Energy Monitoring IoT Application using Stream Reasoning

Varun Shah[1], Suman Datta[2], Debraj Pal[2], Prateep Misra[2] and Debnath Mukherjee[2]
[1]*Union College, NY, U.S.A.*
[2]*TCS Innovation Labs, Tata Consultancy Services, Kolkata, India*

Keywords: Energy Monitoring, Stream Reasoning, Internet of Things, Maintainability.

Abstract: We consider the application of stream reasoning to the problem of monitoring energy consumption of a premises with buildings, each building having multiple floors. The floors have energy meters in several categories such as AC, UPS and Lighting. The objective is to compute the real-time aggregate energy consumption and alert whenever energy consumption thresholds are crossed, at the building, floor or meter-type level, thus determining whether there is overloading. We also want to have a solution that can be easily applied to a large number of floors and buildings. We show how just a few continuous SPARQL queries and performance enhancing rules can implement the solution. Finally we compare the performance of queries with and without the HAVING clause and with and without using entailments from rules.

1 INTRODUCTION

Monitoring the energy consumption of a premises is important as it prevents overloading and power outages, allowing the business to run smoothly. Furthermore, being informed of their current power consumption patterns gives companies the opportunity to develop more energy-efficient approaches to power consumption which would help reduce electricity expenses. More and more energy-efficient technologies are emerging and developing a system that can monitor and measure their performance is of utmost importance.

An energy monitoring system is also one example where the Internet of Things (IoT) technology can be applied. The "things" here are the energy meters and the IoT message payloads are the energy meter readings. These readings are captured at a central processing facility. In this paper, we discuss energy monitoring of premises and buildings and floors within the buildings. The processing consists of generation of alerts when thresholds are crossed.

In this paper, we consider the design of an energy monitoring system (EMS) for a building with multiple floors, each having meters that measure the power consumption of AC, UPS and LIGHTING on that floor. The meters stream dynamic data continuously to an IoT cloud platform housing a stream reasoner. The stream reasoner performs reasoning over the dynamic meter data and background knowledge about the relationships between meters and their location (floor, building). The stream reasoner is used to send out alerts when certain thresholds of power consumption are crossed e.g. building-level, floor-level, meter type level etc. One of the important requirements of EMS addressed in this paper is maintainability: how the system can be implemented such that a large number of buildings can be monitored with only a few continuous queries and rules. The QUARKS (QUerying And Reasoning over Knowledge Streams) stream reasoner (Mukherjee, Banerjee and Misra 2013) has been adopted to implement the energy monitoring system.

The contributions of this paper are:

- Design of an ontology for the energy monitoring application which combines both meter domain and building domain
- A combination of only a few rules and continuous queries using aggregate functions to implement energy consumption monitoring in a premises
- Design of maintainable Continuous SPARQL queries that yield comprehensive and accurate results
- Configuration of sliding windows that perform real-time analytics on dynamic data

Shah, V., Datta, S., Pal, D., Misra, P. and Mukherjee, D.
Energy Monitoring IoT Application using Stream Reasoning.
DOI: 10.5220/0006782006270634
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 627-634
ISBN: 978-989-758-298-1

- Evaluation of the pros and cons of using count-based sliding windows for energy monitoring
- Experimental performance evaluation using energy monitoring use cases.
- An analysis outlining the various advantages and disadvantages of the different types of queries used.

The rest of the paper is organized as follows: Section 2 presents the Problem Statement, Section 3 presents the Related Work, Section 4 presents the Solution Approach, Section 5 presents experiments and results, and Section 6 presents Analyses of results. Section 7 concludes the paper and discusses future work.

2 PROBLEM STATEMENT

We are given a premises having a number of buildings. Each of these buildings have an overloading threshold. Additionally, every floor of each of these buildings has its own overloading threshold. Each floor has 3 meters of type AC_meter, UPS_meter and LIGHTING_meter. Each type of meter has its own overloading threshold as well. Our goal is to develop a system that sends out alerts if, at any given point of time, any of these thresholds is exceeded.

To be more specific, an alert is sent out if, at any point of time, any of the following cases are true:

- All meters in a building have a total consumption greater than or equal to the building threshold.
- All meters on a floor have a total consumption greater than or equal to the floor threshold.
- All AC_meter meters, UPS_meter meters and LIGHTING_meter meters in a building or floor have a total consumption greater than or equal to the AC_meter threshold, UPS_Meter threshold or LIGHTING_meter thresholds for the building or floor respectively.

