Diploma Thesis

“Shop-floor monitoring tool designed for Industry 4.0: Combining wireless sensor networks, web platform and data analytics for industrial environments.”

Xanthopoulos Nikitas

SUPERVISORS

DIMITRIS MOURTZIS
DEPT. OF MECHANICAL ENGINEERING & AERONAUTICS
ATHANASIOS TSAKALIDIS
DEPT. OF COMPUTER ENGINEERING & INFORMATICS

PATRAS, AUGUST 2017
Abstract

Uncertainties in materials, machines, operators and other unexpected events demand real-time decisions reducing significantly the production rate of manufacturing systems, increasing the demand for real-time monitoring and control. Automated acknowledgement and investigation of such failures at the moment they occur interfaced with scheduling services who provide alternatives based on the current situation of resources, results in a flexible, less expensive and resource-aware production.

Easy obtainable, low cost, scalable cutting-edge technologies including wireless sensor networks, cloud services and no-SQL databases have prepared the ground for an architecture which can achieve a monitoring module using mainly open source/hardware tools. Important information such as the current status of resources can be digitalized through the utilization of the measurements gathered from embedded sensors in the shop-floors machine tools. A web platform which is also interconnected with remote services to transmit necessary information using Web 3.0 standards stores, processes and visualizes data.

This thesis is focused on the implementation of those technologies in a platform which is capable of sensing and monitoring the status of the machine tools and several performance indicators. The developed platform is a) constantly connected to a scheduling service in order to provide a resource-aware, adaptive scheduling and condition-based preventive maintenance, and b) remains scalable and flexible in order to provide a solid ground for further development and integrations of services in the area of Industry 4.0.

KEYWORDS: monitoring system, web platform, wireless sensor network, big-data, adaptive scheduling, industry 4.0.
Acknowledgements

The core of the monitoring module presented in this thesis has been developed in the context of the EU-funded project CAPP4SMEs (2014-2016) on behalf of Laboratory of manufacturing Systems, Department of Mechanical Engineering and Aeronautics, University of Patras. My main responsibilities were initially to propose the general architecture and then implement it software-wise. Characteristics and capabilities are presented in chapters 2-3. Additional work done for the matters of this thesis is the interface, between the monitoring module and the scheduling engine (as explained in chapter 4).

Last but not least, I would like to thank:

- Associate Professor D. Mourtzis for supervising and offering the opportunity of working on new technologies,
- my thesis advisors K. Vlachou and C. Giannoulis for their great co-operability while writing and developing this thesis' contents,
- Professor A. Tsakalidis for all his support throughout my University studies.
# Table of Contents

Abstract ................................................................................................................................. 3
Acknowledgements .............................................................................................................. 4
Executive Summary ............................................................................................................. 13

1 State of the art .................................................................................................................. 14

2 Monitoring System Overview ........................................................................................ 17

3 Monitoring System Description ....................................................................................... 18

3.1 Wireless Sensor Network .............................................................................................. 18

3.1.1 Sensors ...................................................................................................................... 18

3.1.2 Microcontroller board .............................................................................................. 18

3.1.3 Wireless Communication ......................................................................................... 19

3.1.4 Gateway ................................................................................................................... 20

3.1.5 Security .................................................................................................................... 21

3.2 Web Platform ................................................................................................................ 21

3.2.1 Server ....................................................................................................................... 22

3.2.2 HTTP Server ............................................................................................................ 22

3.2.3 Database .................................................................................................................. 22

3.2.4 Web application ....................................................................................................... 23

3.2.5 User Interface .......................................................................................................... 24

3.2.6 User Roles ............................................................................................................... 25

4 Monitoring module implementation ................................................................................. 29

4.1 Wireless sensor network implementation .................................................................... 29

4.1.1 Micro-controller board ............................................................................................ 29
10 Abbreviations ...........................................................................................................................................87
11 References ...............................................................................................................................................89
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of the monitoring system.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Non-invasive current sensor.</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Arduino Mega 2560[18].</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>XBee Pro series 1, 2.4GHz[20].</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Arch Linux OS [22] [23].</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>Raspberry Pi Model B+ [21].</td>
<td>20</td>
</tr>
<tr>
<td>3.6</td>
<td>Software architecture</td>
<td>21</td>
</tr>
<tr>
<td>3.7</td>
<td>Document oriented database</td>
<td>23</td>
</tr>
<tr>
<td>3.8</td>
<td>Welcome page.</td>
<td>25</td>
</tr>
<tr>
<td>3.9</td>
<td>Casual user's main page</td>
<td>26</td>
</tr>
<tr>
<td>3.10</td>
<td>Available options as an operator for a machine tool.</td>
<td>26</td>
</tr>
<tr>
<td>3.11</td>
<td>Mean time between failure</td>
<td>27</td>
</tr>
<tr>
<td>3.12</td>
<td>Messages between operators and maintainers.</td>
<td>27</td>
</tr>
<tr>
<td>3.13</td>
<td>Shop floor schedule.</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Microcontroller board functionality</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Gateway functionality</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Rescheduling on the fly flowchart</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Relations between database collections</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>Sensor measurements (dynamically updated)</td>
<td>52</td>
</tr>
</tbody>
</table>
Figure 4.6 Task related GUI page, on top is the original schedule, then the monitored. In second screenshot example task relationships are shown ...........................................53

Figure 4.7 User input for machine specifications ..........................................................54

Figure 4.8 User input for sensor calibration. .................................................................54

Figure 4.9 Main functionality page for operator input. ..................................................55

Figure 5.1 Overview of the communication between the two different services and adaptive scheduling options. ......................................................................................................................58

Figure 5.2 Flowchart of the communication between monitoring system and schedule engine ..............................................................................................................................59

Figure 7.1 Initial schedule ..............................................................................................67

Figure 7.2 Fixing the end time of t1 as the machine keeps processing after t1’s defined end time .......................................................................................................................68

Figure 7.3 A conflict was detected, t1 was monitored to be processed less than 30 seconds before t2’s start time, hence all remaining tasks (t2, t3, t5) are shifted forward 30 seconds .......................................................................................................................68

Figure 7.4 Mazak changed to DOWN status while processing t2, this is the result of the rescheduling ..................................................................................................................69

Figure 7.5 t2 kept processing (total time 10 mins 17 seconds), hence t3 and t5 were shifted forward accordingly. The due time exceeds the user defined maximum schedule delay, so a reschedule is requested ........................................................................................................69

Figure 7.6 This is the resulting schedule, this time Fresadora is assigned a task .......70

Figure 7.7 Fresadora’s sensor node kept sending non-processing status, t3’s start time is shifted forward until this machine’s status is changed to processing .......................70

Figure 7.8 This is the final monitored schedule, taking into account that XYZ’s sensor node sent non-processing values before the assigned task’s (t5) defined end time.....71
List of Code Snippets

Code Snippet 4.1 EmonLib functions ................................................................. 29
Code Snippet 4.2 Microcontroller board program ............................................ 31
Code Snippet 4.3 Service to connect VPN after network initialization .............. 33
Code Snippet 4.4 Service to start gateway metropogram after system startup ...... 33
Code Snippet 4.5 Static IP address settings ....................................................... 33
Code Snippet 4.6 Gateway metropogram example ............................................. 36
Code Snippet 4.7 Persistence example using Ruby and MongoDB ..................... 38
Code Snippet 4.8 Class in Ruby with MongoDB persistence .............................. 38
Code Snippet 4.9 Aggregation example .............................................................. 41
Code Snippet 4.10 Information fusion implementation ........................................ 42
Code Snippet 4.11 Rescheduling on the fly ......................................................... 44
Code Snippet 4.12 Tool usage time estimation External services ......................... 46
Code Snippet 4.13 Library developed to integrate ADVEON (commercial software) to the web platform .......................................................... 48
Code Snippet 5.1 Library developed for monitoring system and scheduling engine connection .................................................................................................... 62
Code Snippet 6.1 Library that integrates a commercial software tool with monitoring system ........................................................................................................ 64
Code Snippet 9.1 Data processing library example in Ruby .................................. 85
Code Snippet 9.2  Schema of XML file generated by LMS MM when a rescheduling is triggered ..........................................................86

Code Snippet 9.3  Example XML file generated by the scheduling service when a rescheduling is triggered. ..........................................................87
Executive Summary

This thesis presents the characteristics, capabilities and evaluation of a cyber-physical monitoring for monitoring and controlling manufacturing systems dedicated for today’s industry. More specifically, this thesis includes:

- The implementation of a wireless sensor network to capture and transfer data from the machines tools.
- The implementation of the software platform to store and analyse the data.
- The implementation of algorithms for resource-aware scheduling and condition-based maintenance.
- The implementation of security and encryption protocols.

Software and hardware have been appropriately developed to sense and analyse some aspects of a machine tool as near to real-time as possible. A web platform receives data from wireless sensor networks installed in shop-floor and processes them for a variety of information extraction. Interoperability of this service has been a major task in order to enable communication with other (research and commercial) existing services.

One direction of the data analysis is focused on the correct identification of completed tasks’ due time (relative to a given schedule), enabling the option of immediate rescheduling if certain conditions are met. In that case, a request for a new schedule that includes necessary shop-floor resources’ information is made to a scheduling external service -an engine that creates schedules given specific parameters-. The latter responds with a schedule that is aware of shop-floor status, making resource aware (re)scheduling feasible.

All components of this system are open-source/open-hardware making it appropriate for further developments, while its technologies are chosen to offer scalability at lowest effort. Moreover, because of its ability to handle diverse and robust data, it offers a solid base for more complicated algorithms from the fields of data mining and machine learning.
1 State of the art

The era of information and technology; an era which can be characterized by the increasing amount of information flowing through wires and other mediums for human-only, human-machine and machine-only consumption. Dedicated computing devices are able to sense\(^1\) some aspects of physical world, enchanted with human input in some cases, creating a first link from physical to the virtual world. By digitizing aspects of physical space, a computing device can process a variety of real life parameters and forward the output for (predefined in most cases) actions, establishing the reverse link from virtual to physical world. This two-way communication can be described as a Cyber Physical system [1], [2], [3].

The overwhelming growth of technology has also shown a great impact in the fields of microcomputers, microcontrollers and sensor boards making the above pattern feasible in terms of cost and complexity. Moreover, Internet's rapid expansion in the last decade has promoted system architectures where these devices are connected between them or/and with larger systems able to perform heavier calculations (Internet of Things [4][5]). These larger systems can be hosted on another lately developed internet-related technology, cloud services, which offer flexibility and scalability when deploying a web platform since the hardware is virtual (more computing resources can be added without having to actually install more hardware). Their providers take care of any effort related to the maintenance of physical servers.

The combination of CPSs, IoT and cloud computing in manufacturing environments (Industry 4.0) can greatly benefit the three pillars of the modern industry: the process optimization, the optimized resource consumption, and the creation of complex autonomous systems [6].

A diversity of sensors [9] can be installed on existing hardware to monitor their performance [16]: accelerometers have already been used for tool wear estimation [10],

\(^1\)in terms of measuring an effect
low-cost non-invasive electrical current sensors for defining the status of the machine tool and for calculating the energy consumption of the machine tool, [7],[17].

In addition, web platforms are suited perfectly in the role of accepting, processing, storing data captured from WSNs and provide cross platform user interaction (web browsers), while existing IT tools able to communicate through web-services can be easily integrated. Time-series data collected from embedded machines are heterogeneous and massive, classifying them as “Big Data”. Although some relational databases with the correct settings can handle Big Data effectively, the effort of doing so in no-SQL databases is much less. Hence, no-SQL databases - schema-less databases that are clustered horizontally- are connected to web platforms in order to serve the latter's needs of creating, updating, serving and deleting data. More complex algorithms from the fields of Information Fusion, Data Mining and Machine learning are used to extract information non visible on “the first look”. Predictive and preventive maintenance [14], control of manufacturing systems are some examples already covered in literature [15].

