Enabling the business-based
Internet of Things and Services

(FP7 257852)

D6.5 Analysis of distributed and centralised BI and integration

Published by the ebbits Consortium

Dissemination Level: Public

Project co-funded by the European Commission within the 7th Framework Programme
Objective ICT-2009.1.3: Internet of Things and Enterprise environments

Document version: 1.0 Submission date: 2013-08-31

Document control page

Document file: D6.5 Analysis of distributed and centralised BI and integration.docx
Document version: 1.0
Document owner: Alexander Schneider (Fraunhofer FIT)

Work package: WP6 – Mainstream Business Systems
Task: T6.3 – Investigation and enhancement of integration mechanisms for distributed intelligence and business intelligence
Deliverable type: P

Document status: ☒ approved by the document owner for internal review
☒ approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Alexander Schneider	2013-07-11	Initial version
0.2	Daniela Fisseler (FIT), Ferry Pramudianto (FIT)	2013-08-13	Business process monitoring, Domain model and
0.3	Matts Ahlsén (CNET), Peeter Kool (CNET),	2013-08-21	Event processing mechanics
0.4	Marek Paralic (IS)	2013-08-22	Device modelling and identification
0.5	Peter Kostelnik (TUK)	2013-08-23	Event ontology and annotations
0.6	Yves Martin (SAP)	2013-08-24	Business intelligence and integration
0.9	Alexander Schneider (FIT)	2013-08-26	Version for internal review finished
			Final version submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Pietro Cultrona (COMAU)	2013-08-29	Accepted with minor comments
Paolo Brizzi (ISMB)	2013-08-28	Accepted with minor comments

Index:

List of Abbreviations:

BAM	Business Activity Monitoring
BPM	Business Process Modelling
BPMN	Business Process Modelling Notation
CEP	Complex Event Processing
EPA	Event Processing Agent
EPC	Event-driven Process Chain
EPN	Event Processing Network
ERP	Enterprise Resource Planning
IERC	IoT European Research Cluster
IoT	Internet of Things
MES	Manufacturing Execution Systems
PSO	Product Service Orchestration
PWAL	Physical World Adaptation Layer
UML	Unified Modelling Language
WSBPEL	Web Services Business Process Execution Language
XMI	XML Metadata Interchange

1. Executive summary

Internet of Things (IoT) applications are characterized by their ability to integrate large number of data sources and support the end user by providing a better user experience. This is accomplished by the vast amount of information that the distributed data sources provide and interpret them in a meaningful way.

For IoT systems the challenge is to properly identify which data sources are available and how to get the data and interpret them in a meaningful way. For this the ebbits platform provides models where the addressed devices and their data (events) are semantically described. The ebbits Entity Manager makes the devices identifiable by providing a unique id within the ebbits system. This allows for correct interpretation of the data coming from the devices. The PWAL (Physical World Adaptation Layer) allows for the integration of devices regardless of manufacturer and supported interfaces. Besides simple event processing of the raw device events Complex Event Processing (CEP) is supported where the gathered data can be processed (e.g. rule processing or filtering) before the data is provided for the application. Both centralized as well as distributed event processing is supported. The data coming from the devices can be used for online applications but will also be persisted in order to use this data for bottom-up historical data analysis.

The context of the user is an important way to identify situations and provide the user the best support. For this a domain model can be defined which allows the ebbits system to identify situations and react in a proper way. The data enrichment is also based on the domain model so that incoming data is identified by the sending sensor and e.g. related to a room that is defined in the domain model. This could be used for example to easily query for "temperature in room 123".

In order to better understand the business processes and to identify optimization potential it is necessary to get detailed information. The ebbits system allows defining a business process model and enriching the gathered data with the process step information automatically. In this way the user could analyse in which process step an unusual amount of energy was consumed and further investigate it. The gathered data can then be used in other optimization tools to reduce the energy consumption.

Existing business systems that are already installed in companies can easily be integrated and the data coming from the ebbits system can be exchanged. This allows for providing the business systems with much more detailed data that can be used to e.g. control the production processes or allow for more detailed analysis of the data and ultimately better business decisions.

2. Introduction

One key aspect of Internet of Things (IoT) applications is that data from various sources will be used to provide new functionalities. These data sources can be potentially spread over the whole world and another aspect is the large number of data sources that can potentially be integrated.

In the ebbits project we will develop a platform that support both distributed as well as centralized data sources. Delivering the right data to such systems also means that collected data needs to be interpreted and probably transformed into a fitting format. For this we have developed an interface to connect any kind of sensor to our platform regardless of manufacturer, wired or wireless connection as well as independent of supported protocols by the device.

We will show how a model-based approach is used in order to enrich gathered information automatically and semantically and what kind of possibilities one can use for processing the gathered events. We process the data in order to get the context of entities and derive situations that are beyond simple comparison of threshold values. This can be used to detect possible machine breakdowns before they happen and a support crew could be informed to do maintenance on this particular machine before production needs to be stopped.

In order to provide data for optimization of e.g. energy consumption a process-based monitoring is included in the ebbits platform. This allows to aggregate data based on the process steps coming from a BPMN (Business Process Modelling Notation) model and allow for a much more detailed analysis than is currently possible.

Distributed and centralized business intelligence (BI) means the usage of data coming from centralized or distributed data sources and the integration of those. For the business aspect it is also important to be able to connect to business applications e.g. Enterprise Resource Planning (ERP) system that have a widespread usage. This is especially important in Manufacturing Execution Systems (MES) because e.g. in production external applications are not allowed to tinker with the processes and take over control of machines.

3. Distributed and centralised intelligence

3.1 Sensors and Devices

Each IoT system is inherently dynamic due to the mobility of physical entities and IoT devices from which it consists of. If we use the terminology of the IoT-A project (IoT-A D4.1, 2011) we can state that IoT applications use virtual entities as digital representations of physical entities, which could be monitored or operated by devices. Devices are either attached to the physical entities directly or the physical entities are in their operation area. The software part of the device that provides information on the entity or enables controlling of the device is a resource. The functionality implemented by the resource is exposed by services. Services provide well-defined and standardised interfaces, hiding the complexity of accessing variety of heterogeneous resources. The interaction with a physical entity can be accomplished via one or more services associated with the corresponding virtual entity (IoT-A D2.1, 2012).

In order to deal with entities, devices and services in IoT applications properly, we need identification scheme(s) that enable resolution in wide sense. By resolution in a wide sense we mean not only mapping names or identifiers to locators or addresses for access purposes, but also discovery and lookup functionalities. Regarding the discovery functionalities, ebbits utilises semantic approaches based on semantically annotated resources, as the majority of current IERC projects do (IERC-AC2 D1, 2013). Details about the use of ontologies can be found in (D4.M36, 2013) and (D9.5, 2013).

Regarding the identification of entities, ebbits provides global service for assigning of unique identifiers. This service is offered by the Entity Manager deployed as a cloud service. Entity Manager assigns world-wide unique identifiers (random generated numbers) that can be persistently stored and used by ebbits applications. Input of the service for creating unique identity is any local name/ID and anytime later could be added further local aliases of created unique identity. Details about identity management for entities provided by ebbits can be found in (D8.8.2, 2013).

If a persistent unique identifier would violate e.g. privacy requirements of an ebbits application, the virtualisation functionality of the ebbits middleware could be used. It has been realised by applying the concept of a semantic-free addressing layer provided by the underlying LinkSmart middleware that uses Virtual End Points to address services. As these end points neither do provide any information about the resources they represent nor can be assumed to be persistent, they allow addressing entities without tracking them – thereby enabling the creation of "virtual" devices/entities exposed as services (D8.8.1, 2012).

The semantic modeling of devices is directly supporting the idea of a Model Driven Architecture. The basic idea is to create the models of all devices, which are used in a concrete application. These models serve as prototypes, which are in the deployment phase tied with the set of physical devices with concrete persistent identifiers assigned by the middleware. Those identifiers enable finding of related physical device instances for the real device already existing in the ontology. The instances of expected physical devices are created in advance following the needs of application. This approach is based on expectations for the context modeling, which must be precisely defined for each application, for example, to be able to determine concrete devices in concrete locations or concrete devices attached to context entities (e.g. animals or robots).

The logic behind dividing devices to models and physical devices is that some of the physical devices are simply based on the same model (even having the same events), e.g. set of the same type of RFID readers (with different persistent identifiers) placed at the same location. To enable the contextual modeling flexibility, the location could be physical location, such as pigsty, but also a specific context entity, such as a door.