3 RELATED WORK

In this section, we discuss related work on energy monitoring and stream reasoning.

(Vijayaraghavan and Dornfeld 2010) discusses how to monitor energy consumption patterns and reduce it to improve the environmental performance of manufacturing systems. This is achieved through stream processing techniques for automatic monitoring and energy consumption. This paper describes a software based approach for automated energy reasoning to support decision making, concurrent energy monitoring, scalable architecture for large data and modular architecture for analysis. The software also includes components to normalize data exchange with a rules engine and complex event processing (CEP) to handle data reasoning and processing.

(Vikhorev, Greenough and Brown 2013) proposes an advanced energy management framework to monitor and manage energy in factory. It also does real time energy data analysis and performance measurement of energy usage. Key performance indicators (KPIs) are used to monitor energy and measuring performance indicators. Further analysis and optimization are done on the result data. Complex event processing technique is used to do real time analysis using a set of tools and algorithms. It also displays the result data in a graph for better visualization.

In related work on stream reasoners, C-SPARQL (Barbieri et al 2009) is an early stream reasoner that defined a language for stream reasoning based upon SPARQL and its windowing mechanisms. CQELS (Le-Phuoc et al 2011) is another stream reasoner which demonstrates good performance and is based on a native approach without using existing data stream management systems. The stream reasoner QUARKS has usability features such as knowledge packets, incremental queries and application managed windows and also shows good performance.

This paper discusses the application of Stream Reasoning using the QUARKS stream reasoner in energy monitoring. Compared to (Vijayaraghavan and Dornfeld 2010), our work is focused on maintainability aspects i.e. application to large number of floors, buildings and premises. We discuss potential for improvement of performance using rules. Also it signals an alert for any kind of threshold violation at any level.

4 SOLUTION APPROACH

We outline the solution approach in this section. We discuss energy monitoring systems and our solution using stream reasoning.

4.1 Energy Monitoring Systems

Energy Monitoring Systems (EMSs) use sensors and

meters that capture energy consumption, occupancy, temperature etc. and create analytical models in order to optimize energy consumption across offices and homes. Right data models combined with advanced machine learning techniques can enable an EMS to continuously improve its behavior, leading to better predictive capabilities and increased accuracy of anomaly detection on an ongoing basis. It is also very important to get real-time visibility and alerting for the anomalies detected. This work implements alerting based on thresholds. It has been tested on energy consumption (classified into appliance categories like Lighting, HVAC etc.) and occupancy data obtained from a large office building.

4.2 Application Ontology

To model the problem, we present our design of an ontology composed of building elements and meter elements. The ontology is presented in Figure 1.

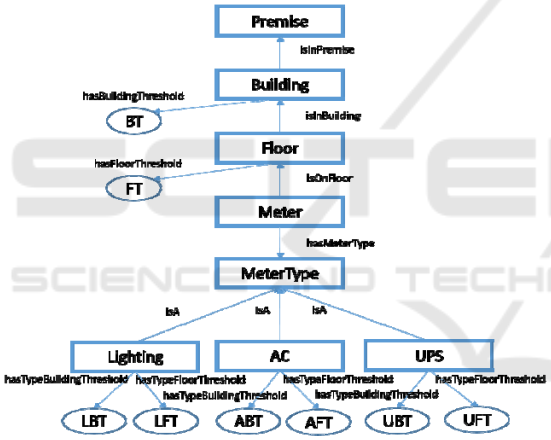

Figure 1: Ontology used in the application.

The rectangles represent “resources” and the ovals are “literals”. The abbreviations are BT: Building Threshold, FT: Floor Threshold, LBT: Lighting Building Threshold, LFT: Lighting Floor Threshold, ABT: AC Building Threshold, AFT: AC Floor Threshold, UBT: UPS Building Threshold and UFT: UPS Floor Threshold.

The Building elements are connected to Premise elements using isInPremise predicate (i.e. Buildings are located in a Premise), the Floor elements are connected to Building using isInBuilding predicate (i.e. Floors are located in a Building). Building elements have a BuildingThreshold property which represents the energy threshold for the aggregate energy consumption of all meters in the building, which when crossed triggers an alert. Similarly, Floors have a FloorThreshold property.

