSTRIAL INFORMATICS
Examiner: Dr. Andrei Lobov and Prof. Jose L. Martinez Lastra
Supervisor: Dr. Andrei Lobov

Keywords: Open knowledge driven system, OKD, manufacturing Execution system, MES, Ontologies, OWL, Orchestration, Orchestrator, ORL, RESTful Web Services

Computer and network technologies are growing rapidly nowadays. For this reason, many doors are opened for implementing some of these technologies in different areas. Currently, manufacturing systems is considered as main consumer of new technologies such as web services and knowledge-based systems. therefore, new terms came to the surface like industry 4.0 and IoT devices. An example could be seen in the eScop project. The concept idea is addressed as designing a knowledge driven manufacturing system which is capable to be applied on various industrial facilities. In this regards, this thesis aims to define MES functions for the Open Knowledge-Driven Manufacturing Execution System (OKD-MES). The study took place on FASTory line located at FAST Lab. In Tampere University of Technology. It is used as a validation case study to proof the implementation of the functions.

The objective of this thesis focuses on employing the MES functions as web services to suit the OKD-MES concept. In this research, the state of art reviews MES functions which were defined at a general level by MESA. Then, these MES functions are specifically defined to work with the layer concept of the OKD-MES which presented by the eScop project. The approach which has been developed for MES functions is a general platform that can be used for all MES functions even though the industry type is different. This approach tends to maximize the flexibility of configuring MES functions in the manufacturing system. At the same time, it minimizes the dependencies in the OKD-MES. With this approach, the user is responsible for defining the description of web services in ontology form as configuration and providing the functionality that suits the manufacturing system as functional scripts. These features of the presented approach allow the user to implement new logic for the MES functions or even use ready-made tools.

Finally, the developed approach is tested with a production scenario. The results of the test show the advantage of using this approach in terms of configurability and simplicity of installation. Nevertheless, the presented approach holds chances for future development.

PREFACE

بسم الله الرحمن الرحيم

الحمد لله رب العالمين و الصلاة و السلام على اشرف الخلق و المرسلين, سيدنا محمد, وعلى اله و صحبه اجمعين. بدايةً, اود ان أشكر أبي و أمي و جميع الأهل و الأصدقاء على دعمهم و مساندتهم طوال فترة دراستي لدرجة الماجستير. كما أود أن أشكر قسم الفاستلاب. على ما قدموه من مساعده و تشجيع لي, و هنا, أود أن أخص بالشكر كل من الدكتور أندري لوبوف و البروفسور جوزيه لاسترا. ثم أخيرا و ليس أخيرا، أتوجه بالشكر إلى فريق إسكوب الذي قدم الدعم و الفرصه لتقديم أطروحتي هذه. حيث ان هذه الاطروحة تمت بدعم من مشروع إسكوب.

In The Name of Allah, The Most Gracious and The Most Merciful

Praise be to God and prayer and peace be upon His prophet and messenger Muhammad, and his family and all his companions. First of all, I would like to thank my father, my mother, my family and my friends for their support for the period of my studies for the master degree. I would like also to thank the FAST-Lab for their help and encourage. And here, I would like to especially thank Dr. Andrei Lobov and Professor Jose Lastra for the help and guidance that they provide to me. Then, last but not least, I thank the eScop team who provided support and the opportunity to make this thesis. As this thesis has been supported by eScop project.

Tampere, 15th Dec,2016

Wael M. Mohammed

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
BPEL	Business Process Execution Language
DC	Device Catalogue
DSC	Distributed Control System
DPWS	Device Profile Web Services
ERP	Enterprise resource planning
GUI	Graphical User Interface
HMI	Human-Machine Interface
HTML	Hypertext Markup Language
HTTP	The Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MES-F1	MES function 1 (Resource Allocation and Status)
MES-F2	MES function 2 (Operations/ Detail Scheduling)
MES-F3	MES function 3 (Dispatching Production Units)
MES-F4	MES function 4 (Document Control)
MES-F5	MES function 5 (Data Collection / Acquisition)
MES-F6	MES function 6 (Resource Allocation and Status)
MES-F7	MES function 7 (Labor Management)
MES-F8	MES function 8 (Quality Management)
MES-F9	MES function 9 (Maintenance Management)
MES-F10	MES function 10 (Products Tracking and Genealogy)
MES-F11	MES function 11 (Performance Analysis)
MSO	Manufacturing System Ontology
OKD-MES	Open Knowledge Driven Manufacturing Execution System
ORL	Orchestration Layer
OWL	Web Ontology Language
PHL	Physical Layer
RPL	Representational State Transfer
RPL	Representation Layer
RTU	Remote Terminal Unit
RFID	Radio Frequency Identifier
SME	Small and Medium-sized Enterprises
SOAP	Simple Object Access Protocol
SPARQL	Simple Protocol and RDF Query Language
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
VIS	Visualization Layer
WSDL	Web Service Description Language

1 INTRODUCTION

This thesis presents an approach for implementing MES functions in open knowledge driven manufacturing system. The task is to redesign MES functions as framework which provides web services to suit the OKD-MES requirements. This implantation takes place on FASTory assembly line in FAST Lab. in Tampere University of Technology. OKD-MES is considered as main stream nowadays in development of manufacturing systems. Therefore, many conferences and journals publish hundreds of articles about knowledge driven system.

This document is divided into six chapters. First chapter is the introduction which shows the motivation, objectives, challenges and limitations of the concept of this thesis. The second chapter illustrates the background of manufacturing execution systems, the international standards related it to and open-knowledge driven system as state of art. The third chapter presents the methodology of developing MES functions. This chapter illustrates the developed approach, tools that are used for development and techniques behind the development. The fourth chapter presents the implementation of the concept on FASTory assembly line. It provides an example of using the presented approach in chapter three. This example is illustrated by using five MES functions. The fifth chapter shows a discussion of the implementation that illustrated in chapter 4. This chapter provides the study case results such as the gained value and the feasibility of the concept and it shows a comparison between OKD system with other manufacturing systems which have been applied on the line as well. The final chapter presents the conclusion of this thesis. Also it points out the future development which can be applied on the concept.

1.1 Thesis Background

Tampere University of Technology was a partner in the eScop project. The eScop project aims to produce a sufficient and general OKD-MES that can be applied on various manufacturing systems. An important part of this project is designing MES functions to work with the OKD-MES. This thesis spots the design phase of MES functions in OKD system. As chapter four shows, the manufacturing system which is represents the target of eScop project is presented with order-product scenario.

1.2 Motivation and objectives

As this thesis shows in chapter two the benefits of using OKD-MES in manufacturing system, the questions is how it would be beneficial to use MES functions as web services in OKD system in order to enhance the manufacturing information system in terms of flexibility, re-configurability, cost, easiness of installation and versatility?

Therefore, the objective of this research work can be addressed as presenting a methodology of designing general web-based framework for MES functions which could be applied on different industry types.

1.3 Problems definition

As mentioned in section 1.2, this study focuses on including MES functions in OKD-MES as web services. This involvement of MES functions in OKD-MES consists of architecture design, implementation and testing the framework with a provided case study as proof of concept. Since this is the objective of this study, this section lists the hypothesis of this thesis as follows:

1. By using web services, the interactions between manufacturing components and the MES functions is expected to be rapid and independent.
2. With enabling the knowledge-driven for configurations, it will be possible to simplify user interactions.
3. Using the concept of employing the logic of MES functions as scripts could provide more flexibility in term of the user choice.

1.4 Limitations and restrictions

Manufacturing systems are categorized as very large topic. Thousands of studies have been conducted to improve and develop the manufacturing systems. This thesis focuses on development of MES functions to be used as web services in an OKD-MES. The previous section showed how MES function supposed to behave. However, reasonable limitations should be taken into account. These limitations depend mainly on the availability of tools to achieve the required target. One of these limitations could be lack of real industrial RTU (Remote Terminal Unit) that supports different web services standards. Also as new topic, OKD system is still under development by many parties and it is not standardized yet. Another limitation could be seen as automation amount in the factory level. As known, some industries still rely on the manual labors. Meanwhile, some other industries are fully automated. These variations of the automation level in the manufacturing system could generate more complexity in this research work.

2 THEORITICAL BACKGROUND FOR BUILDING OKD-MES AND MES FUNCTIONS

Nowadays, manufacturing systems tend to involve all available technologies. The aim of this involvement appears in various examples of technologies usage in industry such as internet and computers, artificial intelligence, cloud processing, etc. one of these technologies is known as KD (knowledge-driven) system. KD comprises the core technology of this thesis. Besides KD, this thesis involves web services technology as well. This chapter discusses the state of art of some technologies which are related to this thesis. It focuses on the vision of using Open Knowledge-driven approach in manufacturing execution system. As presented in the introduction, this study may provide a flexible solution for manufacturing systems.

This chapter contains six sections; first section illustrates the definition of MES layer and MES functions. The second section provides the definition for OKD system in industry. This section shows a comparison between OKD-MES and ISA-95 standard manufacturing system as well. The third section provides a view for implementation of MES functions in OKD system. This section demotes how MES functions can be transformed from general definition as provided by MESA to special definition which suitable for OKD-MES. In forth section, an overview covers web services usage in industry. The fifth section summarizes several APIs (Application Program Interface) which can be used in development phase of MES functions in OKD-MES approach. The final section highlights the relation between the stat of the art and the objective of this thesis.

2.1 Manufacturing Executing systems and MES functions

In late 70’s of last century, MES began to appear in manufacturing systems. At that time, MES has been involved to solve minor problems in manufacturing system. MES was not fully apart from DCS (Distributed Control System) as well. On the other hand, enterprise planning (ERP) layer (material resources planning (MRP) before) and factory floor have been mostly isolated. As described in [1], MES began to be involved to close the gap between both layers. Later on, MES start being modularized since computer science revolution rose in manufacturing systems. Then in the early 90’s in United States, a company called Advanced Manufacturing Research (AMR) proposed the enterprise model (ERP – MES - DCS) [1], [2]. During this period, MES gained more functionality in manufacturing systems. This model led to standardize the functionalities of MES in 1995 by Manufacturing Execution System Association (MESA). Regarding this standardization, 11 MES functions have been defined and merged with ISA-95 standard [3], [7].

ISA-95 standard defines the structure of factory information system into fine layers (see Figure 1)[5]. For ISA-95 standard, level 0, 1 and 2 represents the physical world of the factory like machine, equipment, materials, products, controllers and factory labors. This layer needs to be organized and controlled according to the orders that are issued by the managements. Level 4 contains the management part of the factory. This layer is called ERP and it is responsible to accept orders/ provide products from/to customers. Also ERP interacts with suppliers for providing the required materials. Besides that, ERP includes the human resource and the financial departments. ERP is located on the top of manufacturing systems pyramid which shows that ERP as highest level in the factory structure.

Level 3 is the binding layer between ERP (level 4) and DCS (level 0, 1 and 2). This layer is defined as Manufacturing Execution Systems (MES). MES accepts orders from ERP which are representing the customer needs. Then, MES transforms these orders to schedules in shop floor. On the other hand, MES receives the requirements from shop floor and transforms these requirements to offers in.

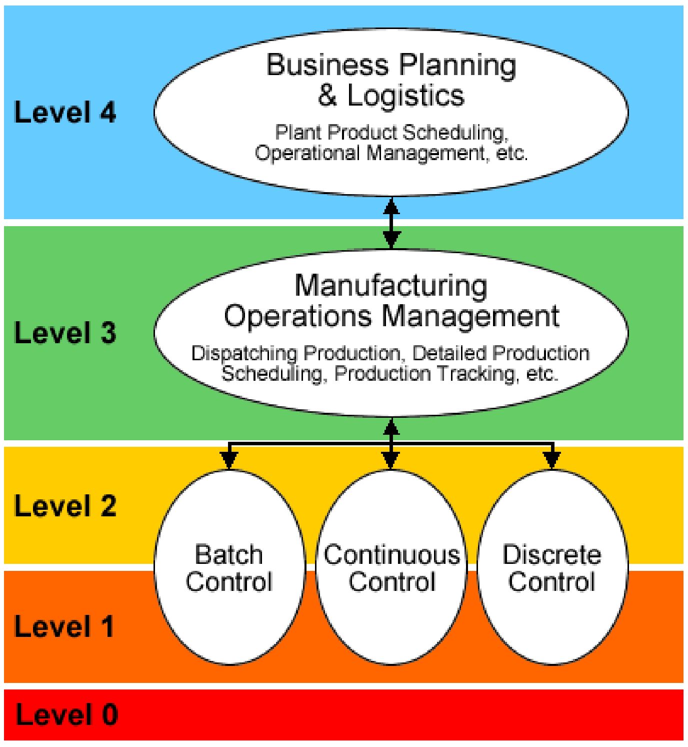

Level 4
Business Planning & Logistics
Plant Product Scheduling, Operational Management, etc.
Level 3
Manufacturing Operations Management
Dispatching Production, Detailed Production Scheduling, Production Tracking, etc.
Level 2
Batch Control
Continuous Control
Discrete Control
Level 1
Level 0

Figure 1: ISA-95 structure by [6]

According to the description of ISA-95, MES has to get standard form in order to achieve the required collaboration. Therefore, MESA is derived to introduce 11 MES functions. Table 1 shows MES functions as defined by MESA:

Table 1: MES functions as defined by MESA [7]

1	**Resource Allocation and Status**	Manages resources including machines, tools labor 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.
2	**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 color 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.
3	**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.
4	**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.
5	**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.
6	**Labor 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.
7	**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.
8	**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.
9	**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.
10	**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.
11	**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.

Regarding to Table 1, MESA aims to define MES functions in general form. The definition may vary according to the involvement in the industry. Nowadays, MES functions are considered as spine of the manufacturing system. These functions provide the needed connection between ERP and factory shop floor.

The generality of MES functions definition increased the adaptation of MES with all different kinds of industries (batch, process and discrete). In this thesis, the focus is directed towards discrete industry. Discrete industry is defined in terms of number of parts of the product and complexity of assembling products [8]. In this manner, many of industrial technology providers build solutions to fit the industry demand for different industry type. The list below shows some of industrial technology vendors with their solutions for discrete manufacturing execution system.

- SIMATIC IT by Siemens [8]
- Discrete manufacturing operations management software by ABB [9]

- E-Business Suite by Oracle [10]
- Proficy for Discrete Manufacturing by General Electric [11]
- Industry solution package by SAP [12]
- Rockwell Automation [13]

All the listed solutions provide the manufacturing system with needed interaction through standalone applications.

2.2 Open-Knowledge-Driven System

Many philosophers tried to define general knowledge. Nevertheless, the majority agreed on the theory of knowledge as “Justified True Knowledge” as Grayling stated in [14]. From technical point of view, knowledge is defined as the ability for providing information [15]. In this context, knowledge represents a set of comprehensive information composed by truth and experience. In industry, knowledge driven system is defined as smart system by some researcher [16], [17]. OKD defines a system as knowledge or semantic information form. This information (knowledge) is defined as set of ontologies. Meanwhile, ontologies have been considered as flexible, extensible and scalable description mechanism [18][19]. In early 90’s, ontology has been formalized by T. Gruber [20]. Ontologies could be defined in many languages in computer science. One of these languages is OWL (Web Ontology Language) by W3C organization [21].

The involvement of knowledge-based concept in several fields has increased nowadays because of internet revolution in the beginning of 21th century. Factory automation is one of the fields which involve the most recent technologies. In this manner OKD-MES (Open Knowledge-Driven Manufacturing Execution System) trends to add more flexibility and configurability to manufacturing execution systems [22].

As Iarovyi and Xu describe in [23], the openness term represents the possibility to be distributed and replaceable in terms of architecture of information. They provide an inclusive illustration for building OKD-MES in [23]. In their article, OKD-MES core is composed by five layers; Physical Layer (PHL), Representation Layer (RPL), Orchestration Layer (ORL), Visualization Layer (VIS) and Interface Layer (INT). See Figure 2.

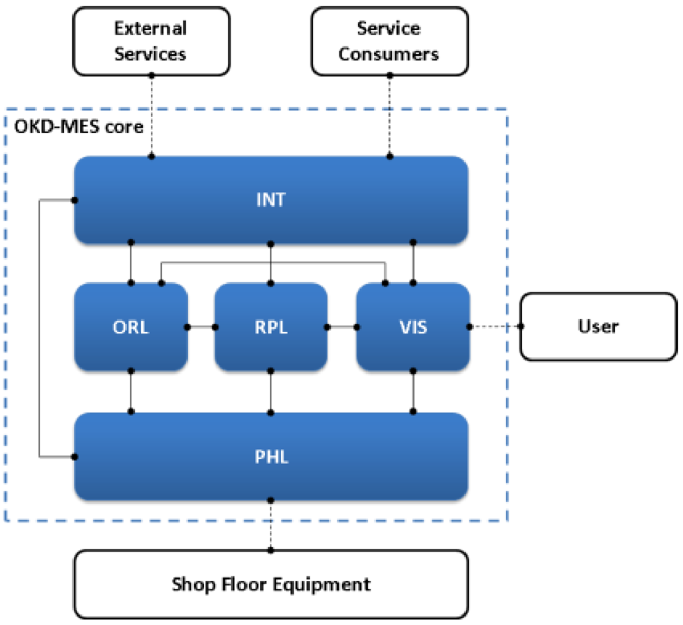


Figure 2: OKD-MES architecture [23]

PHL represents the controlling unit for shopfloor in a factory. As shown in Figure 3, the structure of PHL consists of three main components; WS toolkit, Runtime core and I/O module. Since OKD-MES adapts web services, PHL interacts with other layers via WS toolkit (Web Services toolkit). Runtime core provides logic processes such as structured list or ladder diagram representations. Meanwhile, I/O Module communicates with shopfloor equipment.

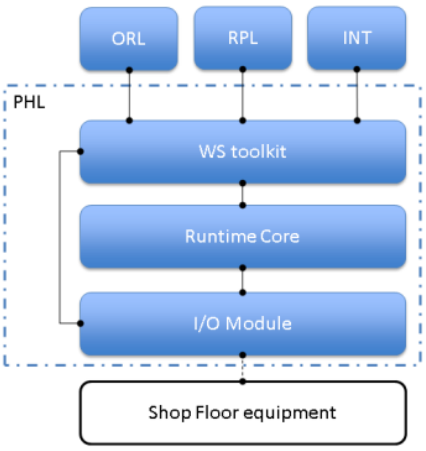


Figure 3: PHL structure [23]

The next layer of OKD-MES core states RPL (Representation Layer). This layer is considered as knowledge provider. As appears in Figure 4, RPL comprises of six components; Device Registration Module, Visualization Provider Module, Ontology Manager Module, SPARQL (see section 2.5) Query Factory, Ontology connector and Ontology Database.

