Comparing SKF and Erbessd sensor integration for predictive maintenance

Jämför sensorintegration från SKF och Erbessd för prediktivt underhåll

William Sjöström

Supervisor : Erik Berglund
Examiner : Anders Fröberg
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar även rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydda mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© William Sjöström
Comparing SKF and Erbessd sensor integration for predictive maintenance

William Sjöström
wilsj389@student.liu.se

ABSTRACT
The purpose of this thesis was to compare two integration’s of sensors, into a system called Enlight, but could in theory have been integrated to most systems. As a pre-study, the specifications and availability of five sensors were researched. From the pre-study, Smart Edge 4.0 and Phantom EPH-V11/10 from Erbessd, were chosen and then integrated. Usability and performance of the integrations were then compared using cognitive dimensions and stopwatch. Phantom from Erbessd was deemed to be more usable, and the integration of Smart Edge 4.0, had better performance.

INTRODUCTION
The usage of predictive maintenance can have many benefits for a company in terms of both maintenance and costs. Predictive maintenance is the main term for work in this context. SKF is a company that has big ambitions in predictive maintenance, and is the company for which this project will be done.

SKF have since 1907, made bearings for all types of industrial use, but have also, in more recent years, been making software and hardware for predictive maintenance. SKF aims to widen their approach to predictive maintenance by integrating the use of 3rd party sensors in their system Enlight Centre. The purpose for integration with 3rd party sensors is to widen their market of obtainable customers. The purpose of predictive maintenance is to, at an earlier stage, detect deviating data which results in predicting the need for maintenance. An example would be, detecting the need for service on a bearing in a wind turbine, using data vibration and temperature data.

To achieve predictive maintenance, monitoring devices are needed, which is usually done with sensors. Vibration, speed(rpm), temperature are some of the main points that, when monitored, enable the use of predictive maintenance, which in this case is the context of bearings. The sensors communicate with the outside world through gateways. Data can then be transferred from a sensor to an interface through the gateway. The job of the interface is to make sure that the data is formatted correctly and thereafter transferred to the correct database or system. This thesis, will in part look at the availability for integrating third party sensors to systems.

PURPOSE
The purpose of this paper is to evaluate two different application programming interfaces(APIs), that will be used to implement an interface that communicates with multiple sensors through a gateway. As a pre-study, five different sensors will be evaluated based on performance and the availability of integration. Thereafter, two sensors will be chosen and integrated to a system called enlight, but this could in theory be replicated and integrated to any system. The integration will be performed with an interface, using an API, provided by the manufacturer of each sensor. The APIs will be evaluated on two factors: performance, and ease of use using cognitive dimensions.

RESEARCH QUESTIONS
What is the difference in ease of use, between the gateway communication API from SKF and the corresponding API from Erbessd, in terms of cognitive dimensions?

What is the performance difference in terms of time between receiving and delivering a packet in SKF’s gateway communication API and Erbessd’s corresponding API?

THEORY
Background
The sensors that will be researched and evaluated, are wireless vibration sensors. Data is measured at the sensors, and then sent to a wireless gateway, that is mounted and connected to a sensor. Then the gateway sends the received data to their respective API. When the data is received by the API, it is then sent forward to the Enlight system through rest posts. The details and specifications of each part will be described below.

Enlight
The system for which the sensors in this paper are to be integrated, is called Enlight. The Enlight system receives and displays different types of measurements from sensors. In the case of this paper, data from the two sensors will be sent to their respective API’s. The API’s will then send their data to the implemented interfaces. At the interface, the received data will be reformatted to meet the post format required by the Enlight API.
Sensors

**Smart edge 4.0**

The SKF smart edge 4.0 has a frequency range of up to 8 kHz on the y and x axis, and 5 kHz on the z axis. It also has a maximum sampling rate of 25.6 kHz. Inside the sensor, there is a semiconductor based temperature sensor with an operating temperature range of -40°C to 120°C. A microphone is also integrated inside that can accurately measure soundwaves between 50 to 120 dB. The smart edge sensor is a triaxial sensor, meaning that it can measure vibration in three different directions. One negative with the smart edge sensor, is that it uses a power cable, making it not fully wireless. Figure 1, shows the workflow for the Smart Edge sensor. From figure 1 it can be seen that there are two ways that the sensor can communicate with its surroundings. A mobile App is used for configuration of the Sensor through bluetooth, and then data is sent from the sensor to SKF’s AWS service through an access point. Figure 2 shows a picture of the Smart Edge 4.0 sensor.