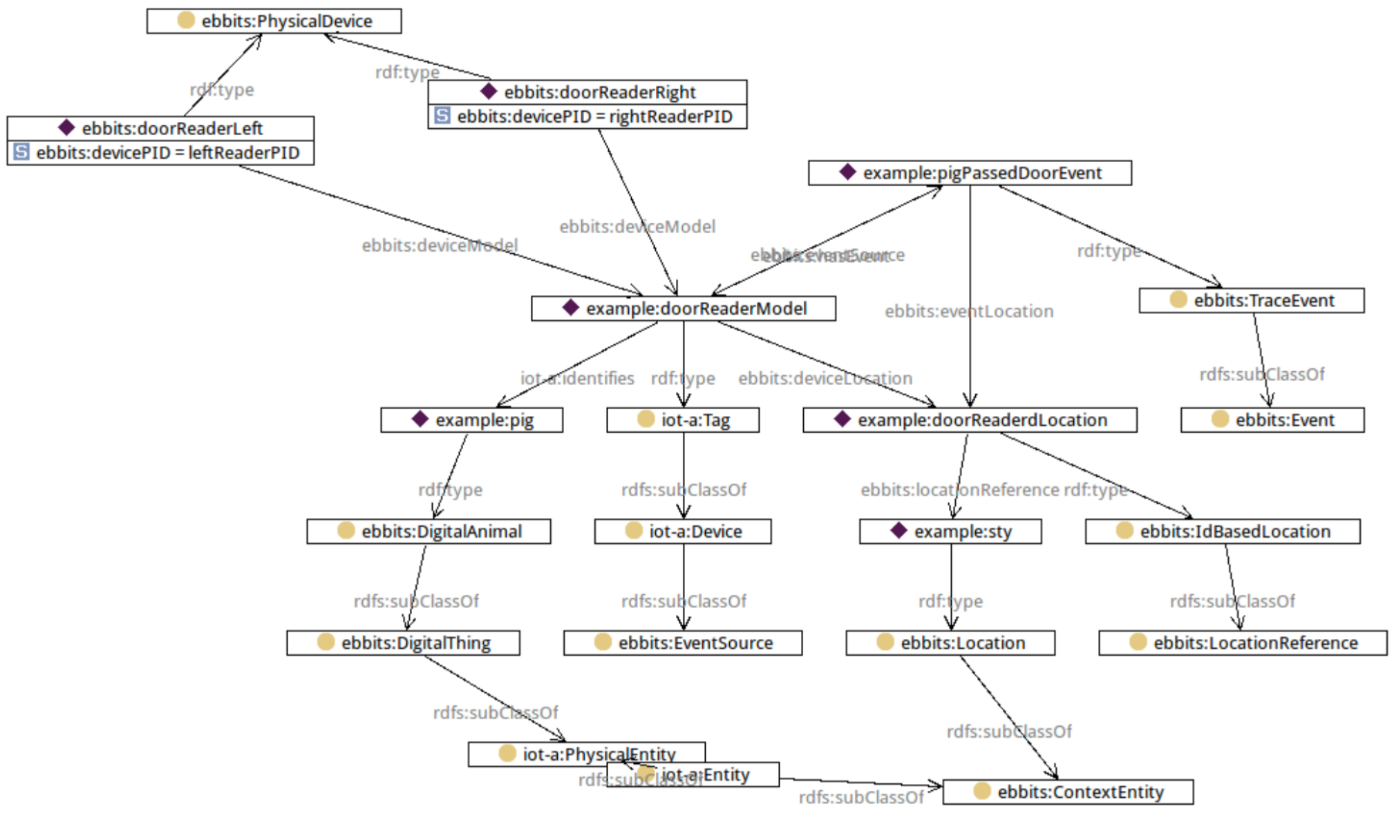


Figure 1: Illustration example of device modeling

The illustration example in Figure 1 shows the two tag readers placed at the pigsty door having the same model (but different persistent identifiers assigned by middleware). This approach enables to easily retrieve whole context surroundings for each device.

The ebbits device model is aligned with IoT-A reference model (IoT-A D2.1, 2012) extended for purposes of ebbits (see also (D4.M36, 2013) for description, how ebbits ontologies were adapted to IoT-A reference model).

The approach of having instances of physical devices in ontology makes sense, where concrete devices are tied to the concrete context, which has to be resolved for each physical device separately.

3.2 Event processing mechanisms

The next step is to show how events and data from devices are handled, stored and distributed. We also show what kind of data annotation can be done automatically and what kind of models and mechanisms are available for this. For a detailed description of the ebbits event and data (thing) management architecture see D7.6.1 (D7.6.1, 2013).

3.2.1 From simple to complex event processing

Event processing in ebbits takes place on several layers in the architecture. Low-level ("raw") events are received from the physical world via the PWAL interface. From then on, several components can process and re-distribute the (possibly transformed) events in different ways.

The LinkSmart event manager is part of the overall ebbits event management architecture. This is an implementation of the basic and well-known publish and subscribe event processing paradigm, and provides a simple form of event semantics by means of Topic lists. The LinkSmart publish and subscribe event manager is suitable when event structure and semantics are less complex and when a loosely coupled distribution mechanism between publishers and subscribers is suitable. In order to support Complex Event Processing (CEP) using additional distribution mechanisms such as direct addressing, Event Processing Agents (EPAs) where introduced. EPAs provide rule processing capabilities, filtering and annotation functionality. Several instances of EPAs can be used to form an event processing network (EPN), in the same processing context (e.g., a server node) or over several processing nodes, such as those involved in a traceability process.

Supporting CEP implies the provision of complex event data structures, as well as semantic models over these events, which is provided by the ebbits ontology framework (which can also support domain, context and service models). The EPAs use the event ontology for matching and for semantic annotations of events and can execute rules expressed over the ontology contents.

The CEP in ebbits also supports event persistency by means of storage services. This allows us to keep event histories, to express rules and queries on past events, which can be used in fall-back and recovery mechanisms.

The LinkSmart Pub-Sub implementation may not be appropriate in a highly distributed environment with large numbers of event producers. For more high frequency event processing a complementary event manager solution has been implemented. The purpose is to provide the "first line" processing of massive event streams such as those that are produced by the sensors in applications like the ebbits manufacturing scenario. The solution is based on the existing 3rd party tools Storm (Storm, 2013) (for data stream processing) and Kestrel (Kestrel, 2013) (for message queuing). These represent techniques currently been applied to applications that process high frequency/volume ("big") data. More details about the Storm/Kestrel based approach is reported in the accompanying deliverable (D7.6.1, 2013).

3.2.2 Centralised and distributed processing

The ebbits system supports the notion of a "thing", which is used to represent any physical or digital entity (c.f. the IoT-A information model mapping to ebbits, (D4.M36, 2013)). The event processing mechanisms with EPAs and event managers are used for distributing the processing intelligence (based on rules). The role of the thing and the corresponding Thing Manager is to coordinate the gathering, transfer and storage of any data and events that can exist for an entity (in present and past).

A basic structure, a Product Service Orchestration (PSO), is used as a "container" for events, data and services that pertain to a thing of certain type. The PSO thus references several of the ebbits ontologies.

Figure 2 below shows an example of a PSO from the ebbits demonstrator application on traceability.

```xml
<?xml version="1.0" encoding="utf-8"?>
<product modelRef="EbbitsProductOntology" type="beef">
  <startCondition>
    <event type="cowBorn"></event>
  </startCondition>
  <stage type="livingInFarm">
    <startCondition>
      <event type="cowBorn"></event>
    </startCondition>
    <thingProperties>
      <services>
        <basicThingData/>
        <!-- Thing meta data -->
      </services>
    </thingProperties>
    <lifeCycle>
      <services>
        <feed modelRef="ServiceOntologyID">
          <frequency type="daysInBetween">1</frequency>
          <!-- Search criteria for selection service -->
         </feed>
        <medication modelRef="ServiceOntologyID">
          <event type="medicationAdministered"/>
        </medication>
        <raisingCondition modelRef="ServiceOntologyID">
          <event type="onStageExit"/>
        </raisingCondition>
      </services>
    </lifeCycle>
  </stage>
  <stage type="transport">
    <startCondition>
      <event type="animalToSlaughter"></event>
    </startCondition>
    <thingProperties>
      <services>
      </services>
    </thingProperties>
    <lifeCycle>
      <services>
        <distance modelRef="ServiceOntologyID">
          <event type="onStageExit"/>
        </distance>
        <time modelRef="ServiceOntologyID">
          <event type="onStageExit"/>
        </time>
      </services>
    </lifeCycle>
  </stage>
</product>
```

Figure 2: A PSO from the traceability demo (partially instantiated)

A Thing is represented by the PSO and the set of software managers (e.g., a Thing Manager and a Service Orchestration Manager) that interprets and acts on the PSO. In this context the EPAs (and event managers) are used to process events, e.g., by filtering and annotation, and also to control the transfers of data and events related to a thing between the steps in a process[1].

3.2.3 Event ontology and semantic annotation of stored data

This section will focus on the semantic annotations extending the support for eventing. The detailed description of ebbits event ontology can be found in (D7.3.2, 2012). The ebbits event model aims to describe the complete structure of all events which are used in a concrete application. These models are created based on application requirements in advance and they are tied with the sources that are producing them. The event models are supporting the eventing process by prescribing the event structures and the connection to the context entities which may be used to extend the basic event information and to retrieve several taxonomies such as hierarchies of event topics etc.

But for purposes of additional analysis of stored historical data, the basic event models often do not contain enough information. Thus the basic event model must be extended with additional information to be easily available for further queries on stored data (see section 3.2.4 Examples of semantic support for historical data analysis). Updates of the event ontology enables the extension of event data by additional information in form of semantic annotations – the pointers to several parts of ontology – which are stored with event data and later used as keys to easily filter and retrieve very specific stored data. This approach practically optimizes the retrieval of stored data. The basic event model extended with semantic annotations is depicted in Figure 3.

