TAMPERE UNIVERSITY OF TECHNOLOGY

BIN ZHANG

DESIGN AND IMPLEMENTATION OF AN INFOSTORE FOR KEY PERFORMANCE INDICATORS

Master of Science Thesis

The topic was approved by the Faculty Council of the Faculty of Automation, Mechanical and Materials Engineering on 5 September 2012.

Examiner: Professor Jose L. Martinez Lastra

PREFACE

The prerequisite for the improvement of efficiency and energy use in industrial facilities is the realization of asset-awareness. The purpose of the thesis work is to design and implement a middleware to realize asset-awareness of manufacturing systems by gathering raw information from the a factory automation testbed based on Service Oriented Architecture, processing the information into Key Performance Indicators, recording both the raw information and indicators, displaying and exposing them on web. The thesis work was carried out in the Factory Automation Systems and Technologies Laboratory of the Department of Production Engineering in Tampere University of Technology. The funding comes from the project eSONIA: Embedded Service Oriented Monitoring, Diagnostics and Control: Towards the Asset-Aware and Self-Recovery Factory.

The work begins with a review on different open source toolkits and frameworks used for the implementation of web services and web applications. In order to investigate the suitability of the toolkits and frameworks, the author tested several of them by coding sample projects. The second phase of the work focused on the theory and previous researches concerning Key Performance Indicators in the domain of factory automation, to seek a suitable model and design a set of appropriate indicators for the testbed. Then in order to extract the indicators, studies on the available information from the testbed and Complex Event Processing technologies were performed, after which the implementation of the web services, data processing and visualization were carried out. Finally comes the fine-tuning of the application.

Next, I would like to express my gratitude to people who have provided me helps and supports during my work.

First of all, I would like to thank my professor Jose L. Martinez Lastra who provided me such a great opportunity to work on such a challenging topic in an international environment.

Then, I am truly grateful to my supervisor Dr. Corina Postelnicu who provided with guidance on my work, on the thesis writing and publication of academic papers.

I also would like to show my gratitude to Axel who gave me advices and guidance on the technologies used in this work, to Andrei and Jani who also gave me advices.

Best wishes to all my colleagues and friends, thank you for your accompany and the joys you brought to me.

Last but the most, I would like to appreciate my parents' supports during the last several years.

Tampere, August 19th, 2012
Bin Zhang

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Machine Automation
ZHANG, BIN: Design and Implementation of an InfoStore for Key Performance Indicators
Master of Science Thesis, 85 pages, 13 Appendix pages
October 2012
Major subject: Factory Automation
Examiner: Prof. Jose L. Martinez Lastra
Supervisor: Dr. Corina Postelnicu
Keywords: Key Performance Indicator, Service Oriented Architecture, Complex Event Processing, SOAP, REST, web application

The efficiency of manufacturing systems becomes more and more significant in today's factories due to the increasingly competitive market. As the rise of energy price, energy saving also becomes vital. The first step to increase efficiency and decrease energy use is to achieve real-time monitoring of the systems at the shop floor. Then the optimization of efficiency and energy use becomes possible. The use of modern information technologies is essential considering the large amount of information generated by the low level facilities.

This thesis work presents the selection of a set of Key Performance Indicators (KPI) to improve the awareness of different manufacturing assets including those related to energy and efficiency. The implementation relies on Service Oriented Architecture deployed by web services and the further processing of the generated events by the application of an Event Processing Language based on rules. The processed data are displayed on web as graphics and updated in real time.

On the other hand, in order to fully exploit the potential of optimization algorithms for efficiency and energy savings in factory automation settings, it is needed to bring together engineers that have knowledge of signal processing algorithms with shop floor engineers that are experienced with real manufacturing processes and factory automation settings. This thesis work exposes the captured raw data and KPIs as a web service so that third party applications can acquire the data via URLs for their own use. The InfoStore is currently populated with data regarding equipment IPC-2541 state events, process/cell/production energy consumption, process/pallet production time, etc regarding a multi robot production line located at the premises of the Factory Automation Systems and Technology laboratory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AOP	Aspect Oriented Programming
API	Application Programming Interface
CEO	Chief Executive Officer
CEP	Complex Event Processing
CRUD	Create, Retrieve, Update, Delete
DAO	Data Access Object
DI	Dependency Injection
DPWS	Device Profile for Web Services
EI	Energy Intensity
EL	Expression Language
EPL	Event Processing Language
ERP	Enterprise Resource Planning
GWP	Global Warming Potential
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoC	Inversion of Control
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
Java EE	Java Enterprise Edition
JDBC	Java Database Connectivity
JMEDS	Java Multi Edition DPWS Stack
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JSTL	JSP Standard Tag Library
KPI	Key Performance Indicator
KRI	Key Result Indicator
MEEP	Measure of Energy Efficiency Performance
MIME	Multipurpose Internet Mail Extensions
MTBF	Mean Time Between Failures
MTTF	Mean Time-To-Failure
MTTR	Mean Time-To-Repair
MVC	Model View Controller
OEE	Overall Equipment Effectiveness
ORM	Object to Relational data Mapping
OXM	Object/XML Mapping
PBT	Persistent, Bio-accumulative and Toxic
PI	Performance Indicator
PLC	Programmable Logic Controller
PMS	Performance Measurement Systems
POJO	Plain Old Java Object
RDF	Resource Description Framework

REST	REpresentational State Transfer
SEC	Specific Energy Consumption
SOA	Service Oriented Architecture
SOA4D	Service Oriented Architecture for Devices
SOAP	Simple Object Access Protocol
SQL	Sequential Query Language
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VTL	Velocity Template Language
WS	Web service
WS4D	Web Services for Devices
WSDAPI	Web Service on Device API
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

1. INTRODUCTION

This chapter introduces the background and problem definition of the thesis work, following the work description. Finally, the outline of the thesis is presented.

1.1. Background

The increasing competiveness in global market nowadays has led to a demand for companies to manage their business more efficiently [1] [2]. Real time monitoring and evaluation of the current states of the key aspects of an enterprise and effective and rapid decision making processes are essential to fulfil today's requirements of flexible production, increased production efficiency, rapid response to customer demands, and high and uniform quality of products and services [2].

On the other hand, traditional indicators (for example, return on investment, market share and rate of defect products) only for financial and quality tracking [3] cannot meet the requirement of a company in the 21st century. The United Nations Conference on Environment and Development in 1992 concluded that sustainable production is the solution for the continued deterioration of the global environment [4]. In addition, as a number of studies, such as [5], indicating the proportional relationship between good environmental and social performance and profits, more and more companies realize the fact that the improvement of the performance in sustainability can establish competitive advantage for a company [3].

Modern manufacturing systems are complex and distributed. A system may be composed of thousands of components and devices or even more. The amount of collectable data for decision making personnel to manage is enormous. Figure 1 illustrates the data flow in the hierarchical pyramid of an enterprise. The data flows among three levels from the process level consisting of a large amount of raw process data to the business level where managers analyse the data and make decisions. [1] reveals that managers are overwhelmed when facing the vast amount of information for rapid and correct decision making. The problem of extracting useful information from the substantial amount of data emerged.

As to the optimization of the efficiency and energy savings, the goals set by the European Council in March 2007 [6] (reduction of 20% of the total energy consumption; 20% contribution of renewable energies to total energy production; 20% reduction of greenhouse gases below 1990 emissions) impose a long term shift from a cost-based competitive advantage to one based on high added value (producing more products with less material, less energy and less waste). Needed energy savings are envisioned to be achieved via process / product / machine tool design and cross-layer

optimization algorithms working with energy and efficiency relevant data gathered from all levels of the enterprise, from shop floor to ERP [7].

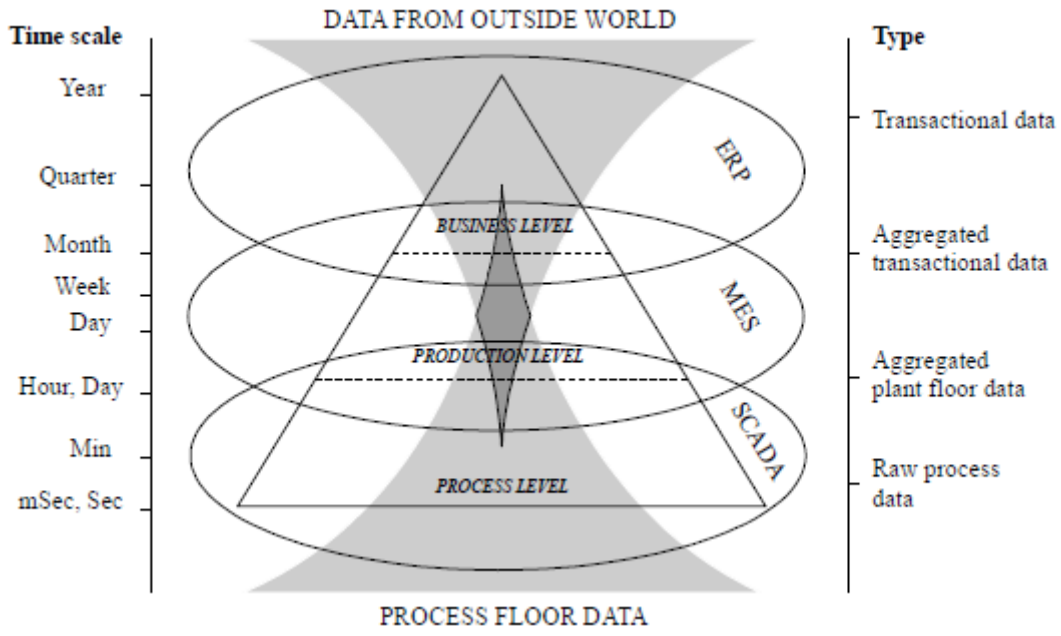


***Figure 1**: Hierarchical structure of an enterprise* [2]

Generally, at design phase, signal processing and optimization algorithm developers work with large datasets tailored for each application domain of interest. Unlike in other fields (e.g. health [8]), in the manufacturing domain, there is a profound lack of such available datasets. In order to develop algorithms for optimization with respect to energy or energy-relevant predictive maintenance, there is a strong need for databases storing real data from various types of factory automation test beds. The data collected must reflect energy consumption of manufacturing processes, products, workstations, transportation devices, routing options, together with production-relevant indicators (machine utilization, timestamps associated to machine breakdowns and idle time) and business relevant indicators (e.g. energy prices during certain time periods).

These data should be made available on the web for all interested parties to access upon request. This would bridge the skills of those that have the know-how on what can be done with the data, with the knowledge of those that have information on how this data can be gathered.

1.2. Problem definition

As is stated by Peter Drucker, the industrial revolutionary, “You cannot manage something you cannot control and you cannot control something you cannot measure” [9], a management team in a company needs to compare the measured figures with the goals and objectives they established and with the figures from other companies in the same sector for decision making [3]. Performance measurement systems (PMS) have emerged to facilitate production managers to improve the performance of manufacturing

plants, such as reliability and productivity. However, it is suggested in [9] that the reason of the collapse of many plants over the last 30 years is the incorrect utilities of the performance measurements. The performance measurements by these companies are all results of previous states. It is impossible to react if failures already occur. There needs to be a set of leading indicators which reflects the current states of the plants.

The solution for successful production management lies in the design of key performance indicators (KPIs) [2] which are used to measure and evaluate the critical aspects of a manufacturing system, a company or any related processes. In other words, they are figures which present the evaluation results of the current states in the enterprise, which are the most concerned. These indicators should be measured frequently and on process basis for rapid reaction on the current states.

On the perspective of sustainability, when designing KPIs, sustainability performance should also be considered, to align with the goals set by the European Council in March 2007 [6]. Asset awareness must refer to energy consumption in addition to the traditionally considered aspects, to be able to manage this consumption and even optimize it wherever possible. The KPI for sustainability performance should highlight six main aspects in sustainable production: energy and material use, natural environment, social justice and community development, economic performance, workers and products as is stated in [4].

For the vast amount of data from the lower levels in an enterprise, the production process can only be managed successfully with the assistant of appropriate information technology, which enables machine-to-machine communication. Furthermore, raw data need to be aggregated and correlated at run-time as well to present meaningful information.

Lastly, the meaningful information needs to be presented to managers so that they can visualize and manage the data for decision-making. The information needs also to be exposed on web for any interested party for visualization or optimization algorithm development. All these statements prompt the following questions.

What are the key performances that are the most concerned for manufacturing systems? How these performances are measured in the domain of factory automation? How the machine-to-machine communication is achieved? How the run time management of vast amount of information is achieved? What techniques can be used to represent the information properly? How the information can be accessed by remote parties?

1.3. Work description

This part outlines the objectives of this work in order to solve questions above. It also describes the methodology that is used to achieve these objectives.

1.3.1. Objectives

1. Communicate with a service-enabled manufacturing system.

2. Design a set of KPIs for discrete manufacturing systems.
3. Run-time management of KPIs.
4. Design proper layout for the KPIs visualization.
5. Implementation of a KPI management framework.
6. Expose the information to other applications.

1.3.2. Methodology

The thesis work is composed of two main stages in order to achieve the objectives. The first is the research stage, in which the author investigates the various studies on KPIs methodologies, frameworks and tools in a variety of fields and extracts proper KPIs in the domain of factory automation. The author also reviews the both back-end and front-end tools and technologies for building web services and web application at this stage. In the second stage, the author first develops the overall architecture for KPI management system. Then he designs the layout for the visualization of the KPIs. Finally the author carries out the implementation.

1.4. Thesis outline

Following this chapter, background knowledge of KPIs and reviews on the concepts and frameworks that are used to enable machine-to-machine communication, complex event processing and development of web applications are presented. In Chapter 3, the test bed used in this thesis work and the design of KPIs are presented. The implementation of machine-to-machine communication, visualization of KPIs and publication of KPIs for interested parties are demonstrated and described in details. The results are shown and discussed in Chapter 4 followed by conclusions in Chapter 5.

2. LITERATURE AND TECHNOLOGY REVIEW

As is stated in problem definition, there is a demand for a set of indicators to reveal the current state of manufacturing systems. The solution for successful production management lies in the design of KPIs. This chapter starts with the explanation of KPIs followed by a series of reviews on the methodologies and frameworks that are developed for the design of KPIs for production processes.

Concerning the vast amount of information from the lower levels in an enterprise, the production processes can only be managed successfully with the assistance of appropriate information technology. In order to gather the necessary information from the lower levels, select useful data and arrange these data into KPIs in real time, information technologies, such as Service Oriented Architecture (SOA) and Complex Event Processing (CEP), are needed. SOA provides the capability of retrieving information pertaining status of manufacturing processes in real time, while CEP is a set of technologies processing and integrating events. Thurs CEP is a viable tool for KPI calculation. The second part of this chapter introduces the technologies that enable machine-to-machine communication and complex event processing.

Another requirement for the monitoring and evaluation of the performance is the visualization. With the advancement of internet, web applications have obtained much interest. In comparison with the traditional desktop applications which are installed separately on single computers, a web application does not need to be installed and can be accessed through any location as long as it is connected to the internet. This advantage makes web applications become popular. Furthermore, with the emerge of open source web application development tools, such as Apache Struts 2 and Spring, developing web applications is not an expensive choice. These tools also enable rapid implementation of web applications. The last part of this chapter concentrates on the introduction of web frameworks, presentation technologies and data persistence technologies for the development of web application.

2.1. Key performance indicators

The definition of KPIs appears on many articles. [1] and [10] define KPIs as “a variable that quantitatively expresses the effectiveness or efficiency, or both, of a part of or a whole process, or system, against a given norm or target”. The definition of KPIs in [11] is referred as a more sophisticated one by [1] which is: “A performance indicator defines the measurement of a piece of important and useful information about the performance of a program expressed as a percentage, index, rate or other comparison which is monitored at regular intervals and is compared to one or more criterion”. [12]

states that "KPIs represent a set of measures focusing on those aspects of organizational performance that are the most critical for the current and future success of the organization". [12] also explored how to distinguish KPIs from similar terms in performance measures- key result indicators (KRIs) and performance indicators (PIs), in which, KRIs are the results of many actions, cover a longer period of time than KPIs, and do not indicate how to improve the results, while, on the other side, PIs lie between KRIs and KPIs which indicate what to do to improve the performance. As is indicated, the difference between KPIs and PIs is the level of importance for the organization, the manufacturing system, etc.

2.1.1. Properties and characteristics of KPIs

Four key properties, listed as follow, need to be considered when KPIs are determined as is mentioned in [2] and [4]:

1. Unit of measurement – the metric in calculating an indicator, for example, watts, numbers, litres, etc.
2. Type of measurement – absolute or adjusted, for example, total amount (of energy consumption per week) or adjust amount (energy used per unit of product per week).
3. Period of measurement – period for calculating an indicator (24/7, daily, weekly).
4. Boundaries – determines how far a company wishes to go for the measurement of an indicator, for example, a production line, entire life cycle or a product.

In addition, KPIs should follow the seven characteristics [12]:

1. Nonfinancial measures (not expressed in monetary unit)
2. Frequently measured (24/7, daily or weekly)
3. Acted on by the CEO and senior management team
4. Clearly indicate what action is required by staff (Staff can understand the measures and know what action can be taken)
5. Measures that tie responsibility down to a team
6. Have a significant impact
7. They encourage appropriate action

2.1.2. Related work

During the last two decades, many researches have been conducted on KPIs in order to improve the competitiveness and sustainability of companies and facilitate manufacturing processes in a wide range of fields. Five general KPIs (safety, efficiency, quality, production plan tracking and employees' issues) are proposed in [2] to enable the comparison with short term and medium term production strategy and goals in the area of production process management. An improved measure methodology based on the general KPIs are developed in [13]. A model, which uses KPIs as production feedback for polymerisation production processes, is developed and described in [1]. In

[14], production efficiency is used as a key performance indicator for energy efficiency evaluation in paper and pulp industry. [15] established an index model using several general and sector-specific KPIs for notebook manufacturing system. A framework for indicators of sustainable production focusing on the aspects of environment, health and safety has been developed in [16]. The framework is developed into five levels. The methodology based on the same framework with expanded indicators for developing and implementing indicators of sustainable production using core and supplementing indicators is presented in [4]. A case study showing the results of testing the methodology is presented in [3]. [17] analyzes the needs for production companies to integrate energy-aware KPIs such as energy intensity (EI) and specific energy consumption (SEC) into their manufacturing systems. The following sections briefly present these work.

2.1.3. General KPIs in production (case 1)

[2] has developed an 8-step iterative model for deriving KPIs from production processes, shown in Figure 2. When defining production goals and objectives in the first step, all key aspects of the organization should be considered. In the second step, it is recommended to use many indicators to reflect production goals and efficiency. Additional and production-specific indicators should be considered when selecting indicators for implementation in step 3. The purpose of setting targets is to ensure the continuous improvement of production processes. When achieving a target, new targets should be set. Step 5 is the most time consuming step including the implementation of all necessary functions for indicator representation. In step 6, periodical communication and evaluation of results are considered a necessity. In addition, it is necessary to establish a system to evaluate, interpret and present results regularly. Actions should be taken in the seventh step to improve the performance continuously. Lastly, new indicators should be introduced and unnecessary indicators should be eliminated.

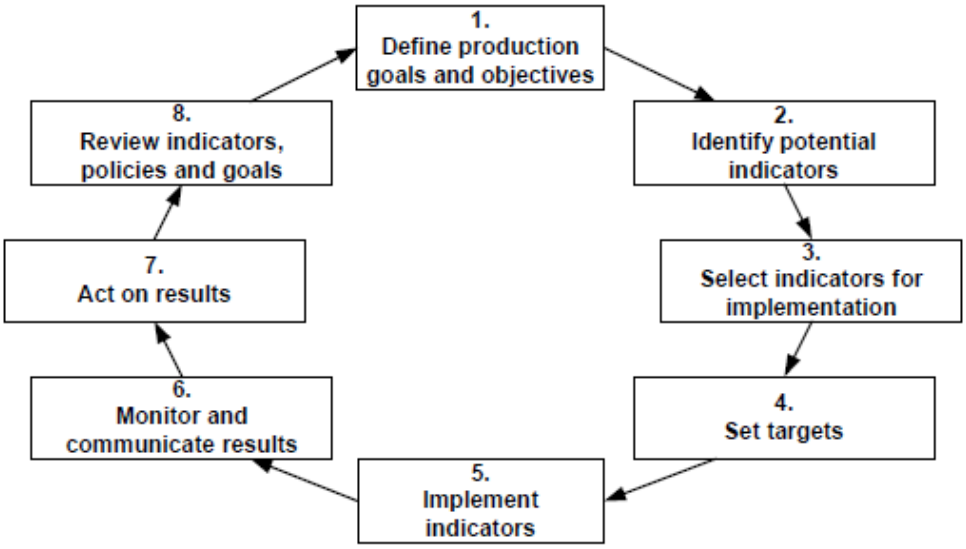


***Figure 2**: Steps for KPIs deriving for production processes* [2]

A set of general KPIs has been developed in [2], including Safety and environment, Efficiency, Quality, Production plan tracking and Employees' issues. Several possible theoretical indicators compose each KPI after normalization and weighting are collected in Table I.

***Table I**: General KPIs for production management*

KPIs	indicators
Safety and environment	Number of accidents at work
	Number of hazardous alarms
	Fresh water consumption
	Waste generated before recycling
	Number of penalties due to releasing waste in environment
Production Efficiency	Efficiency of employees in production
	Infrastructure efficiency
	Material used (total and per product)
	Energy used (total and per product)
	Unit product time
	Quality of internal and external services
	Production shutdowns
Quality	Percent of final products, which do not meet quality criteria
	Percent of raw material, which do not meet quality criteria
	Size of production losses
	Quality of internal and external services
Production plan tracking	Percent of production orders finished late
	Number of penalties
	Percent of production orders finished ahead
Employees' issues	Complete job satisfaction of employees
	Lost work due to injury and illness
	Average length of service of employees
	Employees' proposal for improvements and innovations

All KPIs are allocated into 3 levels according to their importance, in which Safety and environment belongs to level 1 (most important), Employees' issues are allocated in level 3 (least important) and other KPIs in level 2.

The study also reveals the plan to implement a commercially available production information system for run-time data acquisition and the plan to adopt a decision support module to extract relevant data from the low level information and present the KPIs. However, it does not present the result of the implementation of such real-time data acquisition system.

2.1.4. Improved manufacturing performance measures (case 2)

[13] proposed a new methodology for KPIs in which the performance of manufacturing systems is accessed in a qualitative way. The performance indicators selected for manufacturing systems are divided into 6 sections:

1. Safety & Environment
2. Flexibility
3. Innovation
4. Performance
5. Quality
6. Dependability

The research focuses on the KPI of the dependability in which indicators including customer complaints, on-time-in-full delivery to customers, on-time-in-full delivery

from suppliers and overall equipment effectiveness (OEE) are defined. The study collects data in tables and compared the data to the world-class performance. A gap between the actual performance and the world-class is identified as areas for enhancement.

An improved measurement methodology is proposed which is to measure the performance of the production vessels separately, since some of the vessels are in idle state while others are in operation model. Therefore, measuring the performance of the whole discrete production line does not provide the real results. The OEE and the three components of OEE for the five vessels are calculated and provided in line charts.

2.1.5. Production feedback control using production KPIs (case 3)

In order to improve the production performance, [1] has proposed a model of a production feedback control system for a polymerisation plant using production KPIs as reference variables. In order to verify the effectiveness of using production KPIs in the closed loop system, a set of simulation runs are performed on simulation tools such as Matlab, Simulink and Stateflow. As is illustrated in Figure 3, the KPIs that have been chosen specific to the production process are Productivity, Quality and Costs. Although these variables are not directly measurable, they are estimated from low level indicators that can be directly measured. The simulation shows the influence of process variables, such as quality of raw materials, production speed and production schedule, on the values of these KPIs.

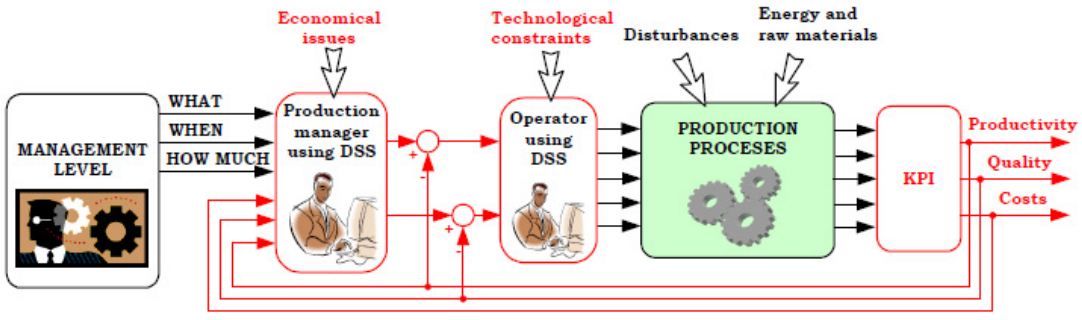


***Figure 3**: Closed-loop control system of production process* [1]

2.1.6. A complete index model for notebook manufacturing (case 4)

[15] has established a complete index model for notebook manufacturing management, in which the most significant key performance indicators in the plant level are defined. Some of the indicators are derived from previous studies in the aspects of productivity, quality, cost and profitability, and cycle time. In addition, the authors also designed new indicators that are specific to the notebook manufacturing lines based on their analysis and their visit to the notebook factories. The data are collected from notebook manufacturers by questionnaire and interviews. The study also indicates that not all the data they collected are acquired automatically by the manufacturers.

2.1.7. Production efficiency as a KPI of energy efficiency (case 5)

The overall production efficiency in paper and pulp production is defined as the time efficiency multiplied by the area efficiency. Sivill and Ahtila discussed the use of area efficiency and time efficiency as KPIs to evaluate energy efficiency in [14] since the decrease of area losses increases the energy efficiency and the increase of time efficiency has a positive effect on energy efficiency. Another fact is that energy efficiency affects the production cost. By increasing the energy efficiency especially during a period of over-capacity, production cost can be decreased. Thus maximising the difference between product price and production cost. Consequently, maximum profitability is achieved.

2.1.8. Indicators for sustainable production (case 6)

Sustainable production is defined by Lowell Centre for Sustainable Production as:"the creation of goods and services using processes and systems that are non-polluting; conserving of energy and natural resources; economically viable; safe and healthful for employees, communities and consumers; and socially and creatively rewarding for all working people". It stresses six aspects of sustainable production: Energy and material use, Natural environment, Social justice and community development, Economic performance, Workers and Products. [4]

Veleva and Ellenbecker demonstrated the trend for standardization of indicators and proposed a standard set of core indicators in these six aspects in [4] as is illustrated in Table II. These indicators aim to measure the issues that are considered common for all production plants.

***Table II**: Core indicators for sustainable production*

	indicators
Energy and material use	Fresh water consumption
	Material used (total and per unit product)
	Energy use (total and per unit product)
	Percent of energy from renewable
Natural environment	Kilograms of waste generated before recycling
	Global warming potential (GWP)
	Acidification potential
	Kilograms of persistent, bio-accumulative and toxic (PBT) chemicals used
Economic viability	EHS compliance costs
	Customer complaints and / or returns
	Organizational openness
Community development and social justice	Community spending and charitable contributions
	Number of employees per unit of product
	Number of community-company partnerships
Workers	Lost workday injury and illness rate
	Rate of employee suggested improvements
	Turnover rate or average length of service
	Average number of hours of employee training
	Percent of workers who report complete job satisfaction
Products	Percent of products designed for disassembly, reuse or recycling
	Percent of biodegradable packaging
	Percent of products with take-back policies

In addition, supplemental indicators should be defined which provide flexibility and production specific performances. Examples of supplemental indicators are provided in [4].

Results of implementing the sustainable production indicators at Acushnet Rubber are illustrated in [3]. The difficulties of using certain indicators are also discussed.

2.1.9. Review on Key Performance Indicators

Table III shows a summary of the previous work, in which some features of the studies are concluded.

***Table III**: Summary of studies on production KPIs*

	Run-time	Flexible	Energy-aware
Case 1: General KPIs for production	no	yes	yes
Case 2: Improved manufacturing performance measures	no	no	no
Case 3: Feedback control using production KPIs	yes	yes	no
Case 4: Index model for notebook manufacturing management	no	no	no
Case 5: Production efficiency as a KPI of energy efficiency	no	no	yes
Case 6: Indicators for sustainable production	no	yes	yes

Although a number of studies on KPI design, methodology and implementation have been carried out in many fields, they have their limits. As is illustrated in Table III, only case 3 collects data for KPIs calculation in real time, but it is in the phase of simulation with software tools. In order words, the study has not been carried out for real manufacturing systems. Case 1 proposed to adopt the general KPIs in a brickworks manufacturing plant by utilizing a commercially available production information system for real-time data acquisition and using a decision support module to extract relevant information from the real- time data. However, the implementation is still at the beginning stage. Case 4 indicates that not all the data collected by the notebook manufacturers are automatically acquired. It can be concluded that the current development of KPIs monitoring system in production lacks run-time data acquisition and extraction of useful data from manufacturing systems.

Secondly, case 1 presents a practical model for KPIs development and general KPIs for production are defined as well. It can only give possible theoretical indicators under each defined KPI. Case 4 utilizes the KPIs from previous work which is common for all sectors, but it still needs to define specific KPIs for the notebook manufacturing. The need of supplementary indicators in case 6 for sustainable production also indicates the fact that although standardization of indicators for production processes is a trend, defining specific indicators according to different manufacturing systems is also necessary because the key performance that needs to be monitored varies from one sector to another.

Last but not least, many researchers include energy-aware KPIs in their studies. Some of them, for example in case 5, aware that energy conservation benefits profitability, while others, for example in case 6, urge the deterioration of environment problems is caused by unsustainable production. By monitoring and evaluating

sustainable indicators, such as energy and resource consumption, it is possible to alleviate environment problems.

All in all, the current KPIs development lacks the run-time data acquisition and extraction. Specific KPIs need to be defined for different manufacturing systems although common indicators can be used in all fields. Because of the urgency of environmental problems, including sustainable indicators is also a must.

2.2. Service oriented architecture

Service Oriented Architecture is a paradigm utilizing autonomous and platform-independent services over the web for distributed applications development. The services can be described, published, discovered and dynamically assembled. The most popular SOA-based technology is web service, which makes use of W3C XML based standards: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI). [18]

A web service involves a service provider, which is capable of publishing events, providing operations and WSDL, and a client which subscribes to the service for event receiving, operation manipulation and WSDL parsing. The service needs to be discovered first, and then subscribed by the client.

Web services are located on resource-constraint devices using Device Profile for Web Services (DPWS) in order that machine-to-machine communication becomes possible [19]. Figure 4 illustrates the underlying protocols of DPWS as a protocol stack.

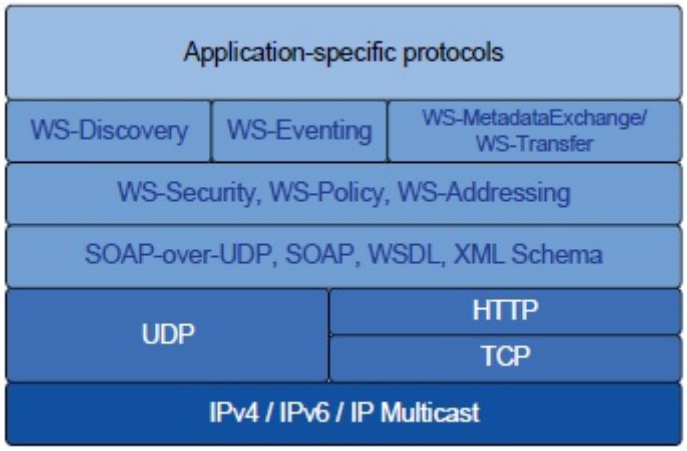


***Figure 4**: Device Profile for Web Services as protocol stack* [19]

2.2.1. Implementation toolkit- WS4D

DPWS can be implemented via a series of different toolkits (e.g. Web Services for Devices (WS4D), Service Oriented Architecture for Devices (SOA4D) and WSDAPI). Each was developed with certain specific use cases in mind.

WS4D stacks were developed by the members of the SIRENA project [20]. Open source stacks developed by the WS4D community are WS4D-JMEDS (Java Multi Edition DPWS Stack), WS4D-gSOAP and WS4D-uDPWS [20].