Meters are associated with Floors via the isOnFloor predicate. It signifies that a meter is located on the specified floor. Meters have meter types which could be either: Lighting, AC or UPS. Each meter type has a building threshold which represents the aggregate energy consumption for all the meters of that type in the building beyond which alerts will fire. Similarly, meter types have a floor threshold for aggregate energy consumption of all meters of that meter type on floors.

It is to be noted that the ontology depicted above is used in the static background knowledge of the energy monitoring application.

4.3 Why Stream Reasoning?

We are dealing with continuous, real-time, dynamic streams of reading data. Had we been working with static data, a query based approach would have been sufficient. However, for dynamic real-time analytics, we must adopt the stream processing approach.

This problem requires us to build associations between meters, floors, buildings and meter types. This requires reasoning over static knowledge as well as dynamic knowledge. An example of a static fact is that a meter, meter_1 is on floor_1. The static fact is thus represented as an RDF (Resource Description Framework) triple:

<meter_1, isOnFloor, floor_1>

Two examples of a dynamic facts are the following:
<reading_1 hasMeter meter_1>
<reading_1 hasValue val>

The reading_1 represents a meter reading streamed from the meter to the stream reasoner after conversion of raw meter data to the RDF triple format. The set of two dynamic facts are part of a Knowledge Packet (KP) in QUARKS. (Triples in a KP are processed atomically by the QUARKS stream reasoner). The first dynamic fact states that the reading originates from meter meter_1. The second dynamic fact states that the reading has a value of “val” (val is a literal which contains the actual energy consumption of the meter).
Using the static fact and the two dynamic facts we can compute the aggregate energy consumption on the floor by a SPARQL query, as follows:

```
Select ?floor sum(xsd:float(?value)) where {
?meter isOnFloor ?floor.
?reading hasMeter ?meter.
?reading hasValue ?value.
```

} GROUP BY ?floor

Further we might define rules that allow us to reason over the background knowledge to derive the building-level energy consumptions: since we know that a meter is on a specific floor and the floor is in a specific building, we can associate a meter with the building using a rule and then compute the sum of all meter readings within the building.

For example, we know from the background knowledge that the meter_1 meter is on floor floor_1 and that it is of type AC_meter. Again, from the background knowledge we know that floor_1 is located in building building_1. Thus we can reason automatically that meter_1 is located in building_1. Similarly, we can also associate AC_meter with building_1 since we know that meter_1 is an AC_meter and is located in building building_1. This reasoning becomes important when we are calculating energy consumption at the meter type level and are required to query all the AC_meter meters in building_1.

The energy consumption reported by meter_1 (i.e. the reading of meter_1) is constantly changing and is an example of continuously streamed dynamic data. Now, we not only have to associate meter_1 with building_1 but also the reading of meter_1 with building_1.

Thus we see that both dynamic facts and static facts are required in the reasoning and querying, necessitating a stream reasoner.

4.4 Design and Architecture

The energy monitoring system described in this paper uses the stream reasoner, QUARKS, which uses query processor and reasoner of Apache Jena. QUARKS accepts streams of facts expressed as triples and encapsulated in data structures called knowledge packets (KP). These dynamic facts are combined with static facts stored in background knowledge in a working memory. Rules are run using the fast Rete reasoner (Forgy 1982). Then continuous SPARQL queries are fired to discover situations which need alerting. These situations are conveyed to listener objects which takes necessary actions for generating alerts. The architecture of the application is depicted in Figure 2.

An important component of the architecture is the “meter data to KP converter”. The meter data is simply an energy reading. Along with the reading, the meter id can also be obtained. The KP converter takes these two information and constructs a knowledge packet consisting of two triples:

<reading_1, hasMeter, meter_1>
<reading_1, hasValue, value>

The KP converter sends the KP to the stream reasoner.