Advanced data analytics provide the IT production systems with meaningful information, transforming them from isolated to adaptive. Several approaches have been reported in literature related to dynamic scheduling. Chryssolouris et al in [61],[62] proposes a dynamic scheduling approach for manufacturing job shops using genetic algorithms and multiple criteria evaluation of alternative schedules. The development of a multi-level adaptive control and scheduling solution for reconfigurable manufacturing environments from a real-time system automation perspective is proposed in [63]. Solution approaches for real-time control of manufacturing systems is also proposed by Monostori et al in [64], where a scheduling system integrated with production monitoring sub-systems is introduced to deal with common daily production disturbances.

Moreover, the data captured by the monitoring systems can be further used for condition-based maintenance of machine tools [65]. Predictive maintenance of machinery gives the ability to ensure product quality, perform just-in-time maintenance, minimize equipment downtime and avoid machine-tool failure [66], [67]. In addition to that, a predictive maintenance platform is proposed by Efthymiou et al
[68],[69] for production systems maintenance, taking into consideration data acquisition systems, knowledge management and a maintenance dashboard.

Yet, most of the reported scheduling methods address the static scheduling problem without considering random events such as random job arrivals, machine-tool failures and maintenance. Moreover, approaches concerning production planning and control are not capable of dealing with real-time data [15]. This thesis is focused on those matters; the proposed monitoring system is developed for the needs of today's industry while remaining flexible for the needs of tomorrow.
2 Monitoring System Overview

One of the main goals of this thesis to sense the status of machine tools on production as close to real time as possible. This has been achieved by combining embedded sensors, the internet and data processing. A wireless sensor network –consisting of nodes which mainly measure current flowing through machine tools' wires- transmits collected data over the internet to the web platform which was deployed on an IaaS. cloud service virtual machine. A no-SQL database stores and serves data while a web application processes it and turns it into readable form for human interaction.

Figure 2.1 Overview of the monitoring system.
3 Monitoring System Description

3.1 Wireless Sensor Network

As mentioned before, the rapid expansion of open source/hardware projects (including the field of microcomputers/microcontrollers) has made feasible the development of low-cost sensor nodes capable of communicating through radio signals. A many-to-one distribution is used, where each node interacts with the microcomputer-gateway. This gateway forwards data over the internet to the web platform. The following subsections include information about the hardware and the software that created such a W.S.N.

3.1.1 Sensors

Non-invasive current sensors transform magnetic field created by current flow around wires into voltage. Specifications of used sensors are 0-30A input current, 0-1V output and 1mA calculated accuracy.

![Non-invasive current sensor](image)

Figure 3.1 Non-invasive current sensor.

3.1.2 Microcontroller board

The output of CT sensors is being driven to analog input pins of a microcontroller board which transforms it into the final form of current (mA). An Arduino Mega 2560 has been picked to do this task because of its' wide open source community and already developed libraries[20]. It is based on ATmega2560 microcontroller which has maximum operating frequency 16Mhz, a variety of I/O lines including digital/analog,
32 general purpose working registers, 256KB internal storage flash memory, 8KB SRAM and 4KBEEEPROM and 10bit A/D converter. The voltage-to-current transformation is achieved software-wise by calling some functions from EmonLib Arduino open source library [19].

![Arduino Mega 2560](image1.png)

Figure 3.2 Arduino Mega 2560[20].

### 3.1.3 Wireless Communication

Measurements are then being transmitted through radio communication which is achieved using the IEEE 802.15.4 [21] protocol. This protocol is suited for small-area low power digital radios. Hardware-wise, XBee radio circuits [22] are installed on each node and the gateway, whose characteristics include a 300 meters indoor and 1600 meters outdoor range, 2.4Ghz transmission frequency, 256 KBps maximum transfer rate and 128-bit encryption. Each node has a preset ID which assigns packets to each machine tool.

![XBee Pro series 1, 2.4GHz](image2.png)

Figure 3.3 XBee Pro series 1, 2.4GHz[22].
3.1.4 Gateway

A microcomputer board collects all transmitted data from the microcontroller boards and sends them to the web platform through HTTP requests. Because of its' wideopen source community, a Raspberry Pi Model B+ [23] is being used as this network's gateway. B+ version consists of a quad-core ARM Cortex-A7 CPU operating at 900Mhz with 1GB of RAM. Arch Linux [24] [25] has been picked as operating system because of its wide open source community support and ease of administration. Moreover, Arch Linux basic installation is a minimal collection of necessary software, a lightweight Linux operating system appropriate for devices with lower computing capabilities. Extra software can be installed through repositories which offer a huge variety of latest-version packages. The metro-program that collects and forwards data is in Python programming language and is registered to run accordingly as a service inside the operating system. If any remote maintenance is needed the gateway can also be accessed through Secure Shell.

Figure 3.4 Arch Linux OS [24] [25]

Figure 3.5 Raspberry Pi Model B+ [23].
3.1.5 Security

The wireless communication inside the W.S.N. is encrypted using 128-bit AES in order to avoid wiretapping inside the shop-floor. The microcomputer board's Linux kernel is also strengthened using specific rules for incoming and outgoing connections, allowed users and running services. Software is regularly updated to the latest version.

3.2 Web Platform

![Software architecture](image)

Figure 3.6 Software architecture
3.2.1 Server

The web platform was deployed and tested on a virtual machine provided from project partners (KTH Royal Institute of Technology [26]) Infrastructure as a Service cloud services for over 1 year. Since 2016 it has been hosted on L.M.S. local servers. In all cases the operating system is Linux based (OpenSUSE, Debian, Ubuntu, Arch Linux distributions have been tested) with minimum requirements 2GB of RAM and 2 CPUs 2.4Ghz. Virtual machines in cloud services come much handier when deploying and maintaining a web platform; no hardware must be monitored or owned. However, the platform can be hosted in any Linux computer. An easily clustered no-SQL database was selected in combination with an easy to develop and maintain web framework as described in the following paragraphs.

3.2.2 HTTP Server

Nginx [41] is an open-source, stable, high-performance server and reverse proxy. HTTP server's responsibility is to serve static files and forward dynamic requests to the specified process. Nginx can be used as a reverse proxy, which means it can split traffic to multiple servers.

3.2.3 Database

MongoDB [34] [35], a noSQL [38] document oriented schema-free database, was selected as the main database because of its natural ability to scale into clusters, good storage of time-series data, speed and aggregation framework. It stores data into BSON [39], an efficient, lightweight, traversable binary-encoded format with a variety of supported data types. Each document represents an instance of a specific object and each collection the collection of a particular type of an object.

Each document has a unique database-wide 12 byte-long ID [40], which is created by the timestamp of creation time, a machine identifier, a process identifier and a counter. MongoDB’s aggregation framework allows the development of custom [36] functions in Javascript programming language which reduce significantly the calculation time when processing large amount of data.
3.2.4 Web application

The web application was developed using Ruby on Rails framework [27] [28], a collection of libraries written in Ruby programming language. R.o.R. uses Model-View-Controller [29], Convention over Configuration [30], Don't Repeat Yourself [31], Active Record [32] design patterns which speed up development and provide more
transparent code hierarchy. In addition, it allows the development of RESTful APIs, types of services that enable interoperability between computers, while it is also open-source and has a wide community that provides a lot of libraries targeting the web. At this point there is only one server hosting the main application but this can be split into separate instances (e.g. a server for sensor measurements, a server for user interaction).

3.2.5 User Interface

Sensor measurements and processed data are visualized using mainly graphs, dynamic and static, with Google Charts, a Javascript library that offers a variety of chart types (gantt, timelines, column, e.t.c.). GUI was developed to provide simple but full functional pages by adding glyphicons (small icons that can be clicked to trigger functions or redirects) and Bootstrap, an HTML/CSS/Javascript front-end framework. Moreover, appropriate database models have been created to host the different types of users (normal users, operators, maintainers, shop-floor managers)
3.2.6 User Roles

Casual User: This user is responsible for inserting facility data in the monitoring system. He is also responsible for setting some important parameters for monitoring system’s internal algorithms, like thresholds between non-processing and processing for each machine tool. He has limited access to visualization and to immediate
decisions, as explained in operator user model.

![Operator user model](image)

**Operator:** He is responsible for all immediate decisions that have to be made in the system regarding a specific machine tool, like setting its status to down and changing its tool magazines.

![Available options as an operator](image)

**Maintainer:** The monitoring system has functionalities related to maintenance of machine tools. He has access to specific pages that contain information about the machining time and is able to send messages to operators through a user interface.
Figure 3.11 Mean time between failure.

Figure 3.12 Messages between operators and maintainers.

Shop floor manager: He has a more general view of the system, while is able to access all data visualization related pages. The overall schedule page (a page that includes graphs in shop-floor, not individual machine, level) is visible only to him and can request a reschedule at any time.
Figure 3.13 Shop floor schedule.
4 Monitoring module implementation

4.1 Wireless sensor network implementation

4.1.1 Micro-controller board

Arduino’s firmware is developed in Wiring, an open source programming framework for micro-controllers. Sketches (programs to be compiled and run on the device) are written in C/C++ language. Arduino's analog input pins connected to a sensor board with 4 current sensors read constantly values in 0..5V range and translate them to current using this formula:

```cpp
//from EmonLib.cpp
double EnergyMonitor::calcIrms(int NUMBER_OF_SAMPLES) {

  if defined emonTxV3
  int SUPPLYVOLTAGE=3300;

  #else
  int SUPPLYVOLTAGE = readVcc();
  #endif

  for (int n = 0; n < NUMBER_OF_SAMPLES; n++)
  {
    lastSampleI = sampleI;
    sampleI = analogRead(inPinI);
    lastFilteredI = filteredI;
    filteredI = 0.996*(lastFilteredI+sampleI-lastSampleI);

    // Root-mean-square method current
    // 1) square current values
    sqI = filteredI * filteredI;
    // 2) sum
    sumI += sqI;
  }

  //Reset accumulators
  sumI = 0;

  returnIrms;
}
```

Code Snippet 4.1 EmonLib functions.
Arduino is programmed to sample every 0.5 seconds and send readings through the serial interface to XBee radio antenna. This is the actual code run on every Arduino:

```c
#include <XBee.h>
#include <EmonLib.h>

#define CURRENT_CALIBRATION 6.2
#define RMS_CAL 424;

XBee antenna = XBee();
uint8_t payload[16];
Tx16Request tx = Tx16Request(0x01, payload, sizeof(payload));

EnergyMonitor spindle;
EnergyMonitor tableX;
EnergyMonitor tableY;
EnergyMonitor tableZ;

volatile double spindle_reading;
volatile double tableX_reading;
volatile double tableY_reading;
volatile double tableZ_reading;

volatile uint8_t *spindleByteArray;
volatile uint8_t *tableXByteArray;
volatile uint8_t *tableYByteArray;
volatile uint8_t *tableZByteArray;

void get_sensor_readings();
void send_sensor_readings();
void double_to_payload();

void setup(){
  Serial.begin(57600);
  antenna.setSerial(Serial);
}

void loop(){
  get_sensor_readings();
  send_sensor_readings();
}

void get_sensor_readings(){
  spindle_reading = spindle.calcIrms(RMS_CAL);
  tableX_reading = tableX.calcIrms(RMS_CAL);
  tableY_reading = tableY.calcIrms(RMS_CAL);
  tableZ_reading = tableZ.calcIrms(RMS_CAL);
}

void send_sensor_readings(){
  float_to_payload( spindle_reading, 0);
  float_to_payload( tableX_reading, 3);
  float_to_payload( tableY_reading, 7);
```
```c
float_to_payload(tableZ_reading, 11);
xbee.send(tx);
}

void double_to_payload( double value, int pointer) {
  uint8_t *temp;
  temp = (uint8_t *)&value;
  payload[pointer] = temp[3];
  payload[pointer + 1] = temp[2];
  payload[pointer + 2] = temp[1];
  payload[pointer + 3] = temp[0];
}
```

Code Snippet 4.2 Microcontroller board program.