WS4D- JMEDS with JMEDS representing Java Multi Edition DPWS Stack is a framework for the implementation of web services, including components- DPWS devices, services and clients, based on DPWS specification using Java aiming for embedded devices with low capacity in memory. [20] The stack implements WS specifications including WS-Addressing, WS-Discovery, WS-Eventing, WS-MetadataExchange/Transfer and WS-Security. Both client and service sides consist of three layers: Communication, Dispatching and Application. A communication manager in the Communication layer ensures the message exchange with the manager on the other side. The manager can transport SOAP message via UDP or HTTP over TCP. Attachment exchange over HTTP is achieved using MIME. It supports both IPv4 and IPv6 connections. The Dispatching layer on client side fulfils the WS-Discovery specification using a *SearchHandler* to discover devices and services. A *Dispatcher* is in charge of sending messages, such as *probe* and *probeMatch* messages. The subscription of services is handled by a *SearchManager* on the service side to manage the event subscribers. Device/Service Registry is used to store discovered devices and services. Instead of invoking a new search, the client can reference the known devices or services via the registry. No changes are needed for Communication and Dispatching layers when using the stack. However, developers need to conduct actual implementation for the Application layer for clients and devices. On the client side, references of devices, services and events from a discovered device are allocated. An *EventSink* is used to receive events. On the side of device, a device can contain many services, while many operations and events can be defined in a service. The *Binding* is used to correlate services to devices. [21]

2.2.2. Spring Web Services

Spring Web Services (Spring WS) is an open source framework aiming to facilitate contract-first SOAP service development. In a contract-first approach, the implementation of the web services is based on a WSDL contract that is created first. [22]

The mapping of incoming XML to handlers in Spring WS is achieved via the endpoint mapping. The *MessageDispatcherSevlet*, which acts as the entry of the framework, receives XML requests, forwards them to the endpoint mapping and invokes corresponding endpoints and interceptors according to the mapping. The *@Endpoint* annotation before the declaration of a class allows the handling of multiple requests in one endpoint class. This can be achieved via the method endpoint mapping using payload root mapping or SOAP action mapping. This mapping routes incoming XML requests to corresponding methods according to the annotated name space and root element if payload root mapping is used or SOAP action if SOAP action mapping is used. Then the XML requests can be parsed using any supported technologies such as standard JAXP APIs (DOM, SAX, and StAX), JDOM, dom4j, XOM as well as marshalling techniques (JAXB 1 and 2, Castor, XMLBeans, JiBX, and

XStream) [22]. A response is generated after the manipulation of requests according to the return value of the handler and serialized into XML format.

A web service implemented with Spring WS focuses on document-driven web services. It generates responses based on the incoming requests. On the other hand, WS4D toolkit requires the discovery and subscription to web services, and then receiving messages until the subscription ends.

2.3. RESTful web services

REST representing Representational State Transfer [23] is an architectural style aiming to release the server from the burden of maintaining complex sessions with clients [24]. In REST, a server holds a variety of resources, each identified by a URI. A client accesses these resources by sending the corresponding HTTP request in the form of such URI and receiving responses. [24] Unlike conventional web techniques (human-friendly HTML format server responses), a RESTful response is represented in XML, JSON or RDF which is consumable by third party applications [25]. HTTP GET requests result in resource status responses, while POST requests is used to alter the status of resources. Besides, other types of HTTP requests such as HEAD (to query the existence of resources), PUT (to update and re-compute) and DELET (to destroy) are also supported [25].

2.3.1. Implementation framework- Spring MVC

Spring is an open source framework to facilitate the development of Java applications by supporting a comprehensive infrastructure. It provides a powerful Model View Controller (MVC) framework for application development. Besides, modules including Inversion of Control (IoC) container, Data access and integration, Web, Aspect Oriented Programming (AOP), Instrumentation and Test are also provided. The framework is organized in modules, thus enabling developers to use the functionalities they need [26]. RESTful web services can be built on top of the Spring MVC framework to deal with HTTP requests and responses. In a RESTful web service, a request is received by using Spring MVC's request mapping mechanism. A response is then generated accordingly represented with a Java Object. Subsequently, it is serialized to XML format using any of the supported XML marshallers (Jaxb2Marshaller, CastorMarshaller, JibxMarshaller, XmlBeansMarshaller and XStreamMarshaller [27]) by Spring's Object/XML Mapping (OXM) module. Besides, responses can also be transformed to other formats including JSON, by configuring Spring's message converter properly.

As opposed to the WS4D-JMEDS toolkit, which uses the SOAP mechanism to discover and subscribe to web services, thus receiving messages until the subscription ends, a RESTful web service implemented with Spring MVC framework enables a third party applications (client) to send HTTP request for resources and the service responds with XML, JSON or RDF format streams reactively.

2.4. Complex event processing

Complex Event Processing (CEP) is defined as a set of tools and techniques for analyzing and handling series of related events in real time in distributed information systems [28]. CEP provides functionalities such as filter, project, join and aggregation of event streams and focuses more on sequential data correlation and complex pattern detection of events [29]. The functionality is realized when events are recognizable to the CEP engine and by defining certain rules. When a series of incoming events matches the pattern defined in the rules, the result is outputted immediately by the CEP engine to the user for further manipulation [29]. The use of CEP enables the downstream applications to be driven by the upstream distributed information systems in real time [30].

2.4.1. Esper

Esper is a java based component developed for rapid development of applications which require to process large volumes of events in real time. Unlike a database which stores data in it and manipulate the data according to the incoming queries, an Esper engine works in an opposite way. It stores the queries in the engine and responds according to the incoming events. Then applications could further manipulate the data which are contained in the response. [31]

In order for the Esper to recognize the events passing through the engine, the events can be represented as JavaBean classes, legacy Java classes, XML document or java.util.Map classes, and these events need to be registered in the engine. The XML represented events used in this thesis work can be registered with event type name and the root element names or a XSD schema document if it is provided. [31]

The queries used in Esper follow Event Processing Language (EPL) syntax, which also need to be stored in the Esper engine. EPL resembles Sequential Query Language (SQL) in the use of select clause. However, instead of using tables, EPL utilizes event streams for data selection.

Figure 5 illustrates the basic setup of an Esper engine to aggregate events. The events can be represented as XML streams or POJOs, etc. They need to be registered in the engine, for example, by their names, so that the engine knows what events are flowing to it. Afterwards, an EPL rule should be defined which guide the engine to aggregate the incoming events. The EPL rule is also associated with a *Listener* class when configured in the engine. As the incoming events flowing into the engine, it detects the events against the EPL rule. Once a match on the rule is detected, the engine populates real-time values, which are selected in the EPL rule, to an object (for example, a *Map* object) and invokes the *update* method in the listener. Thurs the parameters can be retrieved upon the invocation of the *update* method.

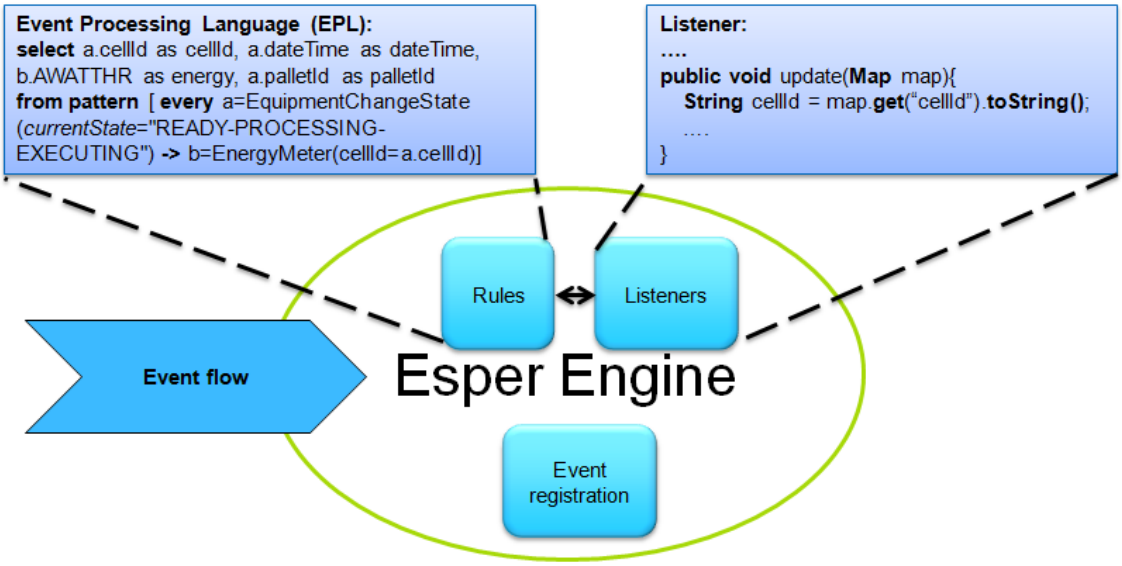


***Figure 5**: Esper engine setups*

2.5. Web application

A web application, as is indicated by its name, is an application that operates on web. Technologies, such as Internet speed, connectivity and client/server, are improving rapidly over the past few years. As a result, the web has become an increasingly popular platform for the building of standard business-oriented enterprise solutions, personal software and a variety of other applications. [32] One benefit of deploying applications on web is that the applications are stored on servers, so the users do not need to install any specific software on their own computers as long as they have a web browser, such as IE, Safari and Firefox, which comes with the operating system or can be installed without effort. Once applications on the server are upgraded, all users of the applications can access the new version by doing nothing. A web application is mainly built on two technologies- Hypertext Transfer Protocol (HTTP) and Java Servlet API [32].

HTTP protocol is a stateless series message exchanges between client and server. The web browser on the client side initiates the message exchange by requesting a static HTML document or dynamically generated document from a web application deployed on a server. However, for the development of web applications, one of the drawbacks is that HTTP is unaware of the relationship among requests. It handles requests separately even from the same client. The only usage of the address in the request is for the return of HTML documents. The other major disadvantage is that HTTP is text based. The text needs to be converted into Java data types, which must occur in both requesting and HTML returning processes. [32]

Java Servlet API is a technology used to expose HTTP to Java platform, which, to some extent, resolves the problems mentioned above. A servlet is a Java object for receiving request objects, processing the data in the request objects and returning response objects including response headers and output streams. The output stream

generates the information in text format. It also provides high-level functions such as a session mechanism, which correlates series of requests from a single client. [32]

2.5.1. Review on architectural patterns

In order to facilitate the building of applications, the concept of architectural patterns has been established. As is defined in [33] and [34], "An architectural pattern is expressing a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them". There exist a substantial numbers of architectural patterns in the field of software engineering. [35] classifies these patterns into eight categories according to architectural views. The classification of architectural patterns is summarized in Table IV. The architectural patterns of Layers in Layered view, Model-View-Controller in User interface view and Client-Server in Component interaction view and their utilities in the field of factory automation are introduced in the following parts.

***Table IV**: A summary of architectural patterns classified according to views*

View	Architectural pattern example
Layered view	Layers, Indirection layer
Data flow view	Batch sequential, Pipes and filters
Data-centred view	Shared repository, Active repository, Blackboard
Adaption view	Microkernel, Reflection, Interceptor
Language extension view	Interpreter, Virtual machine, Rule-based system
User interface view	Model-View-Controller, Presentation-Abstraction-Control
Component interaction view	Client-Server, Peer-To-Peer, Publish-Subscribe
Distribution view	Broker, Remote procedure calls

Layers architecture

According to [35], in the Layers architecture, a system is decoupled into layers in a vertical manner in which each layer provides services that can be used by the higher level layer and uses services that are offered by the lower level layer. The interaction between the adjacent layers is achieved by clearly defined interfaces. Within each layer, horizontal components are also required and they interact through connectors. In a pure Layers architecture, higher level layers can only access lower level layers via the layer beneath.

[36] utilizes a Brower/Server model, which is essentially a Layers architecture, to implement a web-based manufacturing service system for rapid product development in small and medium sized enterprises. The web application is structured into three layers, Client, Server/Application Server and Database Server. The Client layer acquires data from the Server layer, while the Server layer executes the operations of database access from the Database Server. Inversely, the Database Server returns data to Server layer and the data are further forwarded to Client layer.

Another web application using Layers architecture in the domain of manufacturing systems is introduced in [37]. The web application provides virtual environments for dynamic manufacturing tasks planning. The research utilizes a three-tier architecture to implement the system. Similarly to the previous application, the system consists of three

layers, a Web Server, a Database Server and Clients. The Web Server is implemented with Apache HTTP Server, from which users request static and dynamic information. On the other hand, Java database connectivity and socket handler are implemented on the Web Server for data acquisition from Database Server built with MySQL server. Cortona VRML client, embedded in a HTML file, serves as a fast and highly interactive Web3D viewer as plug-ins.

Model-View-Controller architecture

Model-View-Controller (MVC) is an architectural pattern to decompose the design of an application into three distinct parts; the Model, the View, and the Controller. The Model manages the data for the application that are to be represented to users. The View displays the data to users. And the Controller is associated to each View and interacts between the Model and the View informing the Model when a new input has been performed on the View. [38] [39] [40] [35] The MVC design pattern has become more popular in the last few years [41]. In the MVC design pattern, the functional programs in the three parts are developed separately, so the reuse of code becomes possible, thus improving the efficiency of software development [39].

[42] has implemented a web application to share and reuse manufacturing resources using MVC architecture. The web application provides manufacturing resource services, which is a supplement of the existing resource service applications, to discover, manage and integrate manufacturing resources. All the services are implemented as Models using enterprise Java bean, while the View and the Controller of the web application are encapsulated using JSP and servlet separately. [42] states that the use of MVC architecture enables better flexibility and compatibility.

Client-Server architecture

[35] classifies the Client-Server architecture into the category of component interaction view, which concerns issues such as how the independent components interact with each other and how these components are decoupled. In the Client-Server architecture, the system is divided into two independent components, the Client and the Server. The communication between them follows the pattern of request and response, in which a Client requests information from the Server with an ID or an address of the Server. Afterwards, the Server responds to the corresponding Client with the information. Complex architecture can be established using multiple Client-Server architecture, in which a server can either act as a server to clients or act as a client to servers forming an n-Tier-architecture or Layers.

A web-based framework to facilitate the communication between product designers and manufacturers concerning the design and manufacturing of work pieces is proposed in [43]. With the framework, product designers can submit their designed work pieces to the system software. Registered manufacturers in the system can then obtain the design and decide if they want to produce the work piece by submitting an offer to the designer. The applications on both the designers' and the manufacturers' side are

implemented using Client-Server architecture with the system software being the server. On each side, implementation for the establishment of network connection is needed. Furthermore, the Client on the designers side needs to have implementations for authentication and data transfer purposes, while on the manufacturers side, functionalities including understanding and using the services provided by the system as well as accessing and parsing the designed files need to be implemented on the Client.

In [44], a web-based supervision and control system to access the real time data and to remotely control electric load via PLCs is developed. As is proposed in [44], the system is deployed based on a Client-Server architecture, in which the Server communicates with the PLC directly through RS-232 port, while the Client accesses and controls the remote electric load by sending requests to the Server. Therefore, a program to handle the requests from the Client and to communicate with PLC on the Server is needed. In order to show electric load curves for monitoring, a database management system is also implemented on the Server. The using of this structure increases the flexibility of extending the system with new features and the security to control PLCs.

Comparison on architectural patterns

As is summarized in Table V, the architectures reviewed above can be distinctively categorized according to views. Although the Layers architecture can decompose the system into three different parts as the Model-View-Controller architecture, there are major differences. The layers in the Layers architecture are arranged in a vertical manner, so each layer can only interact with its adjacent layers. It is not the case in the Model-View-Controller architecture, in which the Controller and the View are associated. The Controller notifies a change on the View to the Model. However, the data rendered on the View are directly obtained from the Model. Beside, interfaces on each layer need to be clearly defined to achieve the interaction between layers in the Layers architecture, while there is no such requirement in the Model-View-Controller architecture.

On the other hand, there are also similarities in these architectures. As far as the Client-Server architecture is concerned, when a Server acts as a Server to a Client and as a Client to another Server, it becomes a Layers architecture.

***Table V**: Summary of features of Layers, Model-View-Controller and Client-Service architecture*

Architecture	Features
Layers (Layered view)	The system is composed of multiple layers. Each layer uses the services from the lower layer and provides services to the upper. Interfaces for interaction need to be defined in each layer.
Model-View-Controller (User interface view)	Model, View and Controller composes the system in which the Model is strictly independent from the View and Controller. The View directly receives data from the Model, while the Controller notifies the Model when an update on the View occurs.
Client-Service (Component interaction view)	Client and Service are the components in the system. The Client initiates the interaction between the two entities by sending requests, while the Service responds.

2.5.2. Review on web application frameworks

Many frameworks have been developed to facilitate developers for rapid web application development deploying architectural patterns. The following sections briefly introduce these frameworks.

Apache Struts 2

Struts 2 is an open source framework for creating Java web applications deploying MVC design pattern. The MVC pattern is implemented by the action (Model), result (View) and *FilterDispatcher* (Controller). *FilterDispatcher* maps the user request to appropriate action, data for rendering and business logic are defined in model with action component, and view for data rendering can be implemented by presentation-layer technologies, such as JSP, Velocity Tamplate and Freemaker. [45]

Figure 6 demonstrates the process from receiving a user request to view rendering with Struts 2. Each time the framework receives a user request, an *ActionInvocation* class is initiated. The execution of an action starts by calling the *invoke*() method in the *actionInvocation* instance. The method determines if there is a next interceptor. If there is, it passes over the execution of the thread to the *intercept*() method in the interceptor. After all the interceptors are executed, the *actionInvocation* instance invokes the action instance to generate results. Which action to invoke is determined by a user-define configuration file which maps the requests to the actions. Currently, data in results are in Java types. The framework utilizes Object-Graph Navigation Language (OGNL) to convert these results in Java types to string-based HTML data for view rendering. [45]

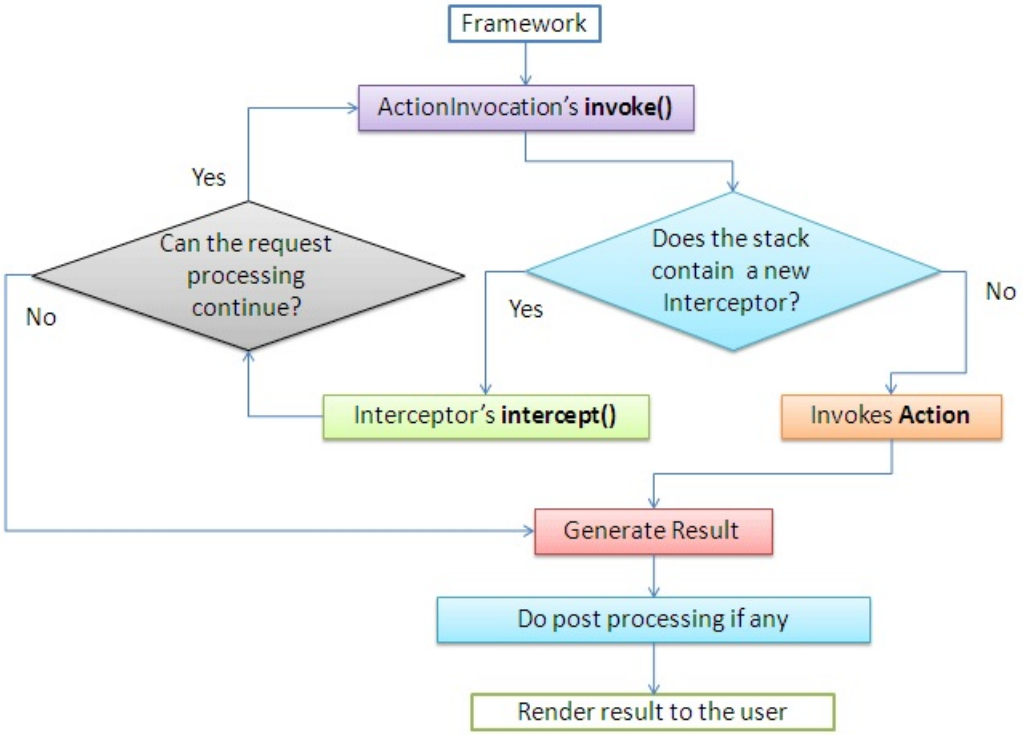


***Figure 6**: Process of action invocation in Struts 2* [45]

Spring Framework

Spring is an open source framework to facilitate the development of Java applications by supporting a comprehensive infrastructure. It provides layers including IoC container, Data access and integration, Web, Aspect Oriented Programming, Instrumentation and Test. However, the framework is organized in modular enabling developers to use the functionalities they need. [46]

One of the features in Spring is the provision of containers for IoC, which is implemented using Dependency Injection (DI) design pattern [47]. The DI design pattern enables external services to be injected into application components without changing the source code of the application [48]. With Spring, the DI can be realized with setter methods and constructors. The container instantiates objects and delivers the resources to components.

As is mentioned, Spring framework also provides a web layer for web applications development. It is complaint with the Model-View-Controller architecture. The design of the framework is around a *DispatcherServlet* that dispatches requests to handlers and provides other functions to assist the development of web applications. Figure 7 illustrates the request flow in a Spring MVC framework. The *DispatcherServlet* known as a Front controller delegates the incoming request to a Controller, which in turn creates a model and returns the model to the Front controller. Afterwards, the Front controller forwards the model to the View template and receives the control back from it with view response. Finally, the Front controller returns the view response to the request.

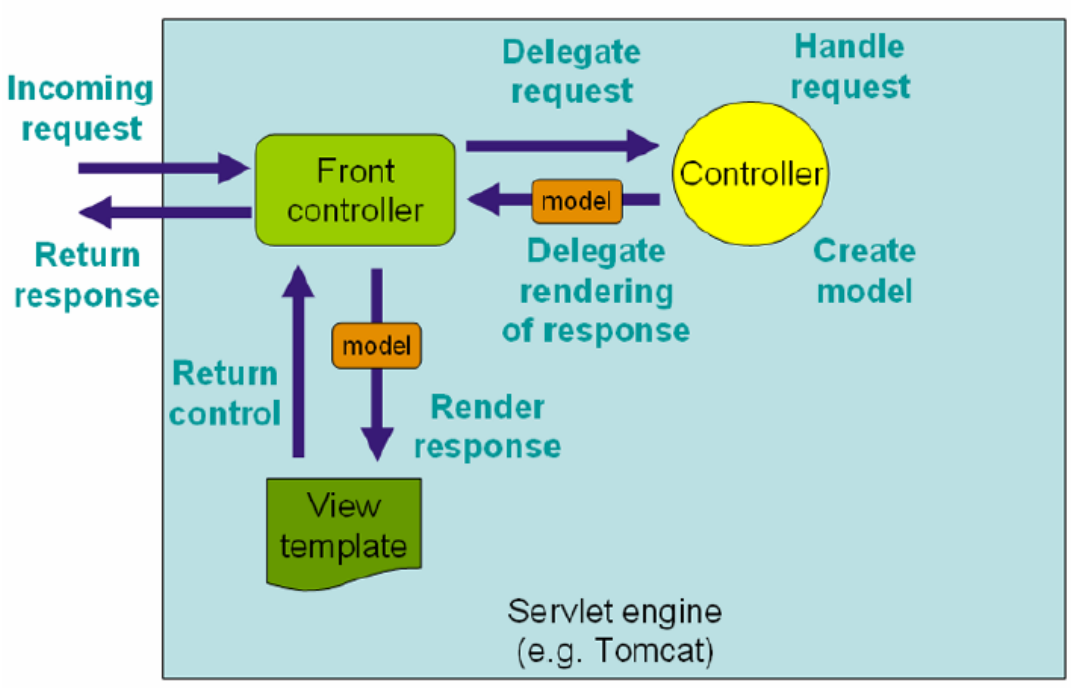


***Figure** 7: Request flow in Spring MVC framework* [46]

Furthermore, Spring framework also supports for the integration of other MVC frameworks such as Struts and WebWork. This feature enables developers with the

knowledge and experiences of other frameworks to use other functionalities provided by Spring framework.

Wicket

Wicket is a component-based framework for developing web applications using regular object oriented Java programming. The developers can program with only Java, HTML and meaningful abstractions in Wicket. Java code is used to implement the behaviour of web applications. For example, developers can create components and their behaviours by inheriting existing components. On the other hand, HTML is used to maintain pure presentation of the applications by constructing only the structure of web pages. In this way, the presentation and logic parts are separated rigidly. Lastly, Wicket provides abstractions for all widgets, for example, links, drop down lists and text fields. Meaningful abstractions can be provided by writing custom components.

To illustrate the architecture of Wicket, how requests are processed is explained. Figure 8 demonstrates the steps for processing a request. In the first step, a request URL is decoded into, for instance, component path, version number and interface name and the decoding results are stored in a *RequestParamters* object. Afterwards, it is the step for determining request target. In this step, many types of requests including bookmarkable pages, shared resources and Ajax requests can be handled. In Figure 8, for example, the request target encapsulates the call from the *ILinkListener* interface on a previously rendered page. The third step handles procedures such as calling links, listeners or Ajax behaviours. Finally, the request target responds to the client by rendering the page, locating a resource or rendering individual components dependant on the whether it is a page request, resource request or Ajax request, respectively. [49]

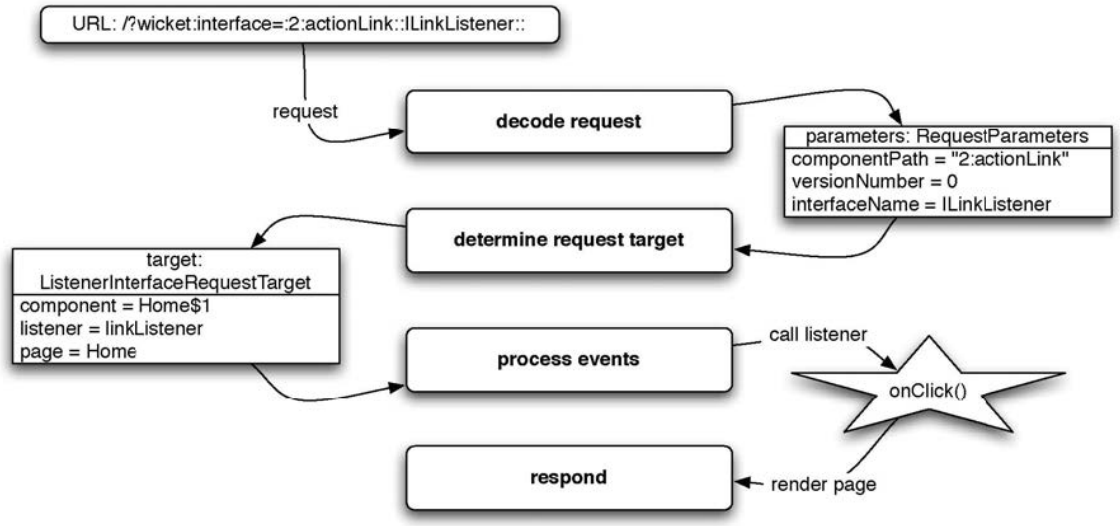


***Figure 8**: Request processing steps in Wicket* [49]

Comparison on web application frameworks

Table VI collects the main features of the reviewed frameworks for web applications development. As is illustrated, all the frameworks are designed complaint with MVC architectural pattern. Struts 2 and Spring frameworks are both request oriented, which

means that developers handles the mapping of requests and views with an xml configuration file for both frameworks. In addition, Struts 2 and Spring frameworks require logic coding in the View implementation, for instance, using JSP or Velocity template, although they also separate the View layer from Model and Controller. Conversely, although Wicket also has request flow in the framework, it hides the flow underneath so that developers does not need handle requests and responds. Instead, during the implementation of web applications using Wicket, one can define functions of components in Java and build the View layer with pure HTML. The components used in HTML and defined in Java are associated with an id attribute. From the view of integration, Struts 2 allows the integration with Spring framework to use extra functionalities and it is possible to integrate Spring framework into other MVC frameworks including Struts and Webwork. However, Wicket does not provide the integration into other frameworks.

***Table VI**: Features of Struts 2, Spirng and Wicket frameworks*

Framework	Architecture	Type	Logic in View	Integration
Struts 2	MVC	Request oriented	yes	Spring
Spring	MVC	Request oriented	yes	Struts, Webwork
Wicket	MVC	Component oriented	no	-

2.5.3. Review on presentation technologies

Many technologies for view representation have been developed and widely used in a variety of web application frameworks. The following parts review these technologies including JSP, Facelets and Velocity.

JSP

JSP representing Java Server Page is a Java-based technology aiming for rapid development of dynamic web applications by applying Expression Language (EL) and JSP Standard Tag Library (JSTL). EL enables simple access to Java bean components by using simple syntax. A simple variable or a nested structure can be accessed. JSTL standardizes structural tasks such as iteration and conditional statement in order that developers define the standard tags and use it in multiple JSP containers. JSP is built based on HTTP and Java Servlet specification. HTTP enables the client-server communication over the Internet. It also requires a simple directory layout to organize configuration information, provided by deployment descriptor, and resources. [50]

Facelets

Facelets is a powerful and lightweight view declaration language for JavaServer Faces technology substituting JSP due to its incomplete support for all the new features in JavaServer Faces 2.0. Besides the support of JavaServer Faces tags and JSTL tags for custom actions and structural tasks respectively, it also supports for Facelets tag libraries for templating purposes. JavaServer Faces HTML tags are used to represent components on web pages. It supports EL as JSP does to reference properties and methods of Java bean components. [51]

Velocity template

Velocity is a Java-based template to tightly separate the view layer from model and controller. As is mentioned in 2.5.2, the view layer in Struts 2 can be implemented with Velocity. Velocity Template utilizes Velocity Template Language (VTL), which can be embedded in HTML documents, to reference methods or variables defined in Java code, or to acquire values from web pages to the background Java programs. The following is an example of VTL.

#set($a = "Velocity")

A VTL statement starts with a '#' symbol following a directive. A variable is referenced with a '$' mark followed by the variable name defined in Java code. In the statement, 'set' directive assigns the variable with the value 'Velocity'. [52]

Comparison on presentation technologies

As is indicated in Table VII, all the three presentation technologies reviewed above utilize certain syntax to reference Java bean properties or methods for data dynamic rendering. All of them allow the execution of logic with structural tasks. JSP and Facelets take the advantage of JSTL for logic representation, while Velocity template uses VTL. One major difference between Facelets and other technologies is that special HTML tags, JavaServer Faces HTML tag library need to be defined to represent web page components while the other two technologies simply use HTML tags.

***Table VII**: A summary of presentation technologies*

	Extension	Component tags example	Directive example	JavaBean reference
JSP	.jsp	HTML tags: <td>1</td>	<c:forEach …> </c:forEach>	${name}
Facelets	.xhtml	JavaServer Faces HTML tag library: <h:outputText value = "1"/>	<c:forEach …> </c:forEach>	${name}
Velocity	.vm	HTML tags: <td>1</td>	#foreach (…) … #end	${name}, $name, !$name, etc

2.5.4. Review of persistence frameworks

To persistent data, persistence frameworks are used to store data in databases. However, since Java model uses classes to represent data, while database uses table, a solution is needed to convert Java objects to Java Database connectivity result sets and vice versa. This section introduces the frameworks and implementations of object to relational data mapping known as ORM.

JDBC

JDBC is the abbreviation of Java Database Connectivity. It is an API offering JDBC classes for database connectivity and operations. It passes SQL statements from Java to a database for operations. The data acquired from the database are Java Database connectivity result sets, the Java program needs to handle the sets by calling specific JDBC classes to convert the database result type to objects. [53]

JPA

Java Persistence API (JPA) is a standard solution introduced to facilitate the mapping of object-oriented models and relational database systems. Nowadays, many persistence products are based on it. The features of JPA are concluded in [54]. First, the objects are Plain Old Java Objects (POJOs) which means no special requirements are needed when designing the models. The object-relational mapping with JPA is driven by metadata, which can be achieved by adding annotations internally or defining XML externally. Then the persistence is non-intrusive since the persistence API does not directly operate the object classes. Instead, the business logic of the application calls API and passes it to persistence objects, thus instructing API to operate on the objects. Thirdly, querying with JPA, one needs not to have the knowledge of the database columns nor database mapping. Java entities and their attributes are utilized in queries which are further translated into SQL by the JPA implementation. Furthermore, features of JPA also include simple configuration, integration and testability.