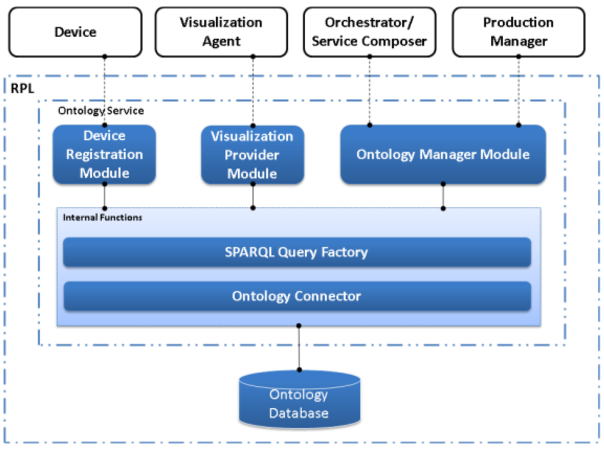


Figure 4: RPL architecture [23]

Device Registration provides information about OKD-MES accessibility. Furthermore, this component is called Services Inventory as well. Visualization Provider supplies VIS with visualization description according to the configuration in the ontology. Meanwhile, Ontology Manager Module manages information flow in RPL. This component communicates with ORL and Production manager (MES functions) via web services. SPARQL Query Factory and Ontology Connector bridges the previous components with Ontology Database. Finally, Ontology Database holds information as ontologies. This component serves query/update request for information manipulation.

Another component in author’s approach is called ORL. ORL manages the execution of process in OKD-MES especially processes and services in PHL. This layer is designed to achieve the generalization concept of OKD-MES. Consequently, ORL is connected with RPL in order to retrieve the information about PHL services and status of the system. According to Figure 5, this layer contains two modules; Services Composer and Orchestration Engine.

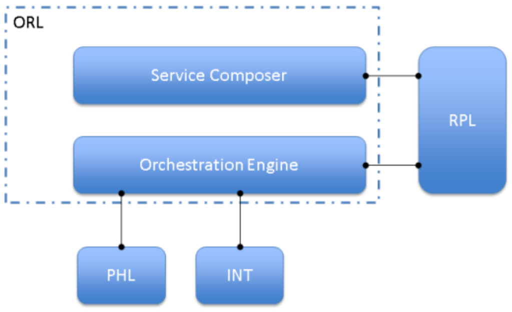


Figure 5: ORL architecture [23]

Service Composer generates lists of tasks which required to be applied on PHL. These tasks depend on the situation of PHL and production status (MES functions). On the other hand, Orchestration Engine manages the execution of these lists on PHL. As well, this module updates RPL with information about processes execution.

Finally, the layer which supports OKD-MES with visualization components is so called VIS. This layer allows the user to interact with the manufacturing system via visual interface browser. As the specified approach in [23], the browser states internet web browser. Accordingly, this layer requires information about requests of users from RPL. Then this Dynamic Composer collects visualization components from Symbol library as SVG elements. These elements are shown in web browser as html page. Similarly, VIS monitors PHL through Visualization Agent. This agent subscribes to PHL events that are connected to visualization functionality such as status of machine events. Then due to PHL events, visualization agent updates the correspondent elements in the html page. See Figure 6.

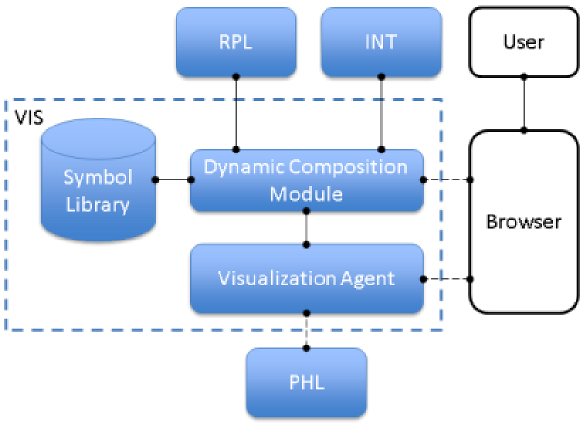


Figure 6: VIS architecture [23]

2.3 OKD systems and MES

The usage of KB (Knowledge-Based) approach in manufacturing system is still under development. As LONG Wen illustrates in [24], the model of ontology for MES contains Process model, Product model, Organization model, Resource model, Information model and Function model. His approach provides a good view about the connections between classes and initial view about how MES ontology will look like. But MES functions did not have a unique entity as model in the ontology. Figure 7 shows the ontology model which is presented by LONG Wen.

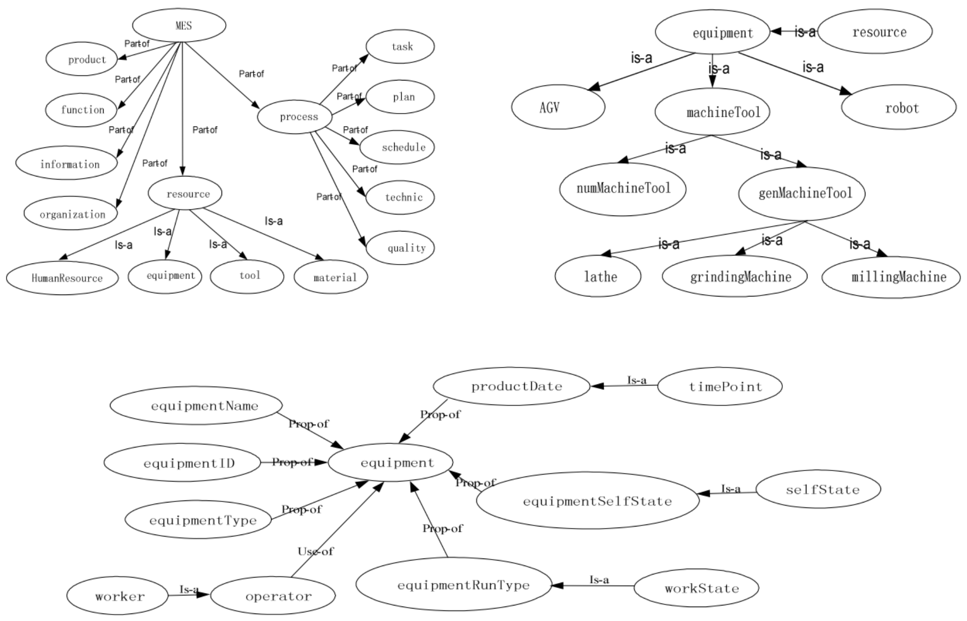


Figure 7: Ontology model [24]

In LONG Wen approach, some of MES functions are inseparable. This means if the user would like to have his/her own function, s/he has to change all the ontology structure. Therefore, the flexibility of editing ontologies is restricted. Furthermore, the user is limited to choose the solution that s/he prefers. In this thesis, one of the targets planned as building separate MES functions ontologies which allow the user to choose the function that s/he would like to use regardless the core configuration of OKD-MES.

2.4 Web services in Manufacturing Systems

Since the revolution of networks and internet has been started in 90's, many of industrial controllers' providers have implemented web services in their products. For this reason, providing SOA (service oriented architecture) by W3C [25] is considered as a necessary requirement for industrial controllers. The main benefit of SOA is to provide a structure of communication between different agent regardless of the differences in the platform or the operating system[25], [26], [27].

As a conclusion in [28], SOA could provide automation with more flexibility and configurability which is required in automation systems. One of SOA deployment in manufacturing systems is DPWS (Device Profile Web Services). DPWS is defined as an implementation of web services that provide security, discoverability and accessibility for WS description. The core of DPWS is based on WSDL, XML schema, SOAP and WS Addressing[29], [30]. As an implementation example in [31], Figure8 shows an implementation for the DPWS with Knowledge-based system for building a Semantic Web Services Framework

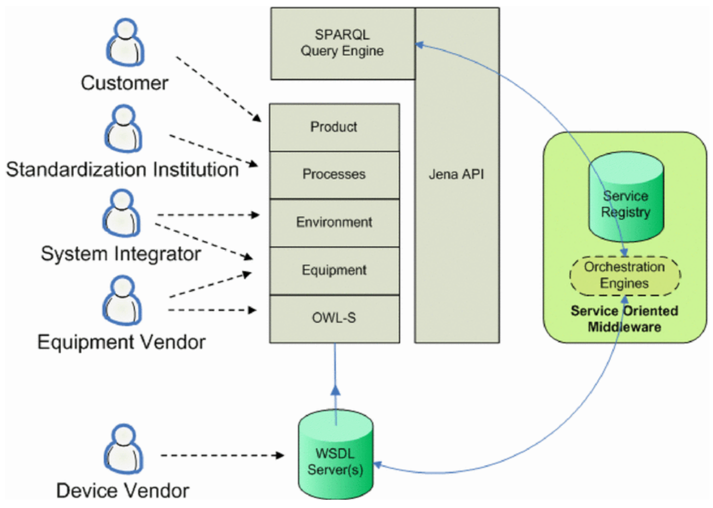


Figure 8:Semantic Web Services Framework using DPWS [31]

Another WS definition provided by W3C is REST architecture (Representation State Transfer). REST is based on HTTP method like GET, POST, PUT, DELTE, etc. The real meaning of HTTP methods is combined in CRUD (Create, Read, Update and Delete) term [32]. The remarkable difference among the other web services appears as representing web resources using a uniform set of "stateless" operations. Furthermore, REST architecture support various kind of data format like xml, html, plain text, JSON message formats. Although REST is not standardized yet, it features light weight processes and fault-tolerant[33] as well. Thus, REST web services are raising more in industrial applications. Furthermore, a vocabulary definition for Hypermedia-Driven Web APIs (HYDRA)[34] has been defined in order to provide a generic structure for web API clients.

2.5 APIs (Application Programming Interface)

Any research and development process needs tools for development. According to the objectives, this thesis trends to develop MES functions in OKD-MES as web services. Therefore, first tool which is necessary for the development is ontology editor. Ontology editors are deployed for structuring OWLs. There are many ontology editors are available such as Protégé, JOE (Java Ontology Editor), Olingvo, etc. Olingvo is a graphical ontology editor based on java API [35]. Moreover, Olingvo provides users with the ability to use SPARQL queries/update on the available ontology.

After building ontologies, a server has to be used to provide query and upgrade services to the ontology using SPARQL language. In this study Fuseki server has been deployed to fulfill this functionality. It also supports REST-SPARQL HTTP format for query and upgrade services [36]. Because of that, Fuseki is considered as suitable tools for serving ontology via HTTP protocol.

As illustrated in the introduction, MES functions have to host certain web services. The choice from variety available APIs depends on the functionality and complexity of the API. Java is one strong candidate which allows the programmer to work in object oriented environment. Also Java can be used on any platform which makes it highly considered for this job [37]. Similarly, C++ and C# are strong APIs as well for building automation applications. But because of the nature of MES functions, C++ and C# could be not the suitable choice since MES functions run as server which may be applied on different platforms. In this matter, java is considered as most likely to be implemented. Even so, after research and conducting some test, NodeJS appeared to be reasonably choice. NodeJS is JavaScript API environment built using chrome v8 engine. The structure of NodeJS is defined as event-driven and non-blocking I/O API. Accordingly, NodeJS is lighter and more efficient API for building web applications. Besides that, NodeJS employs npm open source library. The npm is a package manager for NodeJS. As a fact, the npm is considered as the largest open source library in the world [38]. Consequently, NodeJS provide the capability to the user to include many open source modules through npm.

2.6 Summary

As the objective of this thesis to present a methodology for implementing the MES Functions as web services, this chapter focused on the related fields such as MES and MES functions, knowledge-based systems, the concept of OKD-MES, the cutting edge of the web services and the available tools that can be utilized for building the framework. As this thesis focuses on the implementation of the MES functions as web services, the approach which is illustrated in the coming chapters combines the main features of the listed fields in this chapter.

As the problem definition presented the hypothesis of this thesis, the lack of having MES solution which supports the manufacturing systems with MES functionalities through web services. Thus, this chapter intended to highlight the manufacturing execution systems definition, the open-knowledge driven approach for MES, the web services in the industrial use and the current used APIs for building applications. In the next chapter, the reader will be presented with a methodology of building MES functions platform for web-based and knowledge-based manufacturing execution system.

3 THE METHODOLOGY OF DESINING THE MES FUCNTIONS

The vision of developing MES functions is to present configurable and replaceable tools which allow the user to edit or to replace the implementation of MES functions. This chapter presents the methodology of developing MES functions to work side by side with OKD-MES. This methodology illustrates a general and robust model for MES functions. Hence this model could be used for all MES functions.

This chapter contains four sections. First section illustrates the model of MES functions that can be used for all MES functions regardless industry type. While the second section shows the tools that can be used for developing MES function. And the third section presents the technique of developing MES functions. Finally, the fourth section presents the validation process for the provided model.

3.1 The Approach

Since MES functions are going to be implemented in OKD systems, the target is to present MES functions as layer of OKD-MES. This layer is considered as an optional layer. In other words, it can be used partially or completely according to the user needs. Figure 9 shows the general interaction between OKD-MES layers and MES functions layer.

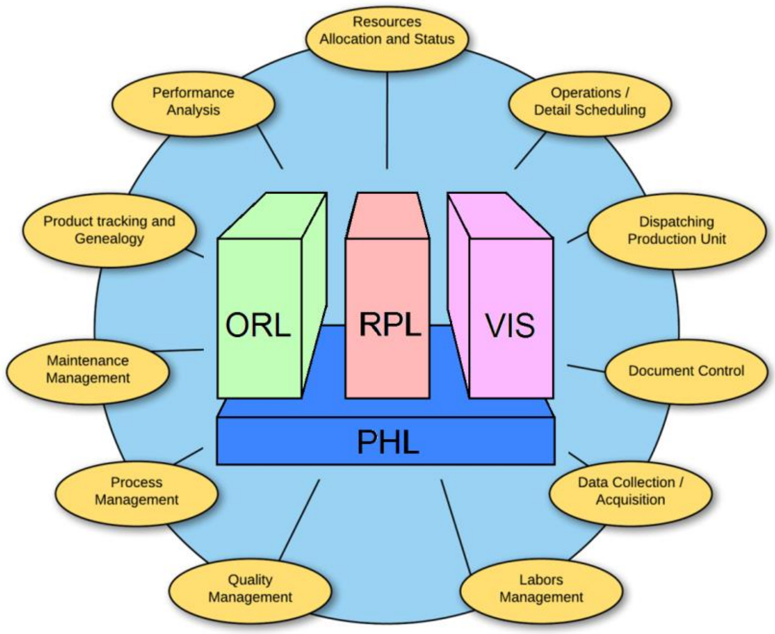

Resources Allocation and Status
Performance Analysis
Operations / Detail Scheduling
Product tracking and Genealogy
Dispatching Production Unit
ORL
RPL
VIS
Maintenance Management
Document Control
PHL
Process Management
Data Collection / Acquisition
Quality Management
Labors Management

Figure 9: MES functions in OKD-MES

As presented in the objectives of this thesis, MES functions have to be generalized despite of the differences in the functionality. Building MES functions platform using KBS and web services is one approach may achieve this requirement. Therefore, the model of MES functions may contain configurable parts (parameters, functions logic and configurations) which represent the differences between these functions. All other parts of the model will be the same for all MES functions. The provided model contains three main separate components. First component is Ontology Manager (OM) which is designed to manage the query and upgrade ontology services. The second component is the Service Manager (SM). This part is fixed for all functions and provides RESTful services to communicate with OKD-MES layers and/or MES functions. The third part is Function Manager (FM) which is responsible to provide the functionality of MES functions by providing logic aspect for MES functions. Figure 10 shows MES function model.

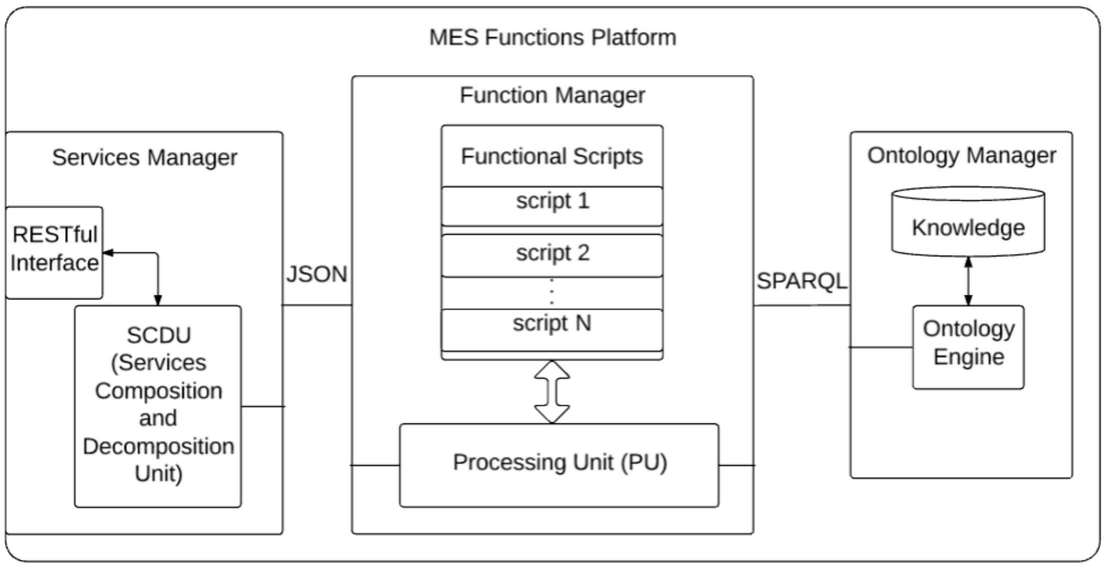


Figure 10: MES Function Model

The user of the MES functions platform is expected to study the manufacturing which the platform will be implemented in. Figure 11 depicts the user activity diagram. Once an appropriate case where the platform can be used, the user starts populating the ontology model with the existent knowledge such as MEs Function will be used, the logic of these MES functions, other components in the manufacturing system such as the factory shopfloor. Next, the user is subjected to prepare the functional scripts which includes the logic for the MES functions. Finally, the user can deploy the platform in the manufacturing system.