- CURRENT CALIBRATION defined in line 4 is a parameter expected by EmonLib and depends on a resistor in sensor board (300 Ohm in LMS sensor board) and is used in lines 34-37.
- RMS CAL defined in line 5 is another parameter expected by EmonLib and represents the number of samples that will be collected and averaged in each measurement.
- Lines 7-9 initialize radio concerned variables. Line 7 initializes the instance that will be used from XBee's library and provides functions to access XBee hardware. Line 8 is the actual payload that will be sent over RF, a 16 size Byte array. Line 9 is another instance from a class provided by XBee library and is responsible for controlling antenna's output (used in line 59).
- Lines 11-14 initialize instances of EmonLib's class responsible to sense and calculate current.
- Lines 16-19 initialize variables that will store sensor readings, one for each sensor.
- Lines 21-24 initialize variables needed to cast sensor readings from double type to byte array so they can be later on stored in payload. In the microcontroller board used, each float and double consists of 4 Bytes.
- Lines 26-28 declare the functions that will be used.
- Lines 30-38 define the setup method, in which serial, radio and sensor reading instances are being set properly before starting action.
- Lines 40-42 define the main loop function that will be repeated until the power is off. Inside it, two functions that are explained later are called.
• Lines 45-50 define the function that reads data collected from EmonLib and stores them to (global) program variables.

• Lines 52-59 define the function that casts (transforms, explained in the following bullet) previous bullet’s results to necessary format and sends data by calling XBee library's function in line 58.

• Lines 62-70 define a function that is needed to cast doubles (and floats in this board) into byte array.

---

4.1.2 Microcomputer - Gateway

4.1.2.1 Operating system

As mentioned in chapter 2 the microcomputer board is a Raspberry Pi with Arch Linux operating system. While developing the monitoring module for the EU founded project
some challenges including connecting the gateway to a VPN, setting static IP address and starting the gateway metro program automatically raised. Hence, some system services were installed in “/etc/systemd/system/” and “/etc/systemd/network/” directories and enabled, as presented in the following code snippets:

```
[Unit]
Description=VPN connection
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
ExecStart=/usr/bin/openvpn /path/to/vpn/configuration/file
RestartSec=5
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Code Snippet 4.3 Service to connect VPN after network initialization

```
[Unit]
Description= python script

[Service]
ExecStart=/usr/bin/python2 /path/to/gateway-metroprogram.py
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

Code Snippet 4.4 Service to start gateway metroprogram after system startup

```
[Match]
Name=eth0_static

[Network]
DNS= X.X.X.X
Address= X.X.X.X/24
Gateway= X.X.X.X
```

Code Snippet 4.5 Static IP address settings

Also, to strengthen gateway's security, a firewall has been set up accordingly by closing all ports for incoming connections. If SSH is needed for updates/maintenance, it can be setup as a service to listen to a specific port (which must be allowed from the firewall) and set it to decline root logins, accept a specific maximum number of login tries and allow connections from specific IP addresses only. To do so, the file /etc/ssh/sshd\_config has to be edited.
4.1.2.2 Metroprogram

The gateway is responsible for collecting measured data from WSN nodes through XBee hardware and propagates them to the web platform. An XBee radio with USB adaptor was connected to Raspberry Pi, accessible in “/dev/ttyUSB{Y}” (Y corresponds to 0,1,2...) file system location. The Python metro-program is the following code listing:

```python
from xbee import XBee

import serial, datetime, re, sys, time, struct, requests, json, ssl, os, subprocess, threading, socket, smtplib

url = 'http://X.X.X.X/daq_input/index'
headers = { 'content-type': 'application/json' }

ser = serial.Serial('/dev/ttyUSB0', 57600)
xbee = XBee(ser, escaped=True, callback=handlePacket)
resetMin = datetime.datetime.now().minute
closed=False

def main():
    while closed == False:
        try:
            loopMin = datetime.datetime.now().minute
            loopSec = datetime.datetime.now().second

            if loopSec % 15 == 0 and resetedXBee == False:
                resetedXBee = True
                print '\nreseting serial...\n'
                xbee.halt()
                ser.close()
                time.sleep(0.001)

            elif loopSec % 15 == 1 and resetedXBee == True:
                resetedXBee = False

            except KeyboardInterrupt:
                xbee.halt()
                ser.close()
                closed=True
                break

        except KeyboardInterrupt:
            xbee.halt()
            ser.close()
            closed=True
            break

def handlePacket(data):
    try:
        values = data.values()[0:1]
        valuesStr = values.pop()
```
```python
def send_message(spindle, rpm, tableX1, tableY1, tableZ1):
    try:
        print("spindle: "+str(spindle))
        print("x axis: "+str(tableX1))
        print("y axis: "+str(tableY1))
        print("z axis: "+str(tableZ1))
        print("c axis: "+str(tableC1))
        print("b axis: "+str(tableB1))
        print("toolchanger: "+str(tool1))
    except requests.exceptions.Timeout:
        print("*************** SLOW INTERNET ***************")
    except requests.exceptions.ConnectionError:
        print("*** CONNECTION ERROR ***")
    except ssl.SSLError:
        print("*** SSL ERROR ***")

    thread = Thread(target = send_message, args = (spindle, tableX1, tableY1, tableZ1))
    thread.start()
    thread.join()

    spindle0= (valuesStr).encode("hex")[0] + (valuesStr).encode("hex")[1]
    spindle1= (valuesStr).encode("hex")[2] + (valuesStr).encode("hex")[3]
    spindle2= (valuesStr).encode("hex")[4] + (valuesStr).encode("hex")[5]
    spindle3= (valuesStr).encode("hex")[6] + (valuesStr).encode("hex")[7]
    spindle= spindle0+ spindle1+ spindle2+ spindle3

    tableY0= (valuesStr).encode("hex")[8] + (valuesStr).encode("hex")[9]
    tableY1= (valuesStr).encode("hex")[10] + (valuesStr).encode("hex")[11]
    tableY2= (valuesStr).encode("hex")[12] + (valuesStr).encode("hex")[13]
    tableY3= (valuesStr).encode("hex")[14] + (valuesStr).encode("hex")[15]
    tableY= tableY0+tableY1+tableY2+tableY3

    tableZ0= (valuesStr).encode("hex")[16] + (valuesStr).encode("hex")[17]
    tableZ1= (valuesStr).encode("hex")[18] + (valuesStr).encode("hex")[19]
    tableZ2= (valuesStr).encode("hex")[20] + (valuesStr).encode("hex")[21]
    tableZ3= (valuesStr).encode("hex")[22] + (valuesStr).encode("hex")[23]
    tableZ= tableZ0+tableZ1+tableZ2+tableZ3

    tableX0= (valuesStr).encode("hex")[24] + (valuesStr).encode("hex")[25]
    tableX1= (valuesStr).encode("hex")[26] + (valuesStr).encode("hex")[27]
    tableX2= (valuesStr).encode("hex")[28] + (valuesStr).encode("hex")[29]
    tableX3= (valuesStr).encode("hex")[30] + (valuesStr).encode("hex")[31]
    tableX= tableX0+tableX1+tableX2+tableX3

    spindle= round(struct.unpack('!f', spindle.decode("hex"))[0], 4)
    tableX1= round(struct.unpack('!f', tableX.decode("hex"))[0], 4)
    tableY1= round(struct.unpack('!f', tableY.decode("hex"))[0], 4)
    tableZ1= round(struct.unpack('!f', tableZ.decode("hex"))[0], 4)
    rpm=int(rpm,16)
```

print "rpm: " + str(rpm)
payload= {'machine_id': '56210bed35f67c536a000014', 'spindle': spindle, 'tableX': tableX1, 'tableY': tableY1, 'tableZ': tableZ1}
request=requests.post(url, data=json.dumps(payload), headers=headers, verify=False)
except requests.ConnectionError:
    pass
except serial.SerialException:
    pass
main()

Code Snippet 4.6 Gateway metropogram example

- Lines 1-2 import already developed (open source) libraries used.
- Lines 4-11 initialize some global variables needed, like the URL of the web platform, headers for the requests and initialization of serial. Also, as provided by XBee library, the function handlePacket() (defined in line 38) becomes a callback function which will be called every time XBee radio has captured data.
- Lines 13-35 define the main function. It is the main loop of the program, in which every 15 seconds the serial is restarted (a lot of delay from data buffering has been observed otherwise).
- Lines 38-84 define the function that will be called every time XBee antenna hardware has data. In this function data are being cast from byte arrays to floats and threads are created to call the next function, send\_message()
- Lines 87-102 define the function that makes the actual request to the web platform which contains sensed data. To do so, data are encoded to JSON format, a lightweight data interchange format inspired by Javascript object literal system [59].
- Finally, in line 104 main function is called.
Figure 4.2 Gateway functionality
4.2 Web application implementation

The web application, as mentioned in chapter 2, has been developed in Ruby programming language. Ruby is a dynamic object oriented general purpose programming language designed mid-1990s [44]. It has come under great appreciation after the release of Ruby on Rails, a Ruby framework (group of packages) focused on building web applications. Its main ideas are also presented in chapter 2.

Database schemas have been declared through Ruby classes containing the correct fields and relations. This was done via Mongoid [45] package, a Ruby-MongoDB driver. An example of such a declaration can be seen in the following code snippet:

```ruby
1. require 'lib_machines'
2. require 'lib_daq_datum'
3.
4. class DaqDatum
5.   include Mongoid::Document
6.   include Mongoid::Timestamps
7.
8.   field :spindle, type: Float
9.   field :tableX, type: Float
10.  field :tableY, type: Float
11.  field :tableZ, type: Float
12.  field :current_fusion, type: Integer
13.  field :task_id, type: String
14.  field :tool_id, type: String
15.
16.  belongs_to :machine
17.
19.  index({current_fusion: 1})
20.  after_create :set_things_up
21.
22.  private
23.  def set_things_up
24.    tmpFusion = LibMachines::calculate_information_fusion(self.machine)
25.    self.update_attribute(:current_fusion, tmpFusion)
26.  end
27.
28.  LibDaqDatum::check_current_task(self)
29.  end
30.
31. end
```

Code Snippet 4.7 Persistence example using Ruby and MongoDB.

Code Snippet 4.8 Class in Ruby with MongoDB persistence.
The above class declaration includes the following:

- Lines 1-2 import two libraries that have been developed and will be explained later on.
- Line 5: This is the base module for all domain objects that need to be persisted to the database as documents [46].
- Line 6: Forces the usage of created\_at and update\_at timestamps, two fields that will be automatically take values when a document is created and updated.
- Lines 8-14: Fields of this document/class, from line 8 to 11 are the sensor measurements, 12 is the calculated fusion (status) of the machine that this sensor’ readings are related with and lines 13-14 are taks related fields.
- Line 16 declares the relation of this class (sensor readings) with another class (machine), where one machine instance can have a lot of sensor reading instances.
- Line 18 makes the instance to validate the presence of some fields before it is actually saved in database.
- Line 19 creates an index to current fusion field so it can be accessed faster.
- Line 20 attaches a function that is called after the instance is created and before it is saved.
- Lines 23-33 define a function that is called as explained in the above bullet and calculates the current fusion of the machine, as it will be explained later, and calls a function that returns the task that is being attached to this machine at this moment of operation so its information will be stored for later usage.