Hibernate

Hibernate is an Object/Relational Mapping implementation based on JPA designed to assist developers working with both Object-Oriented software and Relational Databases. Since the hard coding to map data represented in objects to relational databases is time consuming and cumbersome, Object/Relational Mapping technique deployed by Hibernate provides an easy means of mapping data from an object model to a relational data model. [55]

Comparison on persistence frameworks

As can be seen from Table VIII, although it is possible to operate database directly using SQL with JDBC API, it exposes some weaknesses. First, the object and relational data mapping has to be done manually with JDBC which requires a lot of hard coding in the Java program. For JPA and its implementation, one only needs to map the object-oriented model to the relational data with annotation when creating the model or by adding external XML mapping file. Furthermore, using SQL as the querying language, one needs to have detailed knowledge on the database tables, columns and the mapping between the relational data and Java objects. Once the modification on database occurs, SQLs concerned have to be modified universally. It is not the case with JPA and its implementation Hibernate. Once the ORM is configured with annotation or XML, one can use entities and their attributes to construct the queries directly. In case of modification of database, only changing the ORM configuration is enough.

***Table VIII**: A Summary of persistence frameworks*

	Querying language	Automatic ORM
JDBC	SQL	No
JPA	Object queries	Yes
Hibernate	Object queries	Yes

3. RESEARCH METHODS AND MATERIALS

As is presented in theoretical background, a variety of researches have been conducted for the design of KPIs for manufacturing systems. There also exist multiple toolkits and frameworks for the communication of service-enabled systems, development of web applications and run time management of events. This chapter describes how these techniques are utilized to implement a run time monitoring system for KPIs.

It starts with the introduction to the overall architecture of the system and a brief introduction to the test bed. Then the design of KPIs for the test bed using the methodology from theoretical background is described. Finally, how the KPI management system is developed into a web application and how it is exposed as a web service are presented.

3.1. Introduction to overall architecture

The overall system architecture is designed and shown in Figure 9. Local devices (1 and 2) publish their information in web services through Ethernet or wirelessly. The local server at 3 can subscribe to these web services thus receiving information from the devices. The local server also exposes its data as web services so that other applications can make use of them.

The estimation of KPIs can be achieved at all locations. On one hand, KPIs can be acquired directly from local devices at 1 and 2 if the processing of data is performed at device level. The advantage of it is that data processing at device level alleviates the complexity of aggregation and correlation of data at higher level since low level devices know the context of information. On the other hand, KPIs can also be computed and correlated at locations 3, 4, and 5. The advantage of this method is that it provides great flexibility for generating new and complex KPIs. No change is needed at device level when new KPIs are designed and calculated at higher level. This thesis work computes KPIs at location 3.

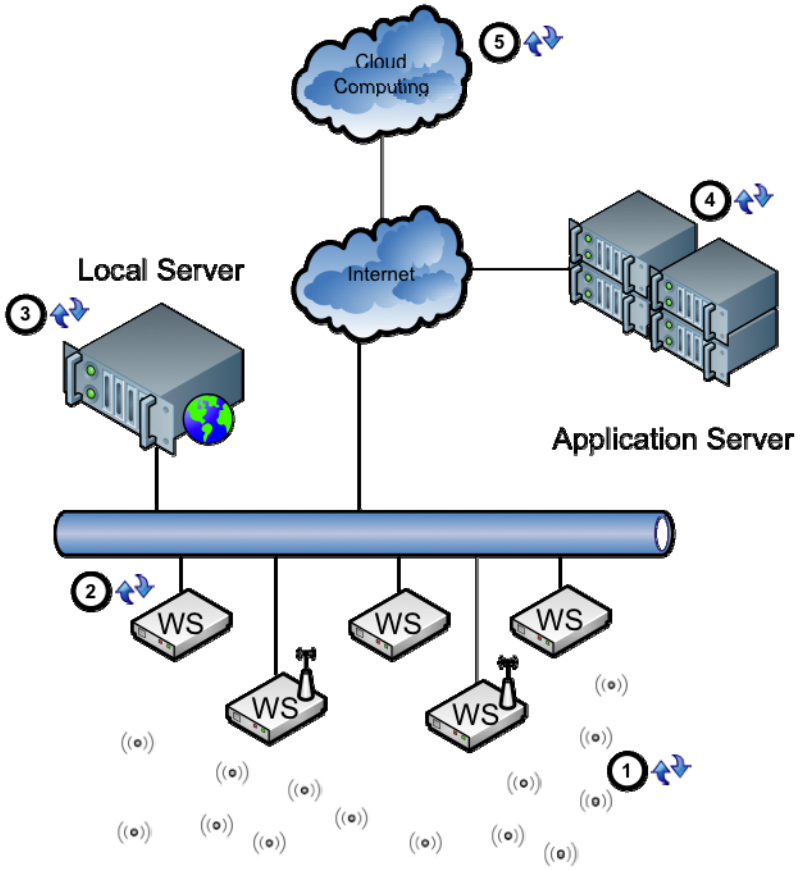


***Figure 9**: Overall system architecture*

3.2. Introduction to test bed

The test bed used in this work is a production line physically located at the premises of the Factory Automation Systems and Technology laboratory, Tampere University of Technology. Figure 10 depicts the layout of the test bed. It consists of ten robots simulating the production of mobile phones by performing assembling operations including frame drawing, screen drawing and keyboard drawing. There are three different types of components in shape and three in colours for each shape making a total variant of products to be 729.

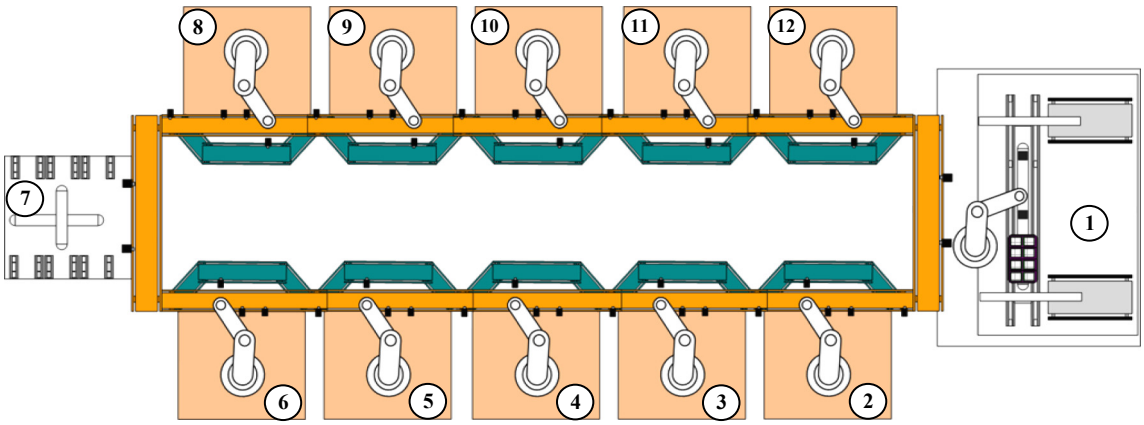


***Figure 10**: Layout of Fastory production line*

Robot 1 is in charge of completing the production. Once a product arrives at cell 1, it checks if all the three assemblies are completed. If completed, it is dispatched to the tray. Otherwise, it is forwarded to cell 2 for further assembly. In addition, a machine vision system located in cell 1 is used for quality inspection. The machine vision system inspects whether the completed products meet the quality criteria. If not, it also specifies which component does not meet the criteria, the frame, the keyboard or the screen. Cell 7 is a buffer which stores multiple products.

On condition that one robot is occupied by a product, while another one arrives, it is transferred to the next robot by using the bypass conveyor.

Information of the test bed is published in web services by S1000 controllers [56]. These web services can be discovered and subscribed by any DPWS client. In addition, the S1000 controllers also support manual configuration of message subscribers. This work utilizes the latter option to register the subscriber URL in the subscription list (Figure 11).

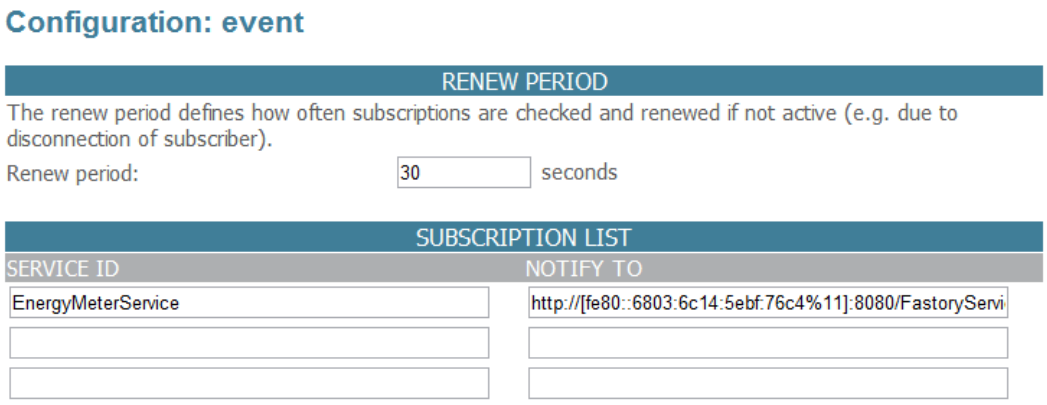


***Figure 11**: Subscription configuration on S1000 controllers*

Each cell exposes web services notifying the status of the robot and the conveyor system. The machine vision system exposes another web service about quality inspection. In addition, each cell is embedded with an E10 energy analyzer [57] publishing the energy consumption related information as a web service. All available information is shown in Table IX. As can be seen, the information from the manufacturing system is atomic, which requires aggregation and correlation at higher level to gain KPIs. As is mentioned in Section 2.4, complex event processing technology is needed to aggregate these raw data. However, before that, it is necessary to design a set of KPIs for the test bed.

***Table IX**: Events from the test bed*

No	Message	Description
1	EquipmentChangeState	The message contains information of cell ID, event ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot number 2-6 and 8 to 12, as well as a time stamp.
2	EquipmentChangeState	The message contains information of cell ID, event ID, pallet ID, the current state, the previous state of the robot number 1, as well as a time stamp.
3	QualityInspection	The message indicates the quality information including pallet ID, the quality of frame, screen and keyboard, the quality of the inspection result as well as a time stamp.
4	EnergyMeter	The message contains energy consumption (3-phase active energy, active power, reactive energy, reactive power, apparent energy, apparent power, Root Mean

		Square (RMS) current, RMS voltage as well as line frequency) located in each working cell. The information is published at the time interval of five seconds.
5	NotificationMessage	The message contains pallet transfer status on conveyors including cell ID, 'fromZoneId', 'toZoneId', pallet ID, event ID and time stamp.
6	Values	The message from the THL sensors contains values for temperature (in degree Celsius), relative humidity (in percentage) and ambient light (in lux).

3.3. Design of KPIs for test bed

In the level of production management, five principal KPIs were defined in [2], which are Safety and Environment, Production Efficiency, Production Quality, Production Plan Tracking and Employees Issues.

[2] has developed an 8-step iterative closed-loop model of KPI introduction, which is described in 2.1.3. The thesis work adopts the model for KPI design for the test bed.

3.3.1. Define production goals and objectives

The goals and objectives of the test bed can be summarised as follow:

1. To increase the productivity
2. To achieve high and uniform product quality
3. To use energy and material efficiently
4. To reduce greenhouse gas emission to environment
5. To establish production tracking

3.3.2. Define potential indicators

According to the goals and objectives of the test bed and the general set of KPIs for production, a set of potential indicators is defined. Productivity of a manufacturing system relies on the efficiency of the production line. Therefore, Efficiency is one of the potential indicators. Meanwhile, in order to ensure high product quality, quality is an indicator to be monitored and evaluated. The goals also include the requirement for production plan tracking. Concerning the energy and material use and the greenhouse gas emission, although there is an indicator of safety and environment in the general set of KPIs in production, the sustainable production indicators is considered more in details. Therefore, sustainable production indicators of energy and material use and natural environment are chosen. They form the KPIs on sustainability. Furthermore, it is stated in [58] that maintenance functions and reliability of a system are key factors influencing the efficiency of an organization producing high quality products and providing services, reliability is thus one of the KPIs. All in all, the potential indicators defined are summarised below. Each KPI consists of indicators in the standard set.

1. Efficiency
2. Quality
3. Production plan tracking
4. Energy and material use
5. Natural environment
6. Reliability

3.3.3. Select indicators for implementation

According to the available information described in Section 3.2, indicators for implementation are selected. Since there is no message concerning information on orders, and measurement of gas emission, the KPI of production plan tracking and natural environment are not selected. However, the KPI of production plan tracking can be implemented as long as an order entry system is established and the messages containing the order information are provided by the system. The KPIs that will be implemented are Efficiency, Energy, Quality and Reliability. Besides, three other KPIs reflecting the overall status of the entire production line are also to be implemented. All the KPIs for implementation are shown in Table X.

***Table X**: Selected KPIs and KRIs for implementation*

KPI	Indicators	Unit	Description
Efficiency	Unit Energy Consumption	Wh	Energy consumption for producing one product in each cell
	Process Energy Consumption	Wh	Energy consumption per process
	Unit Production Time	second	Time needed to produce one part on each cell
	Unit Processing Time	second	Time needed for a single process
	Production Shutdowns	second	Production shutdown time for a cell
	Cell Production Rate	-	Number of products produced per hour by a cell
Energy	Cell Energy Consumption	Wh	The total energy consumption of a single cell
	Unit Energy Consumption	Wh	Energy consumption for producing one product in each cell
	Process Energy Consumption	Wh	Energy consumption per process
	Energy Consumption per Product	Wh	Energy consumption for producing one product by the production line
Reliability	MTTR	second	Mean time-to-repair of each cell
	MTTF	second	Mean time-to-failure of each cell
	MTBF	second	Mean time between failures of each cell
Quality	Frame Quality Rate	%	Percentage of frame that meets the quality criteria
	Screen Quality Rate	%	Percentage of screen that meets the quality criteria
	Keyboard Quality Rate	%	Percentage of keyboard that meets the quality criteria
	Quality Rate	%	Percentage of final product that meets the quality criteria
Overall	Total energy consumption	Wh	Total energy consumption of the production line
	Pallet Production Time	second	Time needed to produce one part by the production line

KPI	Indicators	Unit	Description
	Line production rate	-	The production rate of the production line on hourly basis
	Total products	-	The total number of products produced by the production line

Efficiency

The indicators selected to show efficiency performance are energy consumption per product, energy consumption per process, unit production time, unit processing time, production shutdowns and production rate.

[59] explains four types of energy-aware KPIs, which are called measure of energy efficiency performance (MEEP).

1. Thermal energy efficiency of equipment
2. Energy consumption intensity
3. Absolute amount of energy consumption-heat value
4. Diffusion rates of energy-efficient facilities/types of equipment

Energy consumption intensity is defined as energy input divided by output, which is useful for measuring and evaluating the detailed unit energy consumption or specific energy consumption. The output can be physical output, for example, number of products or numbers of process, etc, as well as economic values. Concerning the seven properties in section 2.1.1, nonfinancial measures, such as dollars, Euros, should be included in KPIs. Therefore, the selected KPIs for energy consumption intensity are energy consumption per product and energy consumption per process.

In addition, timings regarding to the unit production time and processing time are designed as indicators. The decrease of this KPI indicates the increase of efficiency. Number of parts produced per hour is also defined to indicate the efficiency performance on an hourly basis.

Production shutdowns indicator is the total shutdown time for a single cell when a failure occurs. It illustrates the efficiency of the cells in the aspect of working hours.

Energy

According to [4], total energy use and energy use per unit product should be considered when designing indicators for energy aspect. Furthermore, unit energy consumption, energy consumption per product (by the entire line) and cell energy consumption can be attained from the raw data of the test bed. Therefore, they are selected to be the indicators in this category.

Reliability

In the aspect of reliability, mean time-to-repair (MTTR), mean time between failures (MTBF) [60] [15] and mean time-to-failures (MTTF) [61] are implemented.

MTTR should be measured as the average time interval between the occurrence of a failure and the time when the plant starts production process again. The average time

period from the end of a failure to the occurrence of another failure is measured as MTBF. MTTF is the average time interval between the occurrence of a failure and the happening of another failure.

The boundaries of the above mentioned indicators are all specific to a single station which better identifies the location of an action that has to be carried out when an unsatisfied KPI is received.

Quality

As is mentioned in Section 3.2, a machine vision system is in charge of inspections of the quality of the products and sends notifications of the inspection results. It inspects whether the frame, the screen and the keyboard meet the quality criteria thus concluding the quality criteria of the final products. For quality category, frame quality rate, screen quality rate, keyboard quality rate and product quality rate are the indicators. The first three indicates percentage of each component that meets the quality criteria, while the last present the quality rate of the final product.

Overall

In the selected KPIs above, performance of single cells is most concerned regarding to efficiency and energy use. However, the overall performance of the entire production line needs to be monitored as well. It includes the total energy consumption of the production line, pallet production time, line production rate and total products.

3.4. Implement indicators

This section explains the implementation of indicators (fourth step in the 8-step iterative closed loop model) into a web application. The data source of the application is derived from the notification messages published by the test bed. The data and indicators are also published in a RESTful web service so that third party applications can utilize the resources for analysis and visualization, etc.

Spring WS is used to provide an endpoint thus receiving SOAP messages from the test bed, the endpoint's URL is configured in the subscription list in each S1000 controller, as is described in section 3.2. To process the raw events into meaningful information KPIs on the fly, Esper is used. For persistence purposes, raw data and the KPIs are stored in MySQL database utilizing Hibernate integrated into Spring framework. (Figure 12, left side)

The web application is designed according to the MVC design pattern and the framework for implementation is Spring MVC. Upon user requests with an URL from a browser, a controller requests data from Esper and put them in the view layer via model for real time date representation. For historical data visualization, the controller requests from the database utilizing Hiberate's CRUD functions. The response is rendered as human readable format (graphics) for visualization. (Figure 12, middle)

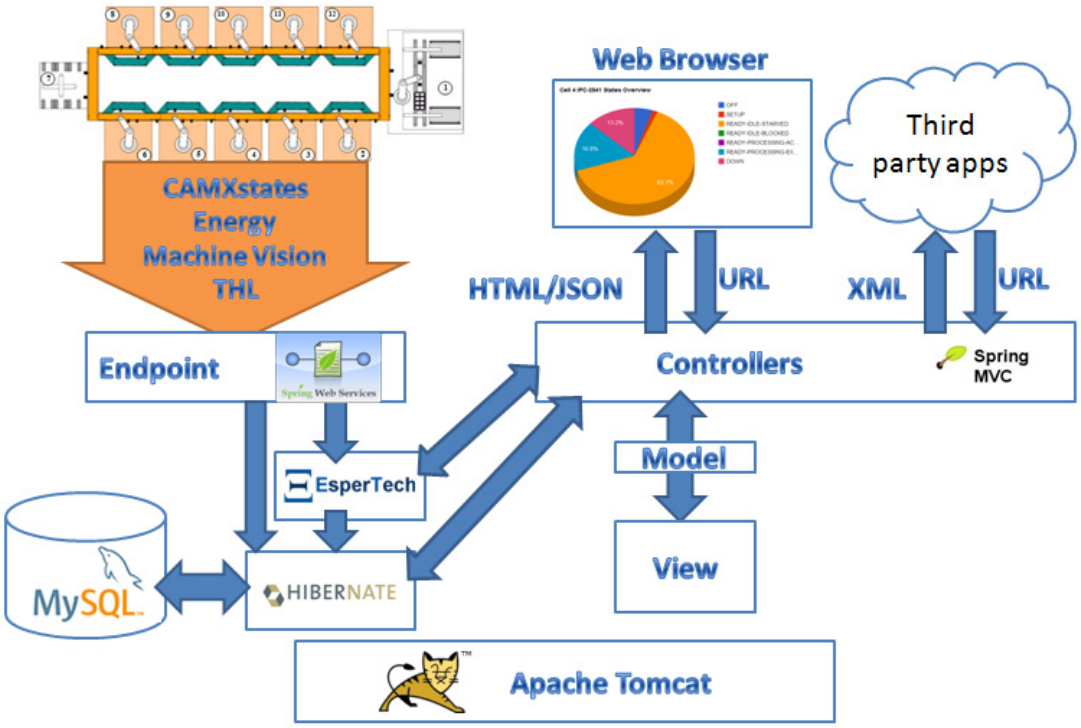


***Figure 12**: Implementation architecture*

Furthermore, the data in MySQL database is also exposed as resources in a RESTful web service using Spring MVC. Third party applications requests resources from the controller. Then the controller selects data from the database and responses the result sets in the format of XML. (Figure 12, right side) The entire project runs on top of an Apache Tomcat server.

3.4.1. Configuration of Java Enterprise Edition project

The configuration of a standard Java Enterprise Edition (Java EE) project relies on a web.xml file located in wepapp/WEB-INF (see Appendix 1 for the project hierarchy). The web.xml file defines servlet and servlet mapping for the incoming requests (Figure 13).

- Spring WS utilizes a *MessageDispatcherSevlet* to dispatch SOAP messages to endpoints. The servlet mapping for *MessageDispatcherServlet* is configured to be "/endpoint/SOAP/*" so that any messages coming to the URL http://{host_name}:port/{project_name}/endpoint/SOAP is handled by the *MessageDispatcherServlet*. In this project, the URL of the endpoint is http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/SOAP.
- A *DispatcherServlet* is responsible for the delegation of URL requests to Spring MVC controllers. The servlet mapping for *DispatcherServlet* is configured to be "/mvc/*" (for the web application) and "/endpoint/REST/*" (for the RESTful web service) so that any URL in the format of http://{host_name}:port/{project_name}/mvc/* and http://{host_name}:port/{project_name}/endpoint/REST/* is handled by *DispatcherServlet*. Therefore, a URL beginning with

http://esonia-controller.rd.tut.fi:8080/FastoryService/mvc/ or http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST is handled by the *DispatcherServlet* and then forwarded to a Spring MVC controller.

- The asynchronous mode of the *DispatcherServlet* is turned on by specifying a true value in *async-support*. In asynchronous mode, when a response of a user request is not available, the *DispatcherServlet* does not respond an empty value immediately. Instead, it waits to respond until a response becomes available or a timeout occurs.
- Additionally, a *ContextLoaderListener* is used to load configuration files other than the default ones. By default, when the name of the *MessageDispatcherServlet* is defined as 'spring-ws', it looks for spring-ws-servlet for loading during initialization. Similarly, if the name of *DispatcherServlet* is spring-mvc, it automatically loads spring-mvc-servlet. When an xml configuration file is used other than the default ones, the *context-param* is used to specify the name of the file from which the *ContextLoaderListener* needs to load.
- A welcome page in the webapp folder can be defined in the web.xml file. It is the first page to render when the application starts.

```
<!-- Context Configuration locations for Spring XML files -->
 <context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/mvc-service.xml
     /WEB-INF/mvc-servlet.xml
     /WEB-INF/applicationContext.xml
     /WEB-INF/rest-servlet.xml
     /WEB-INF/hibernate-config.xml
     /WEB-INF/mvc-security.xml
     /WEB-INF/udp-context.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<!-- Defines the the first page to be rendered -->
<welcome-file-list>
    <welcome-file>home.jsp</welcome-file>
</welcome-file-list>
<!-- Defines the Spring-WS MessageDispatcherServlet -->
<servlet>
    <servlet-name>spring-ws</servlet-name>
    <servlet-class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <async-supported>true</async-supported>
</servlet>
<!-- Map all requests to this servlet  -->
<servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>/endpoint/SOAP/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>*.wsdl</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>rest</servlet-name>
    <url-pattern>/mvc/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>rest</servlet-name>
    <url-pattern>/endpoint/REST/*</url-pattern>
</servlet-mapping>
```

***Figure 13**: Configuration of a Java EE project with web.xml file*

3.4.2. Implementation of SOAP web service

A server side of the SOAP web service is achieved via the implementation of an endpoint provided by Spring WS to handle SOAP messages from the test bed. Figure 14 illustrates the message flow among all the layers.

- A S1000 controller is used to generate SOAP notifications and send them to the server side application implemented with Spring WS.
- A *MessageDispatcherServlet* despatches the messages to the endpoints accordingly.
- The endpoint de-serializes the events to Java objects (marshal) and maps the events to methods accordingly in which the Java objects representing the events are forwarded to a service layer.
- On one hand, these Java objects representing the incoming events are forwarded by the service layer to Esper engine for pre-processing. On the other hand, in the service layer, the Java objects are assigned to other objects that are mapped with relational database tables. These ORM objects are forwarded to a Hibernate layer.
- The ORM objects are stored in MySQL database tables using functions in the Hibernate layer.

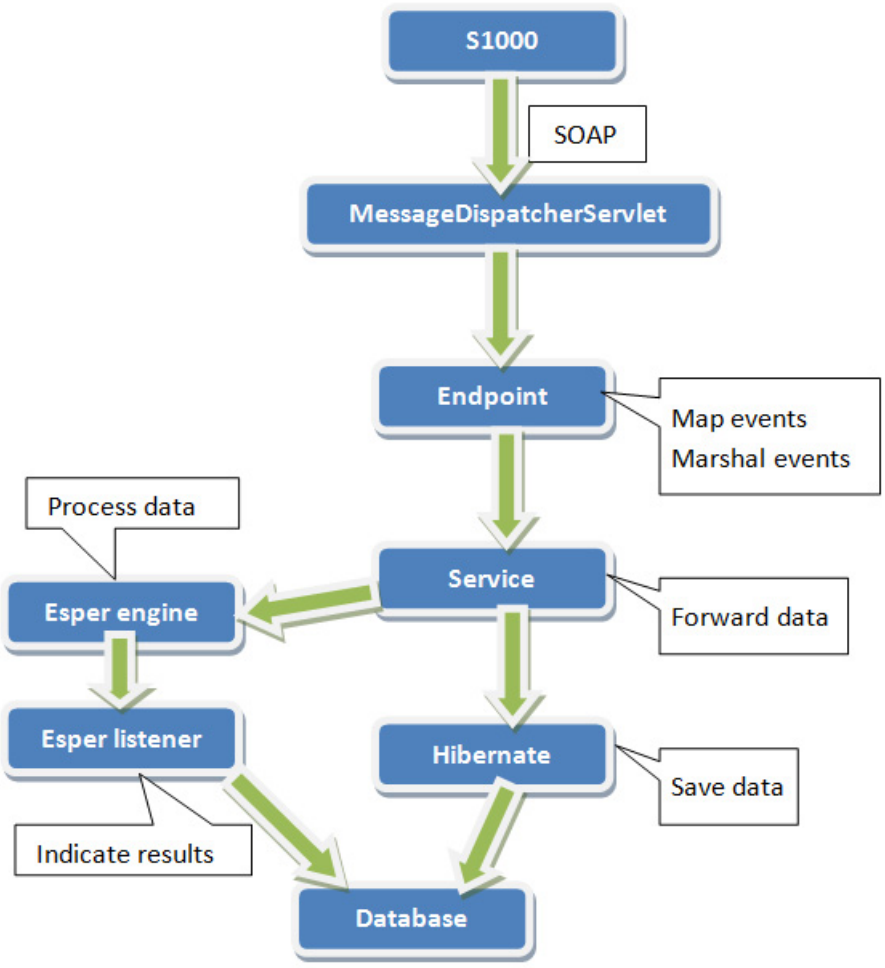


***Figure 14**: Flow Chart for the Implementation of SOAP web service*

- If the Esper engine detects a match with the incoming events, an Esper listener is invoked in which the processing results are assigned to ORM objects and saved in database tables.

Besides, Spring WS supports contract-first web service which means that the WSDL and schema files that are used to describe the messages should be defined beforehand. The following sections utilize robot.xsd and robot.wsdl, which describe the *EquipmentChangeState* message, available in Appendix 2 and 3, to demonstrate the implementation of each layer in Figure 14.

Configuration for Spring WS

First, certain configuration is necessary for creating endpoints for Spring WS in /webapp/WEB-INF folder (Figure 15).

- The class *FastoryEndpoint* is defined as a Java bean with *fastoryService* being referenced from another Java bean in "*constructor-arg*" enabling an object of *FastoryEndpoint* to be created using constructor and the *fastoryService* object being the input parameter during the initialization of the application. This is achieved because of the Spring framework's DI facilities. Besides, some Java codes are also necessary that will be explained later.

```
<bean id="marshallingEndpoint" class="fi.tut.kpimeter.spring.ws.endpoints.FastoryEndpoint">
    <constructor-arg ref="fastoryService"/>
</bean>
<bean id="fastoryService" class="fi.tut.kpimeter.spring.ws.service.impl.FastoryServiceImpl"/>


<bean id="annotationMapping" class="org.springframework.ws.server.endpoint.mapping.
PayloadRootAnnotationMethodEndpointMapping">
     <property name="interceptors">
         <list>
            <bean class="org.springframework.ws.soap.server.endpoint.interceptor.SoapEnvelopeLoggingInterceptor"/>
            <bean class="org.springframework.ws.soap.server.endpoint.interceptor.PayloadValidatingInterceptor">
                <property name="schema" value="/WEB-INF/robot.xsd"/>
                <property name="validateRequest" value="true" />
                <property name="validateResponse" value="true" />
            </bean>
        </list>
    </property>
    <property name="order" value="1" />
</bean>

<bean class="org.springframework.ws.server.endpoint.adapter.MarshallingMethodEndpointAdapter">
    <property name="marshaller" ref="marshaller" />
    <property name="unmarshaller" ref="marshaller" />
</bean>
<bean id="marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
    <property name="classesToBeBound">
        <list>
            <value>fi.tut.kpimeter.spring.ws.schema.robot.EquipmentChangeState</value>
        </list>
    </property>
</bean>

<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
    <description>A message factory to support SOAP 1.2</description>
    <property name="soapVersion">
        <util:constant static-field="org.springframework.ws.soap.SoapVersion.SOAP_12"/>
    </property>
</bean>

<sws:static-wsdl id="robot" location="/WEB-INF/wsdl/robot.wsdl"/>
```

***Figure 15**: Necessary configuration for Spring WS*

- The *PayloadRootAnnotationMethodEndpointMapping* enables the mapping between endpoints and SOAP messages using annotations, in which two interceptors are added for the logging of envelopes of the request and response messages

(*SoapEnvelopeLoggingInterceptor*) as well as the validation of request and response messages (*PayloadValidatingInterceptor*) against the schema file (here robot.xsd as an example).

- The project uses *Jaxb2Mashaller* for serialization and de-serialization between XML and Java objects. By configuring *Jaxb2Mashaller* and setting it to be a property in the *MarshallingMethodEndpointAdapter*, the incoming messages are already de-serialized into Java objects (e.g. of class *EequipmentChangeState,* see Appendix 4) upon the arrival at handlers so that they can be directly handled.
- *SaajSoapMessageFactory* defines the soap version of the incoming messages to be handled by Spring WS. Since S1000 controllers generate SOAP 1.2 messages, *SOAP_12* is defined.
- To publish the WSDL file on web, an ID and the location of the WSDL file (robot.wsdl) are defined for *static-wsdl* element.

Implementation of endpoint

To initiate the message routing, the endpoint URL (http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/SOAP) needs to be registered as a service subscriber in each S1000 controller. The *MessageDispatcherSevlet*, which is the entry of the web service, receives SOAP events, dispatches the events to Spring WS endpoints and invokes the corresponding handlers. A Spring WS endpoint is a Java class annotated with *@Endpoint*. A simplified endpoint for handling *EquipmentChangeState* message is shown in Figure 16.

```
@Endpoint
public class FastoryEndpoint {

    /**
     * Namespace of the request.
     */
    public static final String ECS_NAMESPACE_URI = "http://www.tut.fi/fast/robot";

    /**
     * The local name of the expected request.
     */
    public static final String ECS_REQUEST_LOCAL_NAME = "EquipmentChangeState";

    private final FastoryService fastoryService;

    @Autowired
    public FastoryEndpoint(FastoryService fastoryService) {
        this.fastoryService = fastoryService;
    }

    @PayloadRoot(localPart = ECS_REQUEST_LOCAL_NAME, namespace = ECS_NAMESPACE_URI)
    @ResponsePayload
    public void handleECSRequest(@RequestPayload EquipmentChangeState requestElement,
            SoapHeader soapHeader) throws ParserConfigurationException {
        if(!requestElement.getCellId().equals("XX")){
            fastoryService.saveECSData(requestElement, soapHeader);
        }
        System.out.println("handling EquipmentChangeStateRequest");

    }


}
```

***Figure 16**: Implementation of an endpoint*

- The *@Autowired* marks the constructor of the endpoint indicating the constructor and the Java bean of *fastoryService* (defined in Figure 15) are auto-wired.
- To map an incoming SOAP event to a handler (a Java method), *@PayloadRoot* annotation is used in which local part of the root element (*localpart*) and target name space (*namespace*) are defined in the method thus incoming SOAP events with the same root element and target name space is dispatched to this controller (by *MessageDispatcherServlet*) and handled by the method.
- The handler is also annotated with *@ResponsePayload* so that the return object is used as the response message. This handler uses *void* keyword, so no response message is generated upon the *EquipmentChangeState* message.
- A *@RequestPayload* annotation specifies the input parameter of the handler. The input parameter is already an *EquipmentChageState* object instead of an XML stream because of the use of *Jaxb2Mashaller* and *MarshallingMethodEndpointAdapter*.
- The handler simply forwards the *EquipmentChangeState* object to the service layer by calling the *saveECSData* method defined in the service layer.