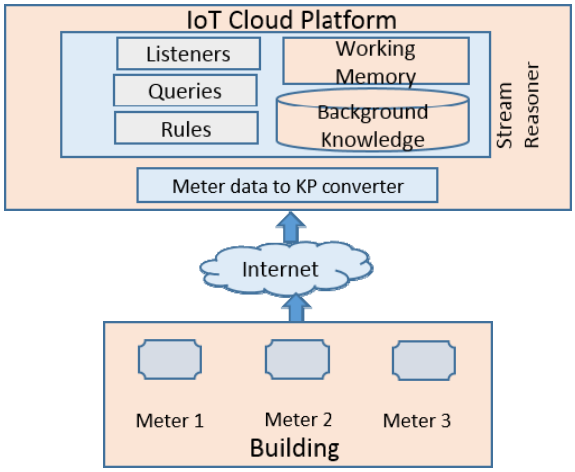

Figure 2: Architecture of the application.

The output generated by the continuous queries are handled by the query listener objects which “listen” to the query and processes the results produced by it. Processing could include printing these results or performing mathematical computations using these results or even comparing these results with an instance variable (like a threshold) and only printing them under certain conditions (like if the threshold has been exceeded).

QUARKS supports windows. A count-based window is used to monitor the meter reading values. The way a count-based window works is that it processes a specified number of events at once. In the case of our problem, the count-based window processes a number of meter readings together. The syntax is: COUNT N, where N is the number of events (KP) maintained in working memory. This count-based window slides by taking N events, processing them and then adding the newest event after deleting the oldest one. This window slides by 1 and only one event is added as another is deleted, thus, maintaining N events at all times.

This sliding count-based window allows us to continuously process meter reading events together which is useful because for the computation we need to perform, we need data from all the meters at a given time. With a sliding window, we can continuously add the latest meter reading while deleting the oldest one. This allows for real-time energy monitoring.

4.5 Solution Details

In order to instantaneously monitor energy, we have designed building-level, floor-level and meter type-level queries that are fired simultaneously, additionally configuring floor-level, building-level and meter type-level thresholds and meter-floor, floor-building, and meter-type relationships in the background knowledge.

Continuously streamed dynamic knowledge is converted into triples that give us information regarding the meters and their readings. These triples could have been constructed in a number of different ways:

- <meter, hasValue, value>: This triple creates a direct association between a meter and its power consumption at a given point in time. However, for two readings of the same meter which have the same value, the system overwrites the reading with the most recent reading observed in the stream. For our purpose, we need readings of each meter separated by time.
- <reading, hasMeter, meter>, <reading, hasValue, value>: Here, we are using two triples to separately store a meter and its reading. With each new event, the value of "reading" (which is a resource in this case) serves as an iterator which helps us identify the chronological order of events. Reading values are stored as "reading1", "reading2", … "reading n". This ensures that the subject is unique for every event, therefore, preventing the overwriting problem we faced with the previous triple.

Now that we have both static as well as dynamic knowledge, let us dive into the reasoning part of this solution. In addition to the background knowledge, we have written a number of rules that the stream reasoner uses to reason on:

Rule 10002: Building Meter Rule
(?meter isOnFloor ?floor)
(?floor isInBuilding ?building)
→(?meter hasBuilding ?building)

In this rule, if a meter is located on a particular floor and that floor is located in a particular building, there is a direct association created between the meter and the building. Both the triples are obtained from the background knowledge.

Rule 10003: Building Reading Rule
(?meter hasBuilding ?building)
(?reading hasMeter ?meter)
→ (?reading hasReadingBuilding ?building)

In this rule, we condense the combination of two triples into one. Here, if a meter is located in a certain building, then the reading resource adopts its value as the building's value. It should be noted that the first triple is an entailment of Rule 10002 described earlier in this section. Therefore, executing this rule will automatically execute the Building-Meter Rule.

rule10005: Reading Meter Type Rule
(?meter hasMeterType ?type)
(?reading hasMeter ?meter)
→ (?reading hasReadingMeterType ?type)

This rule builds an association between the reading and type of meter that is being reported. Here, if a meter has a certain type, then the reading resource adopts its value as the meter type's value.
Rule 10006: Floor Value Rule

(?meter isOnFloor ?floor)
(?reading hasMeter ?meter)
→ (?reading hasReadingFloor ?floor)

Here, if a meter is located on a particular floor, then the reading resource adopts its value as the floor's value. Thus, this rule builds an association between the reading and the floor on which it is recorded.
Rule 10001: Remove Meter Value Rule

(?reading hasMeter ?meter)
(?reading hasValue ?value)
(?reading removeMeter ?meter)
(?reading removeValue ?value)
→ remove(0) remove(1) remove(2) remove(3)

When we wish to delete an event, we simply add the corresponding "remove" triples to the working memory. An example of a remove triple is: <reading1, removeMeter, meter1>. These newly added "remove" triples in the working memory match the meter and value of an existing reading. Once the appropriate triples with the specific ?meter and ?value have been found, they (including the corresponding "remove" triples) are removed from the working memory and are no longer included in any computations, facilitating the "slide".