4.2.1 Data processing

Back-end (more "heavy" calculations) is also developed in Ruby so it would not be necessary to integrate another programming language with Ruby on Rails. Ruby is not indicated as a fast language, however the back-end can be transferred to the language of choice for faster computations with a server-client architecture between "front" (Ruby on Rails) and "back" (language of preference) ends. For the time being, there is not significant delays on the functions of the platform since most heavy operations are completed in MongoDB's aggregate framework [37]. The developed back-end consists of some libraries/modules under “lib/” directory, a directory in Ruby on Rails
applications that makes its contents visible to the main application. The code listing in 7.1 is an example of such a library.

4.2.1.1 Machine Availability

Aggregation functions have been used in order to calculate the machine working periods (e.g. in the last week) from a data-set that consists of 2 measurements per second. Aggregations are completed after three steps, one for matching fields with custom data, one for grouping them and one for sorting them. This is done completely from the MongoDB service and not the R.o.R. application itself. They can be observed in the following code snippet:

```ruby
1. def self.aggregate_working_periods(machine, days)
2. 
3. client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
4. daq_data = client[:daq_data]
5. match = { :$match =>
6.   
7.   :current_fusion => 1,
8.   :machine_id => machine.id,
9.   :created_at => { :gte => Time.now.advance(:days => -days)},
10. }
11. }
12. }
13. group = { :$group =>
14.   
15.   :id => {
16.     :year => { :year => "$created_at" },
17.     :month => { :month => "$created_at" },
18.     :day => { :$dayOfMonth => "$created_at" },
19.     :hour => { :$hour => "$created_at" }
20.   },
21.   :min_minute => { :$min => "$created_at" },
22.   :max_minute => { :$max => "$created_at" }
23. }
24. }
25. sort = { :$sort => {
26.     "_id.year" => -1,
27.     "_id.month" => -1,
28.     "_id.day" => -1,
29.     "_id.hour" => -1,
30. }
31. }
32. aggregation = daq_data.aggregate([match,group,sort])
33. 
34. working_windows = []
```
This aggregation finds the working periods of a machine by scanning latter's sensor data documents:

- Lines 3-4 make the appropriate connection to database and pick the sensored data collection of documents.
- Lines 5-11 define the first step of the aggregation, match phase, in which documents that are related to the given machine, have information fusion output as 1 ("processing") and are created after the given date are picked.
- Lines 13-24 define the second step, grouping, in which the results from first phase are grouped together by the time they were created in minute accuracy.
- Lines 25-31 sort the result from the previous step.
- Finally, line 32 calls the actual aggregate function that will be run on mongodb by passing the three steps as declared on the 3 bullets above. Then, they are pushed into an array and returned as result which is saved in database as well for later usage.

4.2.1.2 Information fusion

The information fusion algorithm embedded in the web application takes as input two parameters and outputs the status of the machine. The first parameter is the output of simple processed sensor data, if one value of currents exceeds a given limit (minimum working current which is specific for each machine and must be measured on system setup) the output equals to 1, or else processing, else the output equals 0, or else non-processing. The second parameter is a value given by a user (operator) and adds one more state, down, for the case the machine is down. The implementation can be seen in the following code snippet:
if operatorInput == 3
    machine.update_attribute(:tmp_final_fusion, 3)
machine.update_attribute(:fusion_last_calculated, Time.now)
return 3
end

if sensorInput == 1 or operatorInput == 1
    machine.update_attribute(:fusion_last_calculated, Time.now)
machine.update_attribute(:tmp_final_fusion, 1)
return 1
else
    machine.update_attribute(:tmp_final_fusion, 0)
machine.update_attribute(:fusion_last_calculated, Time.now)
return 0
end

Code Snippet 4.10 Information fusion implementation.

The table below describes the output of information fusion implementation for two given inputs. X marks any input as accepted.

<table>
<thead>
<tr>
<th>Sensor Input</th>
<th>Operator Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.1 Information fusion.

4.2.1.3 Changing database on the fly

Every time a sensor readings packet is arrived, some calculations are made to analyze if the schedule is being followed or not. This happens the exact moment after the readings are saved. The code snippet below is responsible for rearranging task start and end times.
def self.check_current_task(daq_data)
    machine = daq_data.machine
    status = machine.tmp_final_fusion
    currentTask = machine.tasks.asc(:start_time).where(:modified_start_time.lte => daq_data.created_at).and(:modified_end_time.gte => daq_data.created_at)[0]
    if status == 0
        if currentTask.nil? == true
            return
        elsif currentTask.nil? == false
            return
        end
    elsif status == 1
        if currentTask.nil? == false
            return
        elsif currentTask.nil? == true
            previousTask = machine.tasks.desc(:modified_end_time).where(:modified_end_time.lte => daq_data.created_at)[0]
            nextTask = machine.tasks.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)[0]
            if nextTask.nil? && previousTask != nil
                if LibDaqDatum::check_task_post_conditions(previousTask, daq_data) == false
                    timeWindow = currentTask.modified_end_time - daq_data.created_at
                    currentTask.update_attribute(:modified_end_time, daq_data.created_at)
                    return
                end
            end
        end
    elsif nextTask.nil? == false && previousTask.nil? == false
        if nextTask.modified_start_time - daq_data.created_at <= daq_data.created_at - previousTask.modified_end_time
            if nextTask.modified_start_time > daq_data.created_at
                timeWindow = nextTask.modified_start_time - daq_data.created_at
                nextTasks = Task.all.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)
                nextTasks.each do |task|
                    task.update_attribute(:modified_start_time, task.modified_start_time.advance(:seconds => -timeWindow))
                    task.update_attribute(:modified_end_time, task.modified_end_time.advance(:seconds => -timeWindow))
                end
                return
            end
        end
        if LibDaqDatum::check_task_post_conditions(previousTask, daq_data) == true
            previousTask.update_attribute(:modified_end_time, daq_data.created_at)
        end
        if nextTask.modified_start_time - previousTask.modified_end_time < 60
If calculated information fusion has processing output and there is a task for this machine whose start time is less than now and end time is greater than now, everything is fine.

If calculated information fusion has non-processing output and there is not a task for this machine at the moment, everything is fine.

If calculated information fusion has processing output and there is not a task for this machine at the moment and time period between now and last task (if it exists) is less* than the time period between next task's start time and now, modify last task's end time to now. If previous task's modified end time is greater than next task's start time or one of previous task's post condition tasks, shift all remaining tasks one minute forward. *If not, modify next task's start time to now.
4.2.1.4 Tool usage

In order to measure how much time a tool has been used, a specific field in sensor data documents has been inserted which carries tool information. For the time being the identification of tool in use is done by user input. By using aggregations again sensor
data can be matched by each tool ID, grouped to find the time periods and sorted similar to Listing 4.7.

```ruby
def self.aggregate_tool_usage(tool)
    client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
daq_data = client[:daq_data]
match = {
    :$match => {
        :current_fusion => 1,
        :tool_id => tool.id
    }
}
group = {
    :$group => {
        :_id => {
            :year => {
                :$year => "$created_at"},
            :month => {
                :$month => "$created_at"},
            :day => {
                :$dayOfMonth => "$created_at"},
            :hour => {
                :$hour => "$created_at"}
        },
        :average => {
            :$avg => "$power_consumption"
        }
    }
}
sort = {
    :$sort => {
        "_id.year" => -1,
        "_id.month" => -1,
        "_id.day" => -1,
        "_id.hour" => -1
    }
}
aggregation = daq_data.aggregate([match,group,sort])

minutes = []
aggregation.each do |agg|
    time = Time.new(agg["_id"]["year"].to_i, agg["_id"]["month"].to_i, 
                    agg["_id"]["day"].to_i, agg["_id"]["hour"].to_i)
    minutes.push([ time, agg["average"].round(3) ])  
end
return minutes
end
```

Code Snippet 4.12 Tool usage time estimation

External services

This web platform can be connected with other services over the internet with simple HTTP requests, RESTful APIs or SOAP [47] web services. More specifically, for the needs of the EU founded project, the original schedule was transmitted over the internet from PRODINTEC company using XML and simple HTTP requests. Moreover, tool
information was received over the internet using SOAP protocol from a project partner’s (SANDVIK) commercial software, ADVEON [58]. This has been achieved by developing the corresponding libraries which will provide functions able to handle each service’s requirements. An example is the actual code used to develop the communication with the specific commercial software:

```ruby
1. require 'sandvik_tools_correspondence'
2. require 'open-uri'
3.
4. module SandvikIntegration
5.   def self.get_data
6.     url = 'http://capp4sme.cloudapp.net/DownloadService.svc?singleWsdl'
7.     client = Savon.client( wsd1: url)
8.     response = client.call( :get_files)
9.     xmls = response.body[:get_files_response][:get_files_result][:uploaded_file_detail].select {
10.    |x| x[:application_id]== "SEPC15604" and x[:file_name].match("CAPP") and x[:file_name].match("test").nil?
11.  }
12. xmls.each do |xml|
13.    sandvikCode = xml[:file_name][0..6]
14.    toolName = TOOL_CORRESPONDENCE.key(sandvikCode)
15.    unless toolName.nil?
16.    tool = Tool.find_or_create_by name: toolName
17.    unless tool.nil?
18.    doc = Nokogiri::Slop( open(xml[:result_uri]) )
19.    dcons = doc.search "[text()!='DCON']"
20.    dcons = dcons.select{ |x| x.text == "DCON"}
21.    dconFound = false
22.    unless dcons.last.parent.parent.children.select{ |x| x.name == 'numericvalue'}.nil?
23.        value=dcons.last.parent.parent.children.select{ |x| x.name == 'numericvalue'}.last.text
24.        dconFound = true
25.        tool.update_attribute( :dm, value.to_f)
26.    end
27.   end
28.
29.   doc.html.body.assembly.characteristicvalues.characteristicvalue.each do |ct|
30.      if ct.characteristic.bid.text == "_DESCR"
31.          unless ct.stringvalue.text.index("@").nil?
32.            family1 = ct.stringvalue.text.slice(0..ct.stringvalue.text.index("@")).gsub(/@/,"")
33.            family2 = ct.stringvalue.text.slice(ct.stringvalue.text.index("@")+1).gsub(/@/,"")
34.            tool.update_attribute( :family1, family1 )
35.            tool.update_attribute( :family2, family2 )
36.        end
37.    end
```

else
  family1 = ct.stringvalue.text
  tool.update_attribute( :family1, family1)
  tool.update_attribute( :family2, "NS")
next
else
  field = FIELD_CORRESPONDENCE.key(ct.characteristic.bid.text)
  if ct.numericvalue.attributes.empty?
    value = ct.numericvalue.text.to_f
  elsif defined?(ct.stringvalue) != nil
    value = ct.stringvalue.text
  end
unless field.nil?
  tool.update_attribute( field, value )
end
end
end
end
end
end

Code Snippet 4.13
Library developed to integrate ADVEON (commercial software) to the web platform.

4.2.2 Database schema

Even if no-SQL databases are referred as non-relational, they support relationships between documents like a relational database management system the normal way (document(s) that belong(s) to document(s)) and in case of a document-oriented no-SQL database they also supported "embedded documents", documents embedded inside other documents. The latter provides faster read and update times but has lower limitation in the amount of objects that can be stored and read efficiently [59].
4.2.3 Facility data

The monitoring system is able to store information about each facility that are used a) for processing, b) for informational purposes. Data in the first category include calibrated sensor limits for processing status and data needed for human interaction that has not yet been replaced by sensors. The second category consists of data with more “informal” purposes, descriptions of machine tools, their producer, size, etc. The latter can be inserted by a casual user as described in 3.2.6.
4.2.4 Load balancing

It has been observed that if data from more than 10 sensor nodes are sent to the web platform while it is also serving data for visualization in real-time, RAM usage is being increased until its limits. Also, threads responsible for sessions tend to stay open forever while also increasing the memory usage indefinitely. To counter that, an open source library [48] that kills RoR middleware server (PUMA) workers if they use more than a specific amount of RAM has been used. This measure has no significant effects on losing data or slowing transmission.