Implementation of service layer

The service layer is composed of a *FastoryService* interface and its implementation class *FastoryServiceImpl*. To illustrate the handling of *EuipmentChangeState* object in the service layer, a simplified version of *FastoryServiceImpl* is shown in Figure 17.

```
@Service
public class FastoryServiceImpl implements FastoryService {

    @Autowired
    private DataService dataService;

    @Override
    public void saveECSData(EquipmentChangeState equipmentChangeState, SoapHeader soapHeader) {

        Data data = new Data();
        Timestamp ts = this.parseTimestamp(equipmentChangeState.getDateTime().toString());
        data.setTimeStamp(ts);
        String name = equipmentChangeState.getEventId()+"_"+equipmentChangeState.getCellId();
        data.setName(name);
        String address="";
        /**Omitting:Retrieval of address from soapHeader **/
        data.setValue(address);

        Set<MetaData> metaDataSet = new HashSet<MetaData>();
        /**Omitting:Assign each value of equipementChangeState object fields to metaData Object
         * and put each metaData object in the metaDataSet **/
        data.setMetadataSet(metaDataSet);

        dataService.save(data);
        /**Omitting:Check if the rules are already registered in the Esper engine,
         * if yes do nothing, else register the rules **/
        EsperConfig.getInstance().sendEvents(equipmentChangeState);
    }

    private Timestamp parseTimestamp(String timestamp) {
         return Timestamp.valueOf(timestamp.substring(0, 10) + " " +
                 timestamp.substring(11, timestamp.length()));
    }
}
```

***Figure 17**: Sample code of FastoryServiceImpl class*

- A service layer class is marked with *@Service* to enable component scanning without XML configuration.
- In *saveECSData* method, except for the address which is available from the *soapHeader* object, each value in *equipmentChangeState* is assigned to a *data* object (see section 3.4.6 and 3.4.7) and forwarded to a *save* method in the Hibernate layer.
- The *equipmentChangeState* object is also sent to the Esper engine for processing.

Implementation of Hibernate

The implementation of Hibernate layer consists of the integration of Hibernate into Spring framework using Java beans (see section 3.4.7), the design of ORM (see section 3.4.7), a service layer (e.g. *DataService* interface and the implementing class *DataServiceImpl*) and a Data Access Object (DAO) layer (e.g. *DataDao* interface and the implementing class *DataDaoImpl*).

The Hibernate's service layer represents a business object that provides business functions for high level layers. The simplified *DataServiceImpl* class is shown in Figure 18.

- As a service layer, a Hibernate's service class is also marked with *@Service*.
- The method is annotated with *@Transactional* indicating that the method functions as a database transaction.
- Because it is a business object, specific functions should not be implemented here. It simply calls a method in the DAO layer and uses the *data* object as an input parameter.

```
@Service
public class DataServiceImpl implements DataService {

    @Autowired
    private DataDao dataDao;


    @Transactional
    public void save(Data data) {
        dataDao.save(data);

    }

    /**
     * Omitting other CRUD methods
     */
}
```

***Figure 18**: DataServiceImpl class*

The Hibernate's DAO layer provides the detailed implementation of the functions defined in Hibernate's service layer. A DAO layer implementation class *DataDaoImpl* is shown in Figure 19.

- The DAO class is marked with *@Repository* to indicate that this class resprensts a repository, ensure exception translation and allow component scanning for this class without XML configuration.

- The implementation of save function utilizes the *sessionFactory* instance to acquire a session, begin a transaction and save the input data (here *data* Object) in the database table. If an exception occurs during the saving procedure, the session rolls back the transaction. And finally, the transaction has to be committed by the session.

```
@Repository
public class DataDaoImpl implements DataDao {

    @Autowired
    private SessionFactory sessionFactory;

    public void save(Data data) {
        Session session = sessionFactory.getCurrentSession();

        try {
            session.beginTransaction();
            session.save(data);
        }
        catch (RuntimeException e) {
            session.getTransaction().rollback();
            throw e; // or display error message
        }
        finally {
            session.getTransaction().commit();
        }
    }
    /**
     * Omitting other CRUD functions
     */
}
```

***Figure 19**: DataDaoImpl class*

Implementation of Esper engine and listener

The implementation of the Esper engine involves creating classes that represent incoming events, configuration of the engine, creating EPL rules and adding listeners to the engine. A simplified Esper engine implementation is illustrated in Figure 20.

- The class implements a singleton pattern to make sure that only one instance of the *EsperConfig* class can be created throughout the project using the *getInstance()* method.
- The constructor involves the creation of a configuration instance that represents all configuration parameters and provides an *EPServiceProvider* which can be used to create EPL rules and add listeners. The configuration parameters are defined via an XML file (see Appendix 5) to register the classes representing the incoming events, etc.
- The class provides a method *editRule(Rule rule)* to register the pre-defined EPL rules (see section 3.4.5 for the pre-defined EPL rules) in the engine and assign listeners of the rules according to the rule ID. The listener classes are configured as Java beans in an XML file. This method can be called when a new rule is created from the web page or when the first event arrives at the endpoint.
- It also provides a function that can be called in the service layer to send the incoming events to the engine.

```
public class EsperConfig {
    private static EsperConfig esperConfig;
    private EPServiceProvider epService;
    ApplicationContext context = new ClassPathXmlApplicationContext("../hibernate-config.xml");
    public static EsperConfig getInstance(){
        if(esperConfig==null){
            esperConfig= new EsperConfig();
        }
        return esperConfig;
    }
    public EsperConfig(){
      //configuration of Epser
      Configuration config = new Configuration();
      //register expecting events to the engine.
      config.configure("esper.cfg.xml");
      epService = EPServiceProviderManager.getDefaultProvider(config);

    }
    public String editRule(Rule rule){
        try{
              //get rules
            EPStatement stmt = epService.getEPAdministrator().
            createEPL(rule.getEplRule(), rule.getRuleId());
              //add listener

            if(rule.getRuleId().equals("cell energy consumption")){
                System.out.println("adding listener cellEnergyConsumption");
                stmt.setSubscriber((CellEnergyConsumption)context.getBean("cellEnergyConsumption"));
            }else if(rule.getRuleId().equals("camx states")){
                System.out.println("adding listener camxStates");
                stmt.setSubscriber((CAMXStates)context.getBean("camxStates"));
            }
            /**
             * Omitting adding other listeners to engine
             */
            return "success";
        }catch(Exception e){
            return e.getMessage();
        }
    }
    public void sendEvents(EquipmentChangeState obj){
        epService.getEPRuntime().sendEvent(obj);
    }
    /**
     * Omitting methods for sending other events the engine
     */
}
```

***Figure 20**: Esper engine implementation*

The implementation of a listener for computing the duration of IPC-2541 states based on *EquipmentChangeState* events is shown in Figure 21. It is the engine's task to detect a match of patterns defined in EPL rules. The update method in the listener corresponding to the rule is automatically invoked when a match is detected.

- The listener carries the values of the selected parameters specified in the rule (see Table XI, IPC-2541 States Overview) in a Map object (here *states*) as an input parameter. The selected parameters in the rule are used as keys in the Map while the values as values in the Map. Then one can visit a specific value in the Map by specifying its key.
- The *update* method retrieves the values of all the interested parameters, calculates the time difference between two timestamps, assigns all the values to a *data* object and utilizes the *sessionFactory* to store it in database tables.

```
public class CAMXStates {
    private SessionFactory sessionFactory;

    public void update(Map states){

        Data data = new Data();
        String name ="camx_state_time_"+states.get("cellId");
        data.setName(name);
        Long timeDif= this.timeDifference(states.get("dateTimeA").toString(),
                states.get("dateTimeB").toString());
        data.setValue(Long.toString(timeDif));
        data.setTimeStamp(TimeParser.parseTimestamp(states.get("dateTimeB").toString()));
        Set<MetaData> metaDataSet = new HashSet<MetaData>();
        /**
         * Omitting: Assign each value in the map to metaData objects
         * and put each metaData object in the metaDataSet */
        data.setMetadataSet(metaDataSet);

        Session session = sessionFactory.getCurrentSession();
        try {
            session.beginTransaction();
            session.save(data);
        }
        catch (RuntimeException e) {
            session.getTransaction().rollback();
            throw e;
        }
        finally {
            session.getTransaction().commit();
        }
        /**
         * Omitting: Assign each value in the map to a JSON object
         * for real time rendering in view
         */
    }
    /**
     * @param timeA in String
     * @param timeB in String
     * @return time difference in Long
     */
     private Long timeDifference(String timeA, String timeB){
            Timestamp timeStampA = TimeParser.parseTimestamp(timeA);
            Timestamp timeStampB = TimeParser.parseTimestamp(timeB);
            Long mili = timeStampB.getTime()-timeStampA.getTime();
            return mili/1000L;
    }
}
```

***Figure 21**: A listener for computing IPC-2541 state duration*

3.4.3. Implementation of web application

On the other hand, a web application is designed according to the Model View Controller design pattern, which runs on top of Apache Tomcat server. The Model is implemented in POJO, the Controller on top of Spring MVC and the View with JSP, HTML, jQuery and Google Chart Tools. The implementation of the web service is demonstrated in Figure 22.

- KPIs can be visualized on web browsers when users request with valid URLs.
- The *DispatcherServlet* delegates the URLs to the corresponding controllers and invokes the methods that are mapped for handling the URL requests.
- Each method calls a method in the service layer to retrieve data.
- To request historical data, the method in the service layer calls the Hibernate's *sessionFactory* CRUD functions.
- To request real-time data, the method in the service layer returns a *DeferredResult* object that is set by the Esper listener when Esper engine detects a match between the incoming events and the EPL rules.

- The responses are forwarded back to the service layer and further to the controller. The controller responds the *DispatcherServlet* with the model which assigns the model to its representative in the view.
- Finally, the *DispatcherServlet* returns responses in the format of HTML to the browser for visualization.

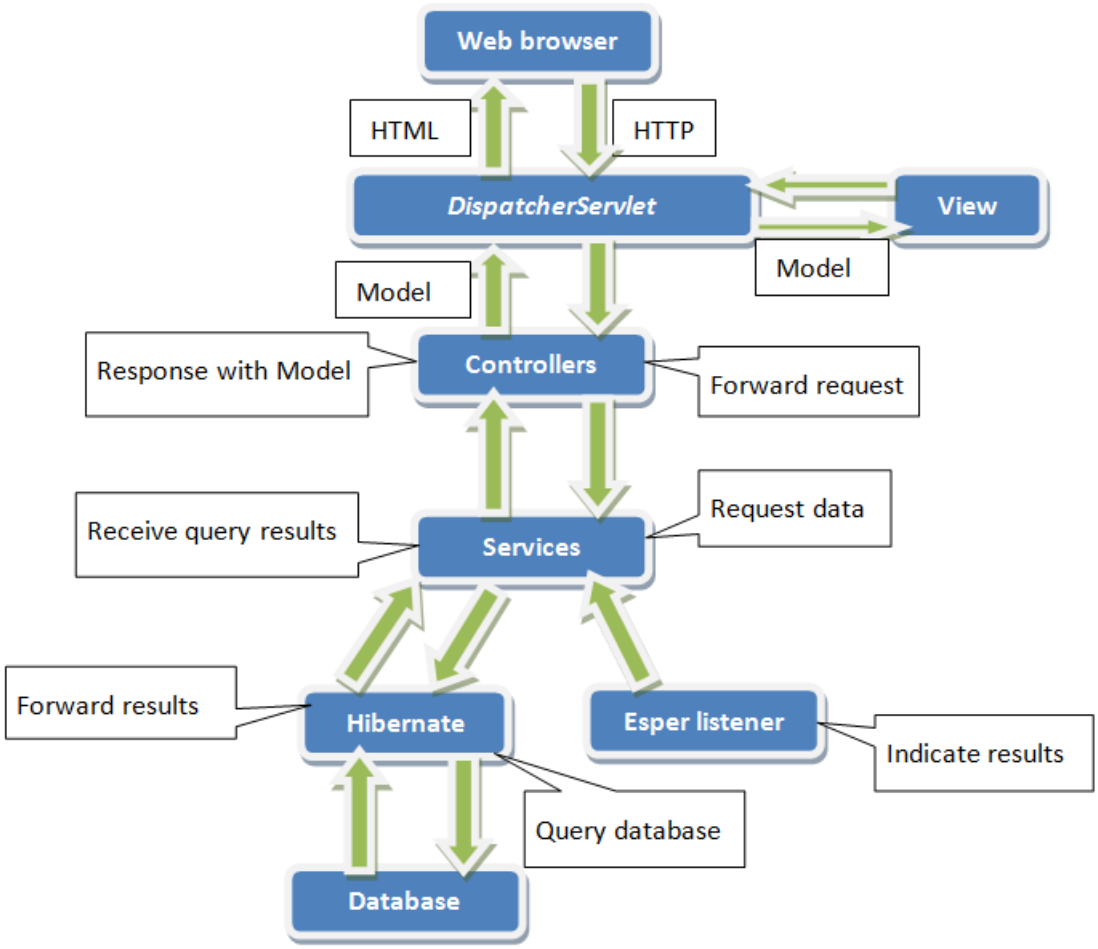


***Figure 22**: Flow chart of the implementation for web application*

Configuration for Spring MVC

Before explaining the implementation of Spring MVC's controller, it is necessary to explain the configuration for the Spring MVC framework in details using the basic XML file illustrated in Figure 23.

- First is to create Java beans for the service classes so that they can be referenced in the controller.
- To enable annotation scanning for controllers, *component-scan* is used here in which the package of the controller classes are specified.
- A *RequestMappingHandlerAdapter* is used here to process the annotations in the class level and method level. Message conveters are registered as the property of the *RequestMappingHandlerAdapter*, so the return of the method can be converted to, for example, JSON (using *jacksonMessageConverter* bean). The property

asyncRequestTimeout determines the duration of the await time for the value to be updated on the view. If the timeout occurs, a new request has to be sent to the controller to keep waiting for a new value.

- Lastly, the logical view name expressed as a String is mapped with the location of the view implementation using *InternalResourceViewResolver*.

```
<!--     Java bean for service  -->
<bean id="kpiViewService"
    class="fi.tut.kpimeter.spring.mvc.service.impl.KPIViewServiceImpl" />

<!--     Other beans for other services omitted-->
 ....

<!--     Enable annotation support in controller classes -->
<context:component-scan base-package="fi.tut.kpimeter.spring.mvc.controllers" />

<!--     Register message converters in an adapter  -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
     <property name="messageConverters">
     <list>
        <ref bean="marshallingHttpMessageConverter" />
        <ref bean="stringHttpMessageConverter"/>
        <ref bean="jacksonMessageConverter"/>
      </list>
      </property>
      <property name="asyncRequestTimeout" value="30000"/>
</bean>

<!--     Configuration of converter beans  -->
 ....

<!--     View name and view implementation mapping  -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/page/"/>
    <property name="suffix" value=".jsp" />
</bean>
```

***Figure 23**: Spring MVC basic configuration*

Implementation of controller

So as to the implementation of controllers, a description of a simplified *RuleController* (Figure 24) is introduced to explain how EPL rules stored in database tables are rendered on the view with the assistance of the *InternalResourceViewResolver*.

- The RuleController class is located in the package *fi.tut.kpimeter.spring.mvc.controllers* as is defined in *component-scan* in Figure 23.
- The class is annotated with *@Controller* for auto scanning.
- The method *setupPage* is annotated with *@RequestMapping* with its value being "/edit". As is introduced in section 3.4.1, any URL beginning with http://esonia-controller.rd.tut.fi:8080/FastoryService/mvc/ is handled by the *DispatcherServlet*. The specified request mapping in the method level ensures the URL http://esonia-controller.rd.tut.fi:8080/FastoryService/mvc/edit to be handled by the method.
- The method calls a *populatesRules(0)* method which requests all the pre-defined rules from the database and responds the result as a list. The rule list is added as an attribute, whose name is "*rules*", in the model. The Rule class is a POJO with fields (*id*, *ruleId*, *eplRule*, *adder*, *timestamp*) that can be accessed with getter and setter methods.
- The method returns a String ("edit") which is the logical view name. Then the *DispatcherServlet* interprets it, according to the *InternalResourceViewResolver*, as

"/WEB-INF/page/edit.jsp", locates it, assigns the model to the view representation and renders the view.

```
@Controller
public class RuleController {

    CEPRuleService cepRuleService;

    @Autowired
    public RuleController(CEPRuleService cepRuleService){
        this.cepRuleService=cepRuleService;
    }

    @RequestMapping(value="/edit", method = RequestMethod.GET)
    public String setupPage(Model model) {
        List<Rule> rules = cepRuleService.populateRules(0);
        model.addAttribute("rules", rules);
        return "edit";
    }

    /**
     * Omitting other methods
     */
}
```

***Figure 24**: Simplified implementation of RuleController*

Implementation of view

The sample code for the view template to render EPL rules is shown in Figure 25 to illustrate how the model is represented in the view.

- A link is created using an HTML "a" tag to specify the relative URL as "edit", mapping to the value of the *@RequestMapping* annotation used in controller. Users can click on the link to view the EPL rules.

```
<!-- from menu.jsp -->
<div id=body_top class="menubuttons">
    <ul>
        <!-- ....other links -->
        <li><a href="edit" id="edit" class="linkhover">CEP Rules</a>

        </li>
        <!-- ....other links -->
    </ul>
</div>

<!-- from edit.jsp -->
<div id="body_right">
    <table class="device_table">
      <tr id = "title" class="device_header"><td></td>
        <td>Name</td>
        <td>Author</td>
        <td>Time</td>
      </tr>
    <c:forEach items="${rules}" var="rule">
      <tr id="${rule.id}" class="message_head">
        <td></td>
        <td class="message_clicker">${rule.ruleId}</td>
        <td>${rule.adder}</td>  <td>${rule.timestamp}</td>
      </tr>
        <tr  class='message_body'><td colspan="4"><div class='message_div'>
        <p>${rule.eplRule}</p></div></td></tr>
      </c:forEach>
    </table>
</div>
```

***Figure 25**: Sample code in edit.jsp*

- The view template is located in "/WEB-INF/page/" as "edit.jsp" so that the *DispatcherServlet* can locate it.
- The EPL rules are rendered in a table using HTML tags.

- The model attribute "*rules*" (from Figure 24) is referenced using the notation "${rules}" in a *foreach* function from JSTL library so that the fields in each item (defined as *rule* with the "var" parameter) can be accessed in the view template.

Update graphics in real time

Google Chart Tools is used to represent data as graphics. For persistence purposes, historical data should be loaded in the graphics as the initial view. When new values are available from the production line, they should be represented in graphics without latency. Real-time update of graphics can be achieved using *DeferredResult* class in Spring 3.2.0.M1.

The IPC-2541 pie chart illustrates the accumulated duration of every IPC-2541 state in history. A state change adds a time difference to the sum duration of the state and updates the pie chart in real time. An example on the implementation of IPC-2541 pie chart is introduced in the following part. It begins with the description on how the controller handles the incoming HTTP requests, following the service layer and Esper listener implementation. Afterwards, how the HTTP requests are composed and sent from the view template and how to use Google Chart Tools to implement the pie chart for data rendering are introduced. The last part describes the sample codes for pie chart update when the server replies a JSON object.

In the *KPIViewController* (Figure 26), two methods are created for historical data and real-time data rendering in the pie chart, respectively.

```
@Controller
public class KPIViewController {
    private KPIViewService kpiViewService;

    @Autowired
    public KPIViewController(KPIViewService kpiViewService){
        this.kpiViewService=kpiViewService;
    }
    /**
     * Other methods are omitted
     */

    //Method used to request historical data for IPC-2541 pie chart
    @RequestMapping(value="/getCamxData", method = RequestMethod.GET)
    public @ResponseBody CAMXStateJsonResponse renderCAMXStateChart
        (@RequestParam String cellNum) {
        String colName="";
        /**
         * set camx state names according to the cell number.
         * when cellNum = "1", colName="camx_state_time_1",
         * the colName for camx state must be the same as the name stored in DB
         */
        CAMXStateJsonResponse csjr= kpiViewService.getCamxStatesTime(colName, cellNum);
        return csjr;
    }
    //Method used for real-time data in PIC-2541 pie chart
    @RequestMapping(value="/updateCamxState", method = RequestMethod.GET)
    public @ResponseBody DeferredResult updateCAMXStateChart
        (@RequestParam String cellNumber) {
        return kpiViewService.updateCAMXStateChart(cellNumber);
    }
}
```

***Figure 26**: Controller for IPC-2541 pie chart*

- The *renderCAMXStateChart* method requests historical data when the URL relative to the *DispatcherServlet* is "/getCamxData". The method assigns the cell number according to the request and forwards the cell number to the service layer. It returns

a *CAMXStateJsonResponse* POJO object to contain the IPC-2541 status information including the duration of each state, the cell number and a timestamp. The *@ResponseBody* annoation is used before the return type to ensure that the object is traslated to JSON type by the *RequestMappingHandlerAdapter*(configured in Figure 23).

- The *updateCAMXState* is used for real-time update of the IPC-2541 pie chart. It handles requests when the relative URL to the *DispatcherServlet* is "/updateCamxState". The method returns an object of *DeferredResult* class that is specifically used to support asynchronous servlet. The method merely passes the cell number to the service layer.

Two methods in the service layer (Figure 27) are created to handle requests from the methods above respectively.

```
@Override
public CAMXStateJsonResponse getCamxStatesTime(String colName, String cellNum) {
    CAMXStateJsonResponse csjr = new CAMXStateJsonResponse();
    CAMXStateToDuration camxStateDuration = new CAMXStateToDuration();
    /**
     * Omitting the Initialisation of camxStateDuration object by setting each state 0.0 second
     * to avoid nullPointerException.
     */
    //Request sum of each state duration from database
    List<CAMXStatesEntity> list = dataService.findCAMXStatesTime(colName);
    if(list != null && list.size()>0){
        for (int i=0; i<list.size(); i++){
            //set the actual time of each state in the camxStateDuration object.
            camxStateDuration.getCamxStateDuration().put(list.get(i).getState(),
                    list.get(i).getTotalTime());
            //set the sum time of OFF state in the JSON object- CAMXStateJsonResponse.
            if(list.get(i).getState().equals("OFF")){
                csjr.setOff(list.get(i).getTotalTime());
            }
            /**
             * Omit: Set the sum time of other state in the JSON object in a similar way.
             */
        }
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        Date now = new Date();
        csjr.setTimeStamp(dateFormat.format(now));
        csjr.setCellId(cellNum);
        csjr.setStatus("SUCCESS");

        //To keep the sum of each state from DB in memory
        FastoryServiceStatus.camxStateTimeAccumulator.put(cellNum, camxStateDuration);
    }else {
    csjr.setStatus("FAIL");
    }
    return csjr;
}
@Override
public DeferredResult updateCAMXStateChart(String cellNum) {
    DeferredResult deferredResult = new DeferredResult(new CAMXStateJsonResponse());
    DeferredResultContainer deferredResultContainer = new DeferredResultContainer();
    deferredResultContainer.setCellId(cellNum);
    deferredResultContainer.setDeferredResult(deferredResult);
    Date now = new Date();
    deferredResultContainer.setTimestamp(now.getTime());
    FastoryServiceStatus.camxStateDeferredResultMap.put(deferredResultContainer, now.getTime());
    return deferredResult;
}
```

***Figure 27**: Methods in service layer for IPC-2541 pie chart*

- The *getCamxStatesTime* method requests historical data from the Hibernate layer, assigns the values to the response list and returns it to the controller for rendering. In addition, the method also stores the historical data in memory (in a map object *camxStateTimeAccumulator*). In this way, a new value can be accumulated to the historical data simply by acquiring it from the memory without querying the database.

- The *updateCAMXStateChart* creates a *deferredResult* object and stores it in memory. It is worth noticing that an empty *camxStateJsonResponse* object is used as the input parameter of the constructor when creating the *deferredResult* object. This empty object plays a key role for real-time update. The empty *camxStateJsonResponse* object in the *deferredResult* enables the *DispatcherServelet* to wait without responding HTTP requests. When it is replaced with a new *camxStateJsonResponse* object populated with real-time data, the *DispatcherServelet* replies immediately. To distinguish different cells, the *deferredResult* is stored in a *deferredResultContainer* object which features a *cellId* field. In consideration with the fact that multiple users might be viewing the same chart simultaneously, the *deferredResultContainer* object is stored in the *camxStateDeferredResultMap*, to make sure that all the requests are responded with the real-time data.

As is motioned, the service layer requests historical data from database tables via a Hibernate layer which is implemented similarly to the one for Spring WS. Thurs the following paragraph only focuses on the implementation of the Esper listener which detects the IPC-2541 state change. Figure 28 illustrates the sample code. It merely aims to explain how the Epser listener results are updated in the *deferredResult* in real time, so other functions, such as storing the data in database tables, are removed.

- Once a match is detected by the Esper engine concerning the rule for detecting the state change (see Table XI, IPC-2541 states overview), the *update* method is invoked. Values regarding to the state change can be accessed from the input parameter.
- The method calculates the difference between two timestamps, and then accumulates the value to the historical value in *camxStateTimeAccumulator* map. Afterwards, it assigns the values after accumulation to the *CAMXStateJsonResponse* object.
- Thirdly, the method traverses all the *deferredResult* objects in the *camxStateRequestMap* and obtains the ones with the same cell number. They are temporarily recorded in a list. The *CAMXStateJsonResponse* object populated with the new values is set in all the *deferredResult* objects in the list, which ensures the *DispatcherServlet* to reply with the *CAMXStateJsonResponse* objects to all requests.
- Finally, the *deferredResult* objects in the temporary list are removed.

On the other hand, the URLs together with their parameters (known as HTTP requests), handled by *renderCAMXStateChart* and *updateCAMXState* methods in the controller layer, are created in the view template. They are different from the one introduced for EPL rule rendering example, in which the URL is created with a HTML "<a>" tag so that users can navigate to the page by clicking on the link. Considering the fact that the pie chart should be loaded and updated without user interference, these HTTP requests should be sent automatically from the view template. In addition, the process of sending the requests and retrieval of responses should not interfere with the

display and other processes on the current page. To achieve this, the requests have to be sent to the server asynchronously using jQuery's *get* and *ajax* methods.

```
public void update(Map states){
    String key = "Cell "+states.get("cellId").toString();
    Long timeDif= this.timeDifference(states.get("dateTimeA").toString(),
            states.get("dateTimeB").toString());

    //accumulate duration of states with a map
    CAMXStateJsonResponse csjr = new CAMXStateJsonResponse();
    if(FastoryServiceStatus.camxStateTimeAccumulator != null &&
            FastoryServiceStatus.camxStateTimeAccumulator.get(key) != null){

        Double totalTimeDif = FastoryServiceStatus.camxStateTimeAccumulator.get(key).
            getCamxStateDuration().get(states.get("state").toString());
        totalTimeDif+=Double.parseDouble(Long.toString(timeDif));
        /**
         * Omit: Replace the new time difference with the old in camxStateTimeAccumulator map
         */
        /**
         * Omit: Assign the new duration to CAMXStateJsonResponse object from the camxStateTimeAccumulator
         */
    }
    Map<DeferredResultContainer, Long> camxStateRequestMap = FastoryServiceStatus.camxStateDeferredResultMap;
    Iterator<DeferredResultContainer> caxmStateRequestIterator = camxStateRequestMap.keySet().iterator();
    while(caxmStateRequestIterator.hasNext()){
        DeferredResultContainer deferredResultContainer = caxmStateRequestIterator.next();

        if(deferredResultContainer.getCellId().equals(key)){
            //retain all deferredResult objects being handled in a list
            processedDeferredResult.add(deferredResultContainer);
            DeferredResult deferredResult = deferredResultContainer.getDeferredResult();
            if(deferredResult != null){
                try{
                    //set the deferredResultObject with the new CAMXStateJsonResponse object
                    deferredResult.trySet(csjr);
                }catch (Exception e){
                    e.printStackTrace();
                }
            }
        }
    }
    for(int i=0; i<processedDeferredResult.size(); i++){
        //clear the deferredResult objects that have been handled
        camxStateRequestMap.remove(processedDeferredResult.get(i));
    }
    processedDeferredResult.clear();

}
```

***Figure 28**: Sample code from CAMXStates listener*

To load the pie chart with the historical data, the request should be sent once the page renders. A function (shown in Figure 29) using jQuery's *get* method is implemented to request historical data from the controller.

```
<script type="text/javascript">
var cellNum = 'Cell 2';
$(document).ready(function(){
    requestCamxState(cellNum);

    // calling other functions...
});

var camxStateflag=true;
function requestCamxState(cellNum){
    $.get("getCamxData", {requestType: cellNum},  function(response){
        if(response.status=="SUCCESS"){
            drawCamxStatePieChart(response);
            if(camxStateflag){
                updateCamxState();
                camxStateflag = false;
            }
        }
    });
}
</script>
```

***Figure 29**: Sending HTTP requests using jQuery's get method*

- The *get* method accepts a URL (*getCamxData*) as the destination of the request, a map or string (*cellNum*) that is sent to the server with the URL and a callback function that handles the response from the server.
- In the callback function, a method *drawCamxStatePieChart* is called to draw the pie chart using the server response (*response*). The server response is the JSON object translated from *camxStateJsonResponse*.
- Then, it calls the *updateCamxState* method for chart updates.

Next is to illustrate the *drawCamxStatePieChart* function to present the server response as a pie chart using Google Chart Tools. Figure 30 shows the sample code.

- To use Google Chart Tools, an API (*jsapi*) needs to be imported from Google website.
- Next is to load the visualization API and the pie chart package from *corechart*.
- A container for the chart created with the HTML *div* tag uses the *id* attribute as the identifier.
- In the function, the JSON object is first assigned to a data table with the states as the first column and values the second.
- Then the chart object can be created using *ChartWrapper* which specifies the chart type as pie chart, the chart container's ID, the data table and options.
- Finally, the *draw* function is used to draw the chart.

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load('visualization', '1.1', {packages: ['corechart']});

var camxData ;
var camxPie;
function drawCamxStatePieChart(response) {

    camxData = google.visualization.arrayToDataTable
    ([
      ['state', 'duration in second'],
      ['OFF' , response.off],
      ['SETUP', response.setup],
      ['READY-IDLE-STARVED', response.starved],
      ['READY-IDLE-BLOCKED', response.blocked],
      ['READY-PROCESSING-ACTIVE', response.active],
      ['READY-PROCESSING-EXECUTING', response.executing],
      ['DOWN', response.down]
    ]);
    camxPie = new google.visualization.ChartWrapper({
      'chartType': 'PieChart',
      'containerId': 'camxPieChart',
      'dataTable': camxData,
      'options': {
        'title': response.cellId+' IPC-2541 States Overview',
        'animation':{'duration': 500, 'easing': 'inAndOut'},
        'is3D':true,
        'sliceVisibilityThreshold':0,
        'backgroundColor':'#FFF'
      },
      'view': {'columns': [0, 1]}
    });
   camxPie.draw();
  }
</script>
....
<div id="body_right">
    <div id="chartLayer">
        ....
        <div id = camxPieChart class="nextChart"></div>
    </div>
</div>
```

***Figure 30**: Processing server response into a pie chart using Google Chart Tools*

To update the pie chart with real-time data, HTTP requests should be sent to the server successively. The *updateCamxState* function (Figure 31) implements an asynchronous HTTP request to retrieve data from the server and updates the data table of the pie chart.

- It creates an HTTP request using jQuery's *ajax* function in which the URL along with the data sent to the server are specified.
- In the callback function, the server response can be accessed via the input parameter, with which, the data table is updated. Then one can reuse the chart object to update the chart with the new data table.
- Then the request is repeated by calling the function itself.