[1] In addition to EPAs a number of other manages are also involved in such processes, e.g., see the Traceability Demonstrator description D9.5.

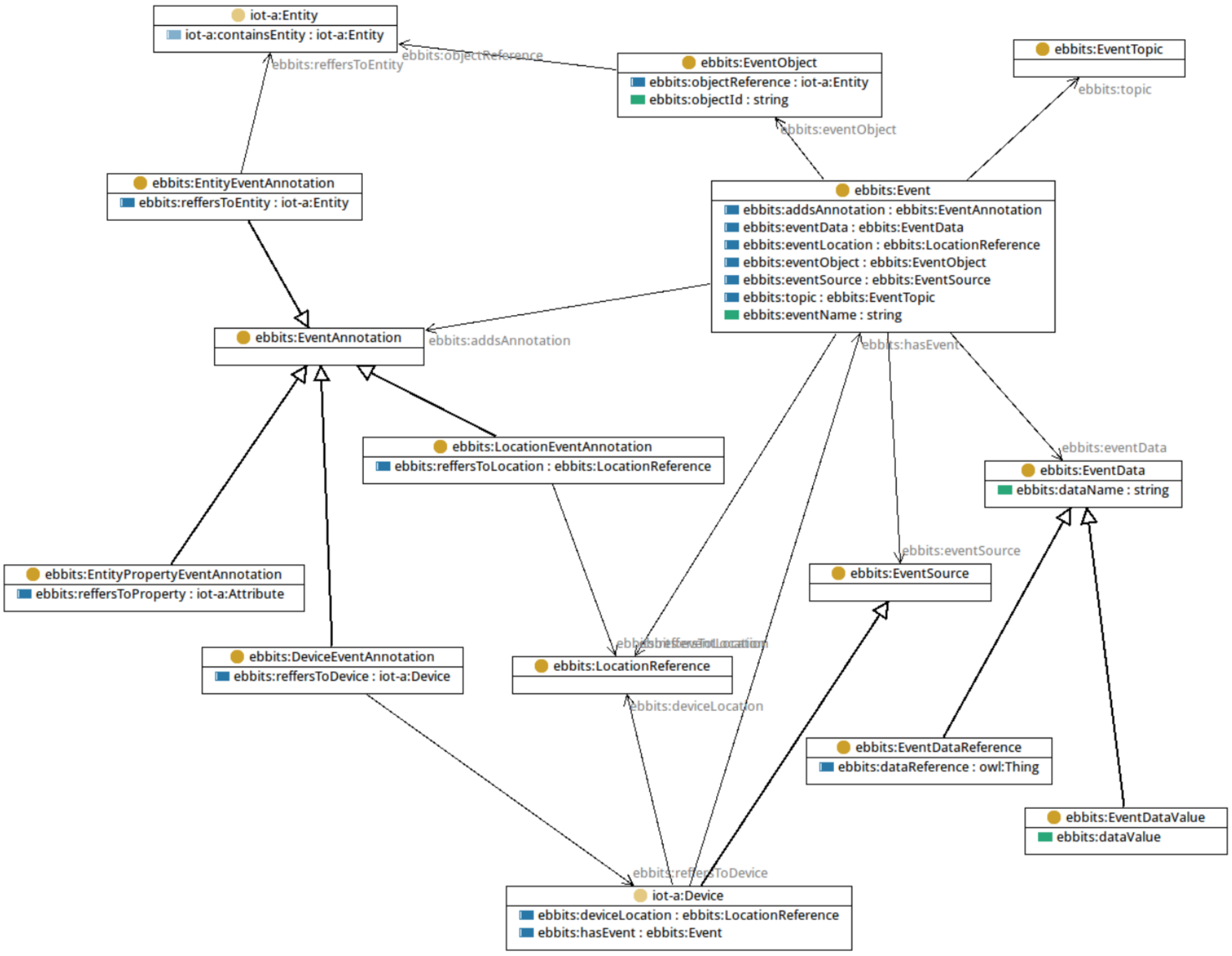

iot-a:Entity
iot-a:containsEntity : iot-a:Entity
ebbits:objectReference
ebbits:reffersToEntity
ebbits:EventObject
ebbits:objectReference : iot-a:Entity
ebbits:objectId : string
ebbits:EventTopic
ebbits:topic
ebbits:eventObject
ebbits:EntityEventAnnotation
ebbits:reffersToEntity : iot-a:Entity
ebbits:Event
ebbits:addsAnnotation : ebbits:EventAnnotation
ebbits:eventData : ebbits:EventData
ebbits:eventLocation : ebbits:LocationReference
ebbits:eventObject : ebbits:EventObject
ebbits:eventSource : ebbits:EventSource
ebbits:topic : ebbits:EventTopic
ebbits:eventName : string
ebbits:EventAnnotation
ebbits:addsAnnotation
ebbits:hasEvent
ebbits:LocationEventAnnotation
ebbits:reffersToLocation : ebbits:LocationReference
ebbits:eventData
ebbits:EventData
ebbits:dataName : string
ebbits:EntityPropertyEventAnnotation
ebbits:reffersToProperty : iot-a:Attribute
ebbits:eventSource
ebbits:EventSource
ebbits:reffersToLocation
ebbits:eventLocation
ebbits:DeviceEventAnnotation
ebbits:reffersToDevice : iot-a:Device
ebbits:LocationReference
ebbits:deviceLocation
ebbits:EventDataReference
ebbits:dataReference : owl:Thing
ebbits:EventDataValue
ebbits:dataValue
ebbits:reffersToDevice
iot-a:Device
ebbits:deviceLocation : ebbits:LocationReference
ebbits:hasEvent : ebbits:Event

Figure 3: Extension of event model with semantic annotations

The idea behind the annotation approach is very simple. The specific events may have attached application (expected analysis) dependent annotations to relevant parts of the ontology. This is modelled using a hierarchy of EventAnnotation concepts which prescribe what data should be additionally stored with event as well as what entities are tied with this data and how these relationships should be interpreted by a rule engine or further data analysis algorithms. As an example Figure 4 shows annotations to context entities, entity properties, devices and locations. The domain ontology containing context entities and their relations serves as the common vocabulary and they are expected to have the uniform interpretation. It is quite straightforward to specify in application logic how to extend event data with very concrete information (for further explanation of common vocabulary and its interpretation see (D4.M36, 2013)). To enable several combinations of additional properties the event may have attached as many annotations as needed, following the needs of further data processing or retrieval. The simple example of concrete event extended with annotations is illustrated in Figure 5.