4.2.5 Security

Security is a great matter in web platforms because of their exposure to the whole internet world; every day new software vulnerabilities come to light. Hence, some actions have been taken to reduce the effects of most cyber-attacks. In the side of the web platform security has been divided into two layers, the operating system layer and the web application layer. It is important to note that the exact same architecture proposed in this thesis can be used in a Local Area Network with no access to the Internet which would definitely secure the majority of potential vulnerabilities of the platform. The application itself does not need or hold sensitive data and no useful information can be extracted from it. Database back-ups take place often through MongoDB's "mongodump" built-in utility which creates a binary export of the contents of the database [55].

4.2.5.1 Operating system layer

Initially, a firewall has been set up, by using Linux internal rules for network connections [49], which allows traffic on ports 80 (for traffic relative to the web application) and 50000, a custom SSH port. To avoid DDOS attacks (an attack method that floods the server's resources and makes it unavailable by sending large amount of requests on small amount of time [54]), these rules are edited correctly to allow connections from specific IP addresses (shop-floor network's and software developer/maintainer's IP addresses). Moreover, SSH server has been configured accordingly to block root login, permit a specific number of login tries and allow connections from specific IP addresses as well. In addition, Nginx with secure
configuration encrypts data transferred between the web platform and clients/gateways to avoid most of wire-tapping attacks using SSL/TLS protocol [50]. To avoid POODLE [56] and other attacks which target older encryption protocols, SSL [57] versions 3.0 and below are disabled. Finally, the server is frequently updated to patch lately discovered bugs.

4.2.5.2 Web application layer

Apart from the welcome page which servers the purpose of aesthetics, every other functionality of the application requires user authentication. User registration is not open to the public since creating new account is one person's responsibility. Communication between gateways and the web application is also done using a registered user's credentials. This is not a web platform targeting the masses, but a specific group of people who are known (maintainers-operators-shopfloor managers), so attacks that inject client-side scripts [51] (from registered users) are excluded. CSRF attacks [52], attacks that target the trust that the application has to a user browser, are prevented by using RoR's built-in CSRF protection and by using POST and GET HTTP requests appropriately.

4.2.6 Graphical user interface

In order to visualize effectively the results of monitoring and processed data a GUI-subsection has been created with a menu on its left and main page contents on center which changes per user request and mainly contains user input and graphs provided by Google [60]. The exact measurements of sensors are displayed in dynamically updated line and column charts (Fig. 4.5), tasks/working periods in time-line charts, power/energy consumption in gauge charts and overall utilization in pie chart and relationships between tasks (post/pre conditions) in org charts. The entire GUI is responsive, meaning that the size of letters/boxes/graphs etc. will re-size according to user's screen size. Machine status can be seen from any page of this subsection in it's menu (all figures of this paragraphs, the menu on top-left), where green colour means non-processing, orange means processing and red means down. Also, when a webcam is connected to the gateway, images of the machine tool are being shown (Fig. 4.6).
Figure 4.5 Sensor measurements (dynamically updated).
Figure 4.6 Task related GUI page, on top is the original schedule, then the monitored. In the second screenshot example task relationships are shown.

User input has been developed using one main theme and it is responsive to screen size as well. Data provided by users are (a) data related to specifications of each resource, (b) sensor calibration and (c) operator input. The first category of data must be provided on first system start-up per factory by one user and mainly regard machine specifications and tool information, but they can be updated at any time (Figure 4.7). They are mostly data used for integration with remote web service and data for record purposes. The second category is needed for the sensor fusion part of information fusion algorithm, the lowest limit of a processing machine which can easily be calculated by setting the machine tool on production and recording the lowest value of the machine when the tool is rotating and when the table is moving in one of the axes (Fig. 4.8). The third category includes real-time decisions from a person in the working environment, functions for invoking external services' communication, manual task editing, enabling/disabling data storage and processing per machine and most importantly, informing the system for a machine failure ("DOWN" state, Fig. 3.9)
Figure 4.7 User input for machine specifications.

Figure 4.8 User input for sensor calibration.
5 Resource aware adaptive scheduling

5.1 Scheduling engine search algorithm[18]

The search algorithm generates feasible scheduling alternatives, where each alternative is considered as a set of resource to tasks assignments that can carry out the workload. The number of alternatives that are generated, is calculated at each decision point of the solution space. The search algorithm uses three adjustable control parameters, namely the maximum number of alternatives (MNA), the decision horizon (DH), and the sampling rate (SR).

The MNA prescribes the maximum number of alternatives that the algorithm can consider. The DH is the time interval that begins at each decision point and dictates the possible assignments that can be performed within the interval. Finally, the SR is the number of paths created after MNA and DH are exhausted. Specifically, MNA controls the breadth of the search and the DH the depth. SR directs the search path towards
achieving higher-quality solutions. The intelligent search algorithm is performed in eight main steps, as shown below:

- **Step 1** Starting at the root, create alternatives by randomly creating assignments for all layers in DH, until MNA is reached.
- **Step 2** For each alternative (Step 1), create SR random alternatives (samples) until all nodes in the branch are searched.
- **Step 3** Calculate the criteria scores for all the samples belonging to the same alternative of Step 1.
- **Step 4** Calculate the alternatives scores as the average of scores achieved by its samples.
- **Step 5** Calculate the utility values of each one alternative/branch.
- **Step 6** Select the alternative with the highest utility value.
- **Step 7** Store the assignments of the selected alternatives
- **Step 8** Repeat steps 1-7 until an assignment has been done for all the nodes of the selected branches

The choice of one alternative over another is performed by evaluating relevant criteria, or attributes in a decision matrix. The algorithm can provide the desired trade-off between solution quality and required execution time, considering both feasibility and availability requirements.

The consecutive steps followed during the decision making are the following. The first step is the determination of the alternatives. Decision implies choice, and choice implies alternatives. An alternative is defined as a possible set of assignments of resources to tasks for the alternatives to be qualified for the consideration. Both feasibility and availability requirements have to be satisfied, therefore, the input from the monitoring system concerning machine tool availability and status is necessary. The second step is the determination of the attributes. Attributes are criteria used to evaluate the alternatives. The correct determination of the attributes is crucial to the effectiveness of the decision method, since they reflect the design and planning objectives of the manufacturing system. The multi-criteria decision making algorithm takes into consideration conflicting criteria of low-time (1), cost (2), and quality (3).

The final step is the determination of the consequences. Consequences, which are needed to evaluate the selected alternatives, are the values of the attributes at the time the decisions are performed. In this work, the set of the generated alternatives is
assessed using a set of performance indicators including production flowtime and resource utilization [18].

5.2 Interface between monitoring module and schedule engine

5.2.1 Architecture

LMS SE uses XML files as input and output. A small controller has been developed to accept HTTP connections and trigger the main algorithm on correct input (paragraph 4.2.2). On LMS MM side, a small library has been developed to form a correct XML temporary file, send it through an HTTP GET request and wait for the results which are also in XML format.

1. Machine tool is detected DOWN or returns from DOWN status

2. Schedule is shifted forward once the originally defined task time is exceed based on the real data

3. Once the defined threshold of the makespan is influenced, scheduling is performed and the monitoring system inform the scheduling system on the status
   - Done Tasks
   - Pending Tasks
   - Ongoing Tasks
   - Machine tools Availability
5.2.2 Algorithm

The original schedule is initially transmitted by LMS SE to LMS MM. LMS MM will periodically, every 1 minute, check for rescheduling factors as explained in 4.2.3. If that leads to a rescheduling, LMS MM collects machines tools' statuses and tasks that have not yet been started. In case of on-going tasks, the remaining time of the task is also calculated. Then, an XML file is created as seen in Appendix A code snippets and sent to LMS SE. The latter takes this XML as input and outputs a new schedule which takes into account the not-started and on-going tasks and replies back to LMS MM accordingly. The structure of the response can be seen in the last XML of Appendix A.

5.2.3 Rescheduling factors

Two questions arise when developing an interface between a scheduling and a monitoring service, when a rescheduling should be triggered and what information should be transmitted. The proposed answer is displayed in Figure 4.2.
As explained in paragraph 3.2.1.3, code listing 3.9, the monitoring system has the ability to identify which task has been completed/is on-going relatively to a given schedule, while it also calculates the correct start and end times of completed tasks. This creates two problems: what happens if a task has been processing longer that it should be and in fact overlaps next task's start time and what happens if the same
situation emerges between a task and one of this task's post-condition tasks. In case of those conflicts, all remaining tasks (tasks whose start time is greater than now) are shifted 1 minute forward. A more complicated approach is to shift only remaining tasks of the machine that is processing the current task, the post conditions of the current task, the post conditions of the post conditions of the current task, but this will be covered in chapter “Future Work”. This will keep going until the current task is over. In case the total delay exceeds a user-defined amount, rescheduling is triggered as explained in paragraph 4.2.2.