```
<script type="text/javascript">
function updateCamxState(){
    $.ajax({
        type:"GET",
        url:"updateCamxState",
        data:{requestType:cellNum},
        success: function(response){
            if(response.status == "SUCCESS"){
                camxData.removeRows(0,camxData.getNumberOfRows());
                camxData.addRows([
                                    ['OFF' , response.off],
                                    ['SETUP', response.setup],
                                    ['READY-IDLE-STARVED', response.starved],
                                    ['READY-IDLE-BLOCKED', response.blocked],
                                    ['READY-PROCESSING-ACTIVE', response.active],
                                    ['READY-PROCESSING-EXECUTING', response.executing],
                                    ['DOWN', response.down]
                                ]);
                camxPie.draw();
             setTimeout(
                     function(){updateCamxState();}, /* Request next message */
                     "50" /* ..after 50 ms */
                 );
            }else {
                setTimeout(
                        function(){updateCamxState();}, /* Request next message */
                        "50" /* ..after 50 ms */
                    );
            }
        },
        error: function(XMLHttpRequest, textStatus, errorThrown){
            setTimeout(
                    function(){updateCamxState();}, /* Try again after 50 ms */
                    50);
        }

    });
}
</script>
```

***Figure 31**: Update the IPC-2541 pie chart*

As is mentioned, the *deferredResult* object set with an empty *camxStateJsonResponse* object in the controller enables the *DispatcherServelet* to wait without responding HTTP requests until a new *camxStateJsonResponse* object populated with real-time data is set to the *deferredResult* or a timeout occurs. In case a populated *camxStateJsonResponse* is set in the *deferredResult*, the callback function in *updateCamxState* function receives the server response as a JSON object immediately. Thurs updating the pie chart in real time. On the other hand, when a timeout occurs, the callback function sends the request to the server again after 50 ms.

3.4.4. Implementation of RESTful web service

Figure 32 illustrates the implementation of the RESTful web service using Spring MVC.

- A client sends a URL request to the web service.
- The *DispatcherServlet* delegates the request to the corresponding handler to process the request. The request mapping is done via a *@RequestMapping* annotation declared before the handler method. For instance, by using *@RequstMapping*(value = "/data/{query}") in front of a handler, requests in the format of http://[host name]/[application name]/[servlet mappings]/data/{query}/ are forwarded to the method, in which *{query}* is a variable used to access the specific resource in the web service. In order to acquire the value of the *query* from the URL, *@PathVariable* annotation is used with the input parameter in the handler.

```
public KPIDataResponse handleKPIDataRequest(
                    @PathVariable(value="query") String query){...}
```

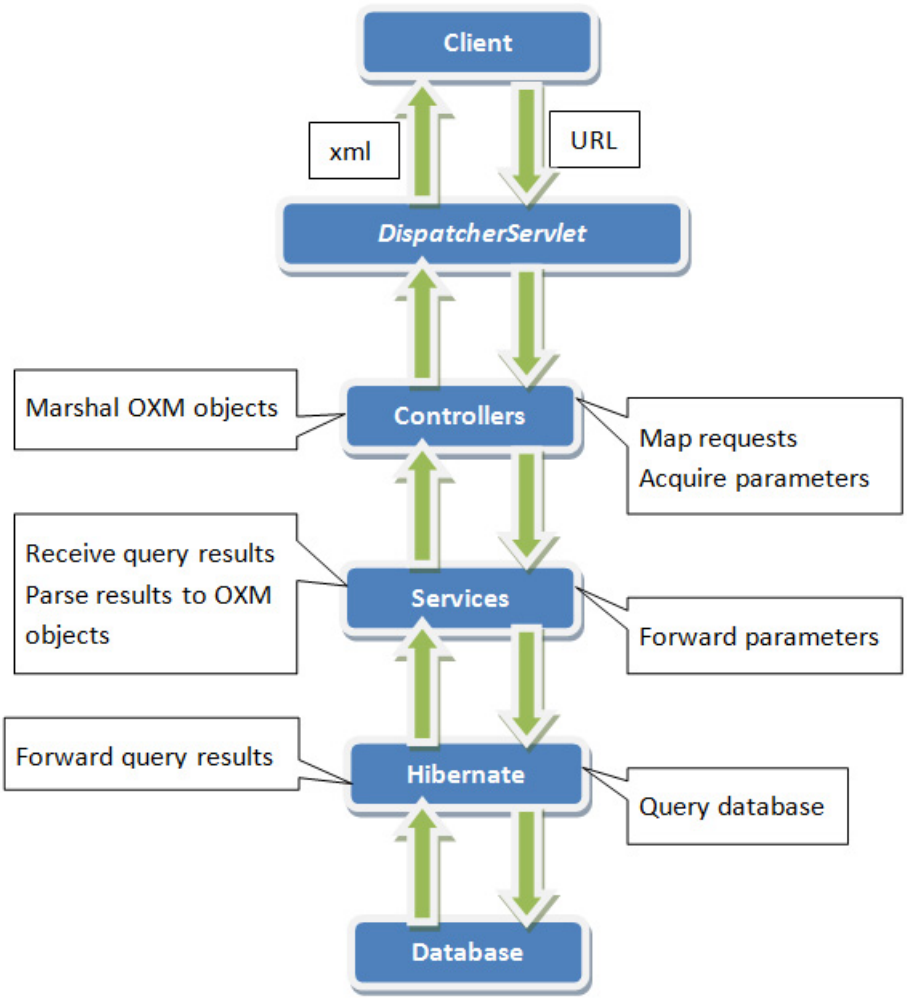


***Figure 32**: Implementation of RESTful web service*

- The parameters are forwarded by the controller to the service layer by calling a method defined in the service class to access database.
- Database access is triggered at the service layer. A Hibernate Query Language (HQL) is constructed and forwarded as the input parameter by calling Hibernate's DAO method to query database tables. In DAO layer, a set of interfaces and implementations is declared with CRUD functionalities utilizing Hibernate's *sessionFactory* instance.
- The Hibernate's DAO method returns the query results as a list.
- The list is further assigned to another Java object which is mapped with the XML in the service layer and subsequently forwarded to the controller for serialization of object to XML stream (marshalling).
- Finally, XML responses are returned to the client as HTTP response.

As can be seen, the implementation of RESTful web service using Spring MVC also relies on the design of controllers. Compared to the controllers for web applications, the controllers for RESTful web service returns XML stream. So there is no need to explain the implementation in details.

3.4.5. EPL rules for KPIs retrieval

EPL rules are designed for each lower-level granularity indicator associated to each area of concern (Table XI), and further mapped to testbed-related information needed as an input. Indicator values are calculated at run time as events are received. Computation of some indicators uses previously calculated results, while in certain cases it is desirable to define customized functions for each indicator of interest, from scratch. Patterns are used in the EPL rules to assist rule evaluation for most indicators.

Calculation of Unit Energy Consumption/ Process Energy Consumption requires two EPL rules, related to robot energy usage upon starting/finishing pallet assembly. The first rule keeps calculating the energy use by multiplying the average power consumption and the time interval between adjacent energy meter events and generates the result as a new event stream '*energyInstance*' using a user-defined function. The second rule is also designed using patterns. The first event in the pattern should be an *EquipmentChangeState* event whose current state is READY-PROCESSING-EXECUTING. Subsequently, the engine starts to gather every *energyInstance* event. The last event in the pattern has to be an *EquipmentChangeState* event whose previous state is READY-PROCESSING-EXECUTING. At this moment, the engine triggers the listener dedicated to this rule to indicate all the energy instances that have been collected so far, the time stamps of the beginning and ending of the execution process, the cell ID, the pallet ID and the recipe number. The sum of the energy instances is the energy consumption for processing a pallet (identified by the pallet ID) and executing a process (identified by the recipe number) in the cell.

Two event types are relevant for the Esper engine to evaluate Unit Production Time/ Unit Processing Time: the first *EquipmentChangeState* event whose IPC-2541 current

state is “READY-PROCESSING-EXECUTING” (i.e. processing has just started), and the immediately succeeding *EquipmentChangeState* event whose IPC-2541 previous state is “READY-PROCESSING-EXECUTING” (i.e. processing has just ended). The difference between the time stamps associated to the events is the product processing duration (indicated by the pallet ID)/ unit processing duration (indicated by the recipe number) in the considered cell.

Computation of Cell Production Rate is based on the number of “READY-PROCESSING-EXECUTING” states for robots within an hour. The result is grouped by cell ID to acquire the number of pallets being produced on each cell. The aggregation result regarding to cell number one indicates the Line Production Rate. The sum of the results of Cell Production Rate on each cell reveals the Total Products indicator.

Cell Energy Consumption is acquired directly from an *EnergyMeter* message originating in the meters installed in the line. EPL rules are not needed to compute Total Energy Consumption (it is computed as a simple sum over Cell Energy Consumption values for all cells).

To compute Energy Consumption per Product, two EPL rules are designed. The first one calculates the energy consumption of the pallet on each cell using the same pattern for calculating Unit Energy Consumption but generates each result as a new event stream. The second rule enables Esper engine to start gathering energy instances from the event streams related to a pallet when the pallet is being executed on cell one. When all the executing processes for this pallet are completed and the pallet flows back to cell number one, the Esper engine exposes all the gathered energy instances related to this pallet. The sum of all these energy instances is the energy consumption for this pallet. Meanwhile, the next round of energy instance gathering process starts.

Mean time to repair (MTTR) is the average time difference between the occurrence of a failure (IPC-2541 DOWN state) and the start of processing (IPC-2541 READY-PROCESSING-EXECUTING state) of the same working cell. To compute Mean Time Between Failures (MTBF), the average time difference between succeeding *EquipmentChangeState* events with a current state mark of “DOWN” is captured. Calculation of Mean Time to Failure (MTTF) relies on the average time difference between “READY-PROCESSING-EXECUTING” and “DOWN” states within the same working cell.

The EPL rule for frame quality, keyboard quality, screen quality and overall quality rate acquisition separately counts the number of *QualityInspectionShort* events when the *frameOk*, *keyOK*, *screenOk* or inspection passed attribute is one and divides the total number of *QualityInspectionShort* events within an hour.

The Pallet Production Time is computed as the time difference between the time stamps when a pallet being processed on cell one (current state in cell one is “READY-PROCESSING-EXECUTING”) and the same pallet is being processed on cell one.

IPC-2541 States Time calculates the duration of each IPC-2541 state in each cell using pattern. One *EquipmentChangeState* event of a robot cell indicates the start of one state as current state. The following *EquipmentChangeState* event from the same

cell whose previous state is the same with the current state indicates the end of the state. The Esper engine selects the time stamps in these two events and the current state value from the first event. The time difference between the two time stamps is the duration of the state. Production Shutdowns is revealed by the time difference between two "DOWN" states from the same cell.

***Table XI**: EPL rule designed for implementation*

KPI	Indicator	Input	EPL rule	Output
Efficiency	Unit Energy Consumption/ Process Energy Consumption	Energy use, states of working cells, cell ID and pallet ID, recipe number	**insert into** energyInstanceStream **select** energyInstance(a.AWATT, b.AWATT, a.dateTime, b.dateTime) as energyInstance, a.cellId as cellId, b.dateTime as dateTime **from pattern**[**every** a=EnergyMeter -> **every** b=EnergyMeter(cellId=a.cellId) **and not** EnergyMeter(cellId=a.cellId)] **group by** a.cellId	Cell ID, energy use and time stamp
			select a.energyInstance as energyInstance, b.cellId as cellId, b.palletId as palletId, b.recipeNum as recipeNumber, b.dateTime as startTime, c.dateTime as endTime **from pattern** [**every** b=EquipmentChangeState(currentState = "READY-PROCESSING-EXECUTING") -> **every** a=energyInstanceStream(cellId = b.cellId) **and not** EquipmentChangeState((currentState="READY-PROCESSING-EXECUTING" or previousState="READY-PROCESSING-EXECUTING"), cellId=b.cellId) -> c=EquipmentChangeState(previousState="READY-PROCESSING-EXECUTING", cellId=b.cellId)] **group by** b.cellId	Cell ID, energy use, pallet ID, recipe number and time stamps
	Unit Production Time/ Unit Processing Time	Time stamps and states of working cells	**select** a.currentState as state, a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.cellId as cellId, a.palletId as palletId, a.recipeNum as recipeNumber **from pattern**[**every** a=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING") -> b=EquipmentChangeState(previousState="READY-PROCESSING-EXECUTING" AND cellId=a.cellId)]	Starting and ending time stamps of working cells being executing, cell ID, pallet ID and recipe number
	Production Shutdowns	States of cells and time stamps	***The DOWN states duration in IPC-2541 states overview***	Starting and ending time stamps of working cells being off, cell ID and the state
	Cell production Rate/ Line Production Rate	States of working cells	**select count**(*) as count, cellId, **from** EquipmentChangeState.**win:time_batch**(3600 sec) **where** currentState="READY-PROCESSING-EXECUTING" group by cellId	Number of executing states of robots within an hour and cell ID
Energy	Cell Energy Consumption	Energy use and cell ID	**select** AWATTHR as robotEnergy, BWATTHR as controllerEnergy, CWATTHR as conveyorEnergy, cellId **from** EnergyMeter	Real time energy use for robots, conveyors and controllers as well as cell ID
	Energy Consumption per Product	Energy use, states of working cells, cell ID and pallet ID	**insert into** unitProductEnergyStream **select** a.energyInstance as energyInstance, b.cellId as cellId, b.palletId as palletId, b.dateTime as startTime, c.dateTime as endTime **from pattern** [**every** b=EquipmentChangeState(currentState = "READY-PROCESSING-EXECUTING") -> **every** a=energyInstanceStream(cellId = b.cellId) **and not** EquipmentChangeState((currentState="READY-PROCESSING-EXECUTING" or previousState="READY-PROCESSING-EXECUTING"), cellId=b.cellId) ->	Cell ID, energy use and time stamp

KPI	Indicator	Input	EPL rule	Output
			c=EquipmentChangeState(previousState="READY-PROCESSING-EXECUTING", cellId=b.cellId)] **group by** b.cellId	
			select a.energyInstance as energyInstance, b.cellId as cellId, b.palletId as palletId, b.dateTime as startTime, c.dateTime as endTime **from pattern**[**every** b=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING", cellId = "1") -> **every** a=unitProductEnergyStream(palletId=b.palletId) **and not** EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING", palletId=b.palletId, cellId = b.cellId) -> c=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING", cellId=b.cellId, palletId=b.palletId)] **group by** b.palletId	Cell ID, energy use, pallet ID and time stamp
Reliability	MTTR	States of cells and time stamps	**select** a.currentState as from_state, b.currentState as to_state, a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.cellId as cellId **from pattern**[**every** a=EquipmentChangeState(currentState="DOWN") -> b=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING" AND cellId=a.cellId)]	Starting and ending time stamp of working cells being down and the time stamp of working cells back to work, cell ID and state change
	MTTF	States of cells and time stamps	**select** a.currentState as from_state, b.currentState as to_state, a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.cellId as cellId **from pattern**[**every** a=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING") -> b=EquipmentChangeState(currentState="DOWN" AND cellId=a.cellId)]	The time stamp of working cells beginning to work and the time stamp when the cells being down, cell ID and state change
	MTBF	States of cells and time stamps	**select** a.currentState as from_state, b.currentState as to_state, a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.cellId as cellId **from pattern**[**every** a=EquipmentChangeState(currentState="DOWN") -> b=EquipmentChangeState(currentState="DOWN" AND cellId=a.cellId)]	Starting and ending time stamps of working cells being down, cell ID and state change
Quality	Frame Quality Rate	Quality of frame processing, quality of keyboard processing, quality of screen processing and production based quality inspection results	**select count**(frameOk, frameOk='1')/**count**(*) as frameRate, **count**(keyOk, keyOk='1')/**count**(*) as keyboardRate, **count**(screenOk, screenOk='1')/**count**(*) as screenRate, **count**(inspectionPassed, inspectionPassed='1')/**count**(*) as overallRate **from** QualityInspectionShort.**win:time_batch**(3600sec)	Number of products that pass the frame quality inspection
	Keyboard Quality Rate			Number of products that passes the keyboard quality inspection
	Screen Quality Rate			Number of products that passes the screen quality inspection
	Overall Quality Rate			Number of products that passes the overall quality inspection
Overall	Total Energy Consumption	Energy use of all cells	***Sum up the Cell Energy Consumption values for all cells***	Total energy consumption in Wh
	Pallet Production Time	States of working cell number 1	**select** a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.palletId as palletId **from pattern**[**every** a=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING", cellId = "1") -> b=EquipmentChangeState(currentState="READY-PROCESSING-EXECUTING",	Pallet ID, starting and ending processing time stamps of each pallet

KPI	Indicator	Input	EPL rule	Output
			cellId=a.cellId, palletId=a.palletId)]	
	Total Products	States of working cells	***Sum up the Cell Production Rate result of each cell***	Total number of products produced on each cell
	IPC-2541 States Overview	States of cells and time stamps	**select** a.currentState as state, a.dateTime as dateTimeA, b.dateTime as dateTimeB, a.cellId as cellId **from pattern**[**every** a=EquipmentChangeState -> b=EquipmentChangeState(previousState =a.currentState AND cellId = a.cellId)]	Cell ID, starting and ending time stamps of each state

3.4.6. Database structure

Captured events and KPIs are stored in database tables as resources of the RESTful web service. In addition, for persistence and user validation purposes, EPL rules and user information are also stored in database tables. Raw data and KPIs are stored in *data*, *metadata* and *data_metadata* tables, EPL rules in *rule_table* and user information in *user_table* and *user_authority_table*. Besides, because of the increase of the size of *data* and *metadata* tables, aggregation functions in MySQL such as 'sum', 'max' and 'min' become slow, which leads to slow rendering of certain graphics, statistics tables are created to speed up the data retrieval process.

Database tables for data and KPIs

The database stores raw data and KPI values in three tables (Figure 33): a *data* table, a *metadata* table and a *data_metadata* table.

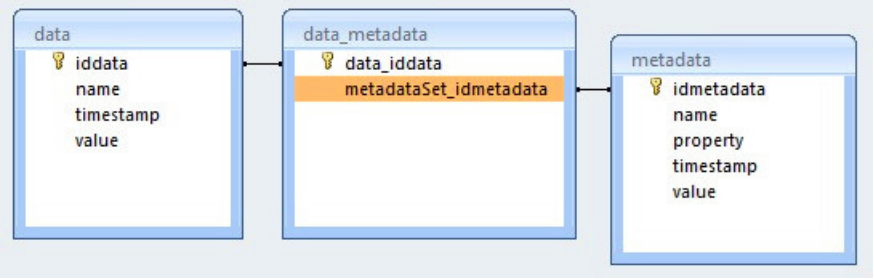


***Figure 33**: Database tables for data: structure and relation*

The *data* table stores:

- the root element name of the incoming xml messages / KPI name (the name column)
- a manufacturing cell identifier (the name column)
- timestamps associated to incoming messages from the line / KPI values (the timestamp column)
- the addresses of the messages / KPI results (the value column)

The *metadata* table stores:

- the values of attributes / elements within incoming messages (the value column).
- the names of the attributes / elements are stored (the property column). In the case of a KPI, the property and value columns are used to further describe the KPI (e.g. via cell IDs / device types / etc.)

- Similar content to the data table in its name and timestamp columns. These columns are left for future usage in case the line needs to be retrofitted.

Figure 34 illustrates a shortcut of how an *EnergyMeter* message is stored in *data* and *metadata* tables. One incoming event increases one row in *data* table, while several rows in *metadata* table, which represents a one-to-many relationship. In order to map the data in *data* table and in *metadata* table, a third table is needed.

***data* Table**

44466	energyMeter5	05/03/2012 17:49:26	http://[2001:708:310:7753:250:C...

***metadata* Table**

1000425	energyMeter5	CWATT	05/03/2012 17:49:26	537.25
1000426	energyMeter5	CVAHR	05/03/2012 17:49:26	21
1000427	energyMeter5	AWATT	05/03/2012 17:49:26	3.70
1000428	energyMeter5	BVA	05/03/2012 17:49:26	73.86
1000429	energyMeter5	BVRMS	05/03/2012 17:49:26	239.32
1000430	energyMeter5	CVRMS	05/03/2012 17:49:26	238.09
1000431	energyMeter5	AVAR	05/03/2012 17:49:26	-93.41
1000432	energyMeter5	CVA	05/03/2012 17:49:26	1098.99
1000433	energyMeter5	CWATTHR	05/03/2012 17:49:26	9
1000434	energyMeter5	AVARHR	05/03/2012 17:49:26	-1

***Figure 34**: Example of data in database*

The *data_metadata* table stores the relations of the data in the above mentioned two tables via id numbers. The table is generated by Hibernate automatically by setting a one-to-many relationship in Hibernate's configuration file or annotating the relationship in POJO with *@OneToMany* annotation. The relationship represented by the *data_metadata* table for the above presented example is shown in Figure 35.

***data_metadata* Table**

44466	1000425
44466	1000426
44466	1000427
44466	1000428
44466	1000429
44466	1000430
44466	1000431
44466	1000432
44466	1000433
44466	1000434

***Figure 35**: One-to-many relation correlation with data_metadata table*

Database table for EPL rules

The *rule_table* (Figure 36) stores EPL rules, including the name of the rule in *ruleid* column, the actual rule in *rule*, the user who added the rule in *adder* and the timestamp indicating the creation time of the rule in *timestamp*.

***Figure 36**: Database tables for rules: structure*

Database tables for users

The user information is stored in two tables (Figure 37). The *user_table* stores the information upon users' registry, including company, email address, first name, last name, password, role of the user and user name. The role of the user in *user_table* can be either 1 or 2 with 1 representing an administrator and 2 a user. This mapping is stored in the *user_authority_table*, which only has two rows (Figure 38).

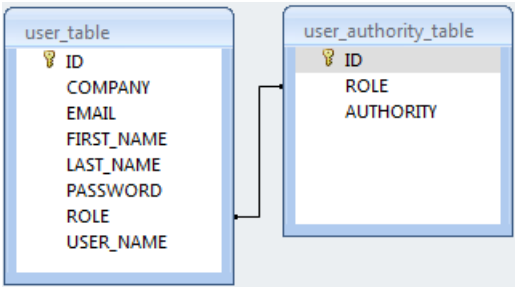


***Figure 37**: Database tables for users: structure and relation*

ID *	ROLE *	AUTHORITY *
1	1	ROLE_ADMIN
2	2	ROLE_USER

***Figure 38**: Data in user_authority_table*

Statistics tables

Four statistics tables are created to speed up aggregation functions, namely, *camx_statistics_table*, *device_statistics_table*, *energy_statisitcs_table* and *production_rate_statistics_table*. The *camx_statistics_table*, the *energy_statistics_table* and the *production_rate_statistics_table* aim to hold the total duration of each IPC-2541 state in each cell, the maximum energy consumption value in each cell, the total number of products manufactured in each cell, respectively, while the *device_statistics_table* maintains the address, the name, the first message timestamp and the last message timestamp from each device. The tables are updated with Hibernate queries when an update in the information concerned occurs, to maintain the latest information in the tables.

3.4.7. Data persistence with Hibernate

On one hand, the application uses Java as the programming language which manipulates data as Java objects. On the other hand, the database regards its data as relational data.

Therefore, Hibernate integrated into Spring framework is adopted to manage the mapping known as ORM.

Integrating Hibernate into Spring framework

Hibernate can be configured via Spring's IoC module by creating Spring beans in a configuration file (Figure 39).

- A *LocalSessionFactoryBean* provides a *sessionFactory* instance which manages the configuration settings of Hibernate and provides CRUD operations for application services. The Hibernate configuration settings can be loaded from an XML file (Figure 40) located in the root of the resource folder. The *configLocation* property defines the location of the XML file.
- Transaction management provided by Spring's AOP module ensures the integrity and consistency of data in database. To enable the transaction management in Hibernate and Spring integration, a *TransactionInterceptor* bean and a *HibernateTransactionManager* bean are used. The *HibernateTransactionManager* bean defines the Hibernate transaction manager and references the *sessionFactory* instance as its property to enable Hibernate's transaction management, while the *TrasactionInterceptor* references the *HibernateTransactionManager* bean as a property to manage Hibernate's transaction management in Spring framework. In addition, the *transactionAttributes* property has to be configured to indicate how the transaction behaves when a nested transaction occurs (transaction propagation). The key property defines the method that utilizes the transaction propagation. For example, 'find*' means that all the methods that starts with 'find' utilize such propagation behaviour. The transaction propagation behaviour defined in this work is 'PROPAGATION_REQUIRED' which supports the current transaction and if no transaction exists, it creates a new one.
- The XML file lists the Hibernate connection parameters, such as driver class, URL, user name and password, the Hibernate connection provider related settings (connection provider class, maximum/minimum poll size for the database, etc), transaction provider class as properties. A list of ORM classes is also registered in the *sessionFacotry* instance as mappings.

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <!-- <property name="dataSource" ref="dataSource" />  -->
    <property name="configLocation">
        <value>classpath:hibernate.cfg.xml</value>
    </property>
</bean>

<bean id="transactionInterceptor"
   class="org.springframework.transaction.interceptor.TransactionInterceptor">
    <property name="transactionManager" ref="transactionManager" />
    <property name="transactionAttributes">
       <props>
        <prop key="save">PROPAGATION_REQUIRED</prop>
        <prop key="find*">PROPAGATION_REQUIRED</prop>
        <prop key="delete">PROPAGATION_REQUIRED</prop>
       </props>
    </property>
</bean>
```

***Figure 39**: Hibernate configuration*

```
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">
        org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">
        jdbc:mysql://130.230.141.228:3306/kpimeter</property>
        <property name="hibernate.connection.username">esonia</property>
        <property name="hibernate.connection.password">ainose</property>

        <property name="hibernate.connection.release_mode">auto</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.generate_statistics">true</property>
        <property name="hibernate.transaction.factory_class">
        org.hibernate.engine.transaction.internal.jdbc.JdbcTransactionFactory</property>
        <property name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.default_batch_fetch_size">16</property>
        <property name="connection.provider_class">
        org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider</property>

        <property name="c3p0.acquire_increment">2</property>
        <property name="c3p0.idle_test_period">300</property> <!-- seconds -->
        <property name="c3p0.max_size">100</property>
        <property name="c3p0.max_statements">100</property>
        <property name="c3p0.min_size">0</property>
        <property name="c3p0.timeout">5000</property>

        <mapping class="fi.tut.kpimeter.hibernate.model.Data" />
        <mapping class="fi.tut.kpimeter.hibernate.model.MetaData" />
        <mapping class="fi.tut.kpimeter.hibernate.model.User" />
        <mapping class="fi.tut.kpimeter.hibernate.model.Rule" />
        <mapping class="fi.tut.kpimeter.hibernate.model.PalletTable" />
        <mapping class="fi.tut.kpimeter.hibernate.model.Message" />
    </session-factory>
</hibernate-configuration>
```

***Figure 40**: hibernate.cfg.xml*

ORM in Hibernate

A Java object can be mapped to relational data using JPA annotations supported by Hibernate. An example of using the annotations can be found in Figure 41.

- A *@Entity* annotation at the class level enables a POJO class to be a persistent entity.
- A *@Table* annotation at the class level allows mapping the name of the persistent class to a database table name.
- A *@Column* annotation before an attribute indicates that the attribute is mapping to the specified column.
- A *@Id* annotation before an attribute informs Hibernate that the values in the corresponding column in this table are unique in each row. This column is also defined as the primary key in this table.
- A *@GeneratedValue* annotation is declared before an attribute to declare that the value in this column is generated automatically.
- A *@Index* annotation before an attribute indicates that the mapped column in the table uses indexing technology to increase the lookup speed.
- A *@OneToMany* annotation before an attribute is used to correlate another table as a one-to-many relationship.
- Besides, every attribute in the POJO class needs a setter and a getter method.

```
@Entity
@Table(name="data")
public class Data{

    @Id
    @Column(name="iddata")
    @GeneratedValue
    private Long id;
    @Column(name="value")
    private String value;
    @Column(name="name")
    @Index(name = "dataNameIndex")
    private String name;
    @Column(name="timestamp")
    @Index(name = "timestampIndex")
    private Timestamp timeStamp;
    @OneToMany(cascade= CascadeType.ALL, fetch = FetchType.LAZY)
    @BatchSize(size = 80)
    private Set<MetaData> metadataSet;


/**
 * getter and setter methods for all the fields
 */
```

***Figure 41**: The use of annotations for ORM*

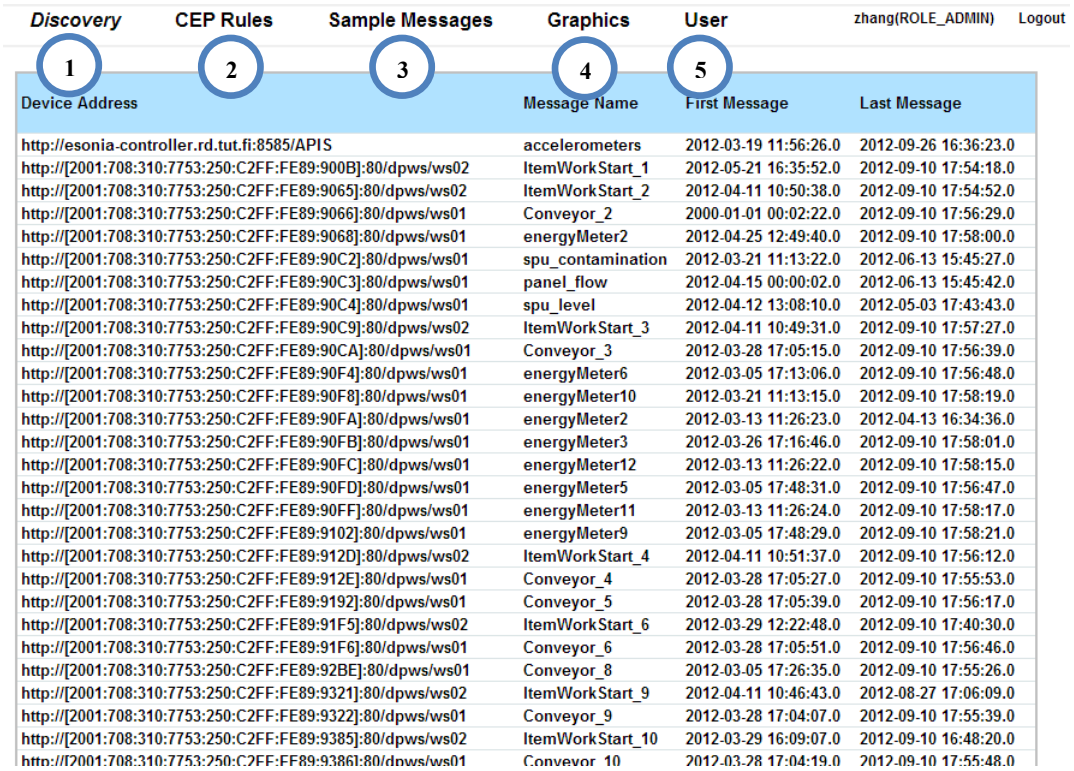
4. RESULTS

This chapter shows the results of the implementation for the web application for KPI management. Then it shows the results of exposing the KPIs in a RESTful web service.

4.1. Results of the web application

The web application is available at the URL: http://esonia-controller.rd.tut.fi:8080/FastoryService. After login to the application, users can visit the 'Discovery' page. As can be seen from the menu on top of Figure 42, the application provides the following functions:

1. Monitoring of device information.
2. Management of CEP rules.
3. Illustration of sample messages.
4. Visualization of historical KPIs and real-time monitoring of KPIs.
5. Management of user account information.