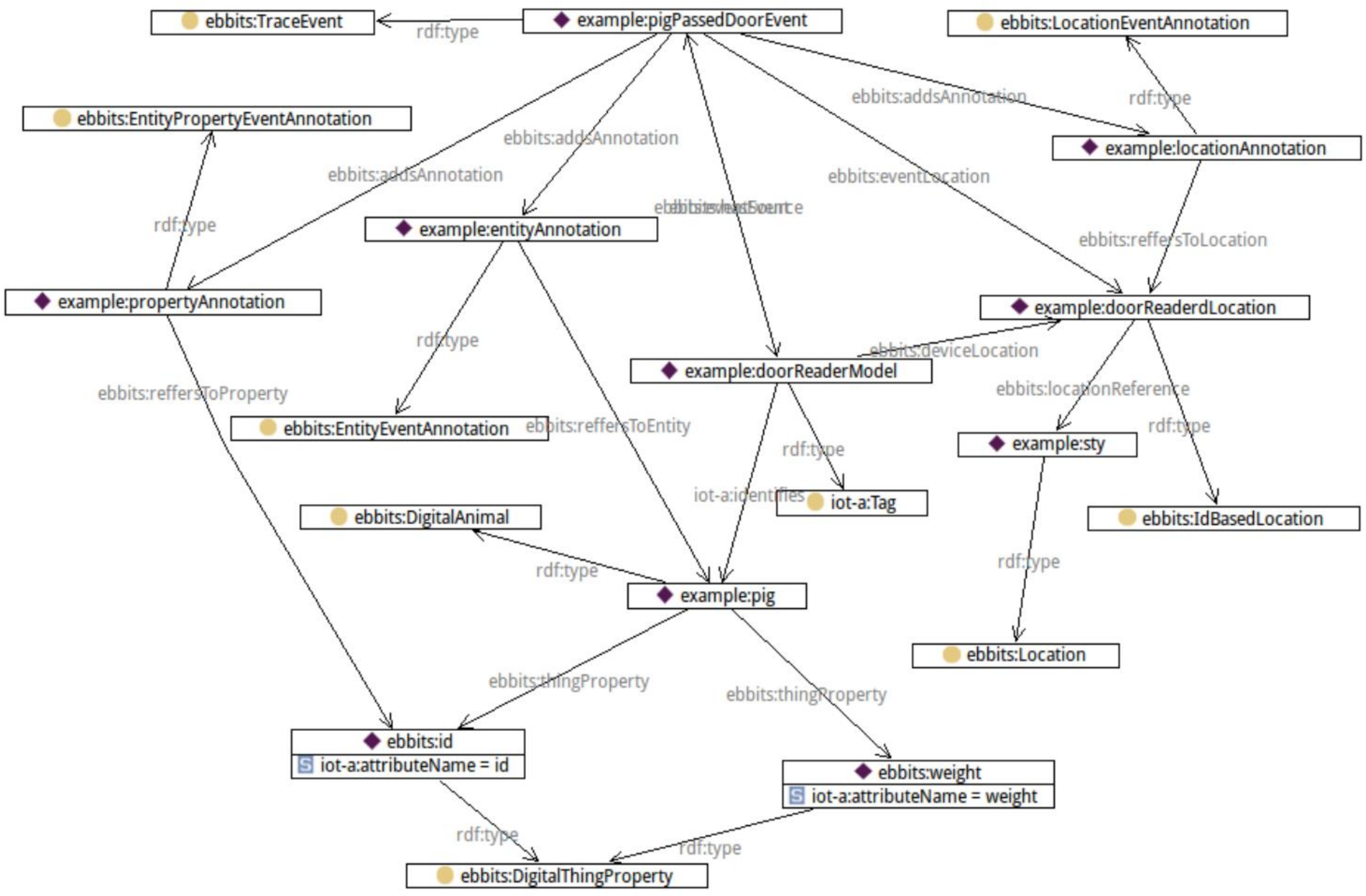


Figure 4: Example of event with semantic annotations

The example shows the event generated by tag reader when a pig passes the door. The event is extended with three annotations which are processed by the rule system in uniform way following the vocabulary interpretation, for example:

- entity annotation referring to the entity type – the pig instance – entity annotation adds reference to the entity ontology instance thus all data pointing to the pigs can be directly retrieved
- entity property referring to the id property – the annotation is interpreted by a rule engine which adds the concrete value of an id property if it is available, thus all data pointing to a concrete entity can be retrieved (this makes sense to be used in combination with the annotation referring the entity type)
- a location reference pointing to the concrete location (does not matter if it is the concrete place or concrete entity) thus all data at concrete location can be easily retrieved.

Following the semantic annotations specified for a concrete event the rule engine will interpret it and extend the stored event data with respective XML structure which will later serve as the WHERE part of query for data retrieval. The extension XML structure would look as follows:

```
<semantic-annotations>

<annotation modelRef=”ebbits:EntityEventAnnotation”>ebbits:pig</annotation>

<annotation modelRef=”ebbits:EntityPropertyEventAnnotation”>

<annotation-property>ebbits:id</annotation-property>
```

```
<property-value>concrete ID added by rule engine</property-value>

</annotation>

<annotation modelRef="ebbits:LocationEventAnnotation">ebbits:sty</annotation>

...

</semantic-annotations>
```

Figure 5: Example of semantic annotations for events

Using this approach the types of annotations can be extended in various ways where each annotation type allows for the uniform interpretation in application logics. Examples of usage of semantic annotations will be described in the following chapter.

3.2.4 Examples of semantic support for historical data analysis

The semantic support for processing of stored data is realised by using the annotations stored with events generated within the ebbits platform. This solution is very straightforward and ebbits specific. The stored data can be easily queried and filtered using the combinations of semantic annotations. Addressing the concrete combination of semantic annotations enables to directly retrieve very specific data which may be used in very specific ways. It can be used both in the real-time phase to support more complex decisions as well as in data analysis which processes the historical data in a bottom-up manner.

We provide two simple examples of bottom-up historical data analysis.

Off-line example: manufacturing scenario

The first usage of semantic event annotations happened in the M24 demo. There were several voltage and ampere sensors attached to several parts of the welding robot. Parts of the robot were modelled as embedded context entities, to each context entity there were attached particular sensors. Event data measured at several parts of robot were semantically annotated to the sensor type and concrete part of the robot. The robot working cycle was monitored for a longer time (in our case several days of continuous running) and annotated data was stored. At the end of the experiment we collected gigabytes of data.

The purpose of the scenario was to create the typical behaviour of each part of the robot using the data measured at each part. The stored data was analysed for each particular part and based on this analysis the behaviour model for concrete parts of the robot was created. This model was created in a bottom-up manner by off-line processing. The results of the analysis were later used in real-time diagnosis in order to detect if a robot's performance is within the boundaries of its typical behaviour, otherwise the alert event was created. Semantic annotations were used to filter and retrieve data tied with the concrete parts of the robot and values measured by sensors attached to particular parts. The example of the bottom-up analysis result is illustrated in Figure 6.

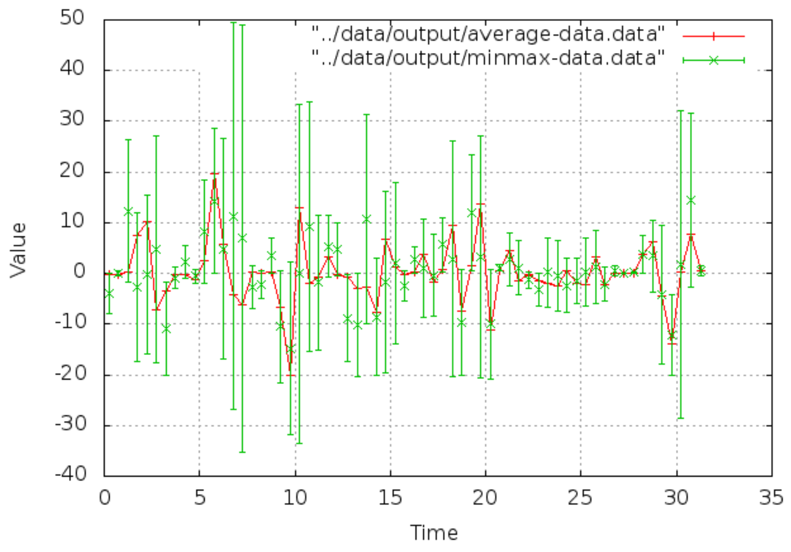


Figure 6: Example of bottom-up data analysis using semantic annotations

The picture shows the result of the analysis of data stored for one concrete sensor attached to a concrete part of the welding robot. The red line shows the average values computed from all robot cycles continually measured over several days. The green line shows the model of typical behaviour of robot part in terms of minimal and maximal values representing the boundaries of normal behaviour.

Farm application scenario

Another – hypothetical – example is the monitoring of drinking patterns for groups of pigs. The scenario can be described as follows: In the pigsty is a drinking place with water. To the drinking place a sensor is attached, continually measuring the amount of water consumed by a group of pigs. For a group of pigs of a certain size there exists the typical drinking pattern during a specific time period. If a deviation of the actual drinking pattern against the typical one occurs there is possibility that a disease in the group of pigs has spread.

The solution is very similar to the manufacturing scenario. In a very simple way the event data produced by each sensor measuring the decrease of consumed water are annotated to the location. The measured data are processed once per time period, a drinking pattern is computed and compared to the typical drinking pattern. When there is the violation against typical pattern, the alerting event is raised.

In this very simple way, the semantic annotations can be used practically for any required bottom-up data analysis.

3.3 Domain model

The domain model is used to combine the devices and their events and process them to get a context. Here we would show how this will be modelled and how this looks like for centralised and distributed setting.

Sensor data delivered by the event manager might not be too useful for the users without knowing the contextual information such what objects are being measured, what properties of the objects the sensor data represent or when the measurements are taken.

Contextual information can be captured from the domain model of the application. In WP5 a development tool is developed to support ebbits developers correlating sensor stream and domain model to achieve contextual information for the application (D5.6.1, 2013).

The tool allows developer to use a model driven approach to correlate sensor stream to the domain objects and their properties visually. After the model is done by the developers, they could generate java code that encapsulates the sensor technology. The generated java artefacts will expose plain java objects (POJOs) whose properties will be updated automatically by the generated code based on the sensor data. This way, developers will get the contextual information without having to know details of the sensor technology (see Figure 7).

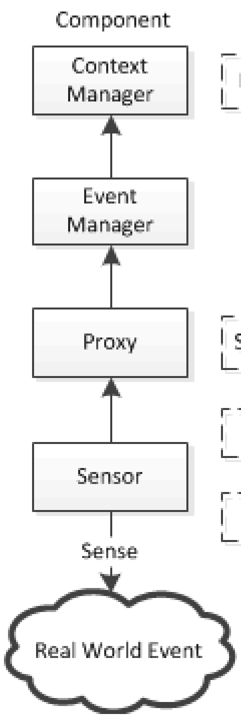


Figure 7: Abstraction of Sensor Data

A visionary scenario of an IoT system is illustrated in Figure 8, in which service providers run their business by providing new services enabled by IoT. This scenario follows the trend of IoT brokerage such as Xively (Xively, 2013). In this scenario, the service providers are required to develop proxies for their sensors and actuators to expose the offered services. The discovery manager will find these proxies using WS-Discovery (WS-Discovery, 2009) protocols and extract the service information such as the address, device capabilities, unit of measurements, etc. It will then update the global knowledge about devices in the ontology which in turn will be synchronised among other discovery managers on the LinkSmart through P2P (Peer to Peer) network so that all of them have the same knowledge about the available services. During development, this knowledge will be used by the application developers to find IoT services based on their needs.