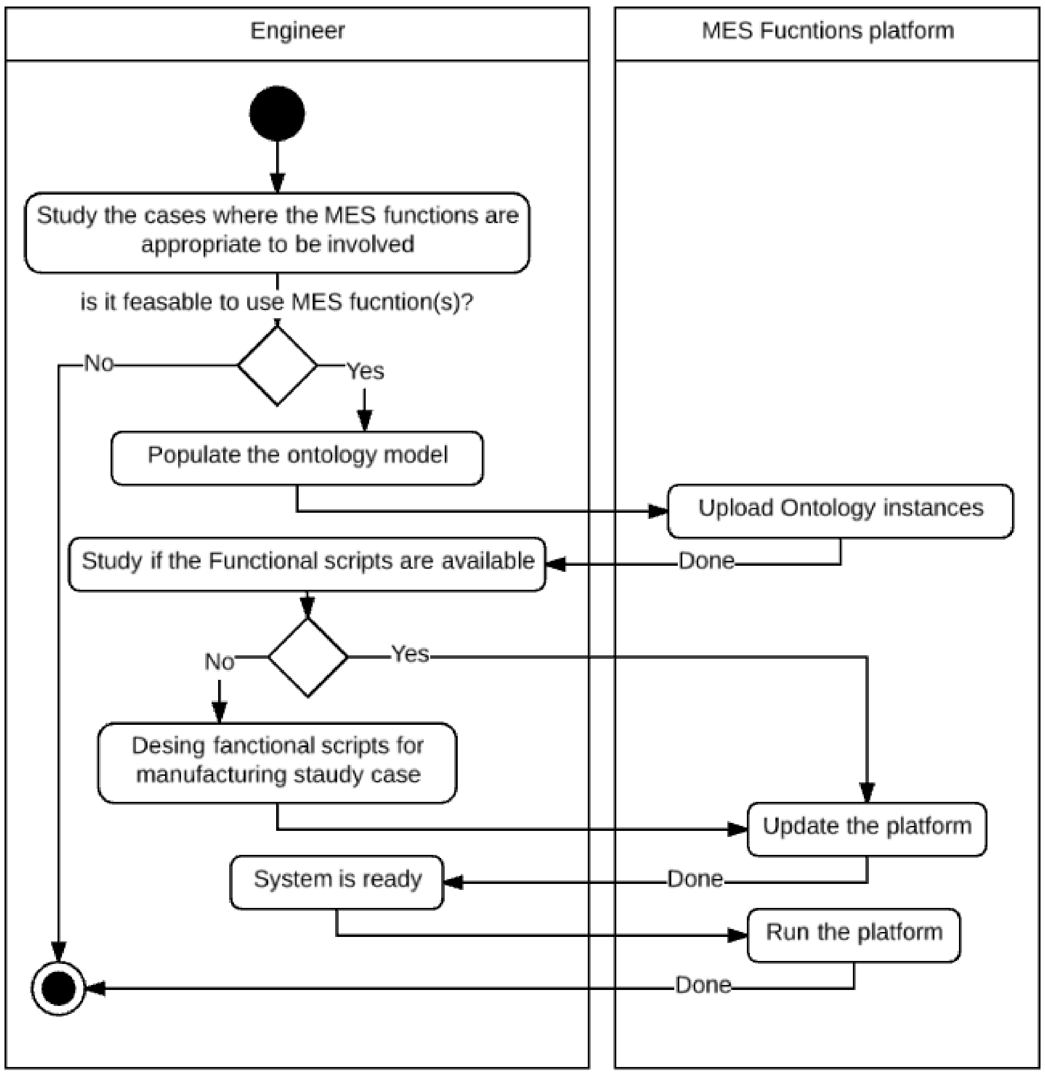

Engineer
MES Fucntions platform
Study the cases where the MES functions are appropriate to be involved
is it feasable to use MES fucntion(s)?
No
Yes
Populate the ontology model
Upload Ontology instances
Study if the Functional scripts are available
Done
No
Yes
Desing fanctional scripts for manufacturing staudy case
Update the platform
System is ready
Done
Run the platform
Done

Figure 11: User activity diagram

3.1.1 Ontology Manager and Ontology Model

MES functions are the core of the manufacturing facility. However, not all MES functions are necessary for all industries. Thus the implementation of MES functions depends on several conditions like industry type, level of automation in the factory and/or the owner needs. The user may choose the tool that s/he prefers. Therefore, these functions must have separate ontology than OKD-MES ontology in order to be easily edited.

Ontologies structure consists four parts; concept (class), attributers, relations and instances. In this chapter the discussion focuses on classes, attributes and relations. Meanwhile, instances are illustrated in the fourth chapter were the implementation takes place. Classes describe the object with one or many vocabularies It represents the meaning of a concept or sub-concept in the domain. An attribute provides information about the class like ID. Or it can be used to define constants or variable for the class. Relations define the connection between different classes. Ontology manager is the component where the user populates the ontology model according to the used case.

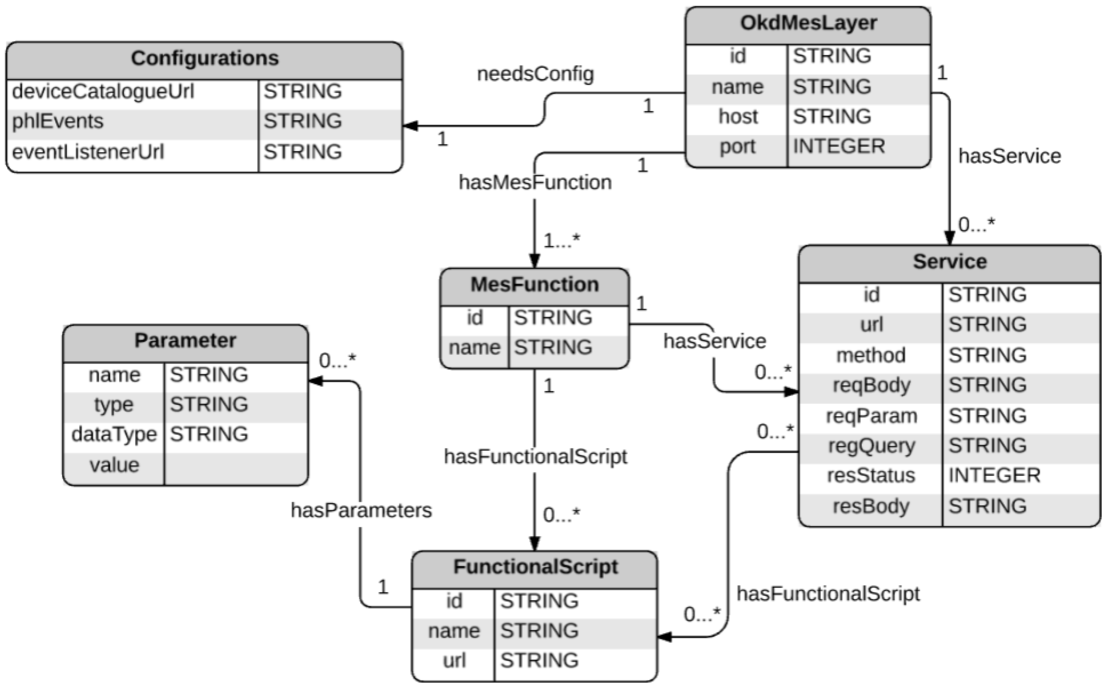


Figure 12: MESF model

As depicted in Figure 12, the model contains six classes; *OkdMesLayer* which represents the MES functions as layer in the OKD-MES. It includes *id*, *name*, *host* and *port* datatypes. Besides, this class holds a one instance in the ontology once it populated by the user. The *OkdMesLayer* is linked to *Configurations*, *Service* and *MesFucntion* classes via *needsConfig*, *hasService* and *hasMesFunction* dataproperties respectively. The *Configurations* class support the layer with the needed preferences. It includes *deviceCatalogueUrl* which is a component in the RPL, *phlEvents* which is comma separated string that contains the PHL event ids. And finally *eventListenerUrl* for configuring the endpoint where the platform will receive events from the PHL.

The Service class contains *id, url, method, reqBody, reqParam, reqQuery, resBody* and *resStatus* datatypes for service composition purposes. In this class, the user defines the services that the platform will serve. The *Service* is linked to *FunctionalScript* class using *hasFunctionalScript* datatype. As appeared in Figure 12, both the OkdMesLayer and the *MesFunction* could has service instance. This means that the platform may serve a service for all platform like i.e. changing the configurations of the platform or Service instance can be a part for the MES function itself. After that, the *MesFucntion* class includes *id* and *name* datatypes and it is linked to *Service* and *FuntionnalScript* classes by *hasService* and *hasFunctionalScript* dataproperties respectively. In this class, the user defines the MES functions that are going to be implemented in the used case. Then the *FucntionalScript* class includes *id*, *name* and *url* datatypes. This class can be part of the MES function or it can be part of a service. In this manner, the user is able to define functional scripts that are called once a service is received or it can be a script runs in the background of an MES function. The *FunctionalScript* class is linked with *Parameter* class using *hasParameter* data property. This parameter can be used as input, output or variable. Finally, the Parameter class includes *name*, *type*, *datatype* and *value* datatypes.

Ontology manager contains two components, information stack which is represented as MESF OWL file and ontology engine which is built by Fuseki server. The ontology manager is responsible to provide query and update services for local ontology. For the global ontology, RPL hosts predefined services. These services are known for MES functions by Service Inventory.

3.1.2 Service Manager

As presented in the introduction, MES functions works in web services domain since OKD-MES approach is targeted. In order to provide web services, MES platform contains the SM. The SM is responsible for interacting with OKD-MES components and MES functions. Besides that, it has to decompose the request for the invoked service and to compose the response for the service. In this context, services manipulation is done by SCDU (Service Composition and Decomposition Unit). SCDU involves HYDRA definition in the structure of the web APIs.

SCDU provides detailed information about invoked service to MES function platform. This information describes the request of the service. Accordingly, MES function model responds to the invoked service via SCDU. Correspondingly, SCDU composes the response in a suitable HTTP standard. On the other hand, SCDU may compos requests as well. This request is required by the platform.

3.1.3 Function Manager

Function Manager is the third component of MES functions model. This component is the core of the platform. It bonds the SM and the OM. It is also the part of the function where calculations take place.

Function Manager holds three parts; FLM (Functions Logic Module) and PU (Process Unit). Functions are predefined scripts by the user which contains logic of MES functions in JavaScript format. The reason behind using JavaScript language that the platform is developed using NodeJS. For more friendly representation, MES functions scripts can be built using STL in IEC-61131-3 standard for future updates. These scripts are different from MES function to another according to the usage of each one.

Secondly, the PU is the part where calculations are occurring. It connects service manager requests and responses with ontology information and calls the required functions according to the matching between ontologies with services. As shown in Figure 10, PU as the main part of the FM is communicating with service manager using JSON notation. Meanwhile, it uses SPARQL queries to communicate with the OM. Figure 13 presents the sequence diagram for the platform.

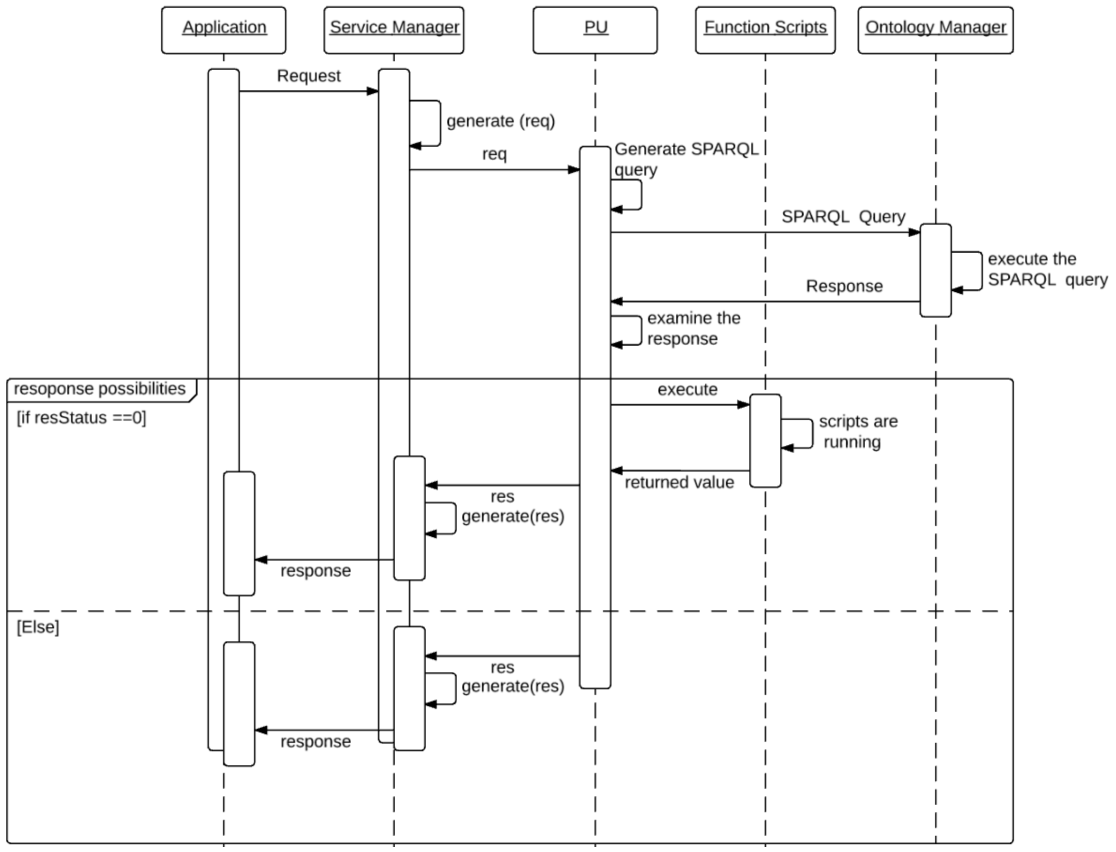


Figure 13: sequence diagram for the platform workflow

As Figure 12 shows, an application requests a service in the platform. Accordingly, the SM notifies the FM about the incoming request and includes the request information to the FM. Then, the FM requests the OM to validate the incoming request. As user populated the ontology model, the OM returns the validation result to the FM. In this manner, the FM could return a response directly to the client or a response contains the specified Functional Script which the service uses in the ontology model. More explanation is provided in the next section.

3.2 Techniques

As mentioned in section 3.1, the platform contains three main parts; the OM, the SM and the FM. This section presents the communication between these parts. It also provides in details how these main parts are collaborating with each other once a service invoked. As shown in Figure 14, the flow of service manipulation starts with validation of the invoked service. The validation process depends on the information in the ontology. If the service is not valid, then SCUD returns error code according to the validation procedure. If the service is valid, then the associated function scrip is called and executed. The response of the invoked service depends on the service. It is also possible that MES function invoke other services in OKD-MES components or in MES functions as well before or after responding for the invoked service.

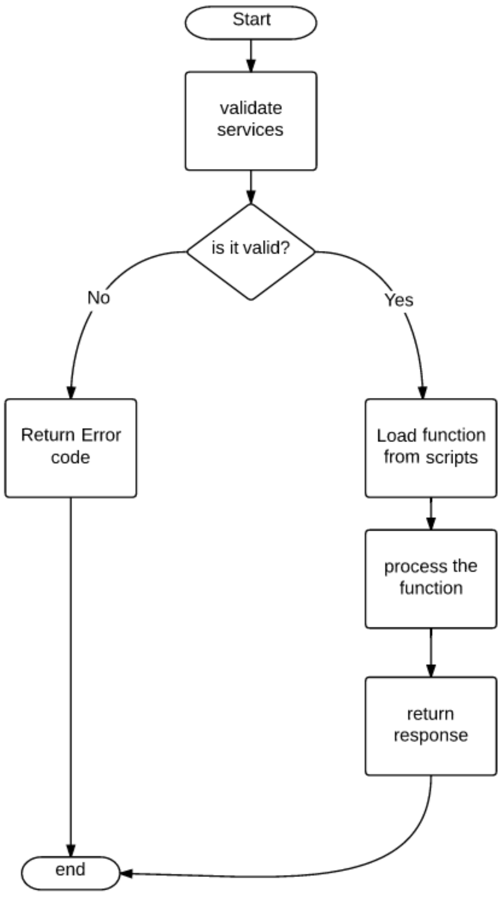


Figure 14: General flowchart for the work sequence of MES functions

This section is divided into four subsections. First subsection shows the interaction between the SM and the FM. The second subsection illustrates the collaboration between the OM and the FM. The third subsection discusses the deployments of script functions and how they are linked with invoked service and information in ontology. Finally, a validation scenario is described in order to build an examination bases for the platform.

3.2.1 Web Service Manipulations

This sub section describes the interaction between components of MES functions model in terms of web services. As mentioned in 3.1.2, the platform uses RESTful web services for communication. The process starts when a client request service from an MES function. The request reaches SCDU via REST interface. SCDU extract all information from the request and send it as JSON object to PU. This object contains information about requested services.

Afterwards, the PU starts validating the service. This validating process queries information from the OM and evaluates the service. The query request is shown in Code 1. The query selects all MES functions, services and functional scripts where the provided requested url and the http method have a match in the knowledge model.

```
PREFIX ms:<http://www.escop-project.eu/#>
SELECT ?OKD_MES ?MES_Function ?Service ?Function
WHERE{
?OKD_MES ms:URL_Value "http://127.0.0.1:3100".
?MES_Function ms:URL_Value "/tracking".
?Service ms:URL_Value "/Products".
?Service ms:HasFunction ?Function.
?Service ms:IsGetHTTPMethod "Get".
}
```

Code 1: SPARQL script for manipulating the invoked services

According to the queried information, the FM proceed in the process of serving the request. Then PU sends JSON Object for the response of the invoked service. Once SCDU receives the response object, it transfers JSON object services which has header and body. Finally, SCDU sends it back to the client. Another usage of the SM is to invoke services in global domain like OKD-MES layers. This service is described in component as so called Services Inventory. Service inventory is component in RPL which hold all information about web services for OKD-MES layers.

3.2.2 Ontology Manipulation

Since MES function server is build using NodeJS, it has the ability to include modules or packages to communicate with Fuseki server. The Fuseki server is used to serve ontologies through http protocol. But this server communicates with clients using SPARQL language. Thus a module has been included in NodeJS to fulfill this gap. The model is open source and downloadable via NPM library. The module is called “sparql-client” [39]. This module is able to return information in JSON format. This feature easies the interaction with ontology manager.

Any JavaScript function which uses sparql-client module has to include the module script in Code 2.

```
var SparqlClient = require('sparql-client');
var util = require('util');
var endpoint = 'http://dbpedia.org/sparql';
```

Code 2: Importing the SPARQL client in NodeJS

Then the query is defined as a string variable in JavaScript. After wards, a client is defined as a new instance of SparqlClient. Finally, the client uses the method “query” to invoke the query in the Fuseki server. See Code 3.

```
var query = " PREFIX ms:<http://www.escop-project.eu/#>
SELECT ?OKD_MES ?MES_Function ?Service ?Function
WHERE{
      ?OKD_MES ms:URL_Value "http://127.0.0.1:3100".
      ?MES_Function ms:URL_Value "/tracking".
      ?Service ms:URL_Value "/Products".
      ?Service ms:HasFunction ?Function.
      ?Service ms:IsGetHTTPMethod "Get".
}";
var client = new SparqlClient(endpoint);
client.query(query);
```

Code 3: Example of using the SPARQL client in the NodeJS

This JavaScript code returns the information as JSON object which is described in sub-section 3.2.3.

3.2.3 The Functional Scripts

As the platform has been illustrated so far, the user has to populate the MESF. Another component that needs the user to define is the functional scripts. These scripts contain the logic for each MES function. But there is one function used by all MES functions. This specific function is required for model use. It requires nothing from the user to modify. This function is web services validation functions. It evaluates the request of services and responds according to the information stored in the ontologies.

Once the validation function is called, it queries the information of the invoked service as shown in previous subsection. Then validation function starts validation by evaluating the result of the query. If the query result is not empty, then the service is correct and validation return with the id of associated function for the service. If the result of the query is empty, then validation function removes body and header query triples and invoke query service again. If the new result is not empty, then validation returns error code 400 (Bad request error [40]). But, if the result is empty again, then validation removes httpMethod triple and request query services again. If the result is empty then validation returns error code 404 (Not found error [40]). If the result is not empty, then validation function returns error code 501 (Not implemented error [40]). Figure 15 shows the flowchart of the validation function.