Device Address	Message Name	First Message	Last Message
http://esonia-controller.rd.tut.fi:8585/APIS	accelerometers	2012-03-19 11:56:26.0	2012-09-26 16:36:23.0
http://[2001:708:310:7753:250:C2FF:FE89:900B]:80/dpws/ws02	ItemWorkStart_1	2012-05-21 16:35:52.0	2012-09-10 17:54:18.0
http://[2001:708:310:7753:250:C2FF:FE89:9065]:80/dpws/ws02	ItemWorkStart_2	2012-04-11 10:50:38.0	2012-09-10 17:54:52.0
http://[2001:708:310:7753:250:C2FF:FE89:9066]:80/dpws/ws01	Conveyor_2	2000-01-01 00:02:22.0	2012-09-10 17:56:29.0
http://[2001:708:310:7753:250:C2FF:FE89:9068]:80/dpws/ws01	energyMeter2	2012-04-25 12:49:40.0	2012-09-10 17:58:00.0
http://[2001:708:310:7753:250:C2FF:FE89:90C2]:80/dpws/ws01	spu_contamination	2012-03-21 11:13:22.0	2012-06-13 15:45:27.0
http://[2001:708:310:7753:250:C2FF:FE89:90C3]:80/dpws/ws01	panel_flow	2012-04-15 00:00:02.0	2012-06-13 15:45:42.0
http://[2001:708:310:7753:250:C2FF:FE89:90C4]:80/dpws/ws01	spu_level	2012-04-12 13:08:10.0	2012-05-03 17:43:43.0
http://[2001:708:310:7753:250:C2FF:FE89:90C9]:80/dpws/ws02	ItemWorkStart_3	2012-04-11 10:49:31.0	2012-09-10 17:57:27.0
http://[2001:708:310:7753:250:C2FF:FE89:90CA]:80/dpws/ws01	Conveyor_3	2012-03-28 17:05:15.0	2012-09-10 17:56:39.0
http://[2001:708:310:7753:250:C2FF:FE89:90F4]:80/dpws/ws01	energyMeter6	2012-03-05 17:13:06.0	2012-09-10 17:56:48.0
http://[2001:708:310:7753:250:C2FF:FE89:90F8]:80/dpws/ws01	energyMeter10	2012-03-21 11:13:15.0	2012-09-10 17:58:19.0
http://[2001:708:310:7753:250:C2FF:FE89:90FA]:80/dpws/ws01	energyMeter2	2012-03-13 11:26:23.0	2012-04-13 16:34:36.0
http://[2001:708:310:7753:250:C2FF:FE89:90FB]:80/dpws/ws01	energyMeter3	2012-03-26 17:16:46.0	2012-09-10 17:58:01.0
http://[2001:708:310:7753:250:C2FF:FE89:90FC]:80/dpws/ws01	energyMeter12	2012-03-13 11:26:22.0	2012-09-10 17:58:15.0
http://[2001:708:310:7753:250:C2FF:FE89:90FD]:80/dpws/ws01	energyMeter5	2012-03-05 17:48:31.0	2012-09-10 17:56:47.0
http://[2001:708:310:7753:250:C2FF:FE89:90FF]:80/dpws/ws01	energyMeter11	2012-03-13 11:26:24.0	2012-09-10 17:58:17.0
http://[2001:708:310:7753:250:C2FF:FE89:9102]:80/dpws/ws01	energyMeter9	2012-03-05 17:48:29.0	2012-09-10 17:58:21.0
http://[2001:708:310:7753:250:C2FF:FE89:912D]:80/dpws/ws02	ItemWorkStart_4	2012-04-11 10:51:37.0	2012-09-10 17:56:12.0
http://[2001:708:310:7753:250:C2FF:FE89:912E]:80/dpws/ws01	Conveyor_4	2012-03-28 17:05:27.0	2012-09-10 17:55:53.0
http://[2001:708:310:7753:250:C2FF:FE89:9192]:80/dpws/ws01	Conveyor_5	2012-03-28 17:05:39.0	2012-09-10 17:56:17.0
http://[2001:708:310:7753:250:C2FF:FE89:91F5]:80/dpws/ws02	ItemWorkStart_6	2012-03-29 12:22:48.0	2012-09-10 17:40:30.0
http://[2001:708:310:7753:250:C2FF:FE89:91F6]:80/dpws/ws01	Conveyor_6	2012-03-28 17:05:51.0	2012-09-10 17:56:46.0
http://[2001:708:310:7753:250:C2FF:FE89:92BE]:80/dpws/ws01	Conveyor_8	2012-03-05 17:26:35.0	2012-09-10 17:55:26.0
http://[2001:708:310:7753:250:C2FF:FE89:9321]:80/dpws/ws02	ItemWorkStart_9	2012-04-11 10:46:43.0	2012-08-27 17:06:09.0
http://[2001:708:310:7753:250:C2FF:FE89:9322]:80/dpws/ws01	Conveyor_9	2012-03-28 17:04:07.0	2012-09-10 17:55:39.0
http://[2001:708:310:7753:250:C2FF:FE89:9385]:80/dpws/ws02	ItemWorkStart_10	2012-03-29 16:09:07.0	2012-09-10 16:48:20.0
http://[2001:708:310:7753:250:C2FF:FE89:9386]:80/dpws/ws01	Conveyor_10	2012-03-28 17:04:19.0	2012-09-10 17:55:48.0

Figure 42: *Device Information*

The 'Discovery' page (Figure 42) lists device information in a table. The first column of the table is the address of the devices, followed by the names of messages

that are published by the devices' services. The last two columns show the time stamps of the first message that the server has received and the time stamps of the most recent messages respectively. By clicking on each row, more detailed information concerning the corresponding device is shown such as the hosting action.

Figure 43 shows a snapshot of the 'CEP Rules' page in which there are two links on the left side, 'Embedded Rules' and 'User Defined Rules'. By default, the 'Embedded Rules' link is enabled in which the EPL rules for KPI retrieval defined by the author are collected in a table. The first column of the table holds the rule names, followed by the user name of the administrator who created the rule and the time when the rules are defined. By clicking on each row, users can see the corresponding EPL rule. An example of how to define a new rule is described in section 4.3.

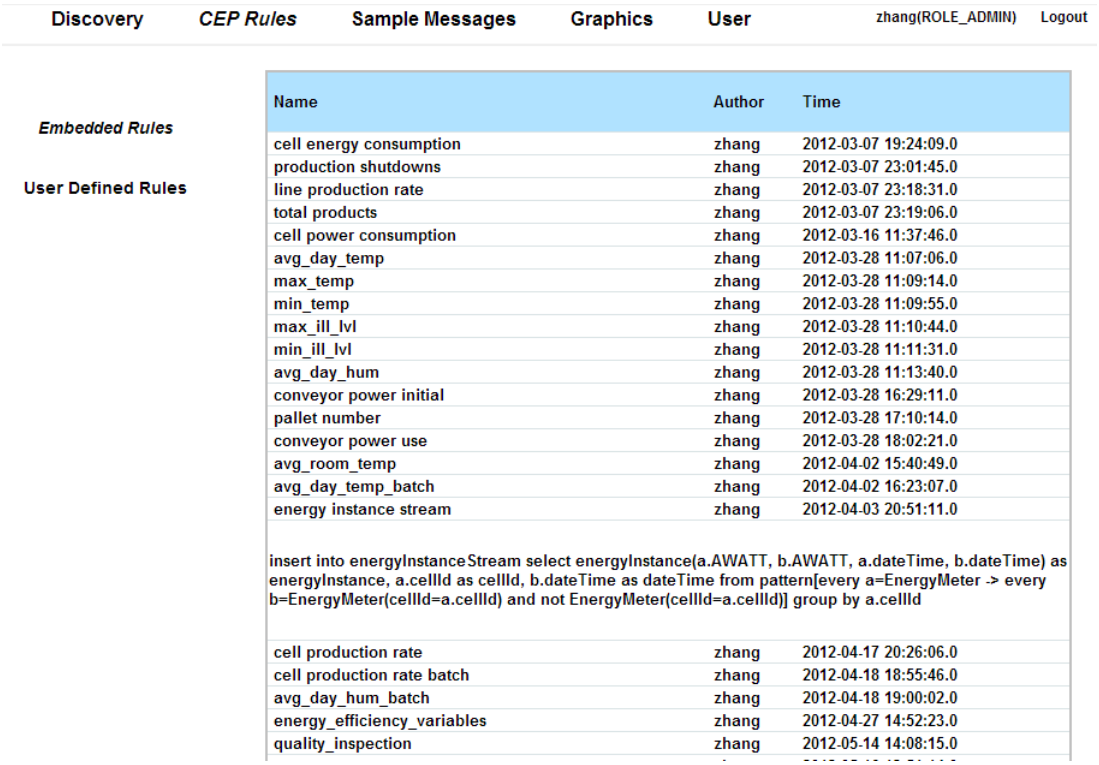


Figure 43: *CEP rules*

Samples of all the messages and descriptions are shown in the page 'Sample Messages' (Figure 44) to help administrators define EPL rules. One can view the sample message and its description by clicking on the name which is also the root element of each XML message. The sample message illustrates the root element, all the attributes and their sample values in the message.

Administrators can view, create and delete user accounts on the 'User' page. This page shows the user name, full name and role in a table. The 'Create' and 'Delete' buttons enable administrators to create user accounts by filling a registry form and delete user accounts, respectively.

Figure 44: *Sample messages*

The 'Graphics' button brings the users to the monitoring of KPIs. The default page (Figure 45) illustrates the users with the indicators belonging to 'Overall'. The buttons ('Efficiency', 'Energy', 'Reliability', 'Quality' and 'Overall') enable users to navigate among all indicators in these categories. A drop down list on the right hand side enables users to select a specific cell. The label on top of each chart enables the graphics to slide up or down, thus moving up or down the succeeding chart.

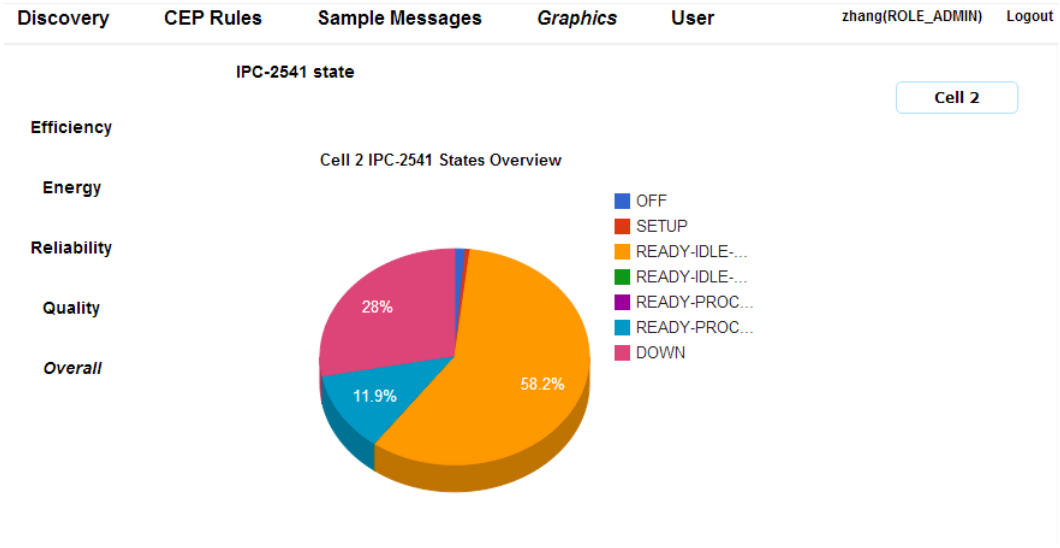


Figure 45: *Graphics*

4.1.1. Visualization of efficiency indicators

The Efficiency session is dedicated to the visualization of indicators for unit energy consumption, process energy consumption, unit production time, unit processing time, production shutdowns and cell production rate.

Unit energy consumption/Process energy consumption

Both historical and runtime unit energy consumption/process energy consumption of a single cell can be visualized with smooth line charts annotated with both pallet ID and recipe number.

Figure 46 illustrates the historical unit energy consumption/process energy consumption in watt hour (Wh) in which a drop down list (Pallet ID) enables user to inspect the energy consumption of a specific pallet, while another drop down list (Recipe) is dedicated to the inspection of energy consumption of a specific process. By specifying a time period, the unit energy consumption/process energy consumption in the time period can be visualized. The range filter enables users to scale the chart with time.

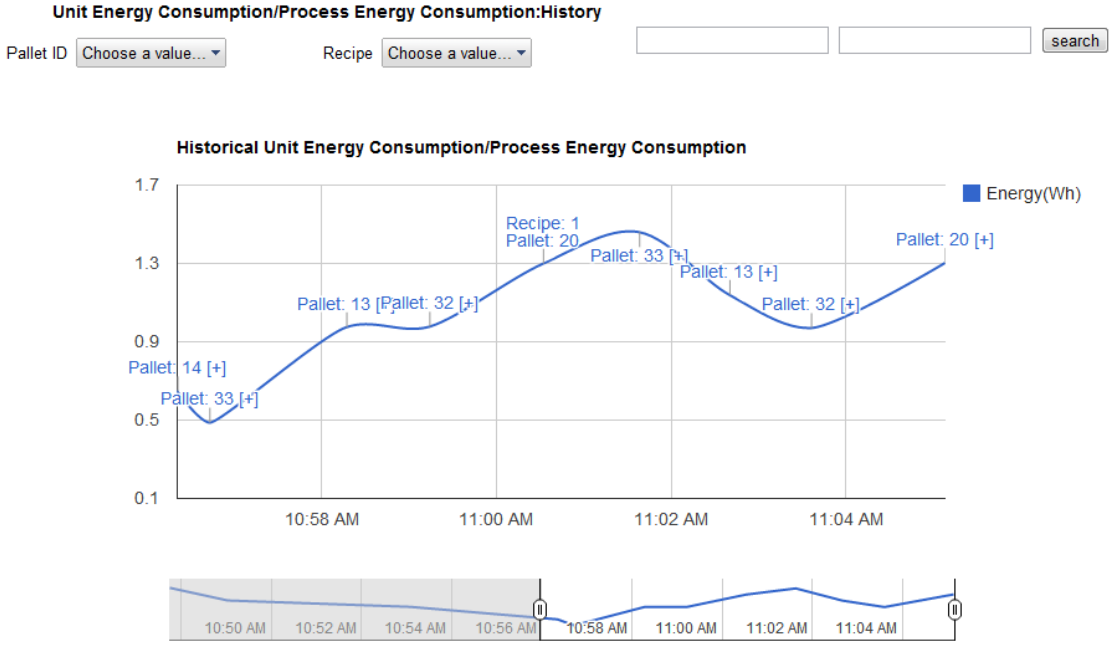


***Figure 46**: Historical unit energy consumption line chart*

The real-time unit energy consumption/process energy consumption smooth line chart (Figure 47) illustrates the unit energy consumption/process energy consumption of a single cell in real time. As a cell completes assembling a pallet, the energy consumption and its annotations are appended to the line.

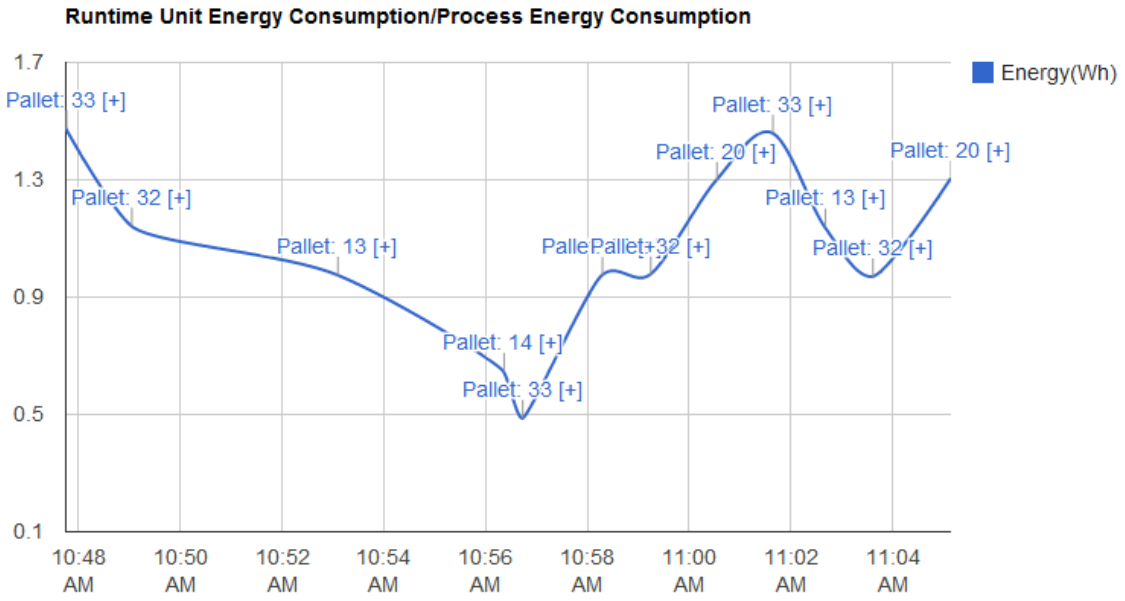


***Figure 47**: Run time unit energy consumption line chart*

Unit production time/Unit processing time

The unit production time/unit processing time in second is displayed with a line chart annotated with pallet ID and recipe number.

By default, the historical unit production time/unit processing time line chart (Figure 48) shows the unit production time/unit processing time within the last 30 minutes since the user opens this chart. However, users can specify a time period by filling out the starting and ending time in the text field. A drop down list can be used to inspect the production time of a specific pallet ID, while another one enables users to visualize the production time of a specific process. A range filter enables users to scale the chart with time.

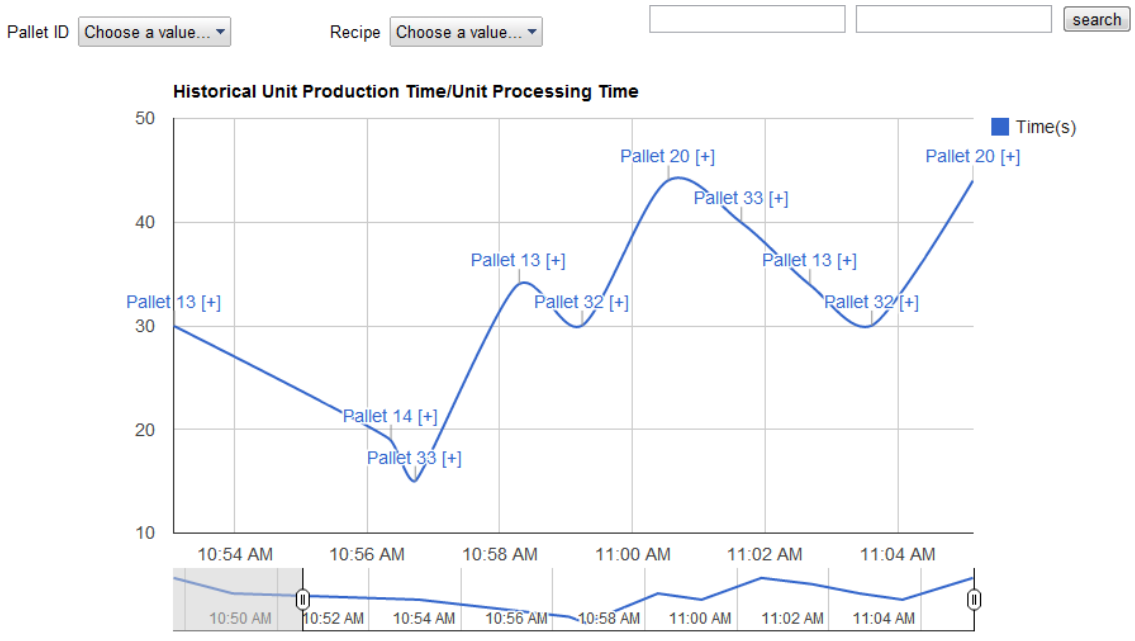


***Figure 48**: Historical unit production time line chart*

Another line chart (Figure 49) provides real-time monitoring of the unit production time/unit processing time. As each cell completes producing one pallet, the unit production time/unit processing time is added to the end of the line.

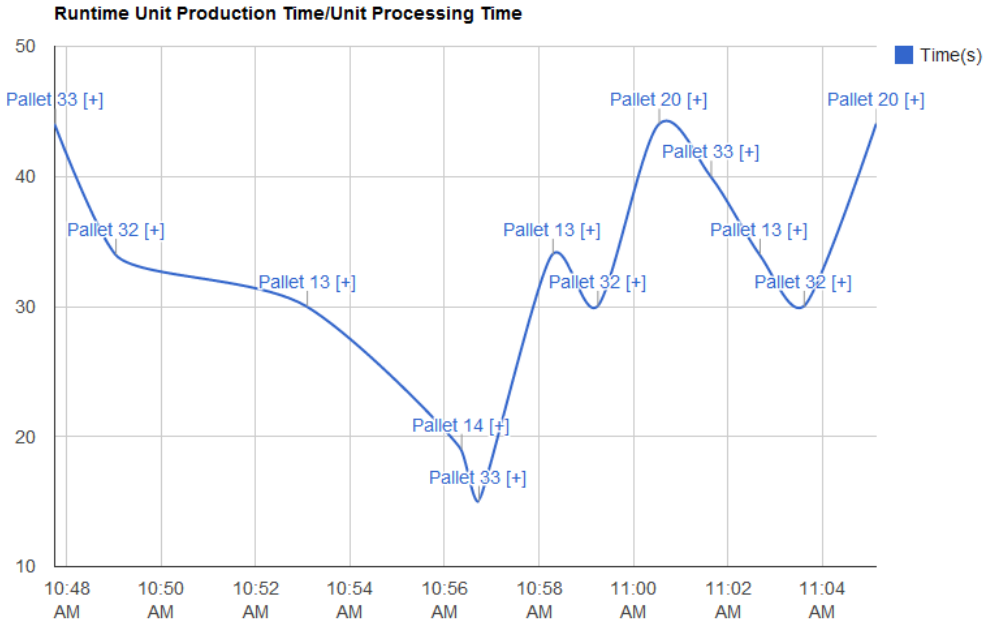


***Figure 49**: Real time unit production time line chart*

Cell production rate

The cell production rate can visualized in plain texts, which indicates the current production rate with numbers and its time stamp, a progress bar, which illustrates the current cell production rate as a bar, and a column chart that shows the historical cell production rate on an hourly basis (Figure 50). The historical cell production rate column chart shows the cell production rate in the past 24 hours by default. However, users can specify a time period by filling out the starting and ending time in the text field.

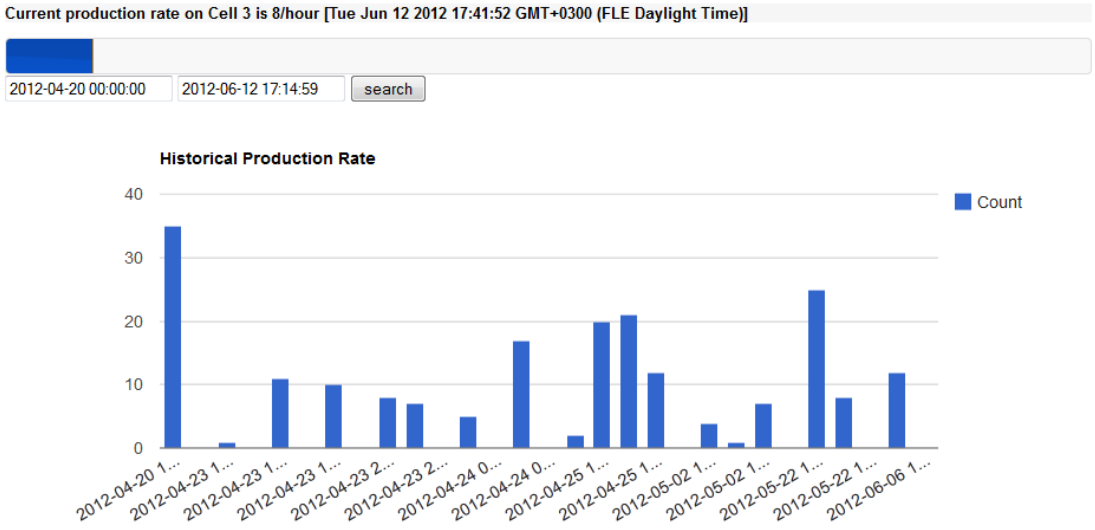


***Figure 50**: Visualization for cell production rate*

4.1.2. Visualization of energy indicators

The Energy session is organized to demonstrate indicators of cell energy consumption and unit energy consumption. In addition, power energy consumption (both historical and run time) is also visualized in this session.

Power consumption

The power consumption for cabinet, conveyor and robot is shown as a line chart (for both historical data and run time data) in watt (W).

In the historical power consumption chart (Figure 51), the Robot, Conveyor and Cabinet buttons enable users to inspect the power trend from the three components separately while an 'All' button resets the chart to reveal all the three components. By specifying a time period in the text fields, users can search for the power consumption trend in the specified time period. A range filter below the line chart provides scaling capability to the chart.

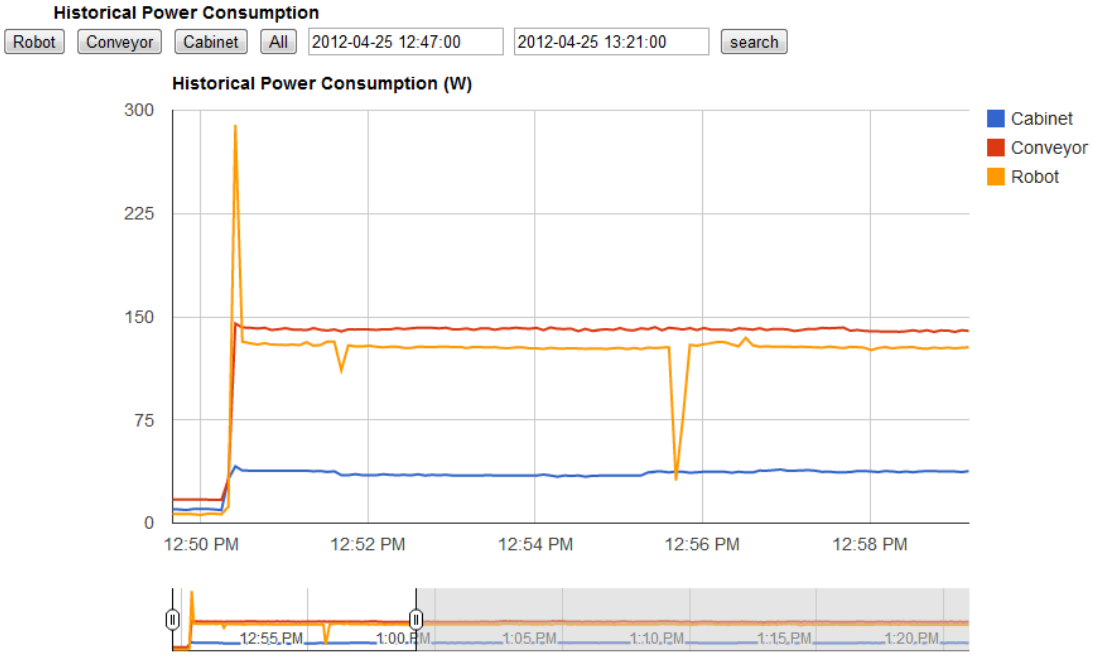


***Figure 51**: Historical power consumption line chart*

With the runtime power consumption chart (Figure 52), users can inspect the power consumption in real time.

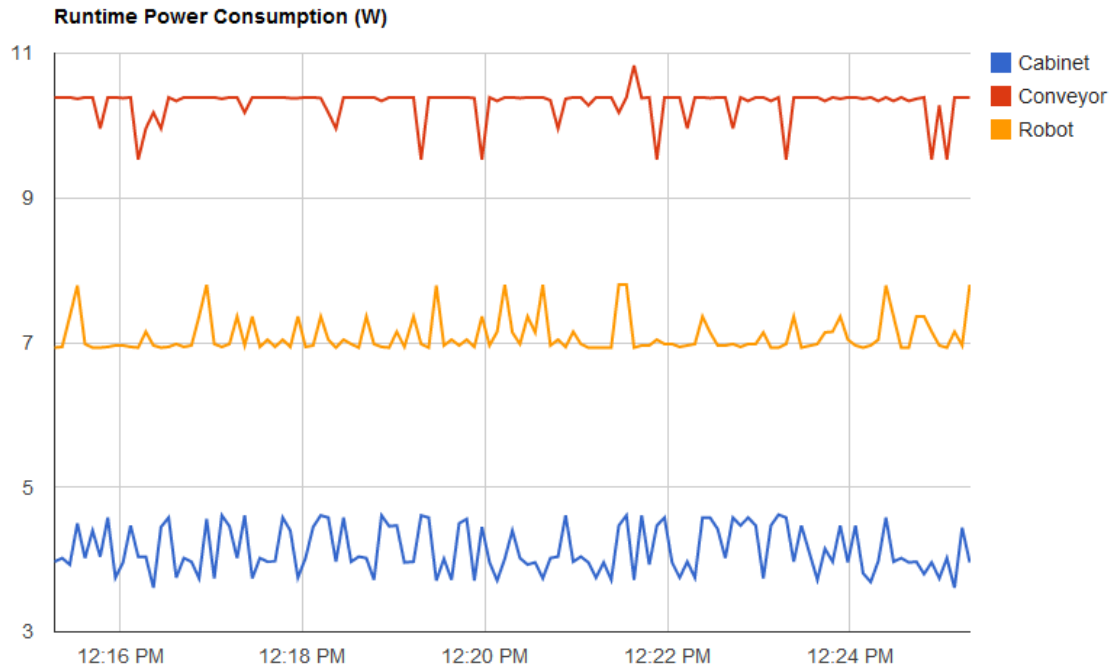


***Figure 52**: Runtime power consumption line chart*

Cell energy consumption

The cell energy consumption of a single cell is visualized as a bar chart (Figure 53) with each bar dedicating to the energy consumption of robot, cabinet and conveyor respectively in watt hour (Wh).

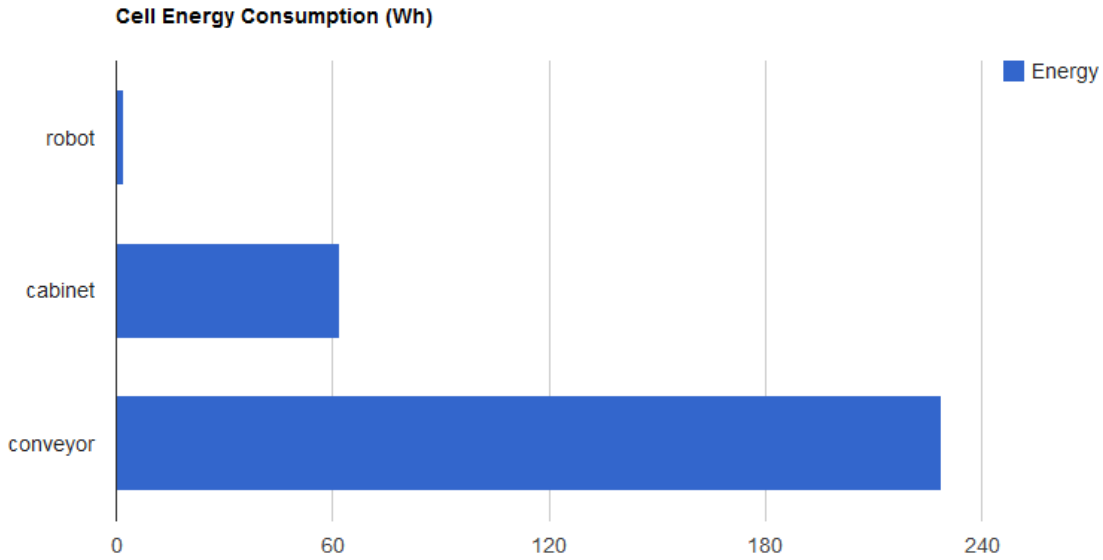


***Figure 53**: Cell energy consumption bar chart*

Energy consumption per product

The energy consumption per product line chart annotated with pallet ID illustrates the energy consumption for the production line to produce one pallet. For example, pallet number 20 is loaded with a piece of paper on cell 1 and enters the line. Then it is being

operated on cell 2, 3 and 4. The line chart plots the total energy consumption of the operations performed for the pallet on all these cells.

By default, a historical energy consumption per product line chart (Figure 54) plots the values within half an hour since the user opens the chart. Users can choose a specific pallet ID and specify a time period for inspection. The range filter below the line chart provides scalability to the chart.