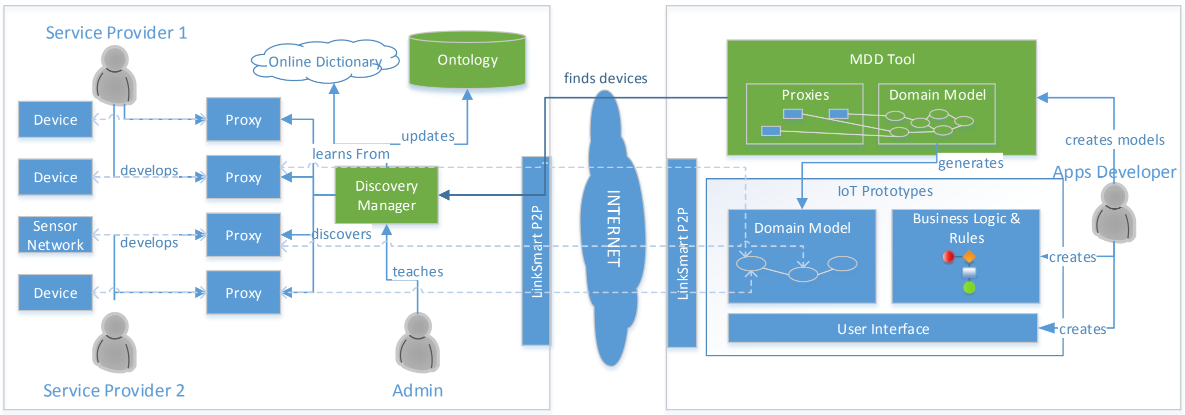
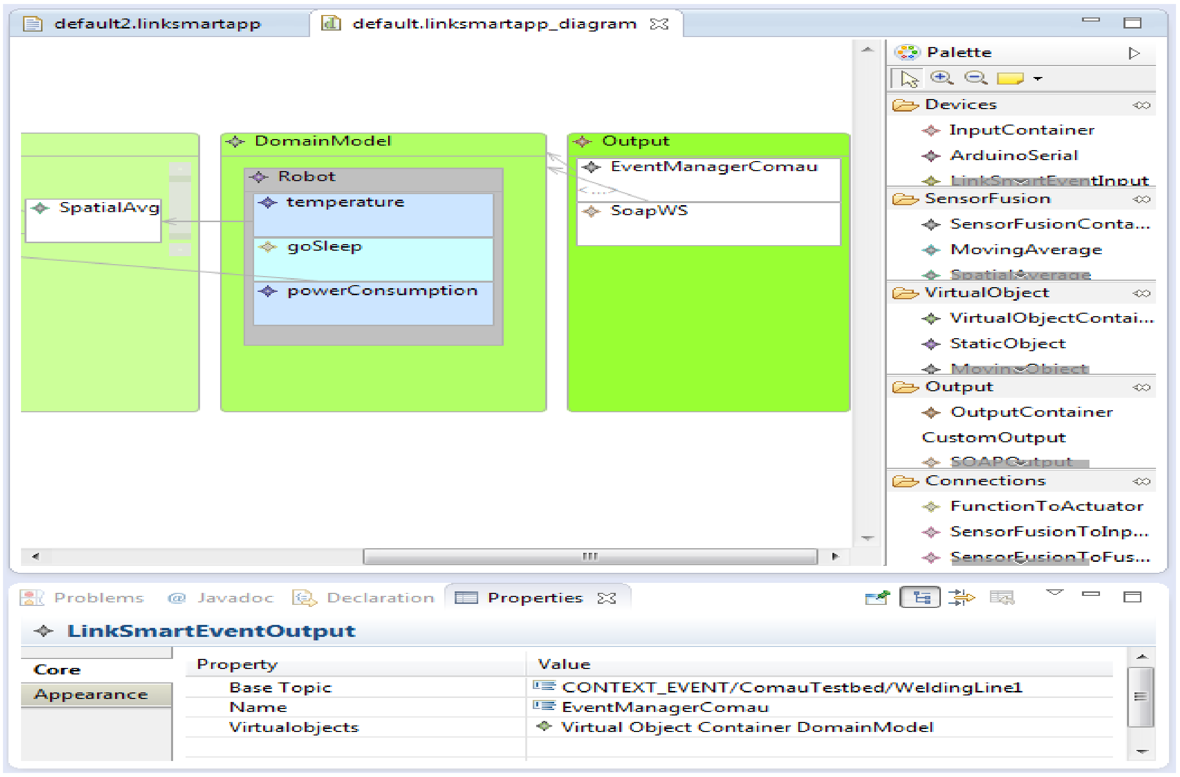


Figure 8: The envisioned prototyping toolkit for IoT

The discovery manager uses an online dictionary to detect different synonymous terms used by different service providers to describe their devices. The technique to handle synonyms is described in (Tsai et al., 2011). The synonyms and device classification structure is constructed in an ontology allowing developers to find devices based on semantic criteria e.g. subset of a device class, capabilities.

The semantic discovery will also be useful during runtime as the application might search for alternative services when the initially bounded service is not anymore available or when it performs below the quality threshold. Although quality of service is beyond the scope of this research, the discovery manager provides the necessary infrastructure for QoS mechanisms.

As depicted on the right side of Figure 9 the apps developer will use the Visual context modelling tool to create the application domain model and connect these domain concepts with the device proxies that have been discovered by the discovery manager. The developers could query the discovery manager to find device proxies according to their semantics such as capabilities and functions as well as their synonyms.



Property	Value
Base Topic	CONTEXT_EVENT/ComauTestbed/WeldingLine1
Name	EventManagerComau
Virtualobjects	Virtual Object Container DomainModel

Figure 9: Visual context modelling tool prototype

After the domain model and the correlation to the sensor is done by the developers, they also need to specify the information needed to configure the input and output streams e.g. if the data comes from the LinkSmart event manager. The developers are required to specify the description of the event manager and the topic of the event so that the generated code is able to find the event manager and subscribe to the corresponding events. They could also choose to publish a LinkSmart event when the properties of domain objects change by linking the domain model to the LinkSmart Event component on the output side.

As depicted in Figure 10, the information about the domain model and the events generated by the domain objects is stored in an XML Metadata Interchange (XMI) format which can be used by the business process engine to correlate the contextual events with the running business process. This will be described in more detail in the next section.

```
<?xml version="1.0" encoding="UTF-8"?>
<linksmartapp:App xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:linksmartapp="http://linksmartapp/1.0">
  <virtualObjectContainer name="DM" output="//@outputContainer.0/@outputs.0
//@outputContainer.0/@outputs.1">
    <virtualObjects xsi:type="linksmartapp:StaticObject" name="Robot">
      <properties name="temperature" fusedInput="//@sensorFusionContainer.0/@sensorfusions.2"/>
      <properties name="powerConsumption"
fusedInput="//@sensorFusionContainer.0/@sensorfusions.1"/>
      <functions name="sleep"/>
    </virtualObjects>
  </virtualObjectContainer>
  <sensorFusionContainer name="SensorFusion">
    <sensorfusions xsi:type="linksmartapp:MovingAverage" name="MovingAverage1"
inputs="//@inputContainer.0/@inputs.0"/>
    <sensorfusions xsi:type="linksmartapp:MovingAverage" name="MovingAverage2"
inputs="//@inputContainer.0/@inputs.1"
property="//@virtualObjectContainer.0/@virtualObjects.0/@properties.1"/>
    <sensorfusions xsi:type="linksmartapp:SpatialAverage" name="SpatialAverage1"
fusion="//@sensorFusionContainer.0/@sensorfusions.1 //@sensorFusionContainer.0/@sensorfusions.0"
property="//@virtualObjectContainer.0/@virtualObjects.0/@properties.0"/>
  </sensorFusionContainer>
  <inputContainer name="InputStreams">
    <inputs xsi:type="linksmartapp:LinkSmartEventInput" name="powerSensor1"
fusions="//@sensorFusionContainer.0/@sensorfusions.0"/>
    <inputs xsi:type="linksmartapp:LinkSmartEventInput" name="thermometer1"
fusions="//@sensorFusionContainer.0/@sensorfusions.1"/>
  </inputContainer>
  <outputContainer name="Output">
    <outputs xsi:type="linksmartapp:LinkSmartEventOutput" name="EventManager:Comau"
virtualobjects="//@virtualObjectContainer.0"/>
    <outputs xsi:type="linksmartapp:SOAPOutput" name="SoapWS"
virtualobjects="//@virtualObjectContainer.0"/>
  </outputContainer>
</linksmartapp:App>
```

Figure 10: XML Serialization of the domain model

3.4 Business-process-based data gathering

3.4.1 Basics

In a production environment such as an automobile manufacture, a lot of business processes can be found. These can include salary accounting, the supply of needed items or the assembly of a certain kind of car.