**EPH-V10 and EPH-V11**

The EPH-V10 and EPH-V11 sensors are two very similar sensors within the Phantom line of Erbessd. As the focus of the paper is regarding integration, the EPH-V10 and -V11 will be discussed in the same category, as integration for one of them will work for the other. Only the specifications for the EPH-V11 sensor will be listed, as it is newer and more modern. From now on, the EPH-V11 sensor will be referred to as the “Phantom Sensor” for ease of reading. The Phantom sensor has a frequency range of up to 10 kHz on the x and y axis, and up to 5.1 kHz on the z axis. Just like the Smart edge sensor, the Phantom sensor is a triaxial sensor meaning measurements can be made on three axles. The maximum sampling rate of the Phantom sensor is 25.6 kHz. There is an integrated lithium battery within the Phantom sensor, which has a battery life of roughly 100,000 measurements. This equates to about 2-4 years, depending on capture frequency. The Phantom sensor communicates with its surroundings using a gateway, which in the case of this paper, sends the data to an API. The EPH-V10 sensor can be seen in figure 3 and EPH-v11 in figure 4. The gateway with which the sensors communicate, can be seen in figure 7.
Languages used in integration
For integration, two languages were used, C# and Golang. C# was needed to be used for integrating with the Erbessd Phantom API. Golang however, was chosen because most of the code written at SKF for sensors, is written in Golang.

Related Work
Wireless Sensor Networking in Smart Homes
In 2008, a study regarding Wireless sensor networking in smart home environments was conducted by Dipak Surie, Oliver Laguionie, and Thomas Pederson[8]. The study is similar in the sense that the setup for experiment, is very similar to the setup of the sensors discussed in this paper. The authors placed several types of wireless sensors within a home, which were connected to a gateway that communicated with an interface. The authors wrote their interface in c#, which is the language that the API for the Phantom sensors is written in. The object manager interface used in the experiment, will behave similarly to the API’s that will be implemented and discussed in this paper. The object manager interface, has four responsibilities, which can be summarized to initialize the sensors, and to provide information regarding the sensors to other interfaces. The authors stated that containing a real-time model of the sensors in the object manager, was of importance for monitoring based on their previous work.[8][9][7]

IoT based Condition Monitoring of Generators and Predictive Maintenance
A paper written by D.Swathi, Dr.T.Anil Kumar, MD.Yaseen, discusses how to conduct predictive maintenance on generators[10]. The sensors used in the study were the ADXL345 accelerometer, and an LM135 temperature sensor. The ADXL345, similarly to the Smart edge and Phantom sensor, is a triaxial vibration sensor. The environment for the authors method, also used a gateway which then sent data to a cloud computing platform. The data was then computed using several formulas and then displayed in a GUI, much like the Enlight system. The main findings of that the authors could conclude, was the importance of real-time data regarding the states of the sensors to achieve sufficient condition monitoring.[10]

Middle ware for Distributed Industrial Real-Time Systems on ATM Networks
In 1996, a paper was written by Ichiro Mizunumat, Chia Shed, and Morikazu Takegakil[4]. The paper discussed an approach for middle ware in distributed industrial real-time system. The authors discuss their implementation of a condition monitoring service called MidART, which operates on a local area network. Although their implementation and environment differs partially from this paper there are some key takeaways that the authors brought up in their paper. One important aspect of having a real-time interface is the ease of use and predictability, which according to the authors is achieved by simplicity and consistency.[4]

METHOD
The following steps were taken to accurately answer the two research questions. During the beginning of this project, a pre-study was conducted. From the pre-study two sensors with
APIs were chosen based on their availability and specifications. Thereafter, two separate implementations of interfaces were made using the APIs supplied by the manufacturers of the sensors, which is SKF and Erbessd. The implemented interfaces were then evaluated based on their performance as well as usability in terms of cognitive dimensions.