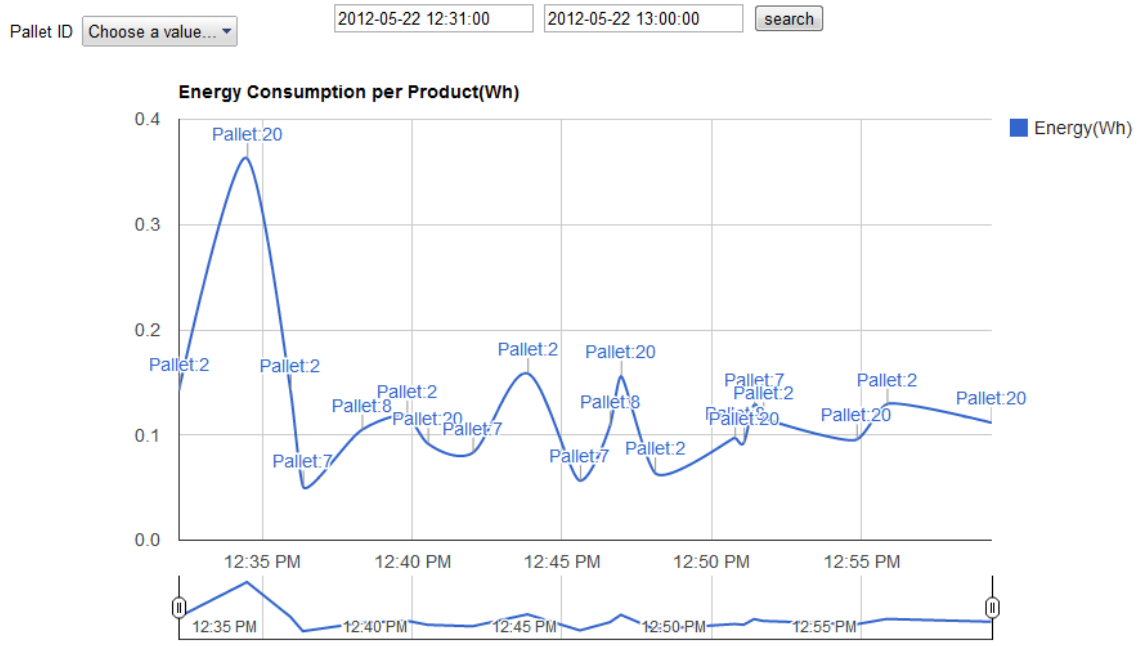


***Figure 54**: Historical energy consumption per product line chart*

Another line chart (Figure 55) shows the energy consumption per product in real time.

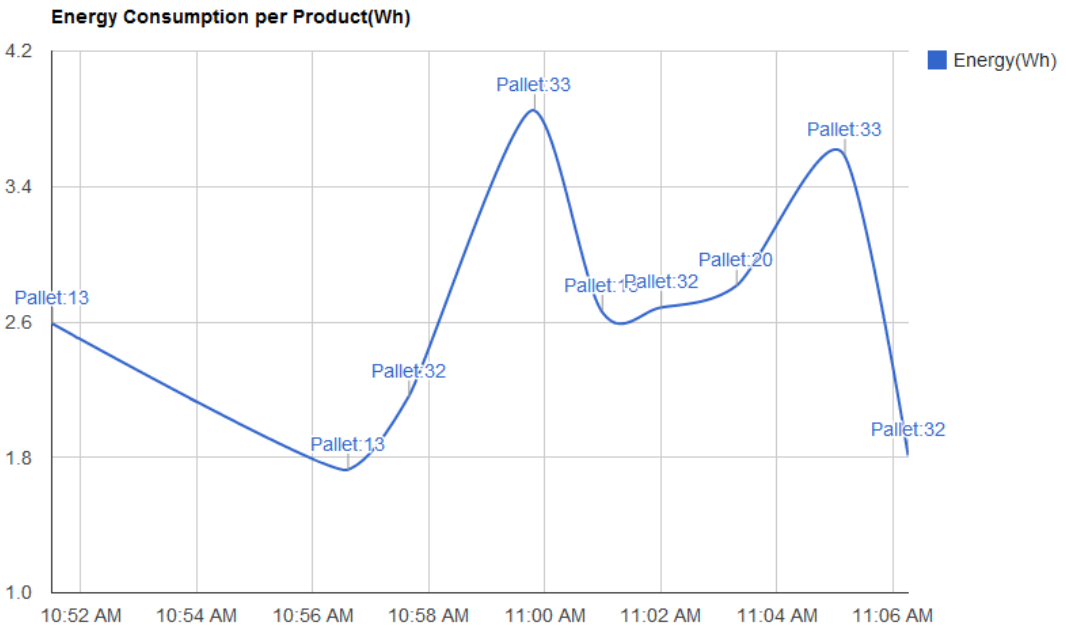


***Figure 55**: Real time energy consumption per product line chart*

4.1.3. Visualization of indicators in reliability

MTTR, MTTF and MTBF compose the reliability session. The three indicators of each working cell are illustrated in the one column chart (Figure 56). The first column displays MTTR, MTTF in the second and MTBF the last. The unit for all the indicators is second. With the change of, for example, MTTR, the height of the corresponding column increases or decreases.

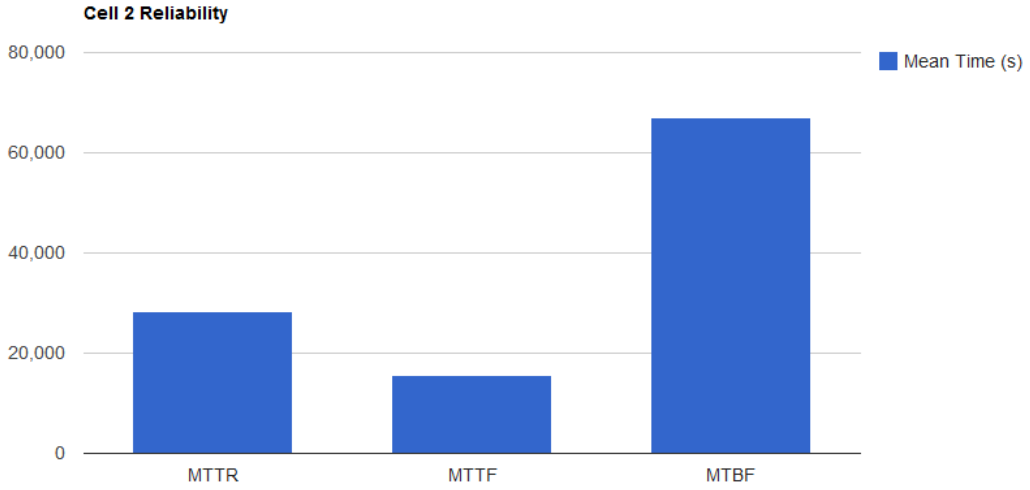


***Figure 56**: Reliability column chart*

4.1.4. Visualization of indicators in quality

The quality rate column chart (Figure 57) shows the percentage of products that pass the quality inspection (blue columns) and that do not pass the quality inspection (red columns) in four categories, the overall, which is the inspection result of the entire product, the keyboard, the frame and the screen drawings. A product that meets the keyboard and frame quality criteria, but does not meet the screen quality criteria increases height of the blue columns in keyboard and frame and the red columns in screen and overall, while decreases the height of the blue columns in screen and overall and the red columns in keyboard and frame.

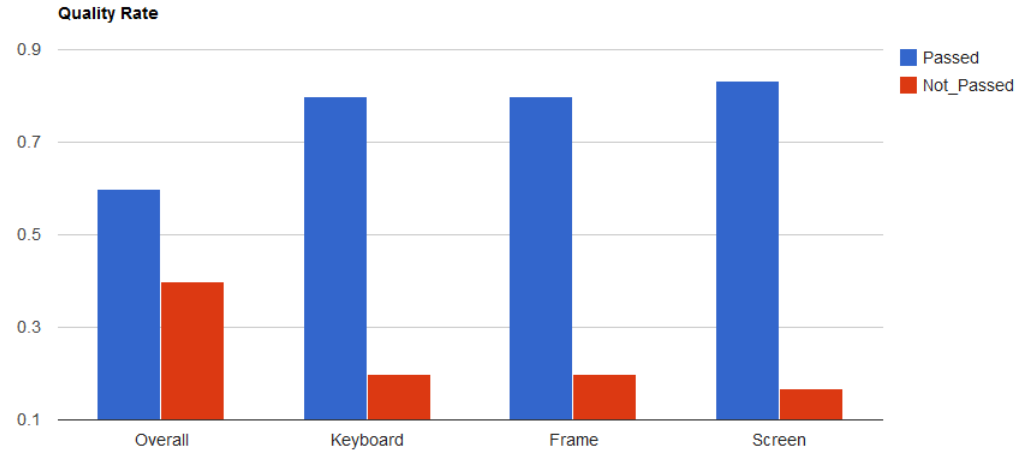


***Figure 57**: Quality rate column chart*

4.1.5. Visualization of indicators in overall

The Overall session is designed to demonstrate indicators including IPC-2541 states overview, total energy consumption and the total products.

IPC-2541 states overview

The IPC-2541 states overview is rendered as a 3D pie chart (Figure 58) with each portion of the pie representing the duration of a state (OFF, SETUP, READY-IDLE-STARVED, READY-IDLE-BLOCKED, READY-PROCESSING-ACTIVE, READY-PROCESSING-EXECUTING and DOWN) in percentage. As soon as one state completes, the corresponding portion increases accordingly. The portion of DOWN state is used to visualize the production shutdowns indicator.

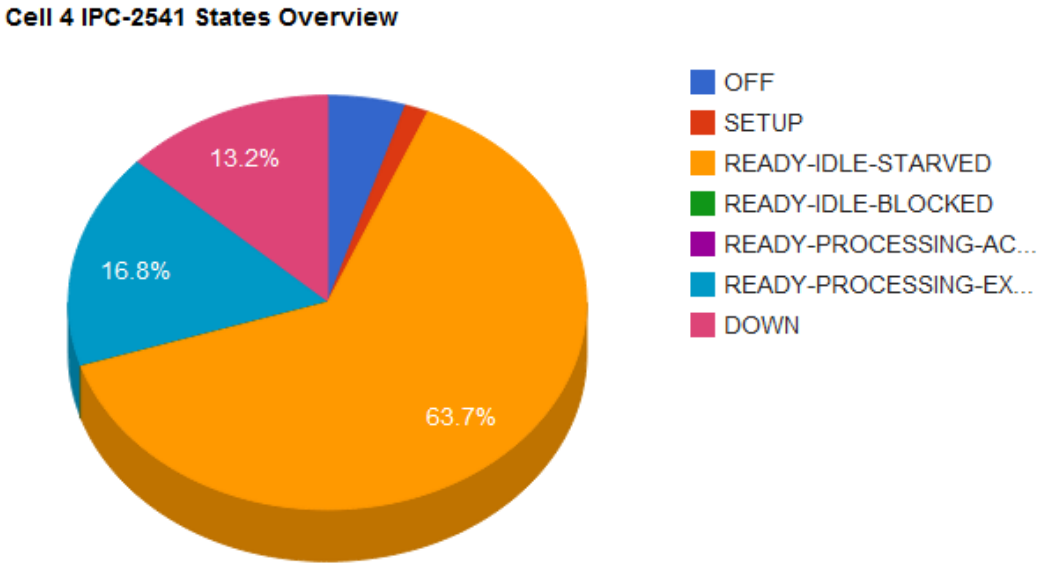


***Figure 58**: IPC-2541 state pie chart*

Total energy consumption

The total energy consumption indicator is illustrated with a bar chart (Figure 59). The first bar represents the energy consumption from all robots. The total energy use by all cabinets is represented with the middle bar, while the last bar indicates the total energy consumption from all the conveyors. The unit for the energy values in this chart is watt hour (Wh).

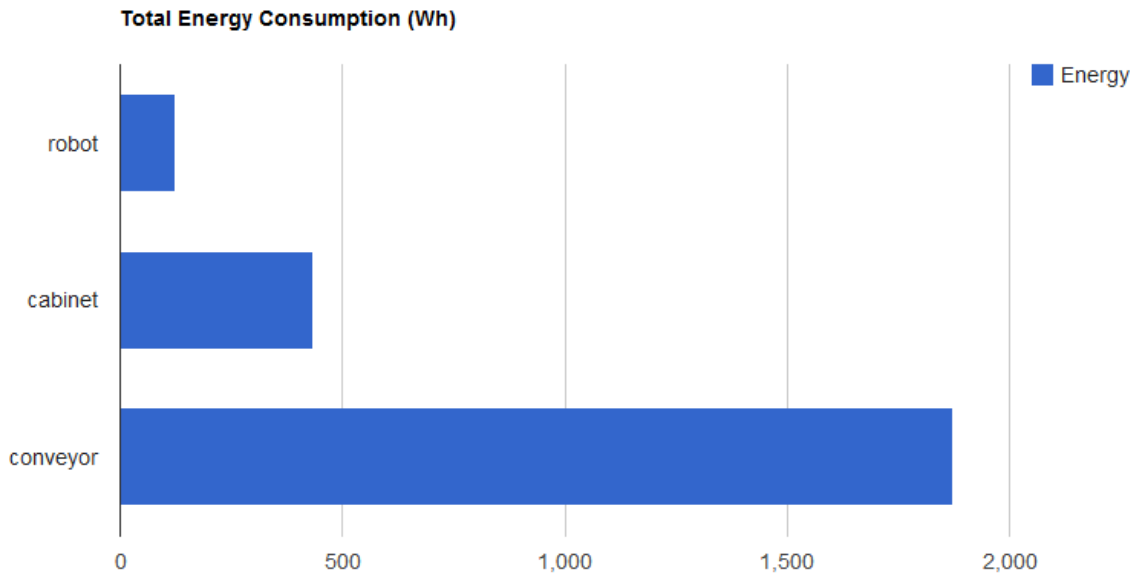


***Figure 59**: Total energy consumption bar chart*

Pallet production time

The pallet production time line chart shows how long a pallet flows on the production line to complete its all production processes.

By default, the historical pallet production time line chart (Figure 60) illustrates the production time of pallets that have been produced on the production line within 30 minutes since a user opens the chart. A drop down list is used to select a specific pallet for inspection. Scalability is also provided with a range filter below the line chart and a form defining a beginning time stamp and an ending time stamp.

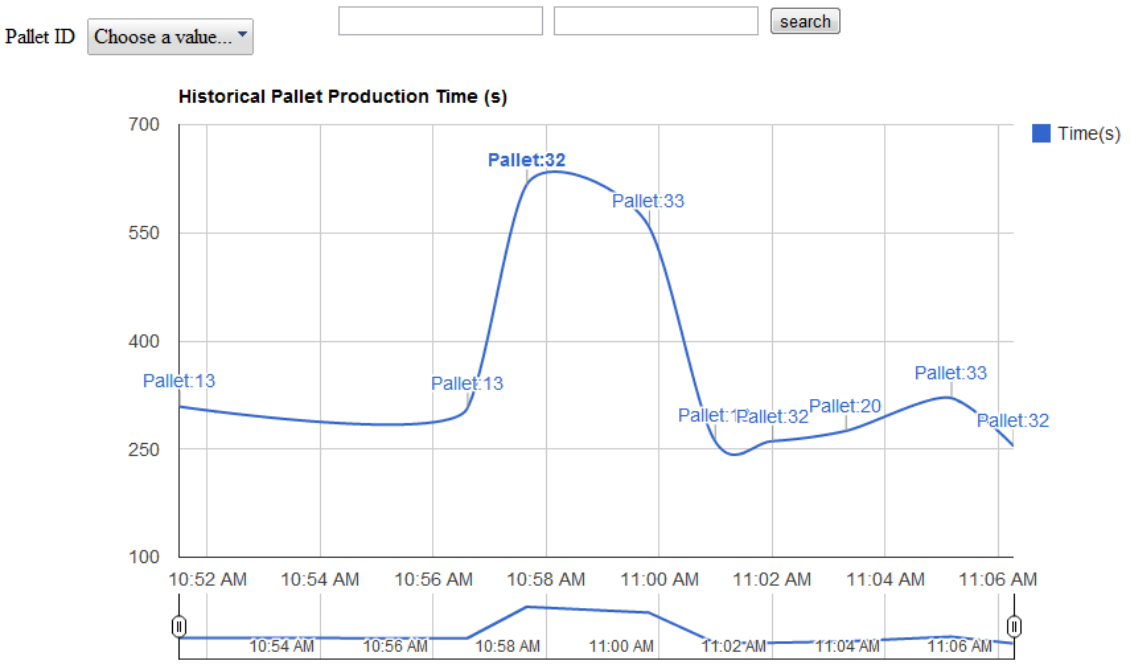


***Figure 60**: Historical pallet production time line chart*

A real-time pallet production time line chart (Figure 61) shows the information in real time.

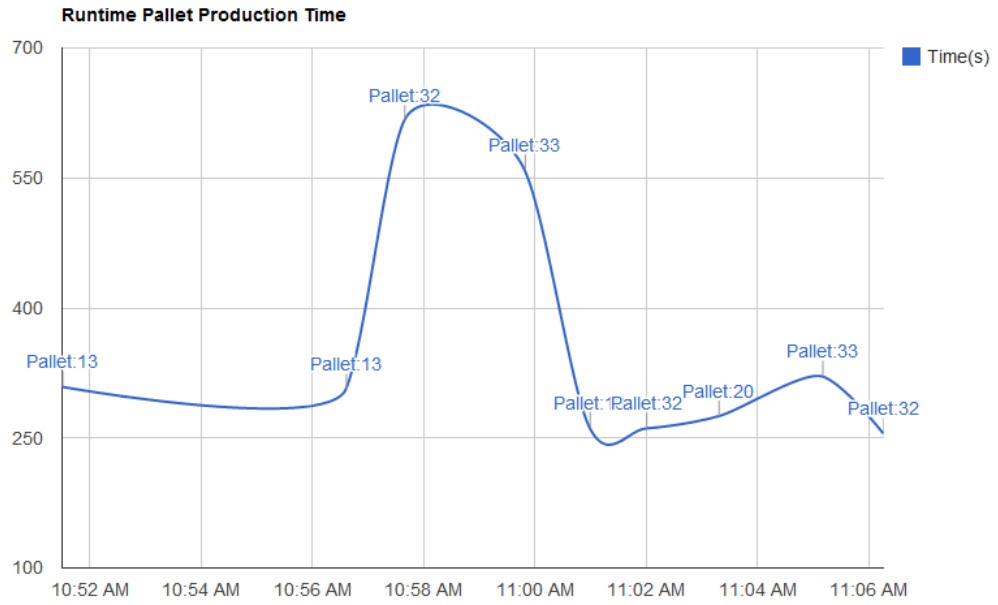


***Figure 61**: Real time pallet production time line chart*

Total products

A column chart (Figure 62) provides users with visualization of total products that have been manufactured on each cell. A new product being produced on one cell increases the height of the corresponding column in real time. The number of total products on cell 1 reveals the number of products being produced by the production line.

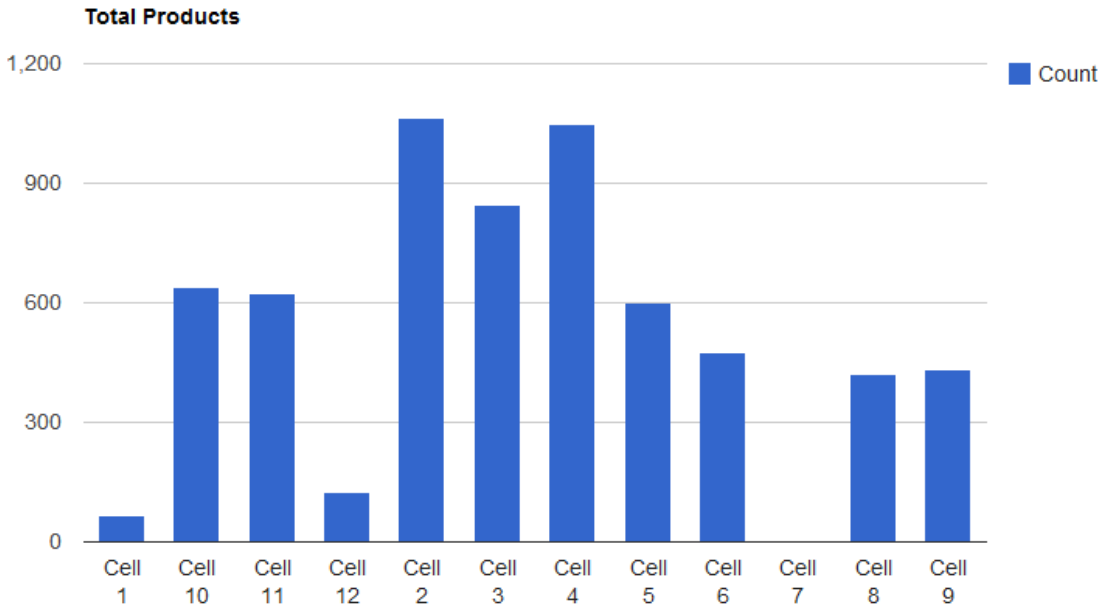


***Figure 62**: Total products column chart*

4.2. Accessing the InfoStore

The web service exposing the data gathered from the factory automation test bed can be found at http://esonia-controller.rd.tut.fi:8080/. The application name is FastoryService. The servlet mapping for handling RESTful web service URL requests is /endpoint/REST/*. Well formed data requests start with http://esonia-controller.rd.tut.fi:8080/ FastoryService/endpoint/REST/. For implementation reasons, Firefox and Google Chrome are preferred browsers at the moment for rendering the XML responses.

4.2.1. The URLs supported by the InfoStore

Three types of requests are currently supported by the InfoStore (Table XII): request of all data names within the database, request of data gathered in a certain time period (longer time periods requested translate to slower response/ larger amount of response data) and request of the last data which has the same name with the one declared in the request URL, the database has stored. All the resources available in the InfoStore including raw data and KPIs can be found in Appendix 6.

***Table XII**: RESTful web service access requests*

	Request	Example
Request of all data names	http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/data/ALL	
Request of data gathered in a certain time period	http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/sdata/{name}/from/{*timestamp*}/to/{*timestamp*}/	Request of all energy related data from cell 5 between 16:10:00 and 16:13:30 on March, 6th, 2012: http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/sdata/energyMeter5/from/2012-03-06 16:10:00/to/2012-03-06 16:13:30
Request of the latest data stored in the database	http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/last_data/{*names of data*}/	Request of the latest energy data from cells 5 and 4: http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/last_data/'energyMeter5','energyMeter4'/

4.3. Defining new indicators

Except for the indicators described in section 3.4.5 and their EPL rules to retrieve them, a user who is granted with an administrator account can create EPL rules to retrieve indicators of interest. The values of these indicators are also accessible in the InfoStore. This section provides a simple example on how users can create such EPL rules and how the indicators can be accessed.

The Create and Delete buttons on the 'CEP rule' page are exposed to administrators. By clicking on the Create button, a form (Figure 63) slides down on which a syntax-correct EPL rule and its name can be edited. The name of a new rule should always be

different from the existing ones. A duplicating name or an ELP rule with incorrect syntax fails the form submission.

For example, a new EPL rule named '*power_factor*' is defined to calculate the power factor of all phases utilizing active power (*AWATT*, *BWATT* and *CWATT*) and apparent power (*AVA*, *BVA*, *CBA*) provided by '*EnergyMeter*' event. Submitting the form registers the EPL rule in the Esper engine.

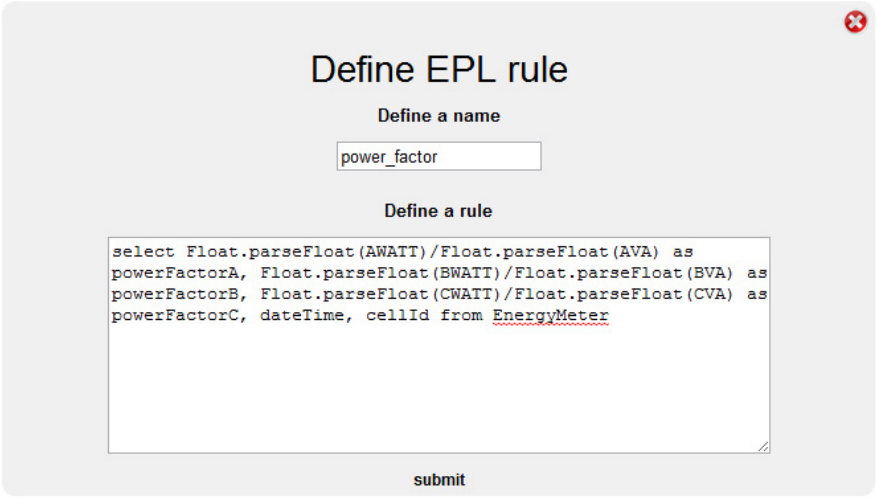


***Figure 63**: Defining an EPL rule*

Afterwards, the '*power_factor*' indicator is accessible in the InfoStore using the URLs described in section 4.2.1. For example, using the URL: http://esonia-controller.rd.tut.fi/FastoryService/endpoint/REST/last_data/'power_factor', the most recent '*power_factor*' indicator is accessed in XML format (Figure 64).

```
▼<ResponseList xmlns="http://www.tut.fi/fast/FastoryService">
  ▼<Responses serviceId="power_factor">
     <Value point="powerFactorB">0.044458598847601824</Value>
     <Value point="cellId">5</Value>
     <Value point="dateTime">2012-06-07T17:58:38.650</Value>
     <Value point="powerFactorC">-0.1851486882786891</Value>
     <Value point="powerFactorA">-0.12495772645846702</Value>
     <Value point="timestamp">2012-06-07 17:58:42.0</Value>
     <Value point="estimatedResponseTime">19ms</Value>
   </Responses>
 </ResponseList>
```

***Figure 64**: Accessing the most recent power factor*

5. CONCLUSIONS

This chapter presents a discussion of results of previous chapters. The implemented KPI InfoStore is compared with previous works. Then there is a discussion on the future development of the application.

5.1. Conclusions on results

This section draws a conclusion on what have been conducted for the KPI InfoStore, what tools or techniques are used for implementation. Then it compares the InfoStore with other KPI frameworks.

5.1.1. Overall

To design and implement the KPI InfoStore for the test bed, the following objectives have been accomplished by using the KPI design model and a variety of web application and web service frameworks and technologies.

A set of KPIs has been designed to monitor the critical performance of the test bed using the 8-step iterative model developed in [2]. During the design and implementation phase, not all the steps are used. The author only designed the production goals, identified the potential indicators, selected indicators for the test bed and implemented these indicators. To select KPIs for the test bed, the author adopts the five principal KPIs defined in [2], the sustainable indicators in [4] and reliability according to [58].

Spring Web Services is used to enable machine-to-machine communication. With the framework, the author implemented a SOAP web service to capture events from the test bed.

Esper engine is used for complex event processing. A set of EPL rules is designed to aggregate and correlate current and historical events for runtime calculation of KPIs. Administrators can also define EPL rules of interests.

The KPI InfoStore is further developed as a web application with MVC design pattern. Spring MVC is used as the overall framework of the web application. Hibernate is utilized as the persistence API for ORM and CRUD operations against MySQL database. The values of KPIs are rendered as charts with the assistance of HTML, JSP, jQeury and Google chart tools.

Lastly, in order for other applications to access the KPIs and raw data, they are published in a RESTful web service using Spring MVC.

5.1.2. Comparison with previous work

As is reviewed in Section 2.1, several researches have already been done for the development of production KPIs. This section compares this work with them.

Previous works usually focus on the methodologies and frameworks of KPI development for production processes. [2] focuses on the methodologies on what kind of indicators should be monitored for production processes, while [4] focuses on the sustainable indicators. The KPIs designed in these researches are not implemented in applications. Although in [1], the KPIs are monitored in real time and used in a feedback control system, it is merely a simulation rather than a real case. [15] designed a set of KPIs for the notebook manufacturing, but the KPIs are not monitored in real time. The energy aspect is not considered in [15] as well.

In this thesis work, methodologies and frameworks for KPI development in manufacturing systems such as the 8-step iterative model and the five principle KPIs developed in [2] and sustainable indicators in [4] are adopted for the design of KPIs for the test bed. The KPI InfoStore is developed into a web application. The advantage is that users can visualize the KPI graphics on web anywhere on condition that a computer is available and connected to internet. The application utilizes modern technologies to enable runtime calculation and monitoring of KPIs so that users can monitor the current states of the test bed. The adding of new KPIs is flexible because instead of receiving KPIs directly from low level devices, the mechanism of KPI acquisition in this work is that raw data are received from devices and the data aggregation and correlation processes are conducted in the application (higher level) using CEP techniques. Therefore, when new KPIs need to be gathered, the user only needs to design new EPL rules. Furthermore, the acquired data are published in a RESTful web service so that other applications can access the data by providing URL requests. The responses containing the data are in XML format.

The drawback of this application is that users need to possess the knowledge of designing EPL rules to manage the KPIs. Besides, when a new web service is introduced in the test bed, the program also needs updates.

5.2. Further work

The thesis work attempts to design and implement a KPI InfoStore for real-time KPI monitoring. The design and implementation of the KPI management system for the test bed follows the first several steps in the 8-step iterative model. The other steps of the model need to be followed. First, a target can be set for every indicator for continuous improvement of the performance. Once the target is achieved, new target can be set. In addition, there should be an alarm mechanism designed so that when the value of an indicator is lower than the tolerable low boundary, the system alarms and alters the colour of the indicator from, for example, green to red. In this way, the managers are released from monitoring the values for long time. Secondly, the KPI management

system needs to be tested for a longer period. Next, in order to improve the performance of the production processes and act on the alarms, action-to-indicator mappings need to be established. For instance, the operations that should be performed when a low value of an indicator is received can be integrated into the InfoStore. Lastly, industrial management experts can review the current indicators and goals of the production line. If it is necessary to set new goals and objectives for the production line, new indicators should be designed by following the 8-step iterative model from the beginning.

The KPIs including Efficiency and Quality originates from the five principle KPIs for production processes. Production plan tracking is not monitored in the system, since the lack of an order entry system in the production line. When an order entry system is established, the production plan tracking should be monitored because it is considered a critical aspect for production processes. Better production plan tracking results improve the level of customer satisfaction. The indicators for production plan tracking can be, for example, the percent of production orders finished late and the percent of production orders finished ahead.

In the aspect of sustainability, this work only monitors the energy relevant indicators for single robot cell and the entire manufacturing system. Besides energy use, other aspects such as material use, waste generated and green house gas generated also have a significant impact on sustainable production. Therefore, future work should also concentrate on the analysis of material use, waste and green house gas generation in the test bed, the design and implementation of measurement systems and the design of indicators for these aspects.

REFERENCES

[1] V. Jovan and S Zorzut, "Use of Key Performance Indicators in Production Management," *Cybernetics and Intelligent Systems, IEEE Conference*, pp. 1-6, 2006.

[2] A. Rakar, S. Zorzut, and V. Jovan, "Assesment of Production Performance by Means of KPI," *Control 2004*, pp. 6-9, 2004.

[3] V Veleva, B Hart, T Greiner, and C Crumbley, "Indicators of sustainable production," *Journal of Cleaner Production*, pp. 447-452, 2001.

[4] V Veleva and M Ellenbecker, "Indicators of Sustainable Production: Framework and Methodology," *Journal of Cleaner Production*, pp. 519-549, 2001.

[5] J Low, "Disclosure and Clobal Capital Markets, Plenary presentation," *2nd International Symposium of the Global Reporting Initiative*, November 2000.

[6] (2012, November) European Commission. [Online]. http://ec.europa.eu/energy/publications/doc/2011_energy2020_en.pdf

[7] (2009, February) European Commission. [Online]. http://ec.europa.eu/information_society/events/ict4ee/2009/docs/files/ec/ec/infso/g2/SmartManufacturing.pdf

[8] G., Kemp, B., Penzel, T., Schlög, A., Rappelsberger, P., Trenker, E., Gruber, G., Zeithofer, J., Saletu, B., Herrmann, W. M., Himanen, S. L., Kunz, D., Barbanoj, M. J., Röschke, J. Värri, A. & Dorffner, G [2] Klösch, "The SIESTA Project Polygraphic and Clinical Database," *IEEE Engineering in Medicine and Biology*, vol. 203, pp. 51-57, 2001.

[9] Richy Smith and R Keith Mobley, "Rules of Thumb for Maintenance and Reliability Engineers," *Chapter 6: Key Performance Indicators*, pp. 89-106, 2008.

[10] Clemens Lohman, Leonard Fortuin, and Marc Wouters, "Designing a Performance Measurement System: A Case Study," *European Journal of Operational Research*, pp. 267-286, 2004.

[11] Jan Smith, *Planning and monitoring your program: first steps in program evaluation*. Sydney: Office of Publish Management, 1992.

[12] David Parmenter, *Key Performance Indicators (KPI): Developing, Implementing, and Using Wining KPIs*, 2nd ed.: John Wiley & Sons, Inc, 2010.

[13] M Munir Ahmad and Nasreddin Dhafr, "Establishing and improving

manufacturing performance measures," *Robotics and Computer Integrated Manufacturing 18*, pp. 171-176, 2002.

[14] L Sivill and P Ahtila, "Paper Machine Production Efficiency as a Key Performance Indicator of Energy Efficiency," *Chemical Engineering Transactions*, pp. 905-905, 2009.