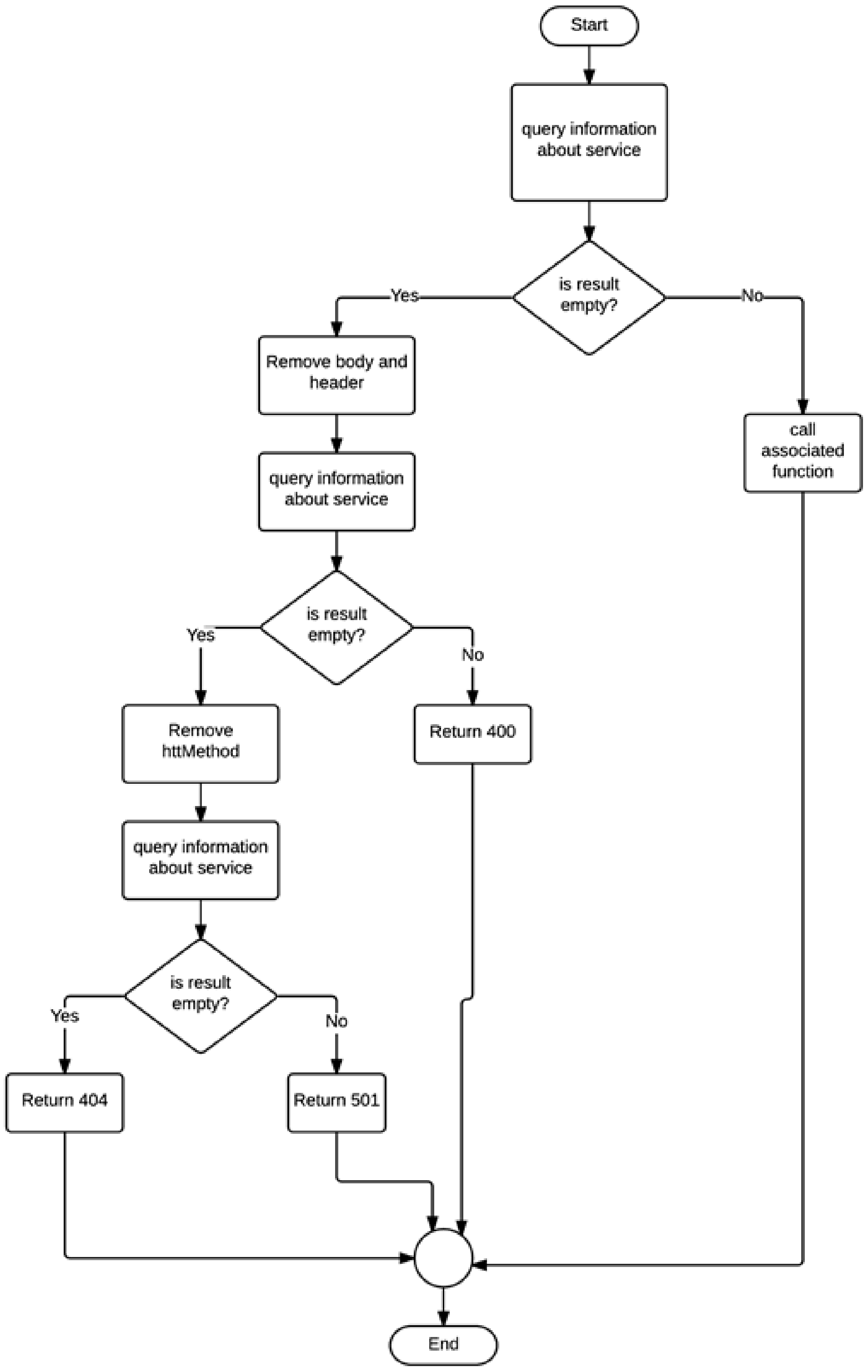


Figure 15: Validation function flowchart

The other associated functional scripts are defined by the user. These scripts are called according to the use as described in the ontology. The concept of calling JavaScript inside JavaScript is achievable by different methods. After doing some tests and research, two main approaches are feasible to be used. First method is by using `eval()` command in JavaScript library [41]. This method executes JavaScript from string datatype. There are two approaches to implement it; first one is by saving all function scripts as text files and read these functions and converts them to string then call it. This approach requires more time for reading the file. The second approach is by providing all function scripts as GET RESTful web services. Then the function executes the response of the services directly. Using `eval()` command has two main drawbacks; firstly, it is hard to pass parameters to the function or to return results from the function. Secondly it is dangerous command because it will be executed in unknown location in the code. Therefore, synchronizing the function with other functions or services is not available.

The second method is by using JavaScript files saved in the server. This method is safer and faster than first method. It works by require command in NodeJS then deal with the function as object. This method easily passes parameters and results and synchronisable. The following code shows how it is implemented:

External JavaScript object (file name: Mul.js):

```
exports.functions={
      mul:function (x,y){
      var z = x*y;
      console.log("The result is : "+z)
      return z
      }
}
```

Code 4: Example of using exports in NodeJS

Main function:

```
var test_function = "Mul"
var mul_function = require('./'+ test_function +'.js').functions;
mul_function.mul(155,2);    // call the external function
```

Code 5: Example of importing a library in NodeJS

3.2.4 Validation Scenario

For validation purposes, a scenario has been included for this thesis. The scenario addresses a production process that can be seen in any factory. As shown in Figure 16, this scenario involves four main MES functions; MES-F1: Resources Allocation/ Status, MES-F2: Operation / Detail Scheduling, MES-F3: Production Unit Dispatching and MES-F10: Products Tracking and Genealogy. In this scenario, three functions will use the provided model. The implementation includes ontology and functions scripts usage.

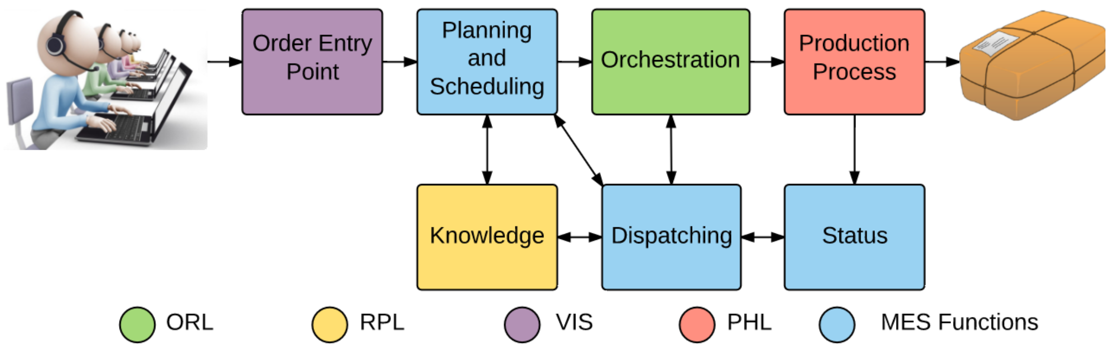


Figure 16: Order to Product scenario

The scenario starts by inserting an order to the OKD-MES through the VIS. The order goes to the MES-F2 which updates the RPL. Then the MES-F2 requires all the orders which needs to be served. Then the MES-F2 requires the ORL to start the production. After that, the ORL proceeds in the production process with the help of the MES-F3. Consequently, the MES-F3 needs both the RPL and the MES-F1 to direct the ORL for the needed production tasks. At the end of the chain, the PHL executes the tasks which are coming from the ORL. At the same time, it updates the MES-F1 with all the changes that happens in the factory equipment.

4 IMPLEMENTATION OF THE CONCEPT

This chapter presents the implementation of the approach of development MES functions which described in chapter three. The implementation focusses on deploying the approach on a testbed industry. Also, implementation includes providing a test scenario for the model. Thus, this chapter is divided into four sections. First section presents the testbed for the developed approach. This test bed is FASTory assembly line. The purpose of using the FASTory is to apply the concept on discrete industrial assembly line. The second section illustrates the readymade OKD-MES by eScop developers. Next section shows the implementation of MES functions model on few functions in MES functions. In this section, the methodology which discussed in chapter three will be applied. The last section shows the test scenarios for the concept which has been deployed for validation of the concept.

4.1 FASTory case-study

FASTory is discrete assembly line. It is used to mock the assembly of mobile phone. The line draws three mobile phone parts (frame, screen and keyboard) with three different colors. Also the line can produce 3 different models of mobile phones. See Figure 17.

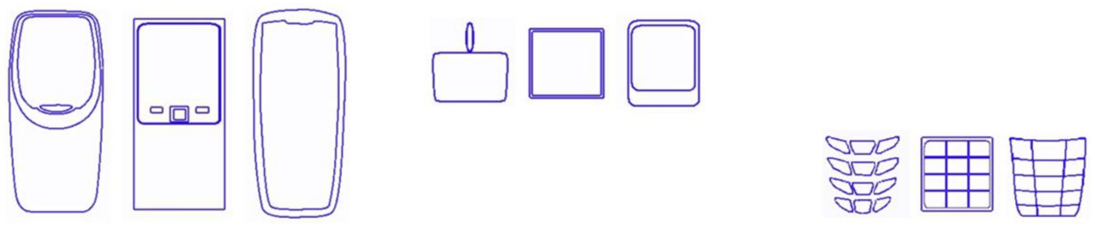


Figure 17: Mobile phone parts[1]

FASTory line contains 12 work station forming loop typology as shown in Figure 18. The line consists of one work stations (WS1) for loading row material and unloading accomplished products, one work stations (WS7) for loading/ unloading pallets to the line. This work station works as buffer for the line. It can hold up to 18 pallets. The remaining ten workstations are identical work stations (2, 3, 4, 5, 6, 8, 9, 10, 11 and 12) which are used for production purposes. Each of these work stations can produce any model with any color of mobile phone.

[1] http://escop.rd.tut.fi:3000/instructions

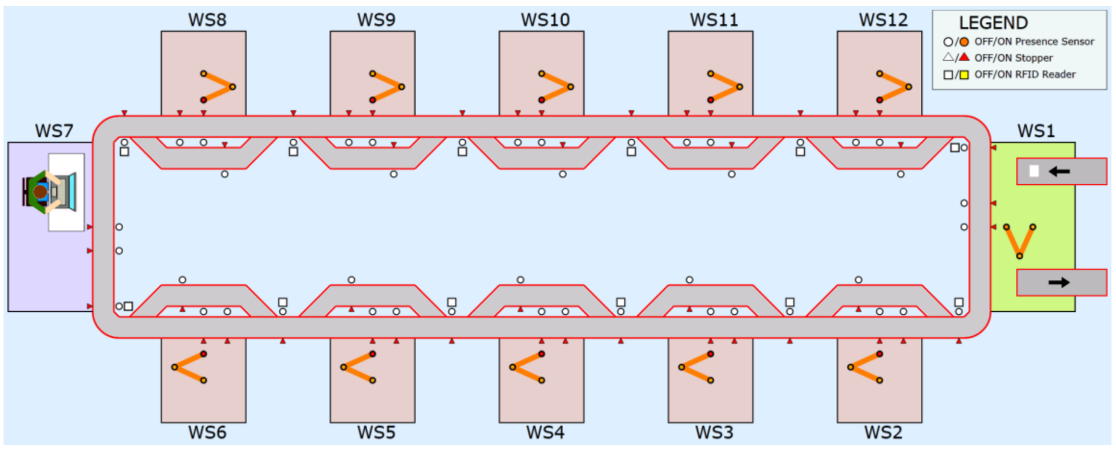


Figure 18: FASTory line typology

Each of 12 work stations contains one conveyor and one robot. For WS 1 and WS7, the conveyor has only main conveyor, meanwhile all other work stations have main conveyor and bypass conveyor. Main conveyor is used for producing process. The bypass conveyor is used when a pallet id required to reach next work stations. In this typology, the assembly line will not be on hold during production process. Also each work station has SONY SCARA robot. In WS1, the robot is responsible for loading and unloading papers to the line. For WS7, robot loads and unloads pallets to the line. On the other hand, for the rest of work stations, Robot draws mobile phone parts.

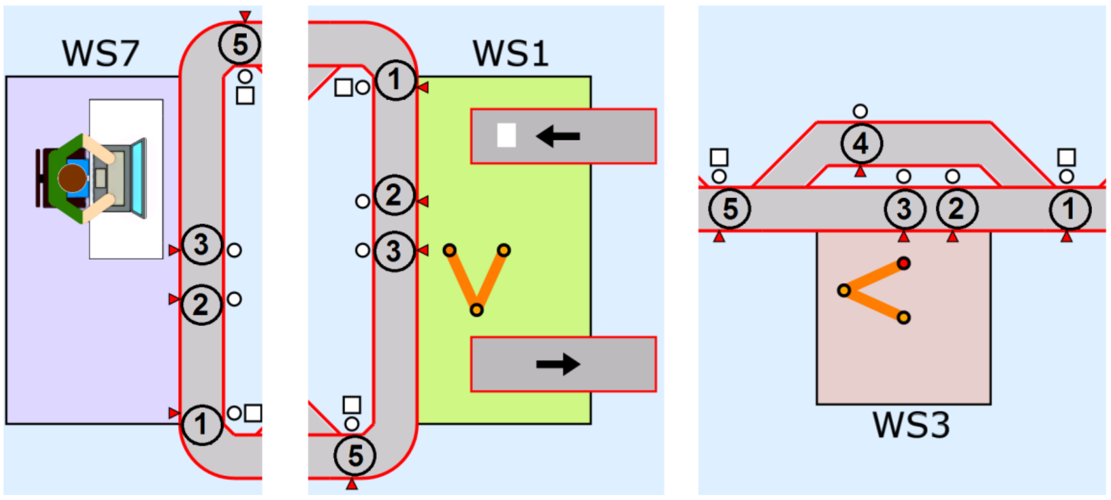


Figure 19: Conveyor zone types in FASTory line

As shown in Figure 19, each conveyor in FASTory line is divided into different zones. There are five different types of zones. In each zone there is one presence sensor that is used for detecting the pallet presence. Also, there is one stopper to stop the pallet when the conveyor is transferring other pallets. An RFID reader is located in each zone 1 for each workstation. This reader reads the pallet's tags for identifying the entering pallet. Table 2 shows the functionality of these zones.

Table 2: Conveyor zones description

Zone ID	Zone Functionality	Availability in FASTory
Z1	Work station entrance. This zone reads the ID of the transferred pallet from previous work station. In this zone the decision could be taken for move the pallet to the work station or to pass it to the next work station	This zone is available in all work stations except ws7
Z2	Internal buffer for the work stations. This zone holds extra pallet in the line if Z3 is occupied	This zone is available in all work stations
Z3	Productions zone. This zone is used to accomplish the task of the work station (loading/ unloading and drawing).	This zone is available in all work stations
Z4	This zone is used for bypass conveyor. There are two functionalities; first one is to decide which pallet will be transferred to zone 5 (from 3 to 5 or from 4 to 5). The other functionality is to hold pallet if zone 5 is occupied.	2, 3, 4, 5, 6, 8, 9, 10, 11 and 12
Z5	Exit zone where the pallet reach the end of work station	This zone is available in all work stations

With this architecture, each robot and conveyor is represented as an RTU (Remote terminal Unit), 12 Conveyors and 12 robots. Each RTU is controlled by INICO S1000 [42]. INICO S1000 a is web-based industrial controller. It supports DPWS (Devices Profile Web Services) mechanism for discovery and manipulating web services. Furthermore, INICO S1000 uses XML/SOAP standard for communications. During eScop project, a reasonable update has been conducted on the S1000. As a requirement of the project, the PHL should support RESTful web services in the communications level.

For this thesis, a replica of the FASTory assembly line is deployed. This replica is known as FASTory simulator. This simulator was developed by eScop development team in order to mimic the real line. The simulator is considered as a handy tool for the development and validation stages during the project [43]. The FASTory simulator represents the physical layer of OKD-MES approach in this thesis. The simulator services can be found in instructions page of the simulator[2]. The FASTory simulator APIs are available in the attached appendix.

4.2 Deployment of OKD-MES on FASTory

OKD-MES has been developed for eScop project by different parties. As described in section 2.2, OKD-MES layers are represented by ontologies. This ontology is separate than MES functions ontology. It is developed by eScop project partners as well. This

[2] http://escop.rd.tut.fi/fastory/instructions

section presents eScop tools[3] (VIS, RPL, ORL and PHL). PHL layer is presented in the previous section. In this section the focus will be on VIS, RPL and ORL and how these layers are prepared and deployed on FASTory line.

4.2.1 Visualization Layer

As any solution for MES, an interface has to be designed for user to interact with manufacturing system. In OKD-MES there is layer so called Visualization layer (VIS). The objective of this layer in OKD-MES covers many topics. In this thesis, VIS is just involved for orders entry.

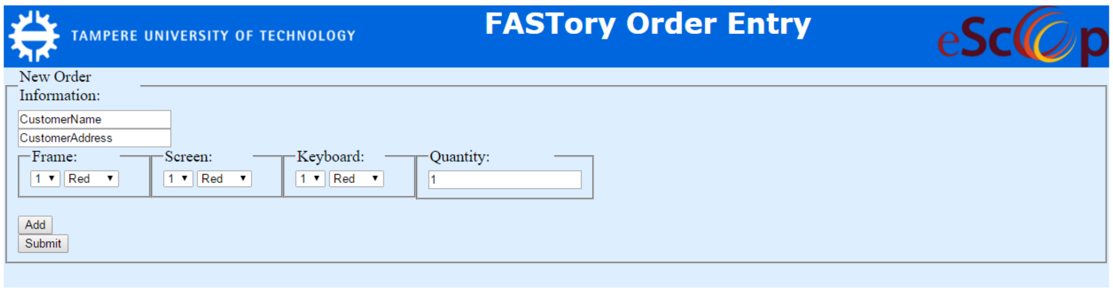


Figure 20: Order entry page in VIS

For implementation an order entry page has been designed to serve the functionality of entering orders. See Figure 20. In this page the user insert order by selecting frame, screen and keyboard for the product. Each order has one product with desired quantity. Then the order is sent to scheduling function (F2: Operation/ Detail Scheduling).

4.2.2 Representation Layer

As described in section 2.2, RPL works as information manipulator in OKD-MES[4]. Thus in FASTory implementation, RPL provides information about all components in OKD-MES. This information could be used for configuration of these components or status of manufacturing system. This study aims to test the collaboration between MES functions with RPL.

RPL provides information as defined in ontologies. These ontologies contain information about all services that are available in OKD-MES layers (MES functions as well) in a block so called ***Service Inventory***. This information allows the requester to retrieve service's URL and service's parameters via service ***Id***. In other words, this information is defined by the user once s/he build the ontology. Therefore, the user is able to choose any tool to fulfill MES functionality. In this thesis RPL has been used

[3] http://www.escop-project.eu/training/pilots/FASTory_Tools.pdf
[4] http://www.escop-project.eu/training/components/RPL_training.pdf

for orders, recipes and pallets information in FASTory line. The RPL APIs are listed in the attached appendix.

4.2.3 Orchestration Layer

Orchestration layer (ORL) is the layer which control shopfloor through PHL. The technique for ORL may vary according to different reasons. As an example, industry type (process, discrete or continuous) affects how the ORL will behave for certain scenarios. Also the orchestration method (i.e. BPEL) affects as well. Another factor may affect the design of ORL is the communication structure of the manufacturing system (i.e. SOAP vs REST).