Through user input, the monitoring system has also the ability to store and process information about machine failures. In such occasions rescheduling is also triggered but the new schedule is adapted not to include the so far broken resources.

```ruby
1. require 'lib_daq_datum'
2. require 'impact_machine_correspondance'
3. require 'builder'
4.
5. module LibImpactIntegration
6.   def self.get_project()
7.     doc = Nokogiri::XML(open('../impact/project_script_generated.xml'))
8.     doc = Nokogiri::Slop(doc.to_s)
9.     Task.all.delete
10.    TaskDefinition.all.delete
11.    Job.all.delete
12.    job = Job.create( :name => 1)
13.    doc.project.orders.order.tasks.task.each do |task|
14.      tf = job.task_definitions.create(:impact_id => task.attributes['id'].value, :name => task.attributes['id'].value)
15.      if defined? task.preconditions != nil
16.        preCond = Array.new
17.        task.preconditions.children.each do |pc_child|
18.          if pc_child.name == "precondition"
19.            preCond.push(pc_child.text)
20.        end
21.        end
22.      tf.update_attributes({:pre_conditions => preCond})
23.      else
24.        tf.update_attributes({:pre_conditions => Array.new})
25.      end
26.      tf.update_attributes({:post_conditions => Array.new})
27.    end
28.    job.task_definitions.all.each do |tf|
29.      tf.pre_conditions.each do |tf_pc|
30.        pre_condition_task = TaskDefinition.where( :name => tf_pc)[0]
31.        pre_condition_task.update_attributes( :post_conditions => pre_condition_task.post_conditions.append(tf.name)) #impact_id
```
def self.get_schedule() #remove Time.now().year etc in startTime, endTime 7-11 lines below
  doc = Nokogiri::XML(open('../impact/out_script_generated.xml'))
  doc = Nokogiri::Slop(doc.to_s)
  Task.all.delete
  tasks = doc.result.alternative.elements
  tasks.each{|task| startTime = Time.at(task.attributes['startTime'].value.to_i/1000)
  startTime = startTime.change({:year => Time.now().year, :month => Time.now().month, :day => Time.now().day })
  endTime = Time.at(task.attributes['endTime'].value.to_i/1000)
  endTime = endTime.change({:year => Time.now().year, :month => Time.now().month, :day => Time.now().day })
  machineImpactId = task.attributes['resource'].value #string e.g. "1"
  machineLocalId = MACHINE_CORR[machineImpactId]
  taskName = task.attributes['task'].value
  machine = Machine.find(machineLocalId)
  dbTask = Task.where( :impact_id => taskName)
  if dbTask.empty?
    machine.tasks.create( :start_time => startTime, :end_time => endTime, :name => taskName, :impact_id => taskName)
  else
    dbTask.update_attributes({:start_time => startTime, :end_time => endTime, :name => taskName})
  end
  end
  TaskDefinition.all.each{|tf|
    task = Task.where( :impact_id => tf.impact_id)[0]
    unless task.nil?
      tf.post_conditions.each{|pc|
        task.post_conditions.create( :post_condition_task_id => pc)
      end
    end
  end
  end
  end
  def self.send_data_for_scheduling() #to be sent to impact
    completedTasks, ongoingTasks, pendingTasks = LibImpactIntegration.get_tasks_status()
    xml_markup = Builder::XmlMarkup.new(:indent => 2)
    xml_markup.instruct! :xml, :version=>"1.0", :encoding=>"UTF-8"
    xml_markup.tag! 'schedule-input' do |si|
      si.resources {
        Machine.all.each{|machine|
          if machine.tmp_final_fusion == 0
            xml_markup.resource( :id=>MACHINE_CORR_INVERT[machine.id.to_s], "completion-time" => 0, "status" => "IDLE")
          end
        end
      end
    end
  end
```ruby
elsif machine.tmp_final_fusion == 1
  currentTask = machine.tasks.where(modified_start_time.lte => Time.now).and(modified_end_time.gte => Time.now)
  unless currentTask.empty?
    remainingTaskTime = ((currentTask[0].modified_end_time - Time.now) * 1000).round #msecs
    xml_markup.resource(:id => MACHINE_CORR_INVERT[machine.id.to_s], "completion-time" => remainingTaskTime, "status" => "BUSY")
  else
    xml_markup.resource(:id => MACHINE_CORR_INVERT[machine.id.to_s], "completion-time" => 0, "status" => "IDLE")
  end
end
end

si.tag!("pending-tasks") {
  pendingTasks.each do |pendingTask|
    xml_markup.pending_task(:id => pendingTask.impact_id)
  end
}

xmlData = xml_markup.target!.gsub(/<to_s \/>/, "")
print xmlData

def self.get_tasks_status()
  completedTasks = Task.where(modified_end_time.lt => Time.now)
  ongoingTasks = Task.where(modified_start_time.lt => Time.now).and(modified_end_time.gt => Time.now)
  pendingTasks = Task.where(modified_start_time.gt => Time.now)
  return completedTasks, ongoingTasks, pendingTasks
end

#used in tasks/check_schedule.rake
def self.check_down_machines
  downMachines = Machine.where(tmp_final_fusion => 3)
  return downMachines
end
```

Code Snippet 5.1 Library developed for monitoring system and scheduling engine connection.

6 Monitoring System Output and Integrations

As mentioned earlier, the monitoring system is capable of connecting with remote software tools through web services. For the needs of the EU founded project, several functionalities and data of this system have been exposed using SOAP web services and simple HTTP requests; the calculation of machine status, available windows, tool usage, power consumption, utilization, actual machining time and remaining
**time between failure** were some of this system’s outputs that were forwarded to project partners.

Moreover, the monitoring system was successfully integrated with a commercial software tool that provides information about machining tools and a scheduling tool that provided (static, resource un-aware) schedules. Data is transmitted using XML format on all occasions.

```ruby
1. require 'sandvik_tools_correspondence'
2. require 'open-uri'
3. module SandvikIntegration
4.
5.   def self.get_data
6.     url='http://capp4sme.cloudapp.net/DownloadService.svc?singleWsdl'
7.     client= Savon.client( wsdl: url)
8.     response= client.call( :get_files)
9.     xml= response.body[:get_files_response][:get_files_result][:uploaded_file_detail].select {  
10.       |x| x[:application_id] == "SECPC15604" and x[:file_name].match("CAPP") and x[:file_name].match("test").nil?
11.   }
12.
13.   xmls.each do |xml|
14.     sandvikCode= xml[:file_name][0..6]
15.     toolName= TOOL_CORRESPONDENCE.key(sandvikCode)
16.     unless toolName.nil?
17.       tool= Tool.find_or_create_by name: toolName
18.     unless tool.nil?
19.       doc = Nokogiri::Slop( open(xml[:result_uri]) )
20.     end
21.     dcons= doc.search "[#text()='#DQN']"
22.     dcons= dcons.select { |x| x.text == "DQN"}
23.     dconFound= false
24.     unless dcons.last.parent.parent.children.select{ |x| x.name == 'numericvalue'}.nil?
25.       unless dcons.last.parent.parent.children.select{ |x| x.name == 'numericvalue'}.last.text.empty?
26.         value=dcons.last.parent.parent.children.select{ |x| x.name == 'numericvalue'}.last.text
27.         dconFound= true
28.       tool.update_attribute( :dm, value.to_f)
29.     end
30.     end
31.
32.   doc.html.body.assembly.characteristicvalues.characteristicvalue.each do |ct|
33.     if ct.characteristic.bid.text == "_DESCR"
34.       unless ct.stringvalue.text.index("@").nil?
35.         family1= ct.stringvalue.text.slice(0..ct.stringvalue.text.index("@")).gsub(/@/,"") #family1
36.       tool.update_attribute( :family1, family1 )
37.     end
38.     tool.update_attribute( :family2, family2 )
```
next
else
  family1= ct.stringvalue.text
  tool.update_attribute( :family1, family1)
  tool.update_attribute( :family2, "NS")
next
end
else
  field= FIELD_CORRESPONDENCE.key(ct.characteristic.bid.text
  if ct.numericvalue.attributes.empty?
    value= ct.numericvalue.text.to_f
  elsif defined?(ct.stringvalue) != nil
    value= ct.stringvalue.text
  end
  unless field.nil?
    tool.update_attribute( field, value )
  end
end
end
end
end
end
end
end

Code Snippet 6.1 Library that integrates a commercial software tool with monitoring system.
7 Case study

Tests run

The following abilities of the monitoring system were tested:

1. Fix the start time and end time of tasks depending on sensor data.
2. Shift related tasks when there is a conflict between a shifted task and its post conditions and same machine’s next task.
3. Request a reschedule when shifting makes the due time of the schedule greater than a defined amount of time.
4. Request a reschedule when a machine changes to DOWN status or returns from DOWN status, if it has a current task it must be restarted on an available machine.
5. Make sure that processing tasks (on working machines) are not affected by reschedules, and in fact their completion time is taken into account by the scheduling engine.
Input – 3 Machine tools

<table>
<thead>
<tr>
<th>Fresadora</th>
<th>Mazak</th>
<th>XYZ</th>
</tr>
</thead>
</table>

Table 7.1 Machines

Input – 5 Tasks

<table>
<thead>
<tr>
<th>Name</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Duration (seconds)</strong></td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>420</td>
<td>300</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>None</td>
<td>t1</td>
<td>t2</td>
<td>None</td>
<td>t4</td>
</tr>
</tbody>
</table>

Table 7.2 Tasks

Master schedule - As received by the scheduling tool

<table>
<thead>
<tr>
<th>t1</th>
<th>Mazak</th>
<th>13:27:50</th>
<th>13:32:50</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>Mazak</td>
<td>13:33:50</td>
<td>13:38:50</td>
</tr>
<tr>
<td>t3</td>
<td>Fresadora</td>
<td>13:39:50</td>
<td>13:44:50</td>
</tr>
<tr>
<td>t4</td>
<td>XYZ SMX SLV</td>
<td>13:28:20</td>
<td>13:35:20</td>
</tr>
<tr>
<td>t5</td>
<td>XYZ SMX SLV</td>
<td>13:36:20</td>
<td>13:44:20</td>
</tr>
<tr>
<td>t1</td>
<td>Mazak</td>
<td>13:27:50</td>
<td>13:32:50</td>
</tr>
</tbody>
</table>

Table 7.3 Initial Schedule
Figure 7.1 Initial schedule
Figure 7.2: Fixing the end time of t1 as the machine keeps processing after t1’s defined end time.

Figure 7.3: A conflict was detected, t1 was monitored to be processed less than 30 seconds before t2’s start time, hence all remaining tasks (t2, t3, t5) are shifted forward 30 seconds.
Figure 7.4 Mazak changed to DOWN status while processing t2, this is the result of the rescheduling.

Figure 7.5 t2 kept processing (total time 10 mins 17 seconds), hence t3 and t5 were shifted forward accordingly. The due time exceeds the user defined maximum schedule delay, so a reschedule is requested.
Figure 7.6 This is the resulting schedule, this time Fresadora is assigned a task.

Figure 7.7 Fresadora’s sensor node kept sending non-processing status, t3’s start time is shifted forward until this machine’s status is changed to processing.
Figure 7.8 This is the final monitored schedule, taking into account that XYZ’s sensor node sent non-processing values before the assigned task’s (t5) defined end time.
<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Start</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>13:27:50</td>
<td>10:33:44</td>
<td>Mazak</td>
</tr>
<tr>
<td>t2</td>
<td>13:37:33</td>
<td>13:48:10</td>
<td>Mazak, XYZ</td>
</tr>
<tr>
<td>t3</td>
<td>13:49:09</td>
<td>13:</td>
<td>Fresadora, XYZ, Fresadora</td>
</tr>
<tr>
<td>t4</td>
<td>13:29:20</td>
<td>10:36:20</td>
<td>XYZ</td>
</tr>
<tr>
<td>T5</td>
<td>13:48:34</td>
<td>13:54:29</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

Table 7.4 The monitored job

7.1.1 Comments

- Results of test 1 can be seen in figure 7.2, 7.8.
- Results of test 2 can be seen in figure 7.3
- Results of test 3 can be seen in figure 7.6
- Results of test 4 can be seen in 7.4
- Results of test 5 can be seen in most of the cases, where processing tasks are not reassigned if their machine’s status is processing, while remaining tasks are reassigned.
8 Conclusion and future work

8.1 Conclusion

This thesis presented the implementation of a platform capable of sensing, analyzing and controlling machine tools and manufacturing systems. The developed platform is a multi-layer system which consists of sensing devices, a wireless sensor network, and a web-based platform. The platform is enhanced with data analysis algorithms in order to identify the status of the machine tools as well as important performance metrics such as:

- Machine tool energy consumption.
- Machine tool utilization.
- Machine tool availability.
- Machine tool current status.
- Machine tool remaining operating time between failure.
- Machine tool actual machining time.
- Task durations assigned to the machine tools.

Following the collected and analyzed data, the platform is also capable of performing resource-aware scheduling and condition-based maintenance of machine tools. Among the main advantages of the proposed platform and its application are:

- Increased awareness on both machine and shop-floor level condition.
- Effective and accurate maintenance of machine tools.
- Increased productivity through condition-based maintenance and adaptive scheduling.
- Increased interoperability and communication among the different systems in the company.
- Increased automation which will support companies to shift towards Industry 4.0 environment.
8.2 Future work

8.2.1 Machine status

Machine status “DOWN” responds to a machine failure, e.g. when a tool occurs. For the time being, the monitoring system is informed about this change only through human (operator) interaction. To avoid that, more complicated algorithms can be used to process sensor data and decide if the machine is down. This can be achieved by using supervised classifiers trained for each machine model; something that requires preparation time when adding new machines in the system.