Now that we have our repertoire of rules, we can implement them in queries at the different levels:

Query 01: Building-level Query

```
SELECT ?building ?threshold (SUM(xsd:float(?value)) as ?sumBuilding) WHERE {
        ?reading hasReadingBuilding ?building.
        ?building hasBuildingThreshold ?threshold.
        ?reading hasValue ?value.
```

```
}
GROUP BY ?building ?threshold
HAVING (SUM(xsd:float(?value)) >=
AVG(xsd:float(?threshold)))
```

This query returns the continuous building-level power consumption if, and only if, the total power consumption of the building meets the building-level threshold at that particular time. There are multiple <building, hasBuildingThreshold, threshold> triples in the result that all have the same threshold value but are continuously added due to the number of events generated by the same building. Therefore, we would not be able to compare the ?sumBuilding to ?threshold as there are multiple threshold values for the same subject and predicate leaving the system confused as to which particular value we are referring to, even though they are all the same! Thus, a good way around that problem is to compare ?sumBuilding with the with the average value of ?threshold since all the values we are considering are the same.

Similarly we could have a floor level query as mentioned below:

```
Query 02: Floor level Query
SELECT ?floor ?threshold (SUM(xsd:float(?value))
as ?sumFloor) WHERE
{
    ?reading hasReadingFloor ?floor.
    ?floor hasFloorThreshold ?threshold.
    ?reading hasValue ?value.
}
GROUP BY ?floor ?threshold
HAVING (SUM(xsd:float(?value)) >=
AVG(xsd:float(?threshold)))
```

```
Query 03: Type Building-level Query
SELECT ?type ?building ?threshold
(SUM(xsd:float(?value)) as ?sumType) WHERE
{
    ?reading hasReadingMeterType ?type.
    ?reading hasReadingBuilding ?building.
    ?type hasTypeBuildingThreshold ?threshold.
    ?reading hasValue ?value.
}
GROUP BY ?type ?building ?threshold
HAVING (SUM(xsd:float(?value)) >=
AVG(xsd:float(?threshold)))
```

This query returns the continuous meter type-level power consumption if, and only if, the total power consumption of that meter type meets the building meter type-level threshold at that particular time.

Similarly, we can have a Type Floor-level Query (Query 04), definition of which is similar to the Type Building-level Query.

Note that the queries above output the location (building, floor) as well as type of meter (AC, UPS etc). This gives the user comprehensive information about the alert.

This design makes the energy monitoring application modular as it gives the user the option to set floor-level thresholds for each type of meter.

All the queries described above implement a count-based sliding window of value 12, since there are a total of 12 different meters in our data set which means that, for any given time, there will be 12 distinct meter readings.

Note that these queries only display the results when one of the thresholds has been met thus saving us the pain of writing code for the listener to check if the sum of energy readings is meeting a threshold and having the queries to function as per our specific requirement.

5 EXPERIMENTS AND RESULTS

In this section we present the experiments, results and analyses.

5.1 Experiments Conducted

The experiments conducted were:
Experiment 1: Compare the query times for the building level query with and without rules, and with and without the having clause
Experiment 2: Repeat the above for the floor level query.

The experiments were conducted on an Intel Core I5 2.67 GHz CPU with 4 GB RAM. As already mentioned in Section 4.1, the system has been tested on energy consumption (classified into appliance categories like Lighting, UPS etc.) obtained from a large office building.

All experiments were run with 1000 events (readings) and the results were averaged.

5.2 Results

Table 1 shows results for building level query. Times are in milliseconds. (Insert time is the time to insert in working memory). The following codes are used in the table: B – building level query, F – floor level query, R – with rules, H – with HAVING.

Table 1: Building level query analysis.

Scenario	Query time	Insert time	Listener time
BRH	3.261	0.3096	0.052
BR	2.389	0.299	0.037
BH	3.578	0.198	0.044
B	2.847	0.171	0.0437

The results for floor level query are in the table (Table 2) below.