For this thesis, ORL is designed to control PHL from pallets point of view. In this manner, ORL subscribes for Z1_Changed in PHL for all workstations (Z2_Changed for WS7). Then once the event is triggered, ORL retrieves PalletID and senderID for requesting list of executable URLs from MES-F3 (Dispatching Production Units). After that MES-F3 provides ORL with list of tasks that needs to be applied for the pallet. See Figure 21.

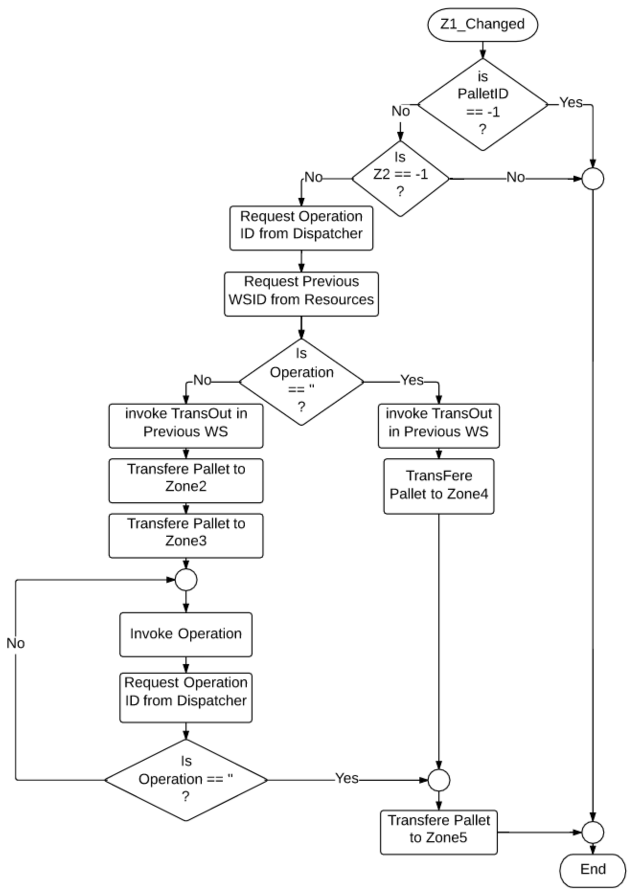

Z1_Changed
is PalletID == -1 ?
No
Yes
Is Z2 == -1 ?
No
No
Request Operation ID from Dispatcher
Request Previous WSID from Resources
Is Operation == '' ?
No
Yes
invoke TransOut in Previous WS
Transfere Pallet to Zone2
Transfere Pallet to Zone3
invoke TransOut in Previous WS
TransFere Pallet to Zone4
No
Invoke Operation
Request Operation ID from Dispatcher
Is Operation == '' ?
Yes
Transfere Pallet to Zone5
End

Figure 21: Flowchart for interaction between ORL and MSE-F3

In this manner, ORL provides fixed and general orchestration for all devices in PHL. However, ORL do not make decisions for pallet to be move to which workstation. ORL which has been tested has developed by eScop project partners[5]. The RESTful APIs for the ORL are listed in the attached appendix.

4.3 MES functions Layer

In chapter three, the architecture of MES functions has been presented. This section shows the implementation of MES functions by using the developed approach. For some of MES functions, MESF is populated to describe the MES functions services and configurations. Then the functional scripts are built to serve the functionality of MES function. These are the steps which the user has to make in order to deploy this concept on his/her factory. For this thesis the implementation covers three MES functions; MES-F1, MES-F2, MES-F3, MES-F5 and MES-F10. See Figure 22.

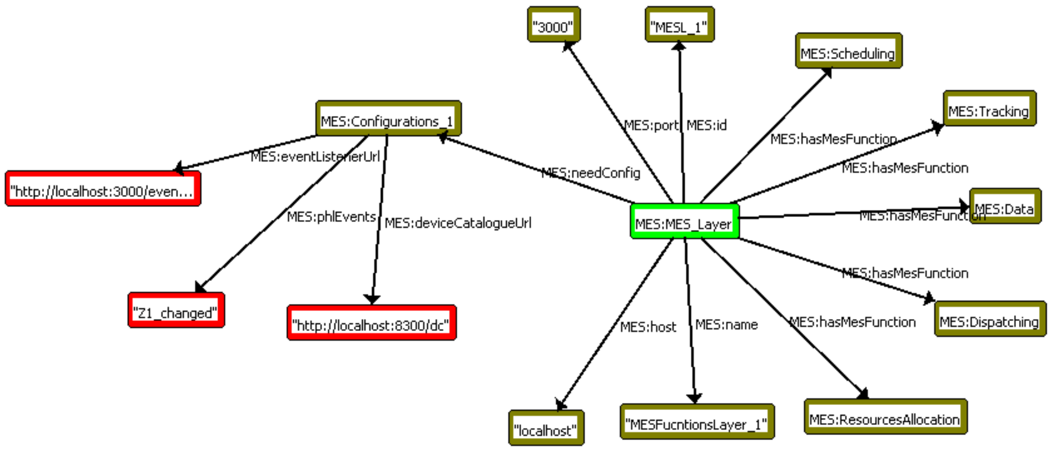


Figure 22: MES functions Layer instances

For the validation purposes, and as presented in section 3.2.4, three MES functions are used for testing the platform. Each of these functions provides several RESTful services that are includes some functionalities. The following table (Table 3) shows the RESTful services which are needed for testing running the validation scenario.

Table 3: MES functions services

#	Function Name	Functionalities	Services	Method	Query	Body	Response
1	Status	onEvent	/eventListenr	POST			200
		getAvailableService	/availalbe services/:id	GET	id		_function
2	Scheduling	onEvent	/eventListenr	POST			200
		getOrder	NeedOrder	GET	id		_function

[5] http://www.escop-project.eu/training/components/eScop_ORL_how_to.pdf

		SetOrder	NewOrder	POST		or-der	204
3	Dispatching	onEvent	/eventListenr	POST			200
		getTask	/nextOperation	GET	id,ws		_func-tion
4	Data	onEvent	/eventListenr	POST			200
		getEvents	/eventrecorder	GET			_func-tion
		getData	/dataRecorder	GET			_func-tion
5	Tracking	onevent	/eventListener	POST			_func-tion
		getProducts	/products	GET			_func-tion

4.3.1 Resource Allocation and Status

For any manufacturing system, resources statues are required to be accessible. This information is used in various implementations like monitoring of the system by VIS, orchestration for PHL and material status for supplement department. In this thesis, MES-F1 (Resource Allocation and Status) is used for providing status of available services in PHL. Table 4 shows the REST services which MES-F1 serves. For first service, MES-F1 responses with available services that are provided by a specified RTU with id. This service responds with a JSON scripts describing the status of the RTU. As described in section 4.1, each drawing workstation provides draw service with different color. The selected color in each work station governs the available services of the workstation. Besides that, the available services of workstations depend on the current position of the pallets in the workstation. For instance, if a pellet is located in zone 3 then TransZone23 services will not be available.

Table 4: MES-F1 services

#	Service Method	Service URL	Service Response
1	GET	/resources/available-services/:id	Return all available service for the RTU which holds the request ID (RTUID)
2	POST	/resources/event-listener	Empty response

As the platform employs the event-driven approach, MES-F1 retrieves the required information directly from the PHL. Thus, MES-F1 listens to post service through a unique URL (/resources/event-listener) for ***Z1_Changed***, ***Z2_Changed***, ***Z3_Changed***, ***Z4_Changed***, ***Z5_Changed***, ***DrawStartExecution***, ***DrawEndExecution*** and ***Pen-Changed*** events that issued by each device. With zone change events, MES-F1 extracts information about status of the conveyor (Busy/Idle) and of zones as well. Meanwhile ***DrawStartExecution*** and ***DrawEndExecution*** events provide information about status of the robot (Busy/Idle). On the other hand, ***PenChanged*** event provide knowledge about the color which the robot able to draw with.

This thesis aims to provide easy configurability for manufacturing systems. By applying the concept which illustrated in chapter 3, the user is required to provide ontologies for the configuration and the services that the function hosts. Besides that, the user is required to provide function models for the system. In this implementation, MES-F1 contains 2 functions; ***EventListener (EventBody)***, ***AvailableServices (RTUID)***. ***EventListener*** function is used once MES-F1 receives a notification about events. This function takes the body of the notification and performs certain calculations to extract a useful information like the status of equipment. ***EventListener*** do not return any value. The sequence diagram in Figure 23 illustrates the interaction for the MES-F1.

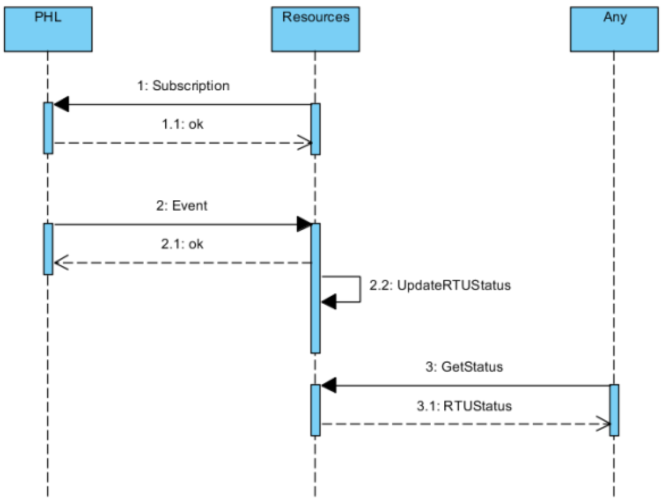


Figure 23: Sequence Diagram of resources function

For ***AvailableServices*** function, one input variable is needed. This variable could be string with RTU id or string with 'ALL' which refers to all RTUs. Then a series of logical operations is required to return the available services of the RTU(s). This function returns JSON object that contains information about statuses and available services in each RTU.

After defining the functions in function module, the user is required to define certain knowledge by using the proposed structure of ontology in 3.1. Figure 24 shows the instances for ontology regarding the implementation of MES-F1 in this thesis.

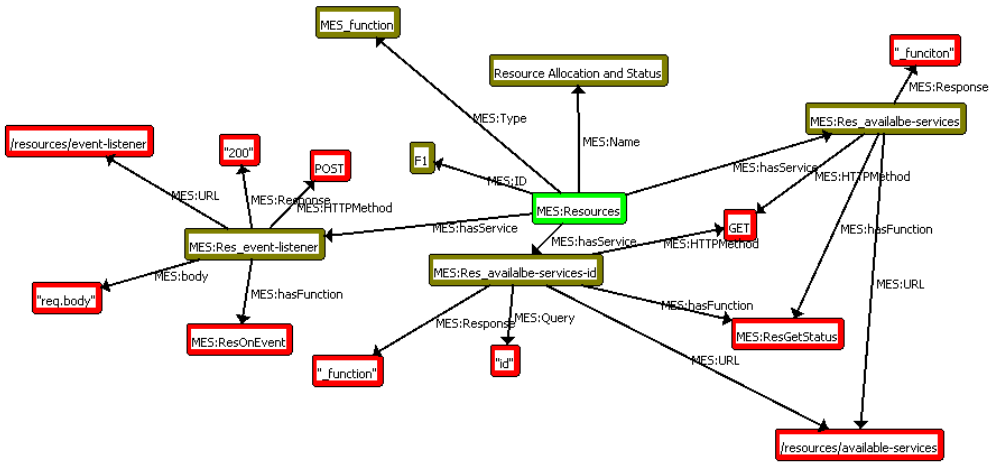


Figure 24: Example of population MESF for MES-F1

As shown, the MES-F1 hosts three services; *Res_available-resources*, *Res_availabe-resources-id* and *RES_event-listener*. The *Res_availabe-resources-id* and *Res_availabe-resources* are *GET* services which returns the available services in the PHL RTUs. The main difference is that *Res_availabe-resources-id* returns the specified RTU by its id while *Res_availabe-resources* returns available services for all PHL RTUs.

4.3.2 Operations/ Detail Scheduling

As defined in ISA-95, MES layer binds ERP with shopfloor of factories. One of the main functionalities that MES has to provide is scheduling and planning. OKD-MES is not a different case. Scheduling orders and equipment is required. MES-F2 provides such functionality. In fact, this function translates incoming orders from customers into schedules for machines.

In this implementation, MES-F2 does not provide schedules for machines since the orchestration is occurred on pallets not machines. Instead, MES-F2 controls the color of the used pen in each workstation. Therefore, the services availability of each workstation is affected. Besides that, MES-F2 connects orders to products. It provides MES-F3 (Dispatching Production Unit Function) with products that need to be produced. This interconnection is described more in the next section where MES-F3 is illustrated. Figure 25 depict the sequence diagram for the MES-F2 interactions.

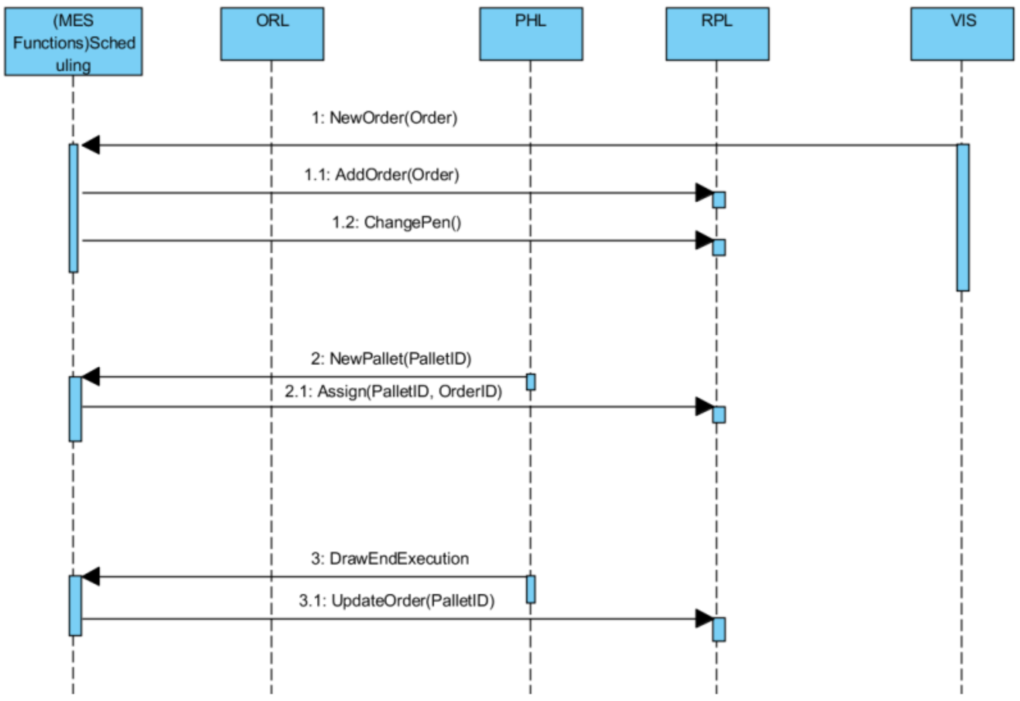


Figure 25: MES-F2 interaction sequence diagram

Regarding this implementation, MES-F2 hosts two services; needOrder and newOrder. NeedOrder service is used when a client request product recipe to produce. This service gets all registered orders in RPL and choose the eldest order then it return it as JSON object. While newOrder is requested once the user inserts a new order through VIS. In this service, MES-F2 register the new order in RPL then it retrieves all registered orders in RPL in order to control the used color in each workstation. Table 5 shows the available services in MES-F2.

Table 5: MES-F2 services

#	**Service Method**	**Service URL**	**Service Response**
1	GET	/scheduling/needOrder	Returns recipe of product regarding to priority of orders
2	POST	/scheduling/newOrder	Return empty response
3	POST	/scheduling/event-listener	Empty response

As all MES functions, MSE-F2 listens to incoming notifications using (/scheduling/event-listener) URL. In this study, MES-F2 listens to one event in PHL. This event is so called PalletLoaded. PalletLoaded event is triggered once a pallet is loaded to the line in workstation 7. MES-F2 registers the pallet in RPL in order to be known for MES-F3. This interaction on is described in more details in next section.

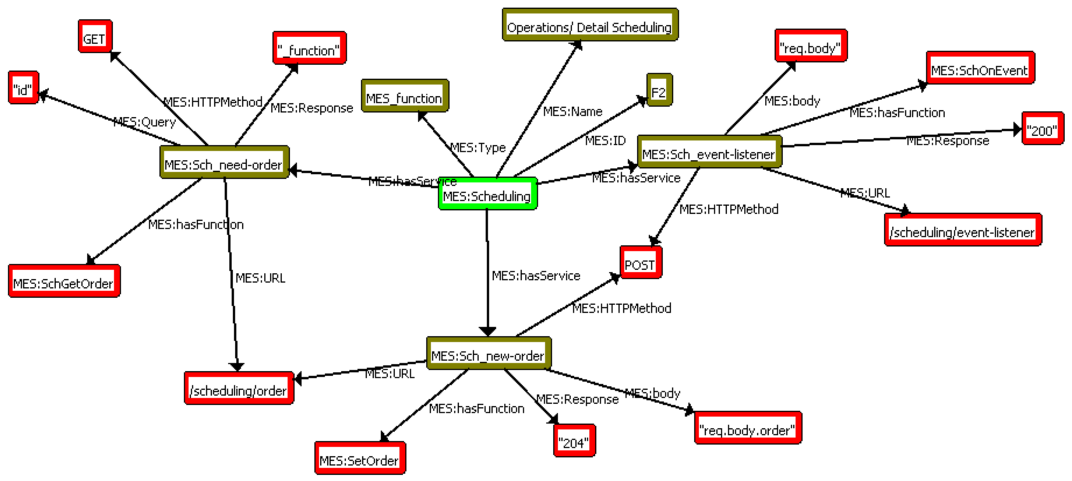


Figure 26: An example of the populated MESF for MES-F2

The functionality model for MES-F2 contains three functions; needOrder (), newOrder () and EventListener (). NeedOrder function retrieves all orders in RPL and chose the order with highest priority. This priority is defined by the user such as registration time or customer urgency for the order. The needOrder function returns the recipe of the product. On the other hand, newOrder function gets the new inserted order to the system and registers the order in RPL. Figure 26 shows a populated example of MESF for MES-F2

4.3.3 Dispatching Production Units

Another MES function which is implanted in this study is Dispatching Production Unit (MES-F3). This function translates orders to task that has to be performed on PHL. This function is hardly connected to ORL since ORL is responsible to control PHL on higher level. This function also requests recipes from MES-F2 in order to provide ORL with list of tasks. As well, MES-F3 interacts with MES-F1 to retrieve information about available services in PHL.