Pre-study
As a first step, a pre-study was conducted where two sensors were selected. The sensor selections was made based on two different variables, availability and specifications. Five different sensors from SKF, Erbessd, Amazon AWS, Emerson and KCF were compared. Firstly the specifications of the sensors were compared. The different values that were interesting to compare were the following:

- Measuring range
- Frequency response
- Sampling frequency
- Battery lifetime
- Operating temperature
- Capture frequency

Thereafter, the availability of the sensors and their ability to be integrated to other systems was researched. Information regarding availability was gathered by contacting the companies, as well as looking at their usage policies through their websites. Availability played a bigger part when choosing sensors, and reaching out to the companies was done in the order that the specification comparison generated.

Implementation
The implementation of the interface using the API, had four main purposes.

- Receiving messages from the API
- Handling the various message types
- Constructing a data packet in JSON for sending[5]
- Sending the data packet to the correct endpoint

Message receiving was handled by the API through predefined functions. Data reformatting and data packet sending were developed. As the authors of [4] stated, simplicity and consistency is key when designing middleware in real time systems. Camel case notations will be used throughout development as according to it reduces mistakes and increases readability according to David W. Binkley, Marcia Davis, Dawn J. Lawrie, and Christopher Morrell[1]. For better comparisons of usage and performance between the two implementations, the implementations were made to be as similar as possible.

Evaluation
When the implementation phase was done, the implementations were evaluated. The performance aspect of the research questions was answered using a method called stopwatch. Stop watch is according to David B. Stewart good for non-interactive programs[6]. The stopwatch was started when a message containing a data point was received at the implemented interface, and stopped when the response from the enlight API was received. To get comparable run-times, five measurements were made for each integration and then compared as average values.

The usability aspect of the research question was answered using cognitive dimensions. The guide for using cognitive dimensions to measure API usability was the base for this comparison[2][3]. Usability was measured based on four different cognitive dimensions.

- Total number lines
- Role-expressiveness
- Premature commitment
- Error-proneness

The two implementations were compared using the four cognitive dimensions, and give one point for each dimension where one implementation was better than the other. Finally, the implementation and API with the better usability were determined based on their respective points.

RESULTS
pre-study
A pre-study was conducted, where five different sensors were compared based on performance. The sensors that were compared were the following:

- Monitron, Amazon Web Services
- AMS Wireless Vibration Monitor, Emerson
- Smart Edge 4.0, SKF
- Smart Diagnostics Vibration Sensor Node (SD-VSN-3), KCF
- Phantom Atex EPH-V11, Erbessd

Looking at the specifications of each sensor in figure 2, we can see that the AMS Wireless Vibration Monitor sensor from Emerson is the most versatile. Unfortunately, Emerson was as of writing not willing to give out their Software Development Kits (SDK). The sensors from Emerson could only be used with their own systems AMS device manager, and AMS machine works. Erbessd however, supplied their customers with an SDK upon purchasing their sensors, which enables Integrability to other systems. Monitron from Amazon Web Services was as of writing this paper, recently released. Therefore there are no public SDKs or APIs available yet, but this might change in the future. Thereafter, the availability of SKF’s APIs were researched. As this paper is being conducted in partnership with SKF, their APIs were available. Smart Edge 4.0 is a newer sensor which as of writing has not been implemented into Enlight. Due to the specifications of Smart Edge 4.0 and Phantom Atex being better than the Smart Diagnostics Vibration Sensor, the availability of KCF’s APIs were not researched. Therefore the sensors and APIs that were chosen are Phantom Atex Wireless Vibration Sensor, and SKF Smart edge 4.0.
The phantom sensors were integrated using their provided API, which can be obtained when purchasing sensors from erbessd. The API receives data from a gateway when both are connected to the same network. The gateway automatically detects the API as a monitor when the API is started, which can be seen in figure 7. The API receives nine different types of messages from the sensors.