Table 2: Floor level query analysis.

Scenario	Query time	Insert time	Listener time
FRH	3.198	0.345	0.122
FR	2.898	0.2685	0.2283
FH	3.253	0.207	0.091
F	3.228	0.223	0.207

6 ANALYSIS OF RESULTS

The results show us that the queries perform best when the HAVING clause is not implemented. It would make sense that removing the HAVING clause would increase the performance of these queries as it allows us to omit querying for each threshold and then comparing the power consumption to those different thresholds. Therefore, each query executes fewer computations and runs faster, leaving the bulk of the work for the listeners to do.

Regarding the performance enhancing rules, the performance enhancement is seen only in specific queries where the corresponding query without rule support would have a large number of patterns. In our case, some improvement is seen in the building level query (Query 01). The building level query is aided by 2 rules: Building Meter Rule (Rule 10002) and Building Reading Rule (Rule 10003), where the former rule output is used in Rule 10003. The Building level query has 3 patterns whereas the same query without rules would need 5 patterns. While the Building Meter rule works on static background knowledge, it will always remain pre-computed in the working memory and is therefore efficient. However Building Reading rule has to be computed every time a reading is received.

The system performs best without the HAVING clause. This is because, removing the HAVING clause means there will be less processing load on the SPARQL processor, therefore, improving performance. Furthermore, we no longer need to query for the threshold of each floor/building/meter type which, again, eases the processing load for the SPARQL engine.

Based on the above, it may seem that removing the HAVING clause and implementing the threshold checking logic in the listener would be the best approach. However, there is a big disadvantage in making the listener compute and manage thresholds. It entails writing a tedious amount of code. In this approach, we are not utilizing SPARQL to its fullest and are depending on regular code to monitor the energy. A big disadvantage with this design is the lack of modularity in the system. This is because, while it is fairly simple to declare instance variables that describe thresholds for different meter types, floors and buildings, it is extremely tedious to do this in cases where there are many buildings in the premises: this means the number of floors increases manifold. By managing threshold overloading within the query itself, we simply access the background knowledge to identify the various thresholds instead of hard coding each of them individually. Also all configuration data including floor, building, meter type etc. and the thresholds are maintained in the background knowledge, making it more convenient for SPARQL continuous queries to access them, and eliminating need for additional code in listener.

7 CONCLUSION AND FUTURE WORK

We conclude that despite the fact that the queries without the HAVING clause perform the best in our use case, it is better to develop queries that implement the HAVING clause since that would reduce the load on the listeners, the lines of code having to be written and would make our energy monitoring system far more modular and maintainable. Moreover, we have shown that the entire energy monitoring system can be implemented using only a few rules and continuous queries.

Further, as currently implemented, our energy monitoring system uses a sliding count-based window which accepts as many values as there are meters. Since the data fed in is ordered by time and meter, the window contains a reading from each of the meters at all times. However, these readings are not necessarily recorded at the same time. This is because of the sliding feature. This is why, we can also consider a tumbling window. This way, the

window will contain all the meter readings at a particular time, and once all the readings have come in, it will delete all of them and add the next specified number of readings.

REFERENCES

Barbieri, D., Braga, D., Ceri, S., Valle, E., Grossniklaus, M. 2009: C-SPARQL: SPARQL for continuous querying. In: Proceedings of the 18th World Wide Web Conference, 1061–1062. ACM.

Forgy, C.L.1982. Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. Artificial Intelligence 19, 17–37.

Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M. 2011. A native and adaptive approach for unified processing of linked streams and linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC, Part I. LNCS, vol. 7031, 370–388. Springer, Heidelberg.

Mukherjee, D., Banerjee, S., Misra, P. 2013. Towards Efficient Stream Reasoning. In: Demey Y.T., Panetto H. (eds) On the Move to Meaningful Internet Systems: OTM 2013 Workshops. OTM 2013. Lecture Notes in Computer Science, vol 8186. Springer, Berlin, Heidelberg.

Vijayaraghavan, A., Dornfeld, D. 2010. Automated energy monitoring of machine tools. Cirp Annals-manufacturing Technology 59: 21-24.

Vikhorev, K., Greenough, R., Brown, N. 2013. An advanced energy management framework to promote energy awareness. Journal of Cleaner Production, 43:103-112.