In ebbits one goal is the implementation of optimization metrics, including energy savings, in manufacturing processes. To implement these metrics first the energy consumption must be determined. To be able to optimize it is obviously not enough to know how much energy in total the assembly of a car consumed. There are different approaches to get a more fine-grained break down of the energy consumption. The consumption information of a machine given by the producer can be used, but it might not be clear if these values can be applied in the current environment and also the consumption might not be the same for every process run. For example a machine might need more energy with time as it requires some maintenance.

Another approach is to measure the consumption of each machine. While this is fine for certain types of data analysis, there might not be enough data for other types of analysis. To have a detailed analysis for one single process step, e.g. assembly of the right door of a car, might not be possible.

By using a business process-based approach to monitor measured values such as energy consumption, we enable a more detailed analysis. In this approach the business process is modelled in required detail and the measured sensor data is mapped to one of the modelled process steps. The data gained can be used to analyse single process steps or to compare a single process to itself over time or to another implementation of the same process in another production site to find out why one site is more efficient than another.

Another advantage of using a business process-based approach is that the processes have to be modeled and the resulting models are a good way to visualize, explain and understand the processes and make them known to employees or customers.

There are different ways to model business processes, for example EPC (Event-driven Process Chain), UML (Unified Modeling Language) (especially activity diagrams), Petri nets, and WSBPEL (Web Services Business Process Execution Language). They all have different disadvantages: EPC and Petri nets have problems with representing complex situations and EPC additionally has no standardization, UML is complex and might be hard to learn and is not often used in non-technical areas, whereas WSBPEL is xml-based and has no graphical representation.

Another possibility to model business processes is BPMN (Business Process Modeling and Notation). It was standardized by the Object Management Group (OMG) in order "to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes" and " to ensure that XML languages designed for the execution of business processes, such as WSBPEL […] can be visualised with a business-oriented notation". (BPMN 2.0, 2011)

3.4.2 Related work

(Estruch et al, 2012) present an approach called EDBPM (Event-Driven Business Process Management). They claim that there is a need in production for an automated detection and handling of particular situations. As events occur regularly in manufacturing they are used to detect these situations: some situations might be detectable through simple events, but there are also situations which can only be found via complex event detection. They model complex event patterns via BPMN 2.0 and make the models executable. These patterns are implemented using the occurrence of simple events and become part of the modeled production process as a sub process loop.

The architecture they propose for the EDBPM IT-solution is shown in Figure 11. There are three layers: a manufacturing connectivity layer to collect and transform events, the adapted BPM engine to execute BPMN-processes including an "enhanced time window event handling" for evaluating complex event processing correlation and "enhanced data processing tasks" for KPIs (Key Performance Indicator) evaluation and the user interface layer which visualizes KPIs and enables user interaction.

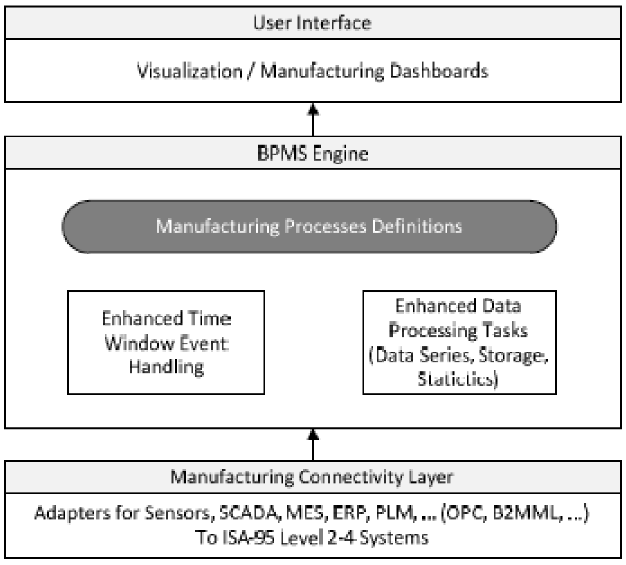

User Interface
Visualization / Manufacturing Dashboards
BPMS Engine
Manufacturing Processes Definitions
Enhanced Time Window Event Handling
Enhanced Data Processing Tasks (Data Series, Storage, Statictics)
Manufacturing Connectivity Layer
Adapters for Sensors, SCADA, MES, ERP, PLM, ... (OPC, B2MML, ...) To ISA-95 Level 2-4 Systems

Figure 11: EDMPM general architecture schema

In order to validate the approach the authors propose that "some manufacturing scenarios […] could be simulated into plant simulator software capable to interact with a BPM companion engine, generating and receiving events to/from deployed process models to carry out the manufacturing management logic including CEP correlation to identify new complex events and BAM (Business-Activity-Monitoring) measures update logic to update KPIs values."

This approach makes it necessary to have an extra layer with adapters for sensors, which are not directly mapped to process tasks. It is not the authors' goal to monitor the energy consumption for later analysis, but to immediately detect unusual events which need a particular handling.

(Ameling et al., 2010) describe demand-side energy management (DSEM). The main goals are to avoid peak demands in production and to flatten load shapes. To do so the energy supply and the energy consumption must be known. In the paper the authors assume that the (local) energy supply can be predicted, but the energy consumption must first be monitored in order to make adaptions in the production process when the energy supply and the energy consumption do not match.

Figure 12 depicts the architecture for the proposed system. The lowest layer provides the required data, e.g. measured data of smart meters for one device or data received from external services, either pushed to or pulled by the event bus which transforms all data into uniform events to which components of upper layers can subscribe to. The event bus can also filter events out, quality-check them and route events based on content.

The complex event processing (CEP) engine in the next layer analyzes processes and combines low level events in order to create complex events based on predefined rules. In the persistency layer events are persisted in a database. In the logic layer the analysis component is responsible for data mining, e.g. discovery of abnormalities such as malfunctioning.

The last layer is the presentation layer, which visualizes the energy consumption on different levels, e.g. on process, device and product level. The authors claim that this can later be connected to external systems, such as a process execution engine, a business process modeling tool, an enterprise resource planning system or a manufacturing execution system in order to give optimization advices to production planners and to enable a semi-automatic production planning.

Their current prototype focuses the device connectivity and the visualization of smart meter information.

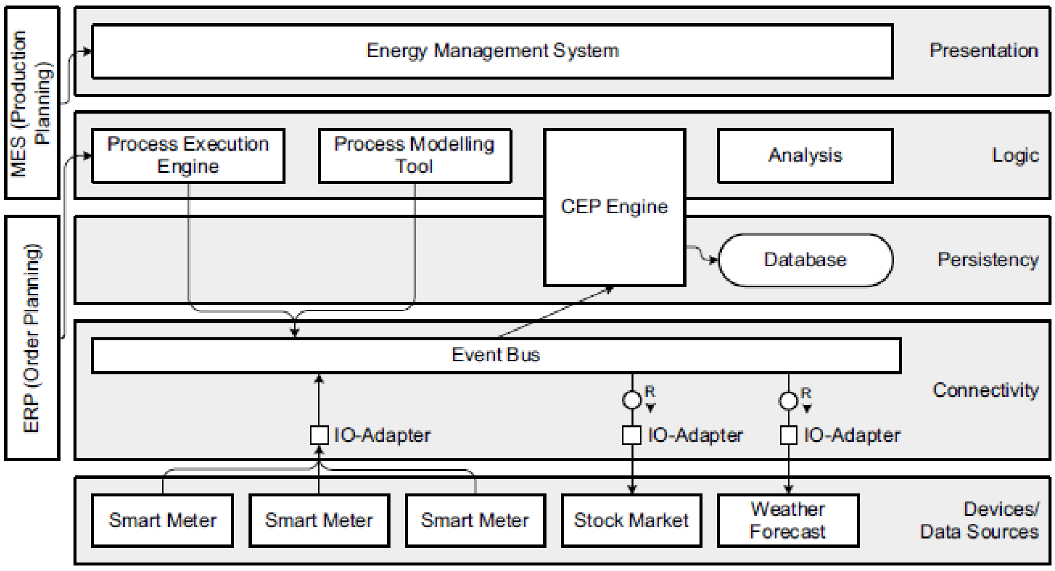


Figure 12: Architecture DSEM

As for now the event handling is done via a CEP Engine. To monitor energy consumption of a specific task, rules would have to be described in the CEP Engine. This is not a graphical tool and requires training.

There are other works like (Ammons et al, 2008) where Business Process Modeling and Execution is combined with CEP. Again the CEP is used for all the event handling and the Business Process Execution provides the CEP with events.

In (Ghose et al., 2009) the authors describe an approach to account and model an organization's carbon emission, more precisely they describe an approach "that helps assess carbon emissions from a process perspective, providing deeper insights into the role of tasks and processes in an organization's total emissions and revealing opportunities for future carbon-aware process re-engineering."

They use the accumulation of process annotations, i.e. carbon emissions in their example, through the process to a selected task. The goal is to model a process with BPMN and during process design, the carbon-footprint is calculated as to guide the analyst to the modeling process. This means that during the entire modeling process the emissions which would have been emitted had the process been executed up to the point of modeling are known.