[15] D Daniel Sheu and Shiao-Lan Peng, "Assessing manufacturing management performance for notebook computer plants in Taiwan," *Int. J. Production Economics 84*, pp. 215-228, 2003.

[16] Vesela Veleva, Jack Bailey, and Nicole Jurczyk, "Using Sustainable Production Indicators to Measure Progress in ISO14001, EHS System and EPA Achievement Track," *Corporate Environmental Strategy*, pp. 326-338, December 2001.

[17] Katharina Bunsea, Matthias Vodickaa, Paul Schönslebena, Marc Brülhartb, and Frank O. Ernstb, "Integrating energy efficiency performance in production management- gap analysis between industrial needs and scientific literature," *Journal of Cleaner Production*, pp. 667-679, 2011.

[18] Marcos Palacios, José García-Fanjul, and Javier Tuya, "Testing in Service Oriented Architectures with dynamic binding: A mapping study," *Information and Software Technology*, pp. 171-189, 2011.

[19] E Zeeb, A Bobek, H Bohn, and F Golatowski, "Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services," *AINA Workshops(1)*, pp. 956-963, 2007.

[20] Elmar Zeeb, Guido Moritz, Dirk Timmermann, and Frank Golatowski, "WS4D: Toolkits for networked embedded systems based on the Devices Profile for Web Serivces," *39th International Conference on Parallel Processing Workshops*, pp. 1-8, 2010.

[21] (2011, May) JMEDS Framework Overview. [Online]. http://ws4d.e-technik.uni-rostock.de/wp-content/uploads/2011/05/StackOverview.pdf

[22] (2012, May) Spring Web Services. [Online]. http://static.springsource.org/spring-ws/sites/2.0/

[23] R.T. Fielding, "Architectural styles and the design of networkbased software architectures," *Doctoral Dissertation, University of California, Irvine, CA*, p. Chapter 5, 2000.

[24] Jeffrey V. Nickerson, Keith D. Swenson Michael zur Muehlen, "Developing

web services choreography standards- the case of REST vs. SOAP," *Decision Support Systems*, vol. 40, no. 1, pp. 9-29, 2005.

[25] Cesare Pautasso, "RESTful web service composition with BPEL for REST," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 851-866, September 2009.

[26] et al Rod Johnson. (2011) Spring Framework Reference document. [Online]. http://static.springsource.org/spring/docs/ 3.1.x/spring-framework-reference/html/

[27] Josh Long and Daniel Rubio Gary Mak, *Spring Recipes: A problem-solution approach*, 2nd ed.: Apress, 2010.

[28] M. Jorge A. Garcia Izaguirre, Andrei Lobov, and Jose L. Martinez Lastra, "OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing," *Industrial Informatics (INDIN), 9th IEEE International Conference*, pp. 205-211, 2011.

[29] Y. Gu, G. Yu, and C. Li, "Deadline-aware complex event processing models over distributed monitoring streams," *Mathematical and Computer Modelling*, 2011.

[30] R. Bhargavi, V. Vaidehi, P.T.V. Bhuvaneswari, P. Balamurali, and G. Chandra, "Complex Event Processing for Object Track in Wireless Sensor Networks," *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference*, pp. 211-214, 2010.

[31] (2007) EsperTech. [Online]. http://esper.codehaus.org/

[32] Don Brown, Chad Michael Davis, and Scott Stanlick, *Struts 2 in Action*. Greenwich, CT: MANNING Publications Co., 2008.

[33] F Buschmann, R Meunier, H Rohnert, P Sommerlad, and M Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester: John Wiley & Sons Ltd., 1996.

[34] Lihua Xu, Hadar Ziv, Thomas A. Alspaugh, and Debra J. Richardson, "An architectural pattern for non-functional dependability requirements," *The Journal of Systems and Software 79*, pp. 1370-1378, 2006.

[35] Paris Avgeriou and Uwe Zdun, "Architectural Patterns Revisited - A Pattern Language," *10th European Conference on Pattern Languages of Programs*, pp. 1-39, 2005.

[36] Hongbo Lan, Yucheng Ding, Jun Hong, Hailiang Huang, and Bingheng Lu, "A web-based manufacturing service system for rapid product development,"

Computers in Industry 54, pp. 51-67, 2004.

[37] Chulho Chung and Qingjin Peng, "Enabled dynamic tasks planning in Web-based virtual manufacturing environments," *Computers in Industry 59*, pp. 82-95, 2008.

[38] F.A Masound and D.H Halabi, "ASP.NET and JSP Frameworks in Model View Controller Implementation," *Information and Communication Technologies, ICTTA'06*, pp. 3593-3598, 2006.

[39] Jie He and Xianhong Xu, "Design of a management information system for Shielding Experimental Reactor ageing management," *Nuclear Engineering and Design*, pp. 103-111, 2010.

[40] László Szirmay-Kalos, Gábor Márton, Tibor Fóris, and Tamás Horváth, "Development of process visualization system: An object-oriented approach," *Journal of Systems Architecture*, pp. 275-296, 2000.

[41] José-Luis Sierra, Baltasar Fernández-Manjón, and Alfredo Fernández-Valmayor, "A language-driven approach for design of interactive applications," *Interacting with Computers*, pp. 112-127, 2008.

[42] Baoli Dong, Guoning Qi, Xinjian Gu, and Xiuting Wei, "Web service-oriented manufacturing resource applications for networked product development," *Advanced Engineering Informatics 22*, pp. 282-295, 2008.

[43] K.-D. Bouzakis, D. Andreadis, A. Vakali, and M.Sarigiannidou, "Automating the manufacturing process under a web based framework," *Advances in Engineering Software 40*, pp. 956-964, 2009.

[44] Wei-Fu Chang, Yu-Chi Wu, and Chui-Wen Chiu, "Development of a web-based remote load supervision and control system," *Electrical Power and Energy Systems 28*, pp. 401-407, 2006.

[45] (2009) VaanNila. [Online]. http://www.vaannila.com/struts-2/struts-2-tutorial/struts-2-framework-tutorial-1.html

[46] (2011) Spring Java Application Framework. [Online]. http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/

[47] (2009) VaanNila. [Online]. http://www.vaannila.com/spring/spring-ioc-1.html

[48] Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo, "EasySOC: Making web service outsourcing easier," *Information Sciences*, p. In

Press, 2010.

[49] Martijn Dashorst and Eelco Hillenius, *WICKET in Action*. Greenwich, CT: Manning Publications Co., 2009.

[50] Robert J. Brunner, *JSP- Practical Guide for Java Programmers*.: Elsevier Inc, 2003.

[51] (2011) The JavaEE 6 Tutorial. [Online]. http://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html

[52] (2007, March) The Apache Velocity Project. [Online]. http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html

[53] Yakov Fain, *Java Programming 24-Hour Trainer*. Indianapolis, Indiana: Wiley Publishing, Inc., 2011.

[54] Mike Keith and Merrick Schincariol, *Pro JPA2 Mastering the Java Persistence API*.: Apress, 2009.

[55] (2004) HIBERNATE Community Documentation. [Online]. http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html/hibernate-gsg-preface.html#d0e94

[56] S1000 User Manual. [Online]. http://www.inicotech.com/doc/S1000%20User%20Manual.pdf

[57] E10 Energy Analyzer. [Online]. http://www.inicotech.com/doc/e10%20brochure.pdf

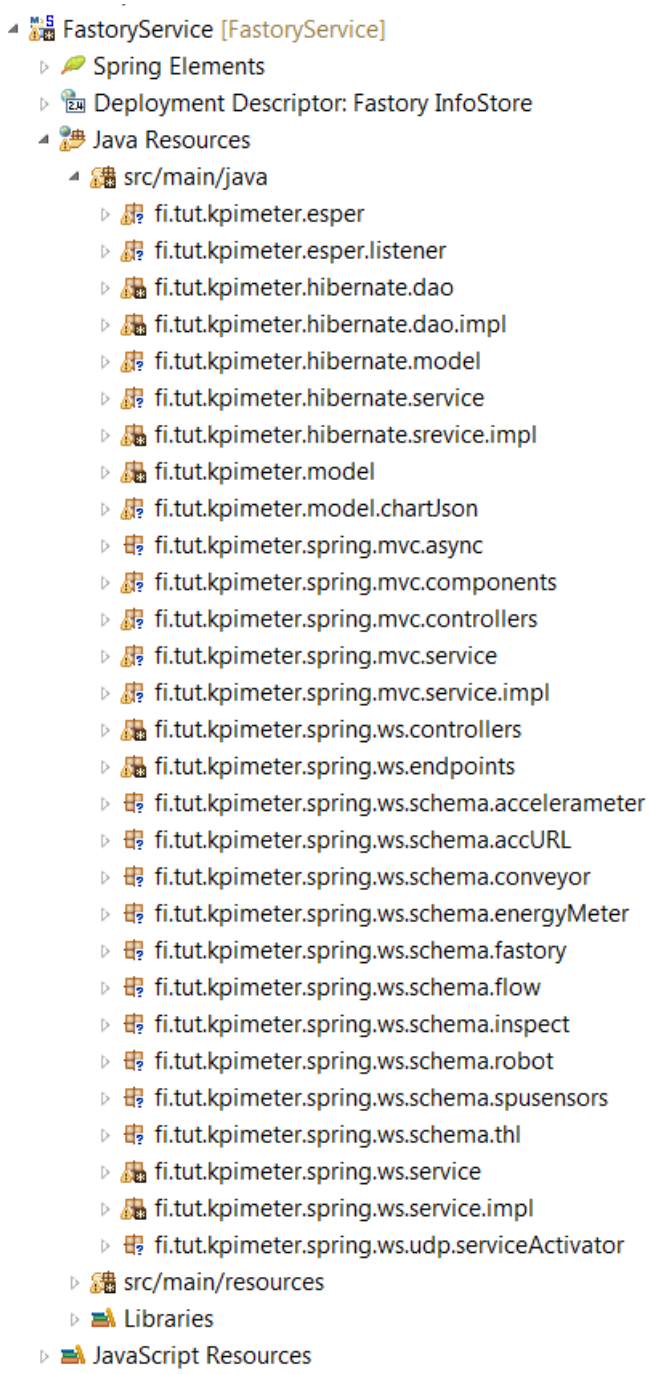
[58] Peter Muchiri, Liliane Pintelon, Ludo Gelders, and Harry Martin, "Development fo maintenance function performance measurement framework and indicators," *Int. J. Production Economics*, pp. 295-302, 2011.

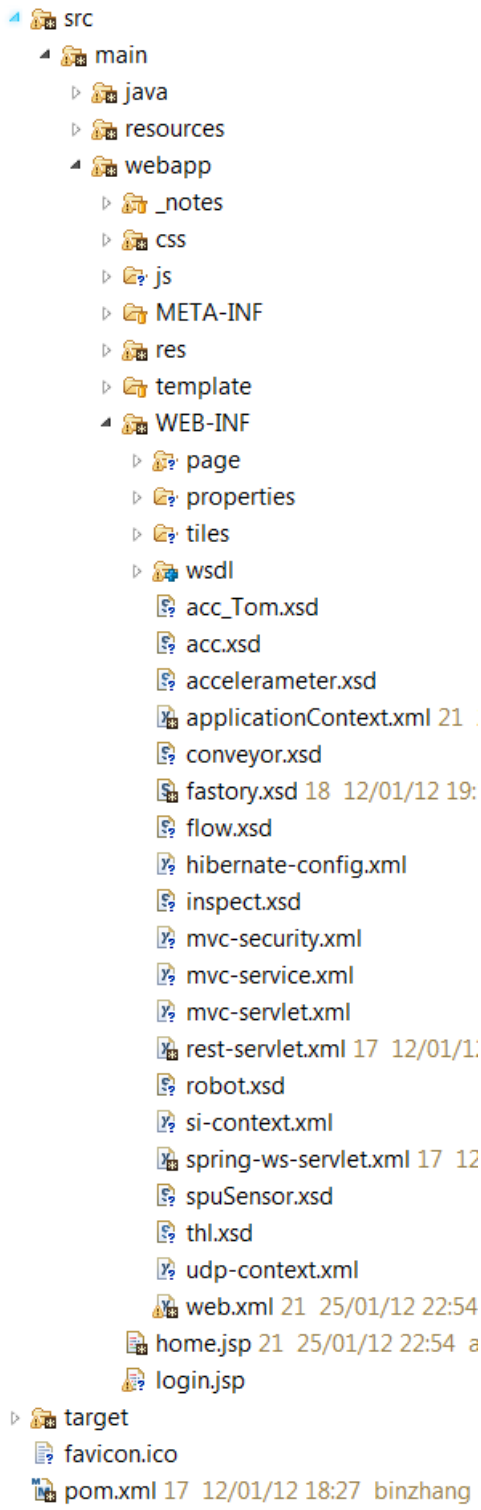
[59] K. Tanaka, "Assessment of energy efficiency performance measures in industry and their application for policy," *Energy Policy 36*, pp. 2887-2902, 2008.

[60] M.C. Eti, S.O.T. Ogajji, and S.D. Probert, "Implementing total productive maintenance in Nigerian manufacturing industries," *Applied Energy*, pp. 385-401, 2004.

[61] Lirong Cui and Haijun Li, "Analytical method for reliability and MTTF assessment of coherent systems with dependent components," *Reliability Engineering & System Safety*, pp. 300-307, 2007.

APPENDIX 1: FOLDER STRUCTURE




src
main
java
resources
webapp
_notes
css
js
META-INF
res
template
WEB-INF
page
properties
tiles
wsdl
acc_Tom.xsd
acc.xsd
accelerameter.xsd
applicationContext.xml 21
conveyor.xsd
fastory.xsd 18 12/01/12 19:
flow.xsd
hibernate-config.xml
inspect.xsd
mvc-security.xml
mvc-service.xml
mvc-servlet.xml
rest-servlet.xml 17 12/01/1
robot.xsd
si-context.xml
spring-ws-servlet.xml 17 12
spuSensor.xsd
thl.xsd
udp-context.xml
web.xml 21 25/01/12 22:54
home.jsp 21 25/01/12 22:54 a
login.jsp
target
favicon.ico
pom.xml 17 12/01/12 18:27 binzhang

APPENDIX 2: SCHEMA FOR *EQUIPMENTCHANGESTATE* MESSAGE

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema targetNamespace="http://www.tut.fi/fast/robot"
        xmlns:tns="http://www.tut.fi/fast/robot"
        elementFormDefault="qualified"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name = "EquipmentChangeState">
   <xs:complexType>
    <xs:attribute name = "dateTime" use = "required" type = "xs:dateTime"/>
    <xs:attribute name = "currentState" use = "required">
     <xs:simpleType>
      <xs:restriction base = "xs:string">
       <xs:enumeration value = "OFF"/>
       <xs:enumeration value = "SETUP"/>
       <xs:enumeration value = "READY-IDLE-STARVED"/>
       <xs:enumeration value = "READY-IDLE-BLOCKED"/>
       <xs:enumeration value = "READY-PROCESSING-ACTIVE"/>
       <xs:enumeration value = "READY-PROCESSING-EXECUTING"/>
       <xs:enumeration value = "DOWN"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name = "previousState" use = "required">
     <xs:simpleType>
      <xs:restriction base = "xs:string">
       <xs:enumeration value = "OFF"/>
       <xs:enumeration value = "SETUP"/>
       <xs:enumeration value = "READY-IDLE-STARVED"/>
       <xs:enumeration value = "READY-IDLE-BLOCKED"/>
       <xs:enumeration value = "READY-PROCESSING-ACTIVE"/>
       <xs:enumeration value = "READY-PROCESSING-EXECUTING"/>
       <xs:enumeration value = "DOWN"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
    <xs:attribute name = "eventId" use = "required" type = "xs:string"/>
    <xs:attribute name = "palletId" use = "required" type = "xs:string"/>
    <xs:attribute name = "recipeNum" use = "required" type = "xs:string"/>
    <xs:attribute name = "toolId" use = "required" type = "xs:string"/>
    <xs:attribute name = "cellId" use = "required" type = "xs:string"/>
    <xs:attribute name = "devType" use = "required" type = "xs:string"/>
    <xs:attribute name = "prodId" use = "required" type = "xs:string"/>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

APPENDIX 3: WSDL FOR *EQUIPMENTCHANGESTATE* MESSAGE

```
<wsdl:definitions targetNamespace="http://www.tut.fi/fast/robot"
          xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
          xmlns:sch="http://www.tut.fi/fast/robot"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
          xmlns:tns="http://www.tut.fi/fast/robot">
 <wsdl:types>
  <xs:schema attributeFormDefault="qualified"
                elementFormDefault="qualified"
                targetNamespace="http://www.tut.fi/fast/robot"
                xmlns="http://www.w3.org/2001/XMLSchema"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="String" type="xs:string" />
   <xs:element name="calibrateRobot" type="xs:string"/>
   <xs:element name = "EquipmentChangeState">
    <xs:complexType>
     <xs:attribute name = "dateTime" use = "required" type = "xs:dateTime"/>
     <xs:attribute name = "currentState" use = "required">
      <xs:simpleType>
       <xs:restriction base = "xs:string">
        <xs:enumeration value = "OFF"/>
        <xs:enumeration value = "SETUP"/>
        <xs:enumeration value = "READY-IDLE-STARVED"/>
        <xs:enumeration value = "READY-IDLE-BLOCKED"/>
        <xs:enumeration value = "READY-PROCESSING-ACTIVE"/>
        <xs:enumeration value = "READY-PROCESSING-EXECUTING"/>
        <xs:enumeration value = "DOWN"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name = "previousState" use = "required">
      <xs:simpleType>
       <xs:restriction base = "xs:string">
        <xs:enumeration value = "OFF"/>
        <xs:enumeration value = "SETUP"/>
        <xs:enumeration value = "READY-IDLE-STARVED"/>
        <xs:enumeration value = "READY-IDLE-BLOCKED"/>
        <xs:enumeration value = "READY-PROCESSING-ACTIVE"/>
        <xs:enumeration value = "READY-PROCESSING-EXECUTING"/>
        <xs:enumeration value = "DOWN"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name = "eventId" use = "required" type = "xs:string"/>
     <xs:attribute name = "palletId" use = "required" type = "xs:string"/>
     <xs:attribute name = "recipeNum" use = "required" type = "xs:string"/>
     <xs:attribute name = "toolId" use = "required" type = "xs:string"/>
     <xs:attribute name = "cellId" use = "required" type = "xs:string"/>
     <xs:attribute name = "devType" use = "required" type = "xs:string"/>
     <xs:attribute name = "prodId" use = "required" type = "xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:schema>
 </wsdl:types>

 <wsdl:message name="EquipmentChangeState">
  <wsdl:part element="tns:EquipmentChangeState" name="EquipmentChangeState"/>
```

```
  </wsdl:message>

  <wsdl:portType name="Robot">
    <wsdl:operation name="EquipmentChangeStateOperation">
          <wsdl:input message="tns:EquipmentChangeState" name="EquipmentChangeState"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="RobotSoap12" type="tns:Robot">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="EquipmentChangeStateOperation">
      <soap:operation
soapAction="http://www.tut.fi/fast/FastoryService/EquipmentChangeStateOperation"/>
          <wsdl:input name="EquipmentChangeState">
            <soap:body use="literal"/>
          </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="RobotService">
    <wsdl:port binding="tns:RobotSoap12" name="RobotSoap12">
          <soap:address location="http://localhost:8080/FastoryService/endpoint/SOAP"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

APPENDIX 4: *EQUIPMENTCHANGESTATE* CLASS

```
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference
Implementation, vJAXB 2.1.10 in JDK 6
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source schema.
// Generated on: 2012.06.11 at 12:23:22 PM EEST
//
package fi.tut.kpimeter.spring.ws.schema.robot;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSchemaType;
import javax.xml.bind.annotation.XmlType;
import javax.xml.datatype.XMLGregorianCalendar;


@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "")
@XmlRootElement(name = "EquipmentChangeState")
public class EquipmentChangeState {

    @XmlAttribute(required = true)
    @XmlSchemaType(name = "dateTime")
    protected XMLGregorianCalendar dateTime;
    @XmlAttribute(required = true)
    protected String currentState;
    @XmlAttribute(required = true)
    protected String previousState;
    @XmlAttribute(required = true)
    protected String eventId;
    @XmlAttribute(required = true)
    protected String palletId;
    @XmlAttribute(required = true)
    protected String recipeNum;
    @XmlAttribute(required = true)
    protected String toolId;
    @XmlAttribute(required = true)
    protected String cellId;
    @XmlAttribute(required = true)
    protected String devType;
    @XmlAttribute(required = true)
    protected String prodId;

    public XMLGregorianCalendar getDateTime() {
        return dateTime;
    }

    public void setDateTime(XMLGregorianCalendar value) {
        this.dateTime = value;
    }

    public String getCurrentState() {
        return currentState;
    }
```

```
public void setCurrentState(String value) {
    this.currentState = value;
}

public String getPreviousState() {
    return previousState;
}

public void setPreviousState(String value) {
    this.previousState = value;
}

public String getEventId() {
    return eventId;
}

public void setEventId(String value) {
    this.eventId = value;
}

public String getPalletId() {
    return palletId;
}

public void setPalletId(String value) {
    this.palletId = value;
}

public String getRecipeNum() {
    return recipeNum;
}

public void setRecipeNum(String value) {
    this.recipeNum = value;
}

public String getToolId() {
    return toolId;
}

public void setToolId(String value) {
    this.toolId = value;
}

public String getCellId() {
    return cellId;
}

public void setCellId(String value) {
    this.cellId = value;
}

public String getDevType() {
    return devType;
}

public void setDevType(String value) {
    this.devType = value;
}
```

```
    public String getProdId() {
        return prodId;
    }

    public void setProdId(String value) {
        this.prodId = value;
    }

}
```

APPENDIX 5: CONFIGURATION PARAMETERS FOR ESPER ENGINE

```
<?xml version="1.0" encoding="UTF-8"?>
 <esper-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns="http://www.espertech.com/schema/esper"
                      xsi:schemaLocation="http://www.espertech.com/schema/esper
                                  http://www.espertech.com/schema/esper/esper-configuration-2.0.xsd">
  <event-type name="QualityInspectionShort"
              class="fi.tut.kpimeter.spring.ws.schema.inspect.QualityInspectionShort"/>
  <event-type name="EnergyMeter"
              class="fi.tut.kpimeter.spring.ws.schema.energyMeter.EnergyMeter"/>
  <event-type name="EquipmentChangeState"
              class="fi.tut.kpimeter.spring.ws.schema.robot.EquipmentChangeState"/>
  <event-type name="ConveyorNotification"
              class="fi.tut.kpimeter.spring.ws.schema.conveyor.NotificationMessage"/>
  <event-type name="THLValue" class="fi.tut.kpimeter.model.THLValue"/>
  <event-type name="FlowMsg" class="fi.tut.kpimeter.spring.ws.schema.flow.FlowMsg"/>
  <event-type name="DiffPressureMsg"
              class="fi.tut.kpimeter.spring.ws.schema.spusensors.DiffPressureMsg"/>
  <event-type name="LevelMsg" class="fi.tut.kpimeter.spring.ws.schema.spusensors.LevelMsg"/>
  <event-type name="MoistTempMsg"
              class="fi.tut.kpimeter.spring.ws.schema.spusensors.MoistTempMsg"/>
  <event-type name="QualityMsg"
              class="fi.tut.kpimeter.spring.ws.schema.spusensors.QualityMsg"/>
  <plugin-singlerow-function name="energyInstance"
                             function-class="fi.tut.kpimeter.esper.listener.TimeParser"
                             function- method="calcEnergyInstance" />
  <plugin-singlerow-function name="powerFactor"
                             function-class="fi.tut.kpimeter.esper.listener.TimeParser"
                             function-method="calcPowerFactor" />
  <plugin-singlerow-function name="timeDifference"
                             function-class="fi.tut.kpimeter.esper.listener.TimeParser"
                             function-method="timeDifference" />
 </esper-configuration>
```

APPENDIX 6: AVAILABLE RESOURCES IN THE INFOSTORE AS OF MAY 5TH, 2012

	Source event	Name	Description
Raw data	ConveyorNotification	Conveyor_2	Raw data from the ConveyorNotification event indicating the pallet transfer status on conveyors including cell ID, 'fromZoneId', 'toZoneId', pallet ID, event ID and time stamp from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		Conveyor_3	
		Conveyor_4	
		Conveyor_5	
		Conveyor_6	
		Conveyor_8	
		Conveyor_9	
		Conveyor_10	
		Conveyor_11	
		Conveyor_12	
	EnergyMeter	energyMeter2	Raw data from the EnergyMeter event indicating the 3-phase active energy, active power, reactive energy, reactive power, apparent energy, apparent power, RMS current, RMS voltage as well as line frequency, cell ID and time stamp from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		energyMeter3	
		energyMeter4	
		energyMeter5	
		energyMeter6	
		energyMeter8	
		energyMeter9	
		energyMeter10	
		energyMeter11	
		energyMeter12	
	EquipmentChange State	EquipmentAlarm_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the current state is DOWN from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		EquipmentAlarm_3	
		EquipmentAlarm_4	
		EquipmentAlarm_5	
		EquipmentAlarm_6	
		EquipmentAlarm_8	
		EquipmentAlarm_9	
		EquipmentAlarm_10	
		EquipmentAlarm_11	
		EquipmentAlarm_12	
	EquipmentChange State	EquipmentInitializationComplete_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the current state is SETUP from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		EquipmentInitializationComplete_3	
		EquipmentInitializationComplete_4	
		EquipmentInitializationComplete_5	
		EquipmentInitializationComplete_6	
		EquipmentInitializationComplete_8	
		EquipmentInitializationComplete_9	
		EquipmentInitializationComplete_10	
		EquipmentInitializationComplete_11	
		EquipmentInitializationComplete_12	
	EquipmentChange State	EquipmentPowerOff_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the current state is OFF from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		EquipmentPowerOff_3	
		EquipmentPowerOff_4	
		EquipmentPowerOff_5	
		EquipmentPowerOff_6	
		EquipmentPowerOff_8	
		EquipmentPowerOff_9	
		EquipmentPowerOff_10	
		EquipmentPowerOff_11	
		EquipmentPowerOff_12	
	EquipmentChange	EquipmentSetupComplete_2	Raw data from the EquipmentChangeSatate

	Source event	Name	Description
	State	EquipmentSetupComplete_3	event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the previous state is SETUP from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		EquipmentSetupComplete_4	
		EquipmentSetupComplete_5	
		EquipmentSetupComplete_6	
		EquipmentSetupComplete_8	
		EquipmentSetupComplete_9	
		EquipmentSetupComplete_10	
		EquipmentSetupComplete_11	
		EquipmentSetupComplete_12	
	EquipmentChange State	ItemWorkComplete_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the previous state is READY-PROCESSING-EXECUTING from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		ItemWorkComplete_3	
		ItemWorkComplete_4	
		ItemWorkComplete_5	
		ItemWorkComplete_6	
		ItemWorkComplete_8	
		ItemWorkComplete_9	
		ItemWorkComplete_10	
		ItemWorkComplete_11	
		ItemWorkComplete_12	
	EquipmentChange State	ItemWorkStart_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the current state is READY-PROCESSING-EXECUTING from cells 2, 3, 4, 5, 6, 8, 9 ,10, 11 and 12.
		ItemWorkStart_3	
		ItemWorkStart_4	
		ItemWorkStart_5	
		ItemWorkStart_6	
		ItemWorkStart_8	
		ItemWorkStart_9	
		ItemWorkStart_10	
		ItemWorkStart_11	
		ItemWorkStart_12	
	EquipmentChange State	MaterialHandlingWorkComplete_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the previous state is READY-PROCESSING-ACTIVE from cells 2, 9 ,10, 11 and 12.
		MaterialHandlingWorkComplete_9	
		MaterialHandlingWorkComplete_10	
		MaterialHandlingWorkComplete_11	
		MaterialHandlingWorkComplete_12	
	EquipmentChange State	MaterialHandlingWorkStart_2	Raw data from the EquipmentChangeSatate event indicating cell ID, tool ID, recipe number, device type, pallet ID, the current state, the previous state of the robot when the current state is READY-PROCESSING-ACTIVE from cells 2, 9 ,10, 11 and 12.
		MaterialHandlingWorkStart_9	
		MaterialHandlingWorkStart_10	
		MaterialHandlingWorkStart_11	
		MaterialHandlingWorkStart_12	
	FlowMsg	panel_flow	Indicates the flow rates and timestamp from a flow meter.
	QualityMsg	spu_contamination	Indicates the quality of the oil and timestamp from the silent power unit.
	MoistTempMsg	spu_moist_temp	Indicates the moisture and temperature of the oil and timestamp from the silent power unit.
	DiffPressureMsg	spu_diffPressure	Indicates the pressure difference of the oil in bar and timestamp from the silent power unit.
	LevelMsg	spu_level	Indicates the level of the oil in percentage and timestamp from the silent power unit.
	THLValue	THL-1-31	Raw message indicating values of temperature (in degree Celsius), relative humidity (in percentage) and ambient light (in lux) from THL sensors number 31, 35 and 38.
		THL-1-35	
		THL-1-38	

	Source event	Name	Description
KPI	THLValue	avg_day_hum	Indicates the average day humidity and sensor ID.
		avg_day_temp	Indicates the average day temperature and sensor ID.
		avg_room_temp	Indicates the average temperature of all sensors.
		max_ill_lvl	Indicates the maximum illumination level.
		max_temp	Indicates the maximum temperature.
		min_ill_lvl	Indicates the minimum illumination level.
		min_temp	Indicates the minimum temperature.
	EquipmentChange State	camx_state_time_2	Indicate the duration, cell ID, device type, starting and ending timestamp of an IPC-2541 state from cells 2, 3, 4, 5, 6, 8, 9, 10, 11 and 12.
		camx_state_time_3	
		camx_state_time_4	
		camx_state_time_5	
		camx_state_time_6	
		camx_state_time_8	
		camx_state_time_9	
		camx_state_time_10	
		camx_state_time_11	
		camx_state_time_12	
	EquipmentChange State	cell production rate batch_2	Indicate the production rate of cells 2, 3, 4, 5, 6, 8, 9, 10, 11 and 12, cell ID and timestamp on an hourly basis.
		cell production rate batch_3	
		cell production rate batch_4	
		cell production rate batch_5	
		cell production rate batch_6	
		cell production rate batch_8	
		cell production rate batch_9	
		cell production rate batch_10	
		cell production rate batch_11	
		cell production rate batch_12	
	EnergyMeter	cell_energy_consumption_2	Indicate the energy consumption from the robot, the cabinet and the conveyor, cell ID and timestamp on cells 2, 3, 4, 5, 6, 8, 9, 10, 11 and 12.
		cell_energy_consumption_3	
		cell_energy_consumption_4	
		cell_energy_consumption_5	
		cell_energy_consumption_6	
		cell_energy_consumption_8	
		cell_energy_consumption_9	
		cell_energy_consumption_10	
		cell_energy_consumption_11	
		cell_energy_consumption_12	
	EnergyMeter	cell_power_consumption_2	Indicate the power consumption from the robot, the cabinet and the conveyor, cell ID and timestamp on cells 2, 3, 4, 5, 6, 8, 9, 10, 11 and 12.
		cell_ power_consumption_3	
		cell_ power_consumption_4	
		cell_ power_consumption_5	
		cell_ power_consumption_6	
		cell_ power_consumption_8	
		cell_ power_consumption_9	
		cell_ power_consumption_10	
		cell_ power_consumption_11	
		cell_ power_consumption_12	
	EnergyMeter	energy_efficiency_variables_2	Besides the raw data in EnergyMeter event, the variables also include 3-phase power factor.
		energy_efficiency_variables_3	
		energy_efficiency_variables_4	
		energy_efficiency_variables_5	
		energy_efficiency_variables_6	
		energy_efficiency_variables_8	

	Source event	Name	Description
		energy_efficiency_variables_9	
		energy_efficiency_variables_10	
		energy_efficiency_variables_11	
		energy_efficiency_variables_12	
	EquipmentChange State EnergyMeter	unit_energy_consumption_2	Indicates the energy consumption for processing one product in Wh, cell ID, pallet ID, start and ending time and timestamp from cells 2, 3, 4, 5, 6, 8, 9, 10, 11, 12.
		unit_energy_consumption_3	
		unit_energy_consumption_4	
		unit_energy_consumption_5	
		unit_energy_consumption_6	
		unit_energy_consumption_8	
		unit_energy_consumption_9	
		unit_energy_consumption_10	
		unit_energy_consumption_11	
		unit_energy_consumption_12	