Table 6: MES-F3 RESTful services

#	Service Method	Service URL	Service Response
1	GET	/dispaching/next-operation	Returns list of operations which ORL requested for managing PHL

According to Table 6, MES-F3 hosts one service. This service is called next-operation. This service is used for reviving list of tasks according to the orders recipe that mapped to a pallet and to workstation available services as well. As illustrated in 4.2.3, ORL listens to Z1_Changed event in PHL for each workstation. Once this event is issued, ORL requests from MES-F3 to provide list of tasks. MES-F3 requests both available services of the workstation and recipe of the order which mapped to the pallet. Then MES-F3 matches the recipe with available service. If there is a match, MES-F3 responds with PHL services IDs. If not, then it responds with empty body. Figure 27 shows the sequence diagram for this function.

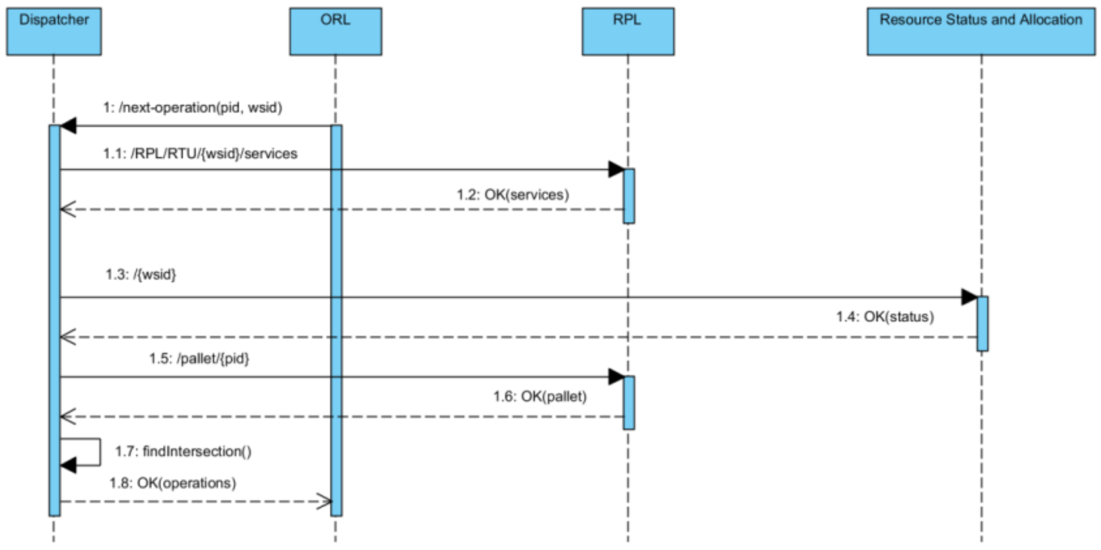


Figure 27: MES-F3 interaction sequence diagram

For MES-F3 there is one function in functionality model. This function is called GetNextOperation (WS, Pallet). This function accepts two input variables; WS which represents workstation ID and Pallet for Pallet ID. As described in 4.2.3, this function is invoked once pallet reaches zone 1 of each workstation. In this manner, and according to FASTory design, there are three possibilities; WS1 for loading and unloading papers, WS7 for loading and unloading pallets and the rest workstations for drawing process. Therefore, MES-F3 firstly requests available services for workstation from MES-F1. If the workstation is down or do not afford any services, then the function returns empty list of tasks. If not, then MES-F3 examines if the pallet is registered in the system or not by querying pallet information from RPL. For pallet status there are three possibilities; unregistered pallet which means the pallet will y unloaded once it reaches WS7. For any other workstations, unregistered pallet passes by the workstation. The second possibility includes registered pallet but without order mapping. In this case, pallet will be mapped to an order once it reaches WS1 otherwise it will pass by workstations. The last possibility addresses the registered and mapped pallet. Here MES-F3 requests MES-F2 to provide recipe for the pallet. If the pallet does not require any process, MES-F3 responds with empty body which means the pallet will pass by all workstations except WS1 which the paper will be unloaded. If the recipe still contains some processes to be conducted, MES-F3 matches the available services with processes that have to be drawn. If the match shows some processes, then MES-F3 responds with id of process which have been matched the available services. if not, then pallet will pass by the work station. An example of populated MESF mode for MES-F3 is depicted in Figure 28. As appear in the figure, has two services; *next-operation* and *event-listener*. As well it contains datatypes related to the MES function, i.e. *name*, *id* and *type*.

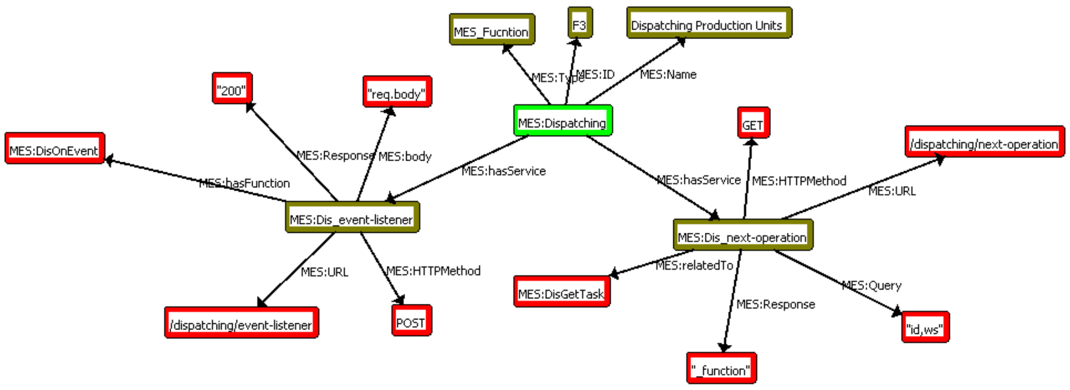


Figure 28: A populated MESF model for MES-F3

1.1.1 Data Collection/ Acquisition

Data analysis is one of the core functionality in any MES. Thus Data collection techniques are main stream in MES development. MES-F5 (Data Collection/ Acquisition) collects data like I/O values, events, invoked services and machine status with respect to time. This data is used then for analysis, maintenance investigation or performance analysis. In this study, MES-F5 is used to collect data of PHL and provide it as web service. Another functionality which is included to this function is events subscriptions in order to notify all MES functions about PHL events.

Table 7: RESTful API for MES-F5

#	Service Method	Service URL	Service Response
1	GET	/data/event-record	The response contains all events triggered in PHL
2	GET	/data/data-record	The response contains all I/O for PHL with respect to a certain period

Table 7 shows the services which MES-F5 provides. Event-record provides all events that have been issued by PHL. While data-record provides all values of I/O in PHL. It also might provide other MES fucnbtions with event notifications in case an MES function does not subscribe to events. See Figure 29. For data-record, a time period is assigned in configuration in ontologies of MES-F5. As shown in Figure 30: MESF for MES-F5, a populated MESF example for MES-F5 contains three functions; GetEventRecord, GetDataRecord and StrartScanning.

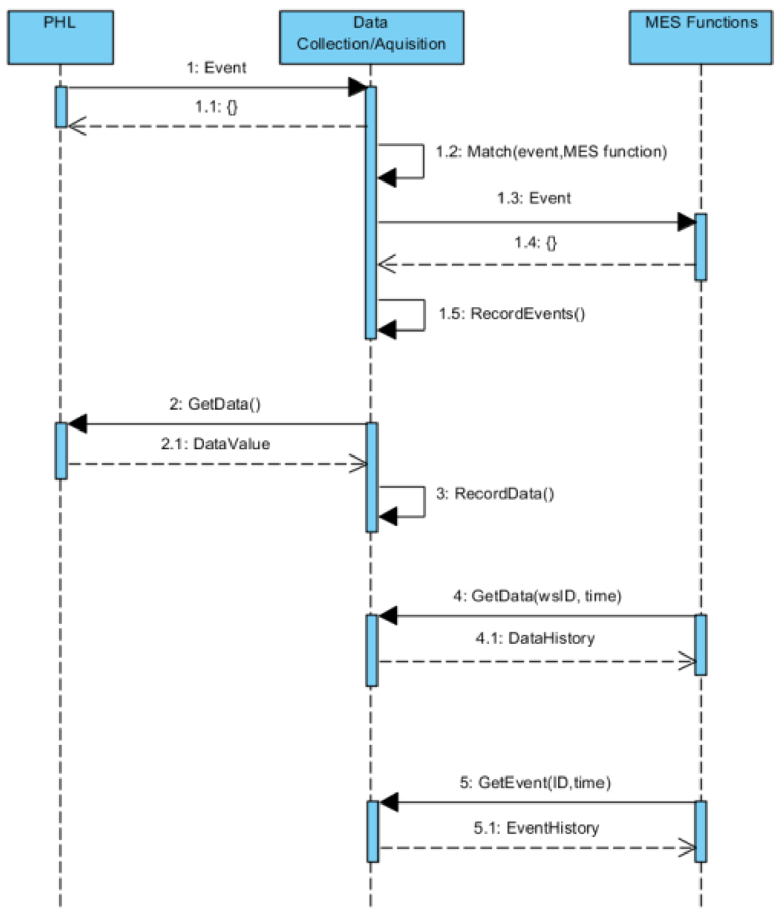


Figure 29: Sequence diagram for interactions of MES-F5

GeteventRecord returns list of all events occurred in PHL. Same as GetEventRecord, GetDataRecord returns list of I/O status with respect to time. Both functions do not accept inputs and return JSON object. StartScanning function scans I/O of PHL. The scanning procedure composes of invoking of REST web services in order to retrieve I/O in PHL. This function called once the MES-F5 is initialized.

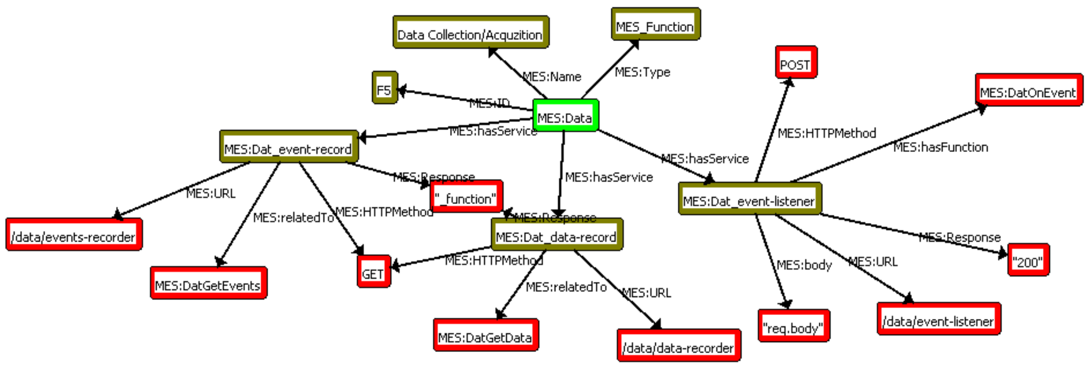


Figure 30: MESF for MES-F5

4.3.4 Products Tracking and Genealogy

As any Factory tracking the product during manufacturing process is necessary. The generated information supports i.e. the quality assurance and fault tracking and fast recovery. In this thesis, the MES-F10 (Products Tracking and Genealogy) has been deployed once **Z_changed**, **StrartDrawingExecution** and **EndDrawingExecution** events are triggered. As depicted in Figure 31, the MES-F10 creates a record for each product listing the location, time and resource which participated for producing the products.

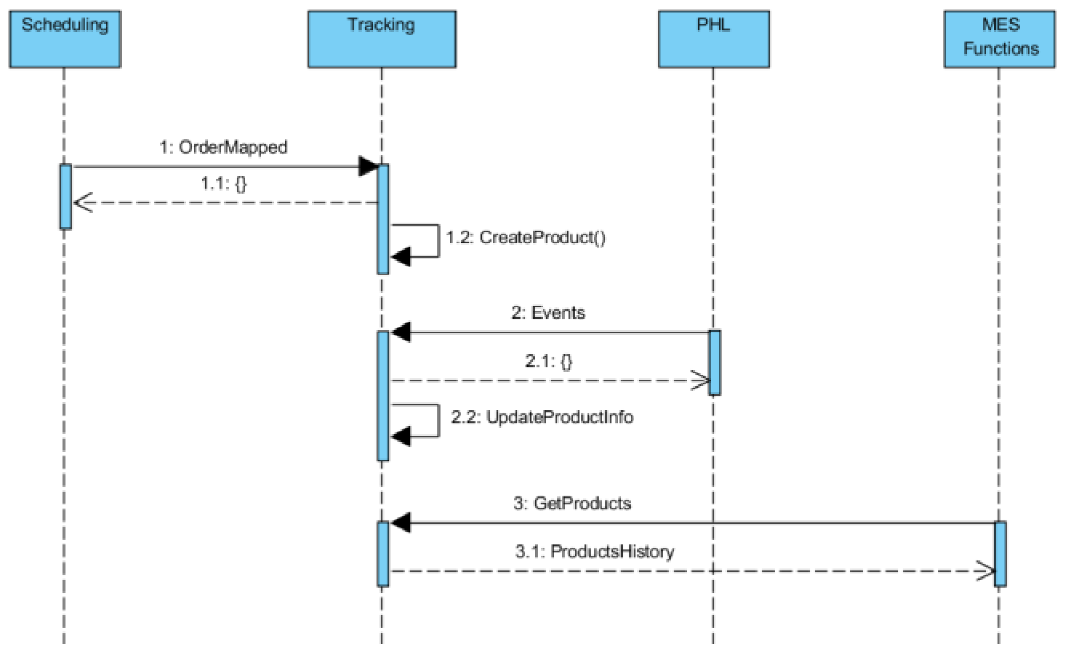


Figure 31: Sequence diagram of MES-F10 interactions

The MES-F10 hosts one service which allows the consumers to retrieve the production tracking information. See Table 8. Once this service is invoked, the MES-F10 responses with a JSON formatted message show each part of the product where it has been manufactured and the time of manufacturing. Such an information could be handy for the performance analysis or fault tracking in a series of product.

Table 8: RESTful APIs for MES-F10

#	Service Method	Service URL	Service Response
1	GET	/products	The response contains all products which have been produced or in progress.

According to the concept of the MES platform, the MESF model can be instanced as shown in Figure 32 for MES-F10. As shown, the MES-F10 hosts two services; Tra_products which is a GET service that provides the user with all products which has been produced in the factory. Tra_event-listener on the other hand, receives the notifications from the PHL.

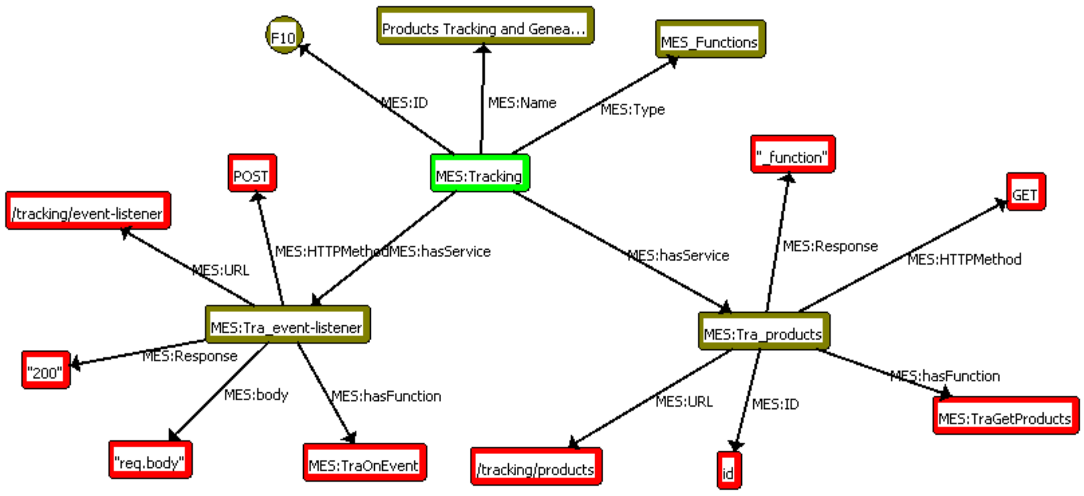


Figure 32: Example of MESF model for MES-F10

5 RESULTS

As illustrated in 3.2.4, the proposed validation scenario highlights a production used-case (from Order to Product). Therefore, the implementation[6] focused on the required MES function that related to this validation scenario. Besides, it is important to mention that this validation scenario shows qualitative results. In other words, the expected results show how well the MES functions are implemented.

The scenario starts by inserting the order via the VIS. The user is able to define the required product by specifying the model and the color of the screen, keyboard and frame. After choosing the required product the user submits the order to the MES-F2. Accordingly, the colour of the used pen in the PHL is changed according to the inserted order. This allows the manufacturing system to balance the load on the PHL in case of multi orders. See Figure 33.

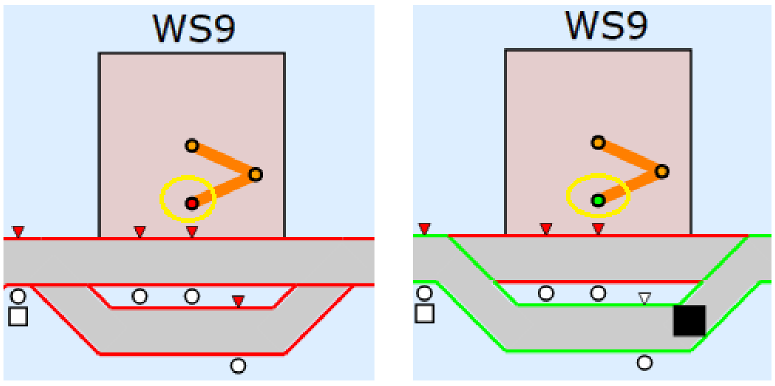


Figure 33: Changes in PHL pen before and after inserting an order

Then, the user manually inserts the pallet in Z1 of WS8. Once the pallet appears in Z1 of the WS8, it will be transferred till it reaches WS1. Each time the pallet reach Z1, the ORL request the status of the pallet. At this stage the pallet is registered in the system but not assigned to an order, therefore the ORL transfer it out.

With the eScop implementation which is the bases of this thesis, a queuing mechanism is employed in the RTUs in order to queue the requests for operations. With this feature, ORL provide high level of controlling equipment in PHL. Meanwhile each RTU in PHL is considered as smart entity which manages itself. See Figure 34.

