Abstract - In order to fully exploit the potential of optimization algorithms for energy savings in factory automation settings, it is needed to bring together engineers that have knowledge of signal processing algorithms with shop floor engineers that are experienced with real manufacturing processes and factory automation settings. This paper presents an open database storing energy-relevant data from a multi-robot flexible factory automation system. The database is currently populated with data regarding equipment CAMX state events, process/cell/device energy consumption and quality. The database is accessible via a URL and will be further populated with data related to production performance, machine failures and temporary stops etc. The intention is to provide algorithm developers with the needed datasets for design of optimization algorithms for energy efficient manufacturing settings.

I. INTRODUCTION

The goals set by the European Council in March 2007 (reduction of 20% of the total energy consumption; 20% contribution of renewable energies to total energy production; 20% reduction of greenhouse gases below 1990 emissions) impose a long term shift from a cost-based competitive advantage to one based on high added value (producing more products with less material, less energy and less waste). Needed energy savings are envisioned to be achieved via process / product / machine tool design and cross-layer optimization algorithms working with energy relevant data gathered from all levels of the enterprise, from shop floor to ERP [19].

Generally, at design phase, signal processing and optimization algorithm developers work with large datasets tailored for each application domain of interest. Unlike in other fields (e.g. health [20]), in the manufacturing domain there is a profound lack of such available datasets. In order to develop algorithms for optimization with respect to energy or energy-relevant predictive maintenance, there is a strong need for databases storing real data from various types of factory automation testbeds. The data collected must reflect energy consumption of manufacturing processes, products, workstations, transportation devices, routing options, together with production-relevant indicators (machine utilization, timestamps associated to machine breakdowns and idle time) and business relevant indicators (e.g. energy prices during certain time periods).

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

This paper presents such an initiative, to collect and make available energy-relevant data collected from a real factory automation multi-robot testbed. All data captured is stored in a MySQL database, which is published as a RESTful web service and can be accessed with a proper URL by third party applications. Development of third party applications is thus made easy.

The paper is organized as follows: Section II describes the background of this work, in terms of technologies used. Section III gives details on the factory automation testbed the data is gathered from. Section IV describes the implementation of the monitoring application, the current structure of the database and how it can be accessed via URLs tailored for the needed expected data types. Section V draws the conclusions and outlines future work.

II. BACKGROUND

A. Service Oriented Architecture (SOA). The Device Profile for Web Services (DPWS)

Service Oriented Architecture is a paradigm for distributed applications development based on autonomous and platform-independent services over the web. The services can be described, published, discovered and dynamically assembled. Web Services are among the most popular technologies used to deploy SOA. Implementations generally rely on W3C XML based standards: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) [1].

Implementations of SOA involve, on, one hand, one or more service providers (entities capable of publishing events and providing operations), and one or more clients, on the other hand. Clients (service requestors) invoke services of interest after having discovered them via a Service Broker.

To make possible machine to machine communication in factory automation settings, devices and applications are exposed to the outside world as Web Services, so that they can be discovered and invoked by any other devices / applications within the line. This is done via the Device Profile for Web Services (DPWS) [2]. Fig. 1 illustrates the DPWS protocol stack.

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

WS4D stacks were developed by the members of the SIRENA project. Open source stacks developed by the
WS4D community are WS4D-JMEDS (Java Multi Edition DPWS Stack), WS4D-gSOAP and WS4D-uDPWS [5].


Besides XML, responses can also be transformed to other formats including JSON, by configuring Spring’s message converter properly.

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

Hibernate is an Object/Relational Mapping (ORM) implementation based on Java Persistence API (JPA) designed to assist developers working with both Object-Oriented software and Relational Databases. Since the hard coding required to map data represented in objects to relational databases is time consuming and cumbersome, Object/Relational Mapping techniques deployed by Hibernate provide an easy means of mapping data from an object model (in POJO) to a relational data model [12]. Hibernate can be integrated in the Spring framework used to access database via proper configuration, which simplifying and standardizing the persistence operations dataset source [13].