8.2.2 Tool usage over time

Moreover, the platform has the ability to store which tool was being used/is being used on which machine and its overall "usage time", meaning the time that the machine was processing and had this specific tool mounted. In the current version this is being achieved by user interaction as well through a corresponding GUI. By using RFID technology, one can attach small circuits capable of transmitting Radio Frequency signals on each tool. Then, an RFID receiver can be connected in the micro-controller board. Each RF transmitter (embedded somehow in the tool, e.g. spindle tool) will have a unique ID that will be transmitted between equal amount of time. Each RF receiver will "lock" and receive data from a specified transmitter. To pair RF transmitters and receivers, one can install one button on each side that will be pressed when the tool is being replaced and declare some custom functions for this purpose.

8.2.3 Advanced rescheduling on the fly: Recursive function

As described in chapter 2, when one task is exceeding its predefined due time some and reaches one of its post conditions’ predefined start times or even the start time of the next task assigned to this machine, all remaining tasks are shifted forward. To avoid shifting unrelated tasks (e.g. tasks that are not in the same machine and have not this task as precondition) a recursive function could be developed that starts with the current delayed task, checks for post conditions and same machine’s next task, their post-conditions and same machine’s tasks e.t.c.
8.2.4 Factory metrics

Since the base of monitoring the production has been set, it can be enhanced to translate additional sensor and human information by adding more sophisticated back end and extra sensor types. Already knowing the power consumption, tool failures, task overall due time, one can relate the efficiency of an order to a specific shop-floor. Hence, specific metrics can be “discovered” for an even higher level monitoring service.

8.2.5 Software topology

This monitoring platform is focused on hosting services for different factories. Incoming data will grow on every new machine that is added in the system. In order to provide stable services, the following steps can be done:

- Split database to a different server and add more clusters when one server’s hardware is not enough. MongoDB provides those functionalities by default.
- Use Nginx as reverse proxy which will forward sensor (only) data to cloned servers running a micro-service dedicated for this cause. Those clones (offered as API) will only save incoming measurements, leaving processing and visualization for the main application.
- In the same way, either use those servers to server HTTP data to user clients, or split them on different ones.
- The main application will be responsible for data processing, hosted on a different server as well.
Appendix A

9.1 Code Listings

```ruby
module LibDaqDatum
  def self.check_task_post_conditions(task, daq_data)
    task.post_conditions.each do |post_cond|
      postConditionTask = Task.where(:impact_id => post_cond.post_condition_task_id)[0]
      machine = task.machine
      if task.modified_end_time >= postConditionTask.modified_start_time.advance(:minutes => -1)
        postConditionTaskDuration = postConditionTask.modified_end_time - postConditionTask.modified_start_time
        postConditionTask.update_attribute(:modified_start_time, postConditionTask.modified_start_time.advance(:minutes => +1))
        postConditionTask.update_attribute(:modified_end_time, postConditionTask.modified_end_time.advance(:minutes => +1))
        nextTasks = Task.all.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)
        nextTasks.each do |task|
          task.update_attribute(:modified_start_time, task.modified_start_time.advance(:minutes => 1))
          task.update_attribute(:modified_end_time, task.modified_end_time.advance(:minutes => 1))
        end
        return false
      end
    end
    return true
  end
end
```
if daq_data.created_at - currentTask.modified_start_time > currentTask.modified_end_time - daq_data.created_at

timeWindow = currentTask.modified_end_time - daq_data.created_at
currentTask.update_attribute( :modified_end_time, daq_data.created_at)
return

elseif status == 1
  if currentTask.nil? == false
    return
  elsif currentTask.nil? == true
    previousTask = machine.tasks.desc(:modified_end_time).where(:modified_end_time.lte => daq_data.created_at)[0]
    nextTask = machine.tasks.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)[0]
    if nextTask.nil? && previousTask != nil
      if LibDaqDatum::check_task_post_conditions(previousTask, daq_data) == false
        timeWindow = daq_data.created_at - previousTask.modified_end_time
        previousTask.update_attribute(:modified_end_time, daq_data.created_at)
        end
      return
    elsif nextTask.nil? == false && previousTask.nil? == false
      if nextTask.modified_start_time - daq_data.created_at <= daq_data.created_at - previousTask.modified_end_time
        timeWindow = nextTask.modified_start_time - daq_data.created_at
        nextTasks = Task.all.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)
        nextTasks.each do |task|
          task.update_attribute(:modified_start_time, task.modified_start_time.advance(:seconds => -timeWindow))
          task.update_attribute(:modified_end_time, task.modified_end_time.advance(:seconds => -timeWindow))
        end
      end
      elsif nextTask.modified_start_time - daq_data.created_at > daq_data.created_at - previousTask.modified_end_time
        if LibDaqDatum::check_task_post_conditions(previousTask, daq_data) == true
          previousTask.update_attribute(:modified_end_time, daq_data.created_at)
          if nextTask.modified_start_time - previousTask.modified_end_time < 60
            nextTasks = Task.all.asc(:modified_start_time).where(:modified_start_time.gte => daq_data.created_at)
            nextTasks.each do |task|
              task.update_attribute(:modified_start_time, task.modified_start_time.advance(:seconds => 60))
              task.update_attribute(:modified_end_time, task.modified_end_time.advance(:seconds => 60))
            end
          end
        end
      return
    else
      return
    end
  end
end
module LibMachines

def self.aggregate_calc_energy_consumption(machine)
  working_periods = LibMachines::aggregate_calc_monitored_processing(machine, 52)
  minutes_worked = 0.0
  processing_consumption = 0.0
  working_periods.each do |wp|
    minutes_worked += wp[1] - wp[0]
    processing_consumption += wp[2] / minutes_worked
  end
  non_working_periods = LibMachines::aggregate_calc_last_week_monitored_non_processing(machine, 52)
  non_processing_consumption = 0.0
  minutes_idle = 0.0
  non_working_periods.each do |wp|
    minutes_idle += wp[1] - wp[0]
    non_processing_consumption += wp[2] / minutes_idle
  end
  working_daq_datum = machine.daq_datum.where(:current_fusion => 1).count
  non_working_daq_datum = machine.daq_datum.where(:current_fusion => 0).count
  overall = working_daq_datum + non_working_daq_datum
  if overall != 0
    utilization = working_daq_datum*100 / overall
    availability = non_working_daq_datum*100 / overall
  else
    utilization = 0
    availability = 100
  end
  return [ processing_consumption, non_processing_consumption, availability, utilization ]
end

def self.aggregate_calc_last_week_monitored_non_processing(machine, weeks)
  client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
  daq_data = client[:daq_data]
  match = {}
  { :current_fusion => 0,
    :created_at => { :gte => Time.now.advance( :days => -1) },
    :machine_id => machine.id
  }
  group = {}
  { :_id => { :year => { :year => "$created_at" },
    :month => { :month => "$created_at" },
    :day => { :dayOfMonth => "$created_at" },
    :hour => { :hour => "$created_at" }#,
    #:minute => { :minute => "$created_at"}
    #:second => { :second => "$created_at" },
  }
...}
154. }
155. sort = { :$sort => {
156.   "_id.year" => -1,
157.   "_id.month" => -1,
158.   "_id.day" => -1,
159.   "_id.hour" => -1,
160.   
161. }
162. }
163. aggregation = daq_data.aggregate([match,group,sort])
164. non_working_windows = []
165. aggregation.each do |agg|
166.   non_working_windows.push([ agg["min_minute"], agg["max_minute"], agg["power_consumption"]])
167. end
168. return non_working_windows
169. end
170. end
171. def self.aggregate_calc_monitored_processing(machine, weeks)
172.   client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
173.   daq_data = client[:daq_data]
174.   match = { :$match =>
175.     {
176.       :current_fusion => 1,
177.       :created_at => { :$gte => Time.now.advance( :days => -1)},
178.     }
179.     :machine_id => machine.id
180.   }
181.   group = { :$group => {
182.     :id => {
183.       :year => { :$year => "$created_at" },
184.       :month => { :$month => "$created_at" },
185.       :day => { :$dayOfMonth => "$created_at" },
186.       :hour => { :$hour => "$created_at" },
187.       
188.     },
189.     :min_minute => { :$min => "$created_at" },
190.     :max_minute => { :$max => "$created_at" },
191.     :power_consumption => { :$sum => "$power_consumption" }
192.   }
193.   }
194.   sort = { :$sort => {
195.     
196.   }
197.   }
198.   aggregation = daq_data.aggregate([match,group,sort])
199.   non_working_windows = []
200.   aggregation.each do |agg|
201.     non_working_windows.push([ agg["min_minute"], agg["max_minute"], agg["power_consumption"]])
202.   end
203.   return non_working_windows
204. end
205. end
206. def self.aggregate_calc_monitored_processing(machine, weeks)
#"_id.minute" => -1

aggregation = daq_data.aggregate([match, group, sort])

working_windows = []
aggregation.each do |agg|
  working_windows.push([ agg["min_minute"], agg["max_minute"], agg["power_consumption"]])
end
return working_windows

def self.aggregate_working_periods(machine, days)
  client = Mongo::Client.new(["127.0.0.1:27017"], :database => 'machine_availability_lms_development')
  daq_data = client[:daq_data]
  match = {
    :current_fusion => 1,
    :machine_id => machine.id,
    :created_at => {
      :gte => Time.now.advance(:days => -days)
    }
  }
  group = {
    :year => {
      :year => "created_at"
    },
    :month => {
      :month => "created_at"
    },
    :day => {
      :dayOfMonth => "created_at"
    },
    :hour => {
      :hour => "created_at"
    },
    #:minute => {
      :minute => "created_at"
    },
    #:second => {
      :second => "created_at"
    }
  }
  sort = {
    "_id.year" => -1,
    "_id.month" => -1,
    "_id.day" => -1,
    "_id.hour" => -1,
    #"_id.minute" => -1
  }
  aggregation = daq_data.aggregate([match, group, sort])
  working_windows = []
  aggregation.each do |agg|
    working_windows.push([ agg["min_minute"], agg["max_minute"]])
  end
  return working_windows
end
```ruby
263. def self.aggregate_avg_overall_power_consumption(machine)
264.   client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
265.   daq_data = client[:daq_data]
266.   group = { :$group => {
267.     :id => {
268.       :year => { :$year => "$created_at" },
269.       :month => { :$month => "$created_at" },
270.       :day => { :$dayOfMonth => "$created_at" },
271.       :hour => { :$hour => "$created_at" },
272.       #:second => { :$second => "$created_at" },
273.     },
274.     :average => { :$avg => "$power_consumption" }
275.   } } ]
276.   sort = { :$sort => {
277.     "_id.year" => -1,
278.     "_id.month" => -1,
279.     "_id.day" => -1
280.   } }
281.   aggregation = daq_data.aggregate([group,sort])
282.   minutes = []
283.   aggregation.each do |agg|
284.     time = Time.new(agg["_id"][:year].to_i, agg["_id"][:month].to_i,
285.       agg["_id"][:day].to_i)
286.     minutes.push([ time, agg["average"].round(3) ])
287.   end
288.   return minutes
289. end

292. end

294. def self.aggregate_avg_working_power_consumption(machine)
295.   client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'machine_availability_lms_development')
296.   daq_data = client[:daq_data]
297.   match = { :$match => {
298.     :current_fusion => 1,
299.     :machine_id => machine.id
300.   } }
301.   group = { :$group => {
302.     :id => {
303.       :year => { :$year => "$created_at" },
304.       :month => { :$month => "$created_at" },
305.       :day => { :$dayOfMonth => "$created_at" },
306.       #:second => { :$second => "$created_at" },
307.     },
308.     :average => { :$avg => "$power_consumption" }
309.   } } ]
310.   sort = { :$sort => {
311.     "_id.year" => -1,
312.     "_id.month" => -1,
313.     "_id.day" => -1
314.   } }
315.   aggregation = daq_data.aggregate([group,sort])
316.   minutes = []
317.   aggregation.each do |agg|
318.     time = Time.new(agg["_id"][:year].to_i, agg["_id"][:month].to_i,
319.       agg["_id"][:day].to_i)
320.     minutes.push([ time, agg["average"].round(3) ])
321.   end
322.   return minutes
323. end
```
320.        "_id.year" => -1,
321.        "_id.month" => -1,
322.        "_id.day" => -1,
323.        "_id.hour" => -1
324.    }
325.  }
326.  aggregation = daq_data.aggregate([match,group,sort])
327.  minutes = []
328.  aggregation.each do |agg|
329.    time = Time.new(agg["_id"]["year"].to_i, agg["_id"]["month"].to_i,
330.                     agg["_id"]["day"].to_i,agg["_id"]["hour"].to_i)
331.    minutes.push([ time, agg["average"].round(3)])
332.  end
333.  return minutes
334. end
335.
336.  def self.calculate_available_windows( machine, duration_in_weeks ) #with s
337.    if duration_in_weeks.equal?( nil)
338.      duration_in_weeks=2
339.    end
340.    end
341.    availabilityArray= Array.new
342.    machine.tasks.asc( :modified_start_time ).each_with_index do |task, index|
343.      if index < machine.tasks.count - 1
344.        unless task.modified_end_time.to_s == machine.tasks.asc( :modified_start_time )[ index + 1].modified_start_time.to_s
345.          availabilityArray.push( [ task.name, machine.tasks.asc( :modified_start_time )[ index + 1].name, task.modified_end_time , machine.tasks.asc( :modified_start_time )[ index + 1].modified_start_time ] )
346.        end
347.      end
348.    end
349.    return availabilityArray
350.  end
351.
352.  def self.machine_setup(machine)
353.    machine.update_attribute( :tmp_operator_input, 2)
354.  self.calculate_information_fusion(machine)
355.  end
356.
357.  def self.machine_operator_input_processing(machine)
358.    machine.update_attribute( :tmp_operator_input, 1)
359.  self.calculate_information_fusion(machine)
360.  end
361.
362.  def self.machine_down(machine, reason_from_user )
363.    unless self.calculate_information_fusion(machine).equal?(3)
364.      current_task= machine.tasks.where( :modified_start_time.lte => Time.now).and( :modified_end_time.gte => Time.now ).desc( :modified_end_time)[0]
365.      unless current_task.nil?
366.         current_task.down_events.create!
367.     end
368.    machine.update_attribute( :tmp_operator_input, 3)
369.  self.calculate_information_fusion(machine)
370.  end
371. end
372.
373.  def self.machine_up(machine)
374.    if self.calculate_information_fusion(machine) == 3
if machine.tmp_operator_task.nil?
  machine.update_attribute(:tmp_operator_input, 0)
else
  machine.update_attribute(:tmp_operator_input, 1)
end