The authors describe three scopes where greenhouse gases are emitted which can be expressed in carbon-dioxide equivalent: scope one are direct emissions which result directly from activities within the organization, such as combustion of fuel. Scope two emissions (indirect) include emissions which are externally emitted, but are brought within the organization such as electricity. The third scope includes indirect emissions which are not part of scope two, e.g. emissions from material.

Their tool for process design maps carbon-dioxide emission to design elements and calculates secondary footprint measures by propagating primary measures through a process model.

To achieve this, among other things, it is necessary to model carbon emissions related entities and their relationship to determine the emissions of a single task. The authors use a top-down and a bottom-up approach for this. The first one is based on the knowledge of the total consumption per period of time and the authors use principles of cost accounting to assign electricity consumption to shared resources and to tasks from a source such as an electricity bill or the measurements from a smart meter, so they do not have to determine all single electricity devices and their average consumption. In the bottom-up approach the consumption of single devices is considered in case they consume a considerable amount of electricity for which the average consumption is known or worth determining.

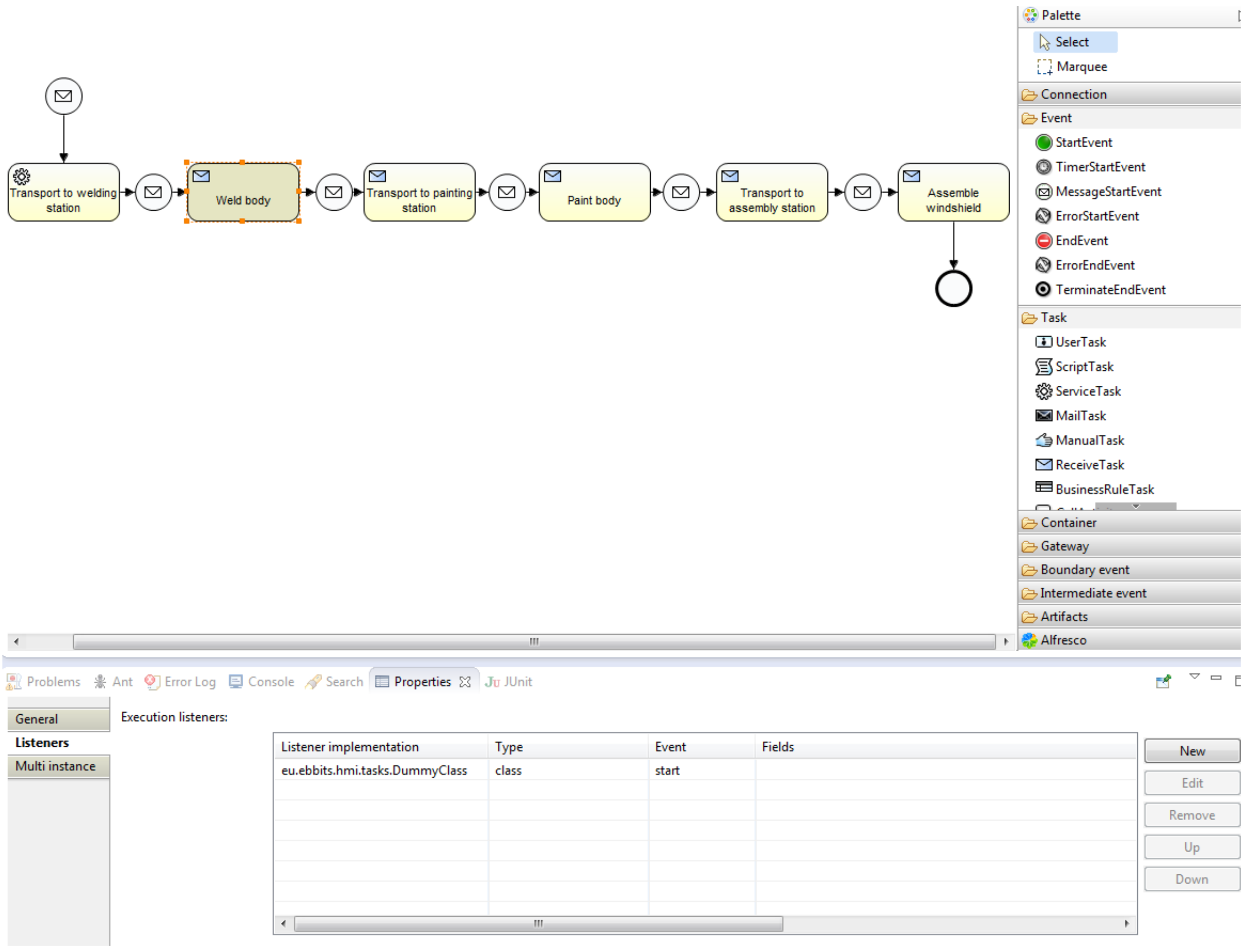
Although this approach maps energy consumption to single process tasks, it is meant for redesigning existing processes and not for monitoring. Values of current process executions are not included, information about the average consumption is used, which might not even be measured, but given by a manufacturer.

3.4.3 Monitoring

BPMN overcomes most of the disadvantages of the other modeling approaches. Additionally there are open-source engines available which provide the possibility to execute BPMN Models, such as the Activiti-engine (Activity, 2013) and an open-source modeling plugin for eclipse (as well provided by Activiti). Activiti is based on Java, XML, and common database technologies which provide the possibility of an easy integration of business systems such as SAP.

Therefore we will use BPMN and Activiti tools in ebbits in order to monitor sensor values like energy consumption. The process engine will receive events via the LinkSmart Event Manager. These events either contain information about sensed data or information about the beginning/end of a process step.

Figure 13 shows a simplified model of a production line in the Activiti Designer which is a plugin for eclipse. The model itself is stored in xml-format. It consists of a MessageStartEvent, a ServiceTask, several MessageCatchEvents and ReceiveTasks alternating and an EndEvent.



Execution listeners:

Listener implementation	Type	Event	Fields
eu.ebbits.hmi.tasks.DummyClass	class	start	

Figure 13: Activiti Designer in Eclipse

For each MessageStartEvent, MessageCatchEvent and ReceiveTask (which are all wait-states waiting for a message or signal to arrive), a message topic can be defined which ends the waiting state and makes the process continue. Events containing data about measured consumption will be enriched and stored in a database from which data can be analyzed.

For each task code can be implemented to help monitor the process. For example, it will be possible to define certain thresholds defining when a sensor value is too high or too low so that certain measures can be taken. The Java code could contain a check and in case the temperature is too high an email is sent to the responsible person.

It would also be possible to build some controlling components into the process (e.g. turn of the robot if it becomes too hot), but for most production environments this will not be desirable as controlling the production is complex and a controlling system is already in place.

3.4.4 Data Enrichment

Events stored in the database by the monitoring application will have additional information in comparison to the events sent via the LS EventManager. The data will contain information about the process instance they were measured in, the process step they belonged to as well as creation time in the database. In case there is already some processing during the execution of the business process (such as determining if a measured value is in the desired range) the results can be persisted as well.

The additional data gives the opportunity to make additional analyses based on processes or process steps. If a car manufacturer has more than one production site producing the same kind of car, he could compare the different process implementations. As customers attach more and more importance to products' ecological footprints, the data might proof helpful to give a breakdown of total consumption for a single car produced.

3.4.5 Tool support

We will use BPMN in combination with Activiti in ebbits to provide a tool for developers helping them implement IoT applications which include business-process based monitoring.

When there are many sensors the topic definition for each of the modeled MessageEvents/ ReceiveTasks would be very time-consuming and prone to error if the modeler had to define everything by hand. Therefore it should be possible to map the MessageEvents/ ReceiveTasks to the topics defined in the domain model.

The file with the XML serialization of the domain model (example shown in Figure 10) will be read by the new tool and the selection of a sensor (or even the selection of a robot) can be done via a menu. The corresponding event topic will be mapped to the selected wait-state, which means that when a process instance of the business process model runs, a wait state will only be left if a message comes with the mapped topic.

There might be several events coming which just update the measured value and which are not supposed to end a wait state. In the process depicted in Figure 13 for example, there might be updates in the "Weld body" process step every second, resulting in events being sent every second. These events might include the current energy consumption of the welding robots and should be persisted, but they should not end the process step. Therefore a second kind of mapping must be included which includes measurement events.

Figure 14 gives an overview of the components involved when developing a monitoring application.