[6] http://www.escop-project.eu/training/

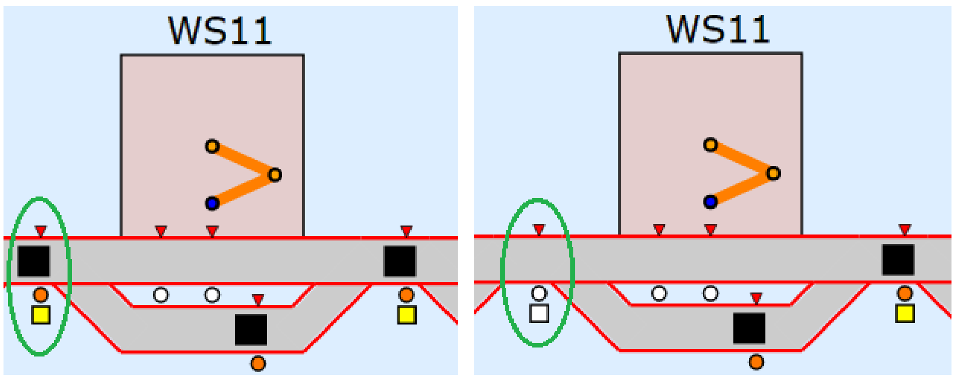
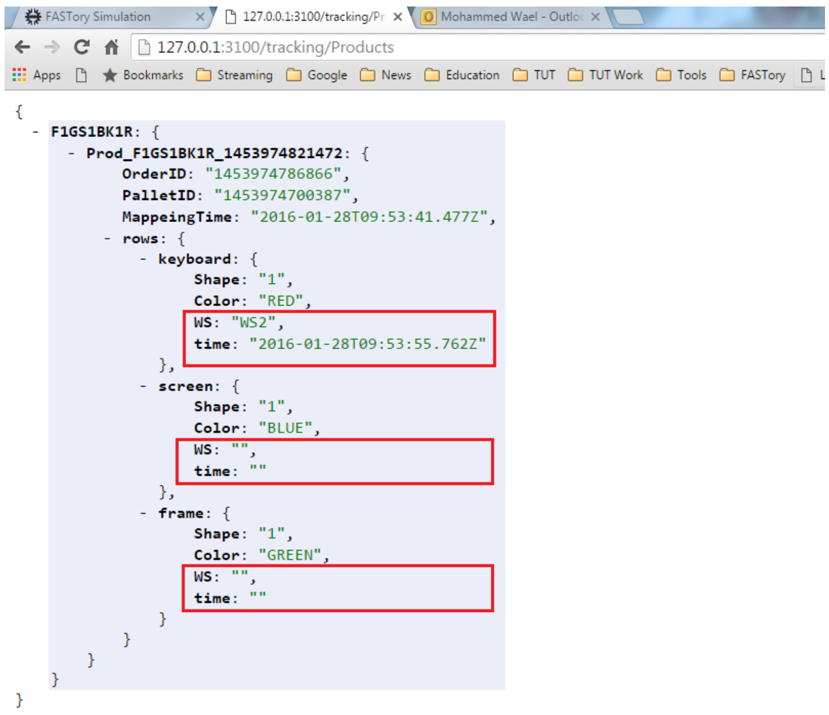


Figure 34: Effect of queuing in PHL

While the pallets transferring between workstations, the MES-F3 provides list of tasks each time pallet reach zone 1 in each workstation. The user can assure that dispatcher is performing well by

1) Observing pallets transferred between work stations.
2) Monitor the properness of choosing correct work stations, the user may get to products tracking page.

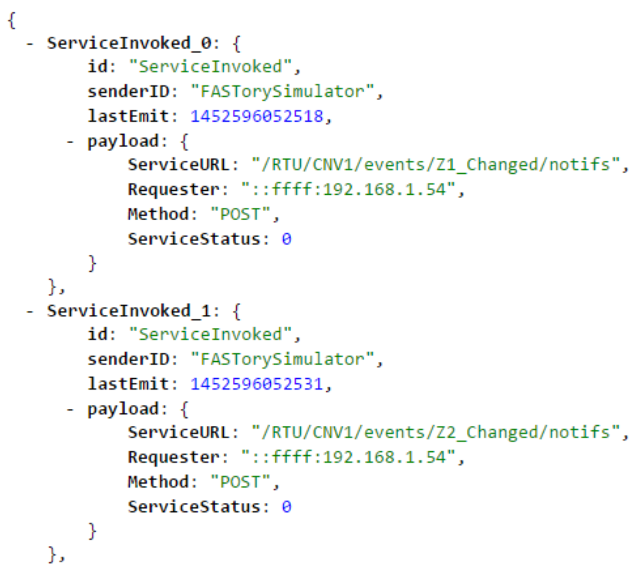
As appears in Figure 35, a product with id **Prod_F1GS1BK1R_1453974821472** which is related to order **1453974786866** and is mapped to pallet that has id **1453974700387** at **2016-01-28T09:53:41.477Z** is in production process. The product is successfully received keyboard part in **WS2** at **2016-01-28T09:53:55.762Z** and still waiting frame and screen.



```
{
  - F1GS1BK1R: {
      - Prod_F1GS1BK1R_1453974821472: {
            OrderID: "1453974786866",
            PalletID: "1453974700387",
            MappeingTime: "2016-01-28T09:53:41.477Z",
          - rows: {
              - keyboard: {
                    Shape: "1",
                    Color: "RED",
                    WS: "WS2",
                    time: "2016-01-28T09:53:55.762Z"
                },
              - screen: {
                    Shape: "1",
                    Color: "BLUE",
                    WS: "",
                    time: ""
                },
              - frame: {
                    Shape: "1",
                    Color: "GREEN",
                    WS: "",
                    time: ""
                }
            }
        }
    }
}
```

Figure 35: Product tracking

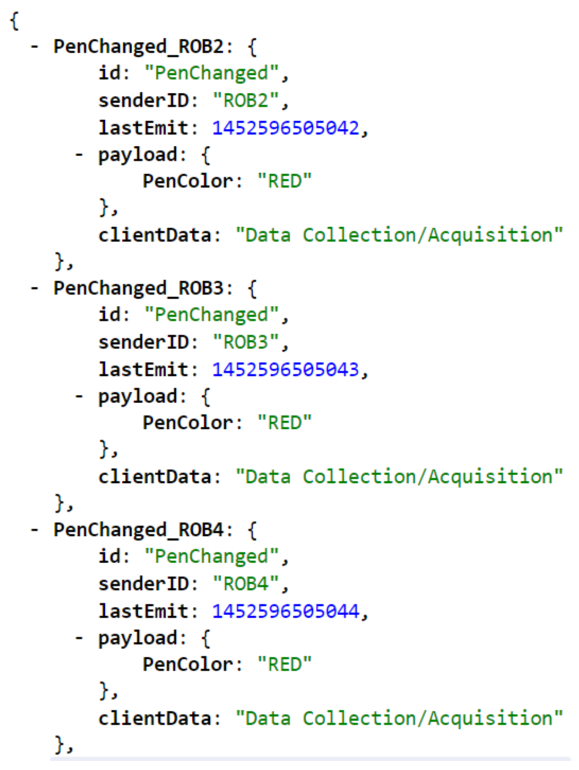
Meanwhile the user is able to monitor the invoked services in the PHL (see Figure 36). The figure shows a JSON formatted description of the service. This information could be used for performance analysis or error tracking.



```
{
 - ServiceInvoked_0: {
       id: "ServiceInvoked",
       senderID: "FASTorySimulator",
       lastEmit: 1452596052518,
     - payload: {
           ServiceURL: "/RTU/CNV1/events/Z1_Changed/notifs",
           Requester: "::ffff:192.168.1.54",
           Method: "POST",
           ServiceStatus: 0
       }
   },
 - ServiceInvoked_1: {
       id: "ServiceInvoked",
       senderID: "FASTorySimulator",
       lastEmit: 1452596052531,
     - payload: {
           ServiceURL: "/RTU/CNV1/events/Z2_Changed/notifs",
           Requester: "::ffff:192.168.1.54",
           Method: "POST",
           ServiceStatus: 0
       }
   },
```

Figure 36: Service recorder

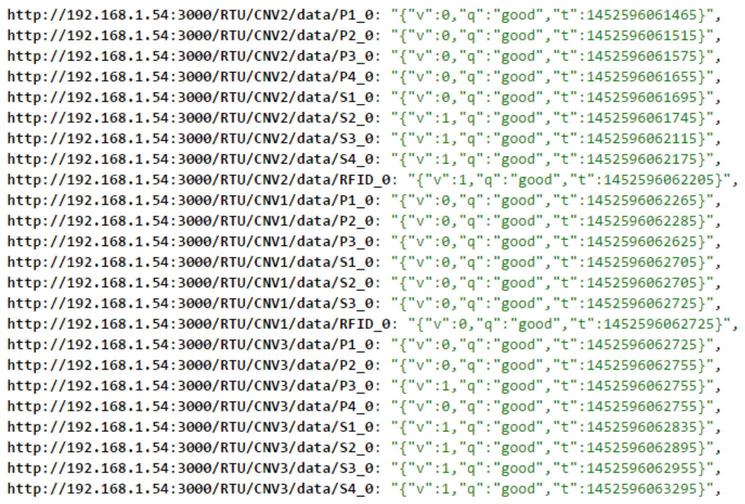
Similarly, the MES-F5 records all notification that triggered in the Manufacturing system. This operation depends on the user configuration of the MES function. Figure 37 presents an example of the notification that has been triggered by the PHL.



```
{
 - PenChanged_ROB2: {
       id: "PenChanged",
       senderID: "ROB2",
       lastEmit: 1452596505042,
     - payload: {
           PenColor: "RED"
       },
       clientData: "Data Collection/Acquisition"
   },
 - PenChanged_ROB3: {
       id: "PenChanged",
       senderID: "ROB3",
       lastEmit: 1452596505043,
     - payload: {
           PenColor: "RED"
       },
       clientData: "Data Collection/Acquisition"
   },
 - PenChanged_ROB4: {
       id: "PenChanged",
       senderID: "ROB4",
       lastEmit: 1452596505044,
     - payload: {
           PenColor: "RED"
       },
       clientData: "Data Collection/Acquisition"
   },
```

Figure 37: Event recorder

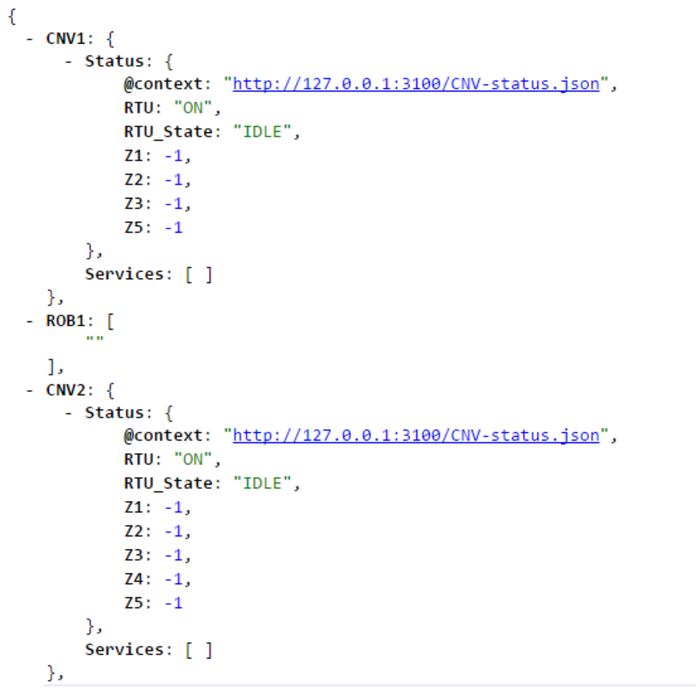
As well, I/Os status is persisted in the MES-F5 (see Figure 38).as appears in the figure, the status of the I/Os is presented as JSON file which contains three elements; v which denotes the value of the I/O. In the FASTory case, the I/Os are binary and may have 0 or 1 value. Then the q which represents the quality of the I/O. this assures that the RTU is connected. The last element is the t which shows the current time in milliseconds when the data is retrieved.



```
http://192.168.1.54:3000/RTU/CNV2/data/P1_0: "{"v":0,"q":"good","t":1452596061465}",
http://192.168.1.54:3000/RTU/CNV2/data/P2_0: "{"v":0,"q":"good","t":1452596061515}",
http://192.168.1.54:3000/RTU/CNV2/data/P3_0: "{"v":0,"q":"good","t":1452596061575}",
http://192.168.1.54:3000/RTU/CNV2/data/P4_0: "{"v":0,"q":"good","t":1452596061655}",
http://192.168.1.54:3000/RTU/CNV2/data/S1_0: "{"v":0,"q":"good","t":1452596061695}",
http://192.168.1.54:3000/RTU/CNV2/data/S2_0: "{"v":1,"q":"good","t":1452596061745}",
http://192.168.1.54:3000/RTU/CNV2/data/S3_0: "{"v":1,"q":"good","t":1452596062115}",
http://192.168.1.54:3000/RTU/CNV2/data/S4_0: "{"v":1,"q":"good","t":1452596062175}",
http://192.168.1.54:3000/RTU/CNV2/data/RFID_0: "{"v":1,"q":"good","t":1452596062205}",
http://192.168.1.54:3000/RTU/CNV1/data/P1_0: "{"v":0,"q":"good","t":1452596062265}",
http://192.168.1.54:3000/RTU/CNV1/data/P2_0: "{"v":0,"q":"good","t":1452596062285}",
http://192.168.1.54:3000/RTU/CNV1/data/P3_0: "{"v":0,"q":"good","t":1452596062625}",
http://192.168.1.54:3000/RTU/CNV1/data/S1_0: "{"v":0,"q":"good","t":1452596062705}",
http://192.168.1.54:3000/RTU/CNV1/data/S2_0: "{"v":0,"q":"good","t":1452596062705}",
http://192.168.1.54:3000/RTU/CNV1/data/S3_0: "{"v":0,"q":"good","t":1452596062725}",
http://192.168.1.54:3000/RTU/CNV1/data/RFID_0: "{"v":0,"q":"good","t":1452596062725}",
http://192.168.1.54:3000/RTU/CNV3/data/P1_0: "{"v":0,"q":"good","t":1452596062725}",
http://192.168.1.54:3000/RTU/CNV3/data/P2_0: "{"v":0,"q":"good","t":1452596062755}",
http://192.168.1.54:3000/RTU/CNV3/data/P3_0: "{"v":1,"q":"good","t":1452596062755}",
http://192.168.1.54:3000/RTU/CNV3/data/P4_0: "{"v":0,"q":"good","t":1452596062755}",
http://192.168.1.54:3000/RTU/CNV3/data/S1_0: "{"v":1,"q":"good","t":1452596062835}",
http://192.168.1.54:3000/RTU/CNV3/data/S2_0: "{"v":1,"q":"good","t":1452596062895}",
http://192.168.1.54:3000/RTU/CNV3/data/S3_0: "{"v":1,"q":"good","t":1452596062955}",
http://192.168.1.54:3000/RTU/CNV3/data/S4_0: "{"v":1,"q":"good","t":1452596063295}",
```

Figure 38: I/O recorder

Finally, the user could read the online status of the RTUs in the PHL. This information is supported by the MES-F1. As depicted in Figure 39, the status of the CNV1 and CNV2 is shown as IDLE. Besides, the user can see the pallet in each zone. In the presented case, all zones of CNV1 and CNV2 are empty. Finally, the services array shows the available services in the RTU.



```
{
  - CNV1: {
      - Status: {
            @context: "http://127.0.0.1:3100/CNV-status.json",
            RTU: "ON",
            RTU_State: "IDLE",
            Z1: -1,
            Z2: -1,
            Z3: -1,
            Z5: -1
        },
        Services: [ ]
    },
  - ROB1: [
        ""
    ],
  - CNV2: {
      - Status: {
            @context: "http://127.0.0.1:3100/CNV-status.json",
            RTU: "ON",
            RTU_State: "IDLE",
            Z1: -1,
            Z2: -1,
            Z3: -1,
            Z4: -1,
            Z5: -1
        },
        Services: [ ]
    },
```

Figure 39: Resources and available services

As shown in the above snapshots, the MES functions has been implemented in web services manner. However, the platform did not perform for complicated implementation due to lack of dependencies management in the platform. However, it has shown a potential for future development.

6 DISCUSSION

Regarding to the previous chapter where the platform has been implemented in FASTory case-study, this chapter presents the author point of view as discussion of the presented solution. This chapter discusses the presented hypotheses in chapter 1, the outcome of the presented approach and a comparison with the effect of using this approach on the FASTory line.

Returning to the listed hypothesis in chapter 1, the usage of the web services is expected to provide independency and fast integration. In this matter, the presented approach illustrated this feature by the conducted tests. As shown, the platform validates the incoming request from different consumers without any relation to the operating system or the programming language that the consumer uses. Since it uses http RESTful services then the approach will function correctly. As well, the provided framework requires the NodeJs to be installed and as known, the NodeJs is available for most used operating systems.

The second point in the hypotheses list considers using the Knowledge-based system could simplify the user interaction. In this regard, the approach involve the knowledge based system in the design phase of the MES functions. As shown, the user defines the MES functions that are going to be used, the hosted services and the logic behind the defined services. As a user interaction, it is required by the user to populate the provided ontology model. It has to be noted that populating the ontology model without a proper UI could requires some skills such as knowledge of using SPARQL language. However, the user expected to provide all necessary information in the populating phase. After that, the framework will function according to the inserted knowledge.

The last point in the hypotheses addresses the usage of the functional scripts as separate logic entity which could increase the flexibility for adapting different choices. This point can be seen in the comparison with ready available solutions for the MES layer. As stated in the second chapter, many technology providers (i.e. Siemens, Rockwell) provide solutions for MES but these solutions tend to perform with certain industry such as monitoring of power plants. Meanwhile, the provided approach allows the user to define the services and the logic which increases the flexibility. Besides, with such solution, the user might employ the presented approach in different industry types since the logic is defined by the user.

Regarding the effect on the case study, the FASTory assembly line has been exploited in several projects such as eSonia, ASTUTE and finally eScop. The line has received a remarkable upgrade in terms of the technology which runs the line equipment. Before eScop project, FASTory assembly line was deploying DPWS with SOAP messaging standards. The orchestrations and the MES functionality concepts were embedded in the WSDL and BPMN files. The following table summarizes the effect that occur on the FASTory line after employing the presented approach.

Table 9: A comparison between before and after eScop implementation.

	Before	After
Technology	SOAP messages and DPWS.	OKD-MES solution with RESTful web services
MES Functionality	The MES functionalities were embedded in the BPMN and WSDL. In this sense, the user provides all the logic as request defined in the BPMN file.	Each MES function is defined and uniquely configured. The user also have the ability to choose the required functionality and then provides the proper logic.
Flexibility, re-configurability and expandability	Even the implementation of DPWS using WSDL and BPMN provided proper orchestration for the pallets in the FASTory line, the manufacturing system required time and effort for any change.	The manufacturing system allows the user to reconfigure the MES functions using the Knowledge model.
Communications	Large overhead due to SOAP standard, high traffic since the shopfloor devices host the WSDLs of the system.	RESTful with JSON body for less overhead. Shopfloor devices are controlled by ORL which runs in the network.

7 CONCLUSION

This thesis has been focusing on the implementation of MES functions as web services in an Open Knowledge-Driven Manufacturing Execution Systems. Firstly, an introduction has been presented which includes the motivation and objectives, the problem definition and the limitations and restrictions. While the second chapter provided a State of the Art in the second chapter. In this chapter, an illustration has been presented on the MES functionality in general and how OKD-MES works according to the eScop project.

The third chapter presented a methodology of building a platform for MES functions which allows the user to implement the MES functions as web services in an OKD-MES. The methodology showed the structure of the platform, the techniques of building the platform. In this manner, the technique includes a description on how the presented platform is built, how it works and how it can be validated. After that, chapter four presented the implementation of the platform in the OKD-MES. In this chapter, the OKD-MES layers and the MES platform are deployed in order to suit the FASTory used-case. Then, chapter five presented the results of deploying MES functions as services in OKD-MEs. As well, a comparison between before and after employing eScop in FASTory assembly line has been presented.