- **RECEIVED_PHANTOM_ACCEL_DATA**
- **RECEIVED_PHANTOM_TEMP_DATA**
- **RECEIVED_PHANTOM_ACCEL_SETTINGS**
- **RECEIVED_PHANTOM_ACCEL_STATE**
- **RECEIVED_PHANTOM_CURRENT_DATA**
- **RECEIVED_PHANTOM_CURRENT_DATA V2**
- **RECEIVED_PHANTOM_ACCUMULATED_CURRENT_DATA**
- **RECEIVED_PHANTOM_RPM_DATA**
- **RECEIVED_PHANTOM_420_DATA**

From the list of message types, the type **RECEIVED_PHANTOM_ACCEL_STATE** was mainly used for integration. The message contains the vibration values in the axial, horizontal, and vertical orientation in mm/s, as well as a temperature measurement in centigrade. When an acceleration state message (RECEIVED_PHANTOM_ACCEL_STATE) is received by the API, it is then reformatted into four different
json objects as a data point for each measurement point. Thereafter the data is sent to the enlight through a http post.

The Erbessd Phantom integration was implemented using c#.

Smart Edge 4.0
Due to an unexpected peak of Covid-19 in India, shipping of the Smart Edge 4.0 sensor was postponed with a couple of months, which meant that no sensors could be acquired in the time span of this thesis. However, an API was given, that retrieved data from several Smart Edge 4.0 sensors, which are currently set up in India. Fortunately, the integration process would have been almost identical given that the sensors could be shipped. The Smart Edge 4.0 sensors are configured to send it’s data directly to AWS (Amazon Web Services). The given API retrieves data from AWS, and thereafter makes the data available to be read on a get endpoint using an event stream. In the case of this thesis, data is continuously read from the event stream get endpoint, reformatted, and finally sent to enlight through a http post. The Smart Edge 4.0 integration was implemented using golang.

Stopwatch
To get as comparable times as possible, both integration’s were implemented as similarly as possible in terms of data reformatting, and data sending. In figure 9, it can clearly be seen, that the the integration API of smart edge 4.0 is more than twice as efficient as that of phantom. The phantom integration has an average running time of 308 milliseconds from receival to delivery. This may be in large part due to the languages that the integration’s were implemented with, but this will be discussed later.

Usability
Total number lines
The total number lines included all the number of lines needed to complete the integration. However, no empty lines used for readability were counted. As can be seen in figure 10, the phantom integration needed less lines to be completed, and therefore gets one point. Out of the box, the Erbessd phantom

![Figure 8. Phantom sensors](image)

![Figure 9. Erbessd Phantom vs. SKF Smart Edge 4.0 run time](image)

![Figure 10. Number of lines](image)
API comes with many parts already optimally implemented, which lets the user quickly do what is desired. One example of this is the message queue with a switch case, that the user then can implement functions for based on message type.

**Role-expressiveness**

The main point of interest in the aspect of Role-expressiveness, in this case is the format of the data packets that are given to the user through the API. Data packets from Erbessd phantom were considerably easier to understand. This argument is based on the fact that their packets contained understandable names to their specific data points. For example, the temperature data point was named "temperature", and the axial vibration values were named rms1, rms2, and rms3 respectively to axial, vertical, and horizontal vibration. This, however, was not the case for Smart Edge 4.0. Their data packets used numbers to represent the different measurements. For example, temperature was represented as "0005", axial velocity as "0002", and so on. This made understanding the data packets problematic. Therefore the Erbessd phantom API wins in this regard, and gets another point.