III. DATA SET

The dataset comes from a production line physically located at the premises of the Factory Automation Systems and Technology laboratory, Tampere University of Technology. The line consists of 10 manufacturing cells, each containing one robot and a pallet-based conveyor system. The layout of the production line is shown in Figure 2, where the robotic work cells are numbered from 2-6 and 8-12. (Cell 7 is a buffer).

The line simulates the assembly of mobile phones. Each of the robot cells are capable of drawing a variety of different frame, keyboard and screen types. In total, the line is capable of drawing 729 different variants of mobile phones providing a suitable research environment for development of new algorithms for production optimization, monitoring, maintenance and control. Cell 7 is a buffer that can store ready or unfinished products, for providing more scheduling. Cell 1 is in charge of determining whether incoming pallets are occupied with finished products that need to be unloaded, or they need further circulation in the line. Quality inspection also takes place here via a machine vision system.

Production line information (e.g. status of robots/products, quality inspection results, energy consumption of...
TABLE I
TESTBED RELATED EVENTS

<table>
<thead>
<tr>
<th>No</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EquipmentChangeState</td>
<td>The message contains information of cell ID, recipe number, device type,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pallet ID, the current state of the robot and a time stamp.</td>
</tr>
<tr>
<td>2</td>
<td>QualityInspection</td>
<td>The message indicates the quality information including pallet ID, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quality of frame, screen and keyboard, the quality of the inspection result</td>
</tr>
<tr>
<td></td>
<td></td>
<td>as well as a time stamp.</td>
</tr>
<tr>
<td>3</td>
<td>EnergyMeter</td>
<td>The message contains energy consumption of the robot, the conveyor and the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>controller located in each working cell published at the time interval of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>five seconds.</td>
</tr>
</tbody>
</table>

IV. IMPLEMENTATION

A. Data gathering: Implementation architecture

Fig. 3 illustrates the implementation architecture. On server endpoint, the web application is designed according to the Model View Controller design pattern, which runs on top of Apache Tomcat servlet. The Model manages the application data. The View displays the data to the user. A Controller is associated to each View and informs the Model when a new input has been performed on the View [14][15][16][17]. In the MVC design pattern, the functional programs in the three parts are developed separately, so the reuse of code becomes possible, thus improving the efficiency of software development [15].

![Fig. 3. Implementation architecture](image)

The Model is implemented in Plain Old Java Object (POJO), the Controller on top of Struts 2 [18] and the View with Velocity [19], HTML and Google Chart Tools. Spring [20][21] and Hibernate [21] map the classes in Model with relational data in MySQL database for better persistence.

Events (exposed as Web Service notifications by the devices in the line) are registered with an Esper engine during the discovery stage by a web service consumer, implemented with Web Service for Devices (WS4D) stack [22], for data selection and manipulation. Web services discovery is initiated via a request from a web browser: the WS4D framework is initiated upon receiving such a request and a client is created in this framework for receiving information from web services within the line. Esper [23] is a Java based component developed for rapid development of applications which require to process large volumes of events in real time. Unlike a database which stores data and manipulates it according to incoming queries, Esper stores the queries in an engine and responds according to incoming events. Then applications can further manipulate data within the response. The queries stored in the Esper engine follow Event Processing Language (EPL) syntax: an SQL-like syntax that operates with event streams instead of tables. The result data in the Esper engine is assigned to instances of classes defined in the Model and stored in database tables.

KPIs can be visualized via a web browser on user request. The corresponding actions retrieve data from the Model, correlate these data to the View and return responses containing HTML to the users for visualization.

The events received from the described testbed, together with Key Performance Indicator (KPI) values evaluated via a Complex Event Processing Esper engine [16] are stored in a MySQL database.

Subscription to testbed events and web service discovery is done via a Client object instantiated by the WS4D framework. Callback methods implemented in this Client class are invoked once a device is discovered, to provide further information related to it (e.g. device serial number/address and service reference instances). Once a service reference instance is retrieved, all service-related events are registered with the Esper engine and can be subscribed to, so that the interested client can receive them.