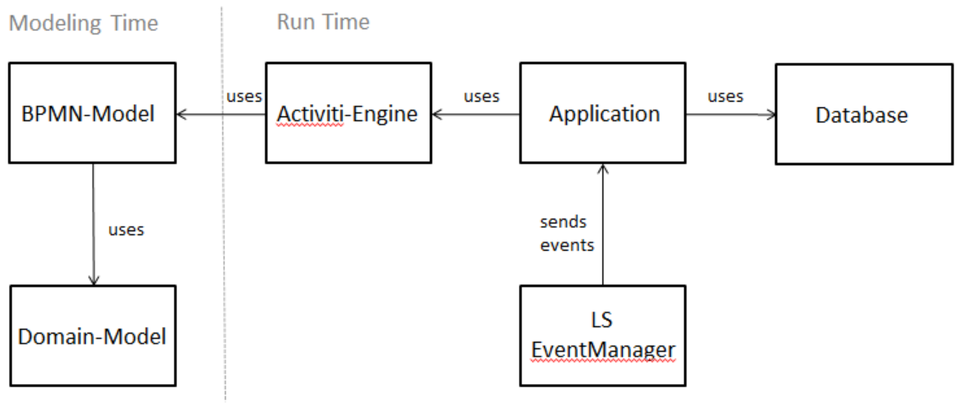


Figure 14: Overview components for tool support

4. Business intelligence and integration

4.1 Integrating Business Intelligence Data for Enterprise Information and Resource Planning Systems

The objectives for ebbits state that "the ebbits architecture contains distributed information systems on the one hand and integrated enterprise information and resource planning systems on the other hand". In this section of this deliverable, it is described how to integrate gathered data from the ebbits middleware into enterprise information and resource planning systems and to do business intelligence analysis on them. Thereby, we will focus on one of the use cases in ebbits, the manufacturing scenario.

This scenario is about automobile manufacturing and described in detail in (D9.4 and D10.4). For such scenarios, the most important enterprise systems are enterprise resource planning (ERP) and manufacturing execution systems (MES). Furthermore, the work processes within a company, which is assembling cars, are executed using orders. For production processes, production orders are used to control production within a company and also to control cost accounting. (SAP PP, 2013)

Such a production order specifies which material is to be processed, at which machine, at what time and how much work is required. This is also depicted in the upper part of Figure 15. Furthermore, such an order also postulates which resources are to be used and how the order costs are to be settled, which is represented graphically in the lower part of Figure 15. (SAP PP, 2013)

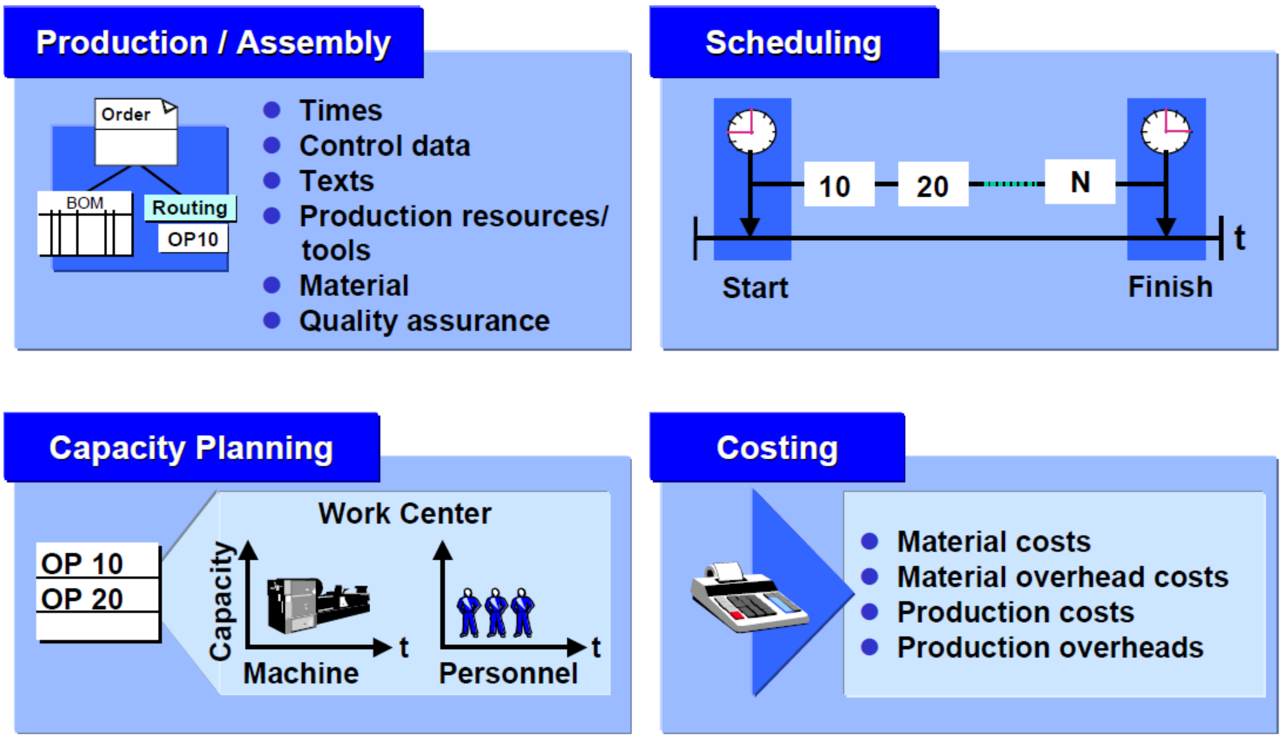


Figure 15: Parts of a Production Order, Source: (SAP PP, 2013)

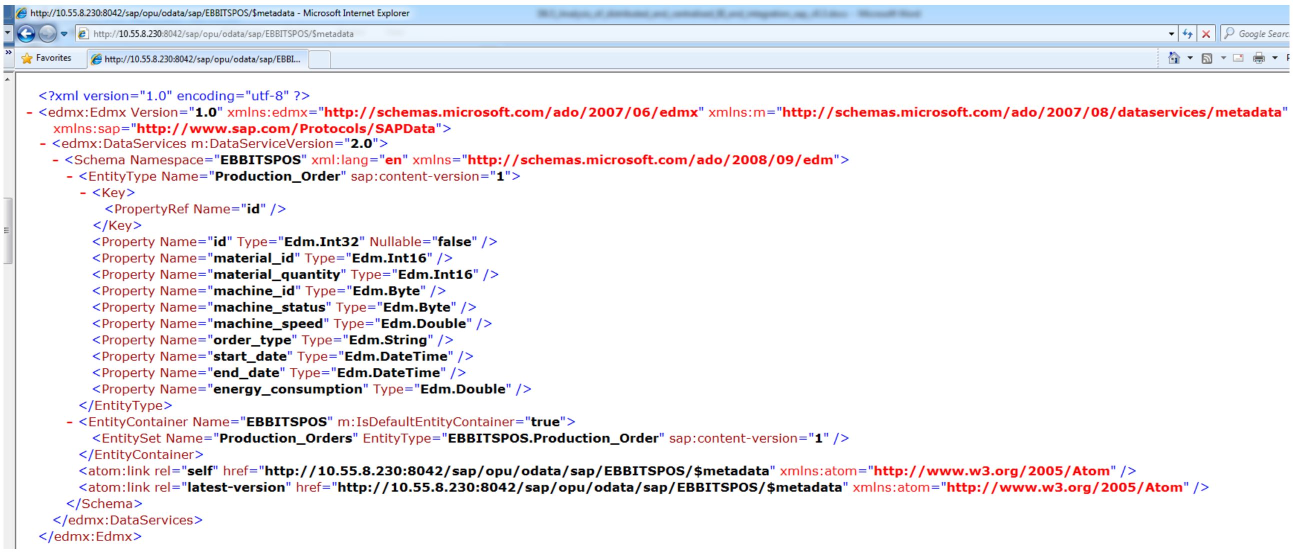
As can be seen from Figure 15 such an order is very complex as it covers a very broad range of different areas and could contain hundreds of value fields (for details the reader is referred to (SAP PP, 2013)). Furthermore not all values will be available at all time, for example during execution shop floor control adds order-relevant data to it to guarantee complete order processing. It depends very much on the business intelligence task at hand, which data should be measured, analysed or reported. On the same level, the integration with business intelligence data gathered and analysed by the ebbits distributed intelligence mechanisms will be subject to the tackled topic.

Due to the inherent complexity and for the easy of exposure, we will show the general integration mechanism for business intelligence data exchange and analyis for decision making involving both, ebbits distributed intelligence and traditional business intelligence mechanisms, by using very simplified examples.

For the data exchange of business intelligence data between the ebbits middleware and enterprise systems we propose to use SAP NetWeaver Gateway connectivity framework. This framework was already introduced in detail in deliverable D6.9 and therefore, only the summary is given here:

Gateway has a RESTful architecture and uses the Atom (Atom Syndication, 2005) and the OData (Open Data, 2013) standards. This type of architecture supports the usage of the standard HTTP GET, PUT, POST, and DELETE methods for server-side resource manipulation. Atom is a pair of standards: the Atom Syndication Format and the Atom Publishing Protocol. Atom is the de facto standard for treating groups of similar information snippets as it is simple, extensible, and allows anything textual in its content. The Atom standards do not specify how data should be encoded within such a group. However, as so much textual enterprise data is structured, there is also a requirement to express what structure to expect in an information snippet. Therefore, the used OData web protocol for querying and updating data extends Atom by defining a set of conventions for specifying structured data. Thereby, the goal of this protocol is to offer database like access to business data – "ODBC for the Web". The OData standard uses an underlying abstract data model called Entity Data Model (EDM) - which is employed by OData services to describe the structure and organization of its resources. OData is also extensible and one such described extension is OData for SAP applications, which contains SAP-specific metadata that helps the developer to consume SAP business data (D6.9, 2013).