7.1 Matching the objective

According to the objective which is pointed out in the first chapter, this thesis is required to present a methodology of designing a MES functions which are suitable for any industry type, able to be configurable and expandable and able to work in the OKD-MES.

In this regards, this thesis provided the platform which can support all the requirement that mentioned above. However, according to lack of some features such as security, dependencies management and programing language dependencies, the platform is expected to function on used cases where the user will not include external (packages that are not in NodeJS or the used by the platform) dependencies in the functional logic.

7.2 Future Work

Due to the lack of the mentioned features, this platform requires:

1. Security implementations

Security is one of the main requirement when it comes to the web applications. The presented approach exposes all services and data to the user without security features. Therefore, this research work could be extended to address the security protocols which assure a secure communication. Besides, a proper encryption might be needed since all the transmitted date is represented as readable text.

2. Dependencies managements

As presented, NodeJS has been used for building the platform as a web application. For NodeJS, developers use public open source libraries provided by the npm repository. In this context, if a user requires some ready-made packages in the logic of the MES functionalities, then these dependencies will be loaded every time the function is called which might affect the performance after running the platform for a while. Therefore, a dependency manager is necessarily needed for managing the dependencies in the platform. As well, the presented platform uses JavaScript language for building the functional scripts. This makes the platform targeted to users whom gained a knowledge of programming in JavaScript.

3. Support the user with GUI

As any application that interacts with users, it is preferred that the application hosts a GUI. As presented in the SOTA, the OKD-MES provides this feature via the VIS layer. But since this platform could be also applied on different manufacturing systems, a proper interface could be considered as a future work.

8 REFERENCES

[1] 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.

[2] D. J. Adler, J. Herkamp, D. Henricks, and R. Moss, “Does a Manufacturing Execution System reduce the cost of production for bulk pharmaceuticals?,” *ISA Trans.*, vol. 34, no. 4, pp. 343–347, Dec. 1995.

[3] M. Georgoudakis, C. Alexakos, A. Kalogeras, J. Gialelis, and S. Koubias, “Decentralized Production control through ANSI / ISA-95 based ontology and agents,” in *2006 IEEE International Workshop on Factory Communication Systems*, 2006, pp. 374–379.

[4] “mes functions mesa - Google Search.” [Online]. Available: https://www.google.fi/search?q=mes+mesda&oq=mes+mesda&aqs=chrome..69i57j0l5.1786j0j4&sourceid=chrome&es_sm=122&ie=UTF-8#q=mes+functions+mesa. [Accessed: 03-Jul-2015].

[5] “ISA95, Enterprise-Control System Integration- ISA.” [Online]. Available: https://www.isa.org/isa95/. [Accessed: 05-Sep-2015].

[6] “BatchControl.com: Now This is Exciting!” [Online]. Available: http://www.batchcontrol.com/s95/s95.shtml. [Accessed: 13-Nov-2015].

[7] “Understanding Manufacturing Execution Systems (MES).” [Online]. Available: https://www.qad.com/Public/Collateral/Freedom%20MES%20White%20Paper.pdf. [Accessed: 04-Apr-2016].

[8] “Siemens MES for Discrete Industries – SIMATIC IT - Industry - Siemens.” [Online]. Available: http://w3.siemens.com/mcms/mes/en/industry/discretemanufacturing/pages/default.aspx. [Accessed: 05-Sep-2015].

[9] “Discrete Manufacturing Operations Management Software from ABB - ABB Industry software case studies and best practices (ABB Operations Management Software for Industries).” [Online]. Available: http://new.abb.com/cpm/industry-software/industry-specific-solutions/discrete-manufacturing. [Accessed: 05-Sep-2015].

[10] “Oracle E-Business Suite | Applications | Oracle.” [Online]. Available: http://www.oracle.com/us/products/applications/ebusiness/resources/index.html. [Accessed: 06-Sep-2015].

[11] “Proficy for Discrete Manufacturing | GE Automation.” [Online]. Available: http://www.geautomation.com/products/proficy-discrete-manufacturing. [Accessed: 06-Sep-2015].

[12] “Discrete Manufacturing - Process Flow Hierarchy - SAP Library.” [Online]. Available: http://help.sap.com/saphelp_46c/helpdata/en/8a/1a1ee54e4211d182be0000e829fbfe/content.htm. [Accessed: 06-Sep-2015].

[13] “Welcome to Rockwell Automation – Leader in Industrial Automation & Information.” [Online]. Available: http://www.rockwellautomation.com/global/overview.page. [Accessed: 06-Sep-2015].

[14] “What is knowledge? - theoryofknowledge.net.” [Online]. Available: http://www.theoryofknowledge.net/knowledge-and-knowers/what-is-knowledge/. [Accessed: 13-Nov-2015].

[15] Randall Davis, Howard Shrobe, and Peter Szolovits, “What Is a Knowledge Representation?,” *AI Magazine*, vol. 14, pp. 17–33, 1993.

[16] M. Saberi, A. Azadeh, Z. Saberi, and P. Pazhoheshfar, “A knowledge management system based on artificial intelligence (AI) methods: A flexible fuzzy regression-analysis of variance algorithm for natural gas consumption estimation,” in *2012 International Conference on Information Retrieval Knowledge Management (CAMP)*, 2012, pp. 143–147.

[17] M. Honma, H. Nakamura, A. Nakano, and S. Tsuruta, “Knowledge refinement approach through incorporating case-based knowledge in maintenance engineer scheduling AI system,” in *1999 IEEE International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC ’99 Conference Proceedings*, 1999, vol. 5, pp. 814–819 vol.5.

[18] B. Ramis, L. Gonzalez, S. Iarovyi, A. Lobov, J. L. Martinez Lastra, V. Vyatkin, and W. Dai, “Knowledge-based web service integration for industrial automation,” in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 733–739.

[19] J. L. M. Lastra, I. M. Delamer, and F. Ubis, *Domain Ontologies for Reasoning Machines in Factory Automation*. ISA, 2010.

[20] T. R. Gruber, “A translation approach to portable ontology specifications,” *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, Jun. 1993.

[21] “OWL 2 Web Ontology Language Primer (Second Edition).” [Online]. Available: http://www.w3.org/TR/owl2-primer/. [Accessed: 06-Sep-2015].
[22] S. Iarovyi, W. M. Mohammed, A. Lobov, B. R. Ferrer, and J. L. M. Lastra, “Cyber Physical Systems for Open-Knowledge-Driven Manufacturing Execution Systems,” *Proc. IEEE*, vol. PP, no. 99, pp. 1–13, 2016.
[23] Sergii Iarovyi and Xiangbin Xu, “Developing Open Knowledge-Driven Manufacturing Execution System,” in *OPEN KNOWLEDGE-DRIVEN MANUFACTURING & LOGISTICS, THE ESCOP APPROACH*, S. Strzelczak, P. Balda, M. Garetti, and A. Lobov, Eds. Warsaw University of Technology Publishing House, Warsaw 2015, pp. 295–310.
[24] 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.
[25] “Web Services Architecture.” [Online]. Available: http://www.w3.org/TR/ws-arch/. [Accessed: 06-Sep-2015].
[26] “SOA and Web Services.” [Online]. Available: http://www.oracle.com/technetwork/articles/javase/soa-142870.html. [Accessed: 06-Sep-2015].
[27] “IBM developerWorks : New to SOA and web services,” 05-Mar-2007. [Online]. Available: http://www.ibm.com/developerworks/webservices/newto/service.html. [Accessed: 06-Sep-2015].
[28] A. W. Colombo, F. Jammes, H. Smit, R. Harrison, J. L. M. Lastra, and I. M. Delamer, “Service-oriented architectures for collaborative automation,” in *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005*, 2005, p. 6 pp.–.
[29] “OASIS Devices Profile for Web Services (DPWS).” [Online]. Available: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01. [Accessed: 15-Nov-2015].
[30] A. Lobov, J. Puttonen, V. V. Herrera, R. Andiappan, and J. L. M. Lastra, “Service oriented architecture in developing of loosely-coupled manufacturing systems,” in *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008*, 2008, pp. 791–796.
[31] A. Lobov, F. U. Lopez, V. V. Herrera, J. Puttonen, and J. L. M. Lastra, “Semantic Web Services framework for manufacturing industries,” in *IEEE International Conference on Robotics and Biomimetics, 2008. ROBIO 2008*, 2009, pp. 2104–2108.
[32] “RESTful Web services: The basics,” 09-Feb-2015. [Online]. Available: http://www.ibm.com/developerworks/library/ws-restful/. [Accessed: 15-Nov-2015].
[33] Ondřej Severa and Roman Pišl, “REST-Enabled Physical Devices in Service Oriented Architecture,” in *Open Knowledge-Driven Manufacturing & Logistics - The eScop Approach*, S. Strzelczak, P. Balda, M. Garetti, and A. Lobov, Eds. Warsaw University of Technology Publishing House, Warsaw 2015, pp. 341–354.
[34] “Hydra Core Vocabulary.” [Online]. Available: http://www.hydra-cg.com/spec/latest/core/#introduction. [Accessed: 18-Apr-2016].
[35] A. Kuutti, A. Dvoryanchikova, A. Lobov, J. L. M. Lastra, and T. Vantera, “A device configuration management tool for context-aware system,” in *2012 10th IEEE International Conference on Industrial Informatics (INDIN)*, 2012, pp. 10–15.
[36] “Apache Jena - Fuseki: serving RDF data over HTTP.” [Online]. Available: http://jena.apache.org/documentation/serving_data/. [Accessed: 13-Sep-2015].
[37] K. Thramboulidis and A. Zoupas, “Real-time Java in control and automation: a model driven development approach,” in *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005*, 2005, vol. 1, p. 8 pp.–46.
[38] “Node.js.” [Online]. Available: https://nodejs.org/en/. [Accessed: 13-Sep-2015].
[39] “sparql-client.” [Online]. Available: https://www.npmjs.com/package/sparql-client. [Accessed: 28-Sep-2015].
[40] “HTTP/1.1: Status Code Definitions.” [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Accessed: 29-Sep-2015].
[41] “eval(),” *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval. [Accessed: 29-Sep-2015].
[42] “Inico Technologies.” [Online]. Available: http://www.inicotech.com/s1000_overview.html. [Accessed: 16-Sep-2015].
[43] Wael M. Mohammed, Andrei Lobov, Borja Ramis Ferrer, Sergii Iarovyi, and Jose L. Martinez Lastra, “A Web-Based Simulator for A Discrete Manufacturing System,” in *IECON 2016 - 42st Annual Conference of the IEEE Industrial Electronics Society*.

9 APPENDIX

9.1 FASTory RESTful

The following tables present the RESTful API of FASTory simulator. More description can be found in http://escop.rd.tut.fi/fastory/instructions

9.1.1 Services

#	Service ID	RTU ID	Method	Service URL	Service Body
1	LoadPaper	ROB1	POST	http://escop.rd.tut.fi:3000/RTU/ROB1/services/LoadPaper	{"destUrl":"http://hostname"}
2	UnloadPaper	ROB1	POST	http://escop.rd.tut.fi:3000/RTU/ROB1/services/UnloadPaper	{"destUrl":"http://hostname"}
3	LoadPallet	ROB7	POST	http://escop.rd.tut.fi:3000/RTU/ROB7/services/LoadPallet	{"destUrl":"http://hostname"}
4	UnloadPallet	ROB7	POST	http://escop.rd.tut.fi:3000/RTU/ROB7/services/UnloadPallet	{"destUrl":"http://hostname"}
5	ChangePenRED	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/services/ChangePenRED	{"destUrl":"http://hostname"}
6	ChangePen-GREEN	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/services/ChangePenGREEN	{"destUrl":"http://hostname"}
7	ChangePenBLUE	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/services/ChangePenBLUE	{"destUrl":"http://hostname"}
8	GetPenColor	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/services/GetPenColor	{"destUrl":"http://hostname"}
9	Draw**	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/services/Draw**	{"destUrl":"http://hostname"}
10	TransZone12	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/TransZone12	{"destUrl":"http://hostname"}
11	TransZone23	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/TransZone23	{"destUrl":"http://hostname"}

12	TransZone35	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/TransZone35	{"destUrl":"http://hostname"}
13	TransZone14	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/TransZone14	{"destUrl":"http://hostname"}
14	TransZone45	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/TransZone45	{"destUrl":"http://hostname"}
15	Z1	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/Z1	{}
16	Z2	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/Z2	{}
17	Z3	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/Z3	{}
18	Z4	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/Z4	{}
19	Z5	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/services/Z5	{}
20	Reset	N/A	POST	http://escop.rd.tut.fi:3000/RTU/reset	{}

* can be replaced with 2, 3, 4, 5, 6, 8, 9, 10, 11 or 12
** can be replaced with 1, 2, 3, 4, 5, 6, 7, 8, or 9

9.1.2 Events

#	Event ID	RTU ID	Method	Event URL	Event Body
1	PaperLoaded	ROB 1	POST	http://escop.rd.tut.fi:3000/RTU/ROB1/events/PaperLoaded/notifs	{"destUrl":"http://hostname"}
2	PaperUnloaded	ROB 1	POST	http://escop.rd.tut.fi:3000/RTU/ROB1/events/PaperUnloaded/notifs	{"destUrl":"http://hostname"}
3	PalletLoaded	ROB 7	POST	http://escop.rd.tut.fi:3000/RTU/ROB7/events/PalletLoaded/notifs	{"destUrl":"http://hostname"}
4	PalletUnloaded	ROB 7	POST	http://escop.rd.tut.fi:3000/RTU/ROB7/events/PalletUnloaded/notifs	{"destUrl":"http://hostname"}
5	PenChanged	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/events/PenChanged/notifs	{"destUrl":"http://hostname"}

6	DrawStartExecution	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/events/DrawStartExecution/notifs	{"destUrl":"http://hostname"}
7	DrawEndExecution	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/events/DrawEndExecution/notifs	{"destUrl":"http://hostname"}
8	LowInkLevel	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/events/LowInkLevel/notifs	{"destUrl":"http://hostname"}
9	OutOfInk	ROB*	POST	http://escop.rd.tut.fi:3000/RTU/ROB*/events/OutOfInk/notifs	{"destUrl":"http://hostname"}
10	Z1_Changed	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/events/Z1_Changed/notifs	{"destUrl":"http://hostname"}
11	Z2_Changed	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/events/Z2_Changed/notifs	{"destUrl":"http://hostname"}
12	Z3_Changed	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/events/Z3_Changed/notifs	{"destUrl":"http://hostname"}
13	Z4_Changed	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/events/Z4_Changed/notifs	{"destUrl":"http://hostname"}
14	Z5_Changed	CNV*	POST	http://escop.rd.tut.fi:3000/RTU/CNV*/events/Z5_Changed/notifs	{"destUrl":"http://hostname"}

* could be replaced with 2, 3, 4, 5, 6, 8, 9, 10, 11 or 12

9.1.3 Data

#	Data ID	RTU ID	Method	Data URL	Response Body
1	S1	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/S1	{"v":" ", "q":" ", "t": " "}
2	S2	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/S2	{"v":" ", "q":" ", "t": " "}
3	S3	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/S3	{"v":" ", "q":" ", "t": " "}
4	S4	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/S4	{"v":" ", "q":" ", "t": " "}

5	P1	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/P1	{"v":" ", "q":" ", "t": " "}
6	P2	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/P2	{"v":" ", "q":" ", "t": " "}
7	P3	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/P3	{"v":" ", "q":" ", "t": " "}
8	P4	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/P4	{"v":" ", "q":" ", "t": " "}
9	RFID	CNV*	GET	http://escop.rd.tut.fi:3000/RTU/CNV*/data/RFID	{"v":" ", "q":" ", "t": " "}

* could be replaced with 2, 3, 4, 5, 6, 8, 9, 10, 11 or 12

9.2 ORL RESTful API

- **Registration in Device Catalogue**

ORL registers in Device Catalogue to allow other willing parties to subscribe for its events. The registration is done at the following URL:

http://<RPL_HOST>/RPL/RTU

- **Registration for appropriate events in Device Catalogue**

Before marking as ready to start, the FASTory process definition is scanned by ORL for all needed events. ORL will ask for event subscription URL at the following URL in RPL:

http://<RPL_HOST>/RPL/RTU/events/<device id>/<event_name>

- **Querying for operations URLs in Device Catalogue**

Every time when ORL needs to perform certain operation on device (eg. TransZoneXY, DrawX), the Device Catalogue is asked for the URL of given operation on given device. The query goes as follows:

http://<RPL_HOST>/RPL/RTU/discovered-operations/<device id>/<operation id>

- **Querying for the next operation drawing operation (or NOP) on workstation's robot**:

To decide whether certain workstation should by bypassed or if there are any operations to perform, the ORL queries the Dispatcher:

http://<RPL_HOST>/dispatcher/next-operation?id=542314&WS=WS5

In return a JSON object with ***operation*** field is returned. If the field contains some value ORL will send the palette to the robot stand and perform this operation and ask for next one, until empty operation is returned. If empty operation is returned before entering robot stand, the palette takes bypass through zone 4.

9.3 RPL RESTful API

Device catalogue, being populated using approach described in the previous section provides access to the events and services URLs based on their identifiers and identifiers of hosting devices. The following services are implemented in device catalogue:

- ***GET: /RPL/RTU* –** Gets a list of registered devices.
- ***POST: /RPL/RTU* –** Inserts a new device in DC. Requires a Device Hello message as a body.
- ***DELETE: /RPL/RTU* –** Clears the list of registered devices
- ***GET: /RPL/RTU/{deviceID}* –** Gets a full description of a device with *"deviceID"* id.
- ***DELETE: /RPL/RTU/{deviceID}* –** Removes a description of device with *"deviceID"* id from DC
- ***GET: /RPL/RTU/{deviceID}/{type}/{opID}[?fields=links.self|all]* –** Gets information about event or service by id of host (*"deviceID"*) and id of event or service (*"opID"*). "*type*" may be "events" or "services" and identifies if it is event or a service information about which is requested. The amount of information provided may be controlled by a filter "*fields*". By default, this service provides only a self-link of the operation. If fields would be set to all will provide all available information about the operation. This service is used to request proper URL for event or service based on the knowledge which type of operation is required in which workstation of FASTory line.
- ***GET: /discovered-devices/{deviceId}* –** Gets a link to device description by device ID
- ***GET: /discovered-operations/{deviceId}/{operationId}* –** Gets a URL of the operation with id *"operationId"* hosted on device with id *"deviceId"*. If not exists returns 404.
- ***GET: /events* –** Gets a list of URLs of all discovered events
- ***GET: /events/{deviceId}/{eventId}* –** Gets the URL of an event by its id and id of the device hosting it.
- ***PUT or DELETE: /order*** – a representation for all available orders
- ***GET, PUT, POST or DELETE: /order/{oId*}** – a representation for a single order
- ***PUT or DELETE: /order/{oId}/pallet*** same as **/pallet** – a representation for all assigned pallet
- ***GET, PUT, POST or DELETE: /order/{oId}/pallet/{pid}*** same as **/pallet/{pid}** – a representation for a single pallet
- ***GET: /order/{oId}/recipe*** – a representation of used recipe