**Premature commitment**

To use the API from Smart Edge 4.0, a Server-Sent events connection is needed, which is not clearly stated. The implementer needs to derive this information on their own, which can be deemed problematic. The Erbessd phantom API on the other hand, has the queue that communicates to the API implemented when delivered. Therefore, it is deemed that any premature commitments needed for the Erbessd phantom API are already made, thus giving it another point.

**Error-proneness**

There is a bothersome flaw with the Smart Edge 4.0 API, which causes the user to adapt, and can cause problems if uncaught. Relatively frequently, the Smart Edge 4.0 API will send same data packet twice. If uncaught by the user, this could lead to predictive maintenance formulas used by companies to give inaccurate calculations. However, the problem can easily be solved by comparing the last received packet to the incoming packet. The Erbessd phantom API on the other hand, has not caused any visible error as of writing. Therefore, the Erbessd phantom API gets another point.

**Summary of evaluation**

In the sections above, it has been derived that the Erbessd phantom is appeals more in terms of usability due to “winning” the cognitive dimensions analysis, as can be seen in figure 11. The Erbessd phantom API proved better in all four cognitive dimensions, and results in being better suited in terms of usability.

**Summary of results**

From the results above, it can be seen that the integration for Smart Edge 4.0 was more efficient than the integration for Phantom from Erbessd. It can also be seen that usability was better with Phantom than with Smart Edge 4.0. The reason and meaning behind the results will be discussed in the discussion section.

<table>
<thead>
<tr>
<th>API</th>
<th>Smart Edge 4.0</th>
<th>Phantom Erbessd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number lines</td>
<td>145</td>
<td>143</td>
</tr>
<tr>
<td>Role-expressiveness</td>
<td>Packets hard to understand</td>
<td>Understandable packets</td>
</tr>
<tr>
<td>Premature commitment</td>
<td>SSE client needed, but this needs to personally be derived</td>
<td>All parts that are needed are documented with example implementations</td>
</tr>
<tr>
<td>Error-proneness</td>
<td>Occasional packet duplicates</td>
<td>No errors</td>
</tr>
<tr>
<td>Total points</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

**DISCUSSION**

There are several parts of this thesis that need to be discussed, in terms of both method and results. Firstly, the pre-study that was conducted for this thesis was very heavily dependent on availability, which led to the specifications of the sensors playing no role when choosing. This was very unfortunate, as it would have been more interesting to test and integrate the best available equipment on the market.

Secondly, as the API’s used for integration operated so differently, it was very challenging to compare them on both usability and efficiency. Due to their differing means of operation, the results in terms of efficiency will have less meaning. As a whole, the Erbessd Phantom API was more enjoyable to integrate, as they are open to integration and offer a far more scale able platform. This is not the case with Smart Edge 4.0 from SKF.

Thereafter, the results drawn in the usability chapter are entirely based on my experience when integrating the sensors. Therefore these answers are quite selective but hope to provide an understanding of my experiences with these interfaces. Predictive maintenance is a large part of the emerging takeover of digitization 4.0 and movement of internet of things (IoT) within the industry environment. Therefore, this paper aims to help shed some light on the world of vibrations sensors.

The results for efficiency for the integrations, are affected by the languages used. As different programming languages possess varying performance and usages, the results for efficiency would probably be different if other languages were to be used. Better results for comparison, would have been given if the same language was used for both integrations.

Finally, due to the challenge of comparing two different API’s that function quite differently, the cognitive dimensions chosen were quite scarce. Due to lack of information provided about the inner workings of each API, the cognitive dimensions that I had the knowledge to answer, was quite limited. This leads to the whole usability spectrum not being analyzed, but only discussed from a user standpoint. However, for any other user aiming to accomplish sensor integration, the information about usability should be sufficient.
Further work
Further work in this area, would be to explore the availability of more sensors, and the specifications they contain. The fourth industrial revolution is in full effect, which aims to further improve internet of things, connecting industrial machines to the internet. Therefore I believe this paper is relevant for the future, and more work in this area will be conducted.
REFERENCES