B. Database structure

At the moment, the database stores raw data and KPI values in three tables (Fig. 4): a data table, a metadata table and a data_metadata table.

![Fig. 4: Database structure and relation](image)

The data table stores:
- the root element name of the incoming XML messages / KPI name (the name column)
- a manufacturing cell identifier (the name column)
- timestamps associated to incoming messages from the line / KPI values (the timestamp column)
- the addresses of the messages / KPI results (the value column)

The metadata table stores:
- the values of attributes / elements within incoming messages (the value column).
- the names of the attributes / elements are stored (the property column). In the case of a KPI, the property and value columns are used to further describe the KPI (e.g. via cell IDs / device types / etc.)
- Similar content to the data table in its name and timestamp columns. These columns are left for future usage in case the line needs to be retrofitted.

Fig. 5 illustrates a shortcut of how an EnergyMeter message is stored in data and metadata tables. One incoming event increases one row in data table, while several rows in metadata table, which represents a one-to-many relationship. In order to map the data in data table and in metadata table, a third table is needed.

### data Table


### metadata Table

<table>
<thead>
<tr>
<th>EnergyMeter</th>
<th>03/03/2012 17:49:26</th>
<th>237.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>21</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>3.70</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>73.86</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>238.32</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>238.09</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>90.41</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>1094.99</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>9</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>-1</td>
</tr>
</tbody>
</table>

Fig. 5 Example of data in database

The **data metadata** table stores the relations of the data in the above mentioned two tables via id numbers. The table is generated by Hibernate [17] automatically by setting a one-to-many relationship in Hibernate’s configuration file or annotating the relationship in POJO. The relationship represented by the **data metadata** table for the above presented example is shown in Fig. 6.

The data acquired by the web application is published as a web service so that other applications can access it via an appropriate URL. The URL contains parameters which are used to request expected responses from the database tables.

### data_metadata Table

<table>
<thead>
<tr>
<th>EnergyMeter</th>
<th>03/03/2012 17:49:26</th>
<th>237.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>21</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>3.70</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>73.86</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>238.32</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>238.09</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>90.41</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>1094.99</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>9</td>
</tr>
<tr>
<td>EnergyMeter</td>
<td>03/03/2012 17:49:26</td>
<td>-1</td>
</tr>
</tbody>
</table>

Fig. 6: One-to-many relation correlation with data_metadata table

### Error! Reference source not found.

Fig. 7 illustrates the entire process, from the moment a URL request is received to the moment an XML response is generated:

1. A client sends a URL request to the web service.
2. The controller maps the request to the corresponding handler method that processes the request. The request mapping is done via a RequestMapping annotation declared before the handler method. Parameter values are acquired from the URL.
3. The parameters are forwarded by the controller to the service layer (in **Error! Reference source not found.**) by calling a method defined in the service class to access database data.
4. Database access is done at service layer. A Hibernate Query Language (HQL) is constructed and forwarded as the input parameter by calling Hibernate Data Access Object (DAO) method to query database tables. In DAO layer, a set of interfaces and its implementations are declared with CRUD functionalities utilizing Hibernate’s sessionFactory instance. The sessionFactory is defined as a Spring Java bean in the Spring configuration file, in which database access parameters such as driverClassName
(e.g. for MySQL database, com.mysql.jdbc.Driver), URL (the URL of the database), user name and password are specified for accessing the database.

5. Hibernate DAO method returns the query results as a list or Java object.

6. The list or object is further assigned to another Java object which is mapped with the XML in the service layer and subsequently forwarded to the controller for serialization of OXM object to XML stream (marshalling).

7. Finally, XML responses are returned to the client as HTTP response.

C. Database details

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

Three types of requests are currently supported by the web service endpoints (Table II): request of all data names within the database, request of data gathered in a certain time period (longer time periods requested translate to slower response/ larger amount of response data) and request of the the last data which has the same name with the one declared in the request URL, the database has stored.

Fig. 8 illustrates a snapshot of the response rendered for the latest energy data related to cell 4. The response related to the energy meter located in Cell 4 contains data regarding 3-phase active energy (AWATTHR, BWATTHR and CWATTHR), active power (AWATT, BWATT and CWATT), reactive energy (AVARHR, BVARHR and CVARHR), reactive power (AVAR, BVAR and CVAR), apparent energy (AVAHR, BVahr and CVAHR), apparent power (AVA, BVA and CVA), Root Mean Square (RMS) current (AIRMS, BIRMS and CIRMS), RMS voltage (AVRMS, BVRMS and CVRMS) as well as line frequency (LINEFREQ)

```xml
<ResponseList>
  <RServiceld="energyMeter">  
    <Value point="AVAHR">89.28</Value>  
    <Value point="AVA">74.75</Value>  
    <Value point="AVAR">96.95</Value>  
    <Value point="AVARHR">-96.95</Value>  
    <Value point="CVA">47.00</Value>  
    <Value point="CBVAR">5.90</Value>  
    <Value point="CBVARHR">-5.90</Value>  
    <Value point="CVAHR">36.40</Value>  
    <Value point="CVAHR">36.40</Value>  
    <Value point="CVAHR">36.40</Value>  
    <Value point="CVAHR">36.40</Value>  
  </RServiceld="energyMeter">  
</ResponseList>
```

TABLE III
DATABASE ACCESS REQUESTS CURRENTLY SUPPORTED

<table>
<thead>
<tr>
<th>Request</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request of all data names</td>
<td><a href="http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/data/ALL">http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/data/ALL</a></td>
</tr>
<tr>
<td>Request of data gathered in a certain</td>
<td><a href="http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/data/%5Bname%5D/from/%5Btimestamp%5D/to/%5Btimestamp%5D/">http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/data/[name]/from/[timestamp]/to/[timestamp]/</a></td>
</tr>
<tr>
<td>time period</td>
<td>Request of all energy related data from cell 5 between 16:10:00 and 16:13:30 on March, 6th, 2012: <a href="http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/sdata/energyMeter5/from/2012-03-06">http://esonia-controller.rd.tut.fi:8080/FastoryService/endpoint/REST/sdata/energyMeter5/from/2012-03-06</a> 16:10:00/to/2012-03-06 16:13:30</td>
</tr>
</tbody>
</table>
Besides, there are also attributes illustrating the device information (deviceType and cellId) and time stamp. For performance testing purpose, the developer also added the estimated response time (estimatedResponseTime), which is not included in the original energy message from the production line, to illustrate the data retrieving time from database.

V. CONCLUSIONS AND FUTURE WORK

In order to fully exploit the potential of optimization algorithms for energy savings in factory automation settings, it is needed to bring together engineers that have knowledge of signal processing algorithms with shop floor engineers that are experienced with real manufacturing processes and factory automation settings (i.e. that have access to data, know which data is relevant for the considered testbeds and how this data can be gathered). To achieve this, the easiest approach would be to have large datasets gathered and stored in the cloud, together with information on how the data can be accessed, so that every interested party can make use of it.

This paper presents such an initiative. A web application is developed for a factory automation testbed, to monitor real time data coming from the manufacturing line and store this data in a MySQL database. The database is then exposed as a RESTful web service so that everyone can access it via URLs. The database is populated already with some data concerning energy consumption of processes and workstations, CAMX state events and quality–related information.

Future work includes optimization of the database and query design to gain faster responses and populating the DB with more data and data values.

ACKNOWLEDGEMENTS

This work has received funding from the ARTEMIS Second Call 2009 Programme under Grant Agreement number 100223, correspondent to the project shortly entitled eSONIA Embedded Service Oriented Monitoring, Diagnostics and Control: Towards the Asset Aware and Self Recovery Factory.

REFERENCES