self.calculate_information_fusion(machine)

elsif self.calculate_information_fusion(machine) == 2
  if machine.tmp_operator_task.nil?
    machine.update_attribute(:tmp_operator_input, 0)
  else
    machine.update_attribute(:tmp_operator_input, 1)
  end
  self.calculate_information_fusion(machine)
else
  if machine.tmp_operator_task.nil?
    machine.update_attribute(:tmp_operator_input, 0)
  else
    machine.update_attribute(:tmp_operator_input, 1)
  end
  self.calculate_information_fusion(machine)
end

end

end

def self.calculate_information_fusion(machine)
  sensorInput= self.latest_sensor_input(machine)
  operatorInput= self.latest_operator_input(machine)
  currentTask= machine.tasks.where(:modified_start_time.lte => Time.now).and(:modified_end_time.gte => Time.now)
  unless currentTask.empty? and machine.tmp_operator_task.nil?
    tmpOperatorTask= machine.tasks.where(:id => machine.tmp_operator_task)
    unless tmpOperatorTask.empty?
      unless tmpOperatorTask== currentTask[0]
        machine.update_attribute(:tmp_operator_task, nil)
      end
    end
  else
    machine.update_attribute(:tmp_operator_task, nil)
  end
  if operatorInput.equal?(3)
    machine.update_attribute(:tmp_final_fusion, 3)
    machine.update_attribute(:fusion_last_calculated, Time.now)
    return 3
  elsif operatorInput.equal?(2)
    machine.update_attribute(:tmp_final_fusion, 2)
    machine.update_attribute(:fusion_last_calculated, Time.now)
    return 2
  end
  end

  if sensorInput.equal?(1) or operatorInput.equal?(1)
    machine.update_attribute(:fusion_last_calculated, Time.now)
    machine.update_attribute(:tmp_final_fusion, 1)
    return 1
  else
    machine.update_attribute(:tmp_final_fusion, 0)
    machine.update_attribute(:fusion_last_calculated, Time.now)
    return 0
  end
  end

end

def self.latest_sensor_input(machine)
  accepted_period= Time.now.advance(:seconds => -30)
433.  daq = machine.daq_datum.desc(:created_at)[0]
434.  unless daq.nil?
435.    if daq.created_at >= accepted_period
436.      if daq.spindle > machine.min_processing_current_spindle or
437.        daq.tableX > machine.min_processing_current_tableX or
438.        daq.tableY > machine.min_processing_current_tableY or
439.        daq.tableZ > machine.min_processing_current_tableZ or
440.        daq.tableC > machine.min_processing_current_tableC or
441.        daq.tableB > machine.min_processing_current_tableB or
442.        daq.rpm > 10
443.      return 1
444.    else
445.      return 0
446.    end
447.    return 0
448.  end
449.  end
450.  return 0
451. end
452. end
453. end
454. def self.latest_task_input(machine)
455.  unless machine.tasks.where(:modified_start_time.lte => Time.now).and(:modified_end_time.gte => Time.now).empty?
456.    return 1
457.  else
458.    return 0
459.  end
460. end
461. def self.latest_operator_input(machine)
462.  case machine.tmp_operator_input
463.    when 1
464.      return 1
465.    when 2
466.      return 2
467.    when 3
468.      return 3
469.    else
470.      return 0
471.  end
472. end
473. end
474. def self.create_test_tasks(machine)
475.  machine.tasks.destroy
476.  a = machine.tasks.new({:start_time => Time.now.advance(:hours => -1), :end_time => Time.now.advance(:hours => 2), :name=>'t1'})
477.  b = machine.tasks.new({:start_time => Time.now.advance(:hours => 2), :end_time => Time.now.advance(:hours => 4), :name=>'t2'})
478.  c = machine.tasks.new({:start_time => Time.now.advance(:hours => 5), :end_time => Time.now.advance(:hours => 6), :name=>'t3'})
479.  d = machine.tasks.new({:start_time => Time.now.advance(:hours => 6.5), :end_time => Time.now.advance(:hours => 8), :name=>'t4'})
480.  e = machine.tasks.new({:start_time => Time.now.advance(:hours => 8.5), :end_time => Time.now.advance(:hours => 10), :name=>'t5'})
481.  a.save
482.  b.save
483.  c.save
484.  d.save
def self.create_batch_daq_datum(machine, n, hours)
  hours.times do |h|
    timeNow = Time.now.advance(:hours => -h)
    stableTimeNow = timeNow
    for i in (1..n)
      if i < 150
        daq = machine.daq_datum.create!( { :spindle => 1, :created_at => timeNow } )
        daq.update_attribute( :current_fusion, 0 )
      elsif i >= 150 && i < n - 150
        daq = machine.daq_datum.create!( { :spindle => 0, :created_at => timeNow } )
        daq.update_attribute( :current_fusion, 1 )
      else
        daq = machine.daq_datum.create!( { :spindle => 1, :created_at => timeNow } )
        daq.update_attribute( :current_fusion, 0 )
      end
      timeNow = timeNow.advance(:seconds => 1)
    end
  end
end

Code Snippet 9.1 Data processing library example in Ruby.

 1. <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 2.   <xs:element name="reschedule">
 3.     <xs:complexType>
 4.       <xs:sequence>
 5.         <xs:element name="machines">
 6.           <xs:complexType>
 7.             <xs:sequence>
 8.               <xs:element name="machine" maxOccurs="unbounded" minOccurs="0">
 9.                 <xs:complexType>
 10.                   <xs:sequence>
 11.                     <xs:element type="xs:string" name="machine_id"/>
 12.                     <xs:element type="xs:string" name="machine_type"/>
 13.                   </xs:sequence>
 14.                 </xs:complexType>
 15.               </xs:element>
 16.             </xs:sequence>
 17.           </xs:complexType>
 18.         </xs:element>
 19.       </xs:sequence>
 20.     </xs:complexType>
 21.   </xs:element>
 22. </xs:complexType>
Code Snippet 9.2 Schema of XML file generated by LMS MM when a rescheduling is triggered:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<reschedule>
  <machines>
    <machine>
      <machine_id>1</machine_id>
      <machine_type>milling</machine_type>
    </machine>
    <machine>
      <machine_id>2</machine_id>
      <machine_type>milling</machine_type>
    </machine>
  </machines>
</reschedule>
```
Code Snippet 9.3 Example XML file generated by the scheduling service when a rescheduling is triggered.

10 Abbreviations

LMS: Laboratory for Manufacturing System and Automation

CAPP4SMEs: Computer Aided Process Planning for Small and Medium Enterprises

IC: Integrated Circuit

CPS: Cyber Physical System
**SOAP**: Service Oriented Architecture

**IoT**: Internet of Things

**IIoT**: Industrial Internet of Things

**KPI**: Key Performance Indicator

**CPPS**: Cyber Physical Performance Systems

**WSN**: Wireless Sensor Network

**RoR**: Ruby on Rails

**IaaS**: Infrastructure as a Service

**XML**: Extended Markup Language

**CT Sensor**: Current Transformer

**A**: Ampere(s)

**mA**: milli Ampere(s)

**V**: Volt(s)

**CPU**: Central Processing Unit

**RAM**: Random Access Memory

**SRAM**: Static Random Access Memory

**EEPROM**: Electrically Erasable Programmable Read-Only Memory

**A/D**: Analog to Digital

**KBps**: Kilo Bytes per second

**SSH**: Secure Shell

**HTTP**: Hyper Text Transfer Protocol

**AES**: Advanced Encryption Standard

**RESTful**: Representational State Transfer

**API**: Application Programming Interface

**JSON**: Java Script Notation

**BSON**: Binary J SON

**UI**: User Interface
GUI: Graphical User Interface

VPN: Virtual Private Network

IP: Internet Protocol

USB: Universal Serial Bus

SOAP: Simple Object Access Protocol

TLS: Transport Layer Security

CSRF: Cross-site Request Forgery

DDOS: Distributed Denial of Service

SSL: Secure Socket Layer

LMSMM: LMS Monitoring Module

LMSS: LMS Scheduling Service

11 References


25. https://www.archlinux.org/


27. http://rubyonrails.org/


29. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

30. https://en.wikipedia.org/wiki/Convention\over\configuration


32. https://en.wikipedia.org/wiki/Active\record_pattern

34. https://www.mongodb.com/
36. https://docs.mongodb.com/v3.2/core/map-reduce/
37. https://docs.mongodb.com/manual/aggregation/
40. https://docs.mongodb.com/v3.2/reference/method/ObjectId/
42. http://getbootstrap.com/2.3.2/
43. https://docs.python.org/2/library/json.html
44. https://en.wikipedia.org/wiki/Ruby\(programming\_language\)
45. https://docs.mongodb.com/ruby-driver/master/mongoid/
47. https://en.wikipedia.org/wiki/HTTP\%3AL\%2F\%2F
52. https://en.wikipedia.org/wiki/Cross-site\%request\%forgery
55. https://developers.google.com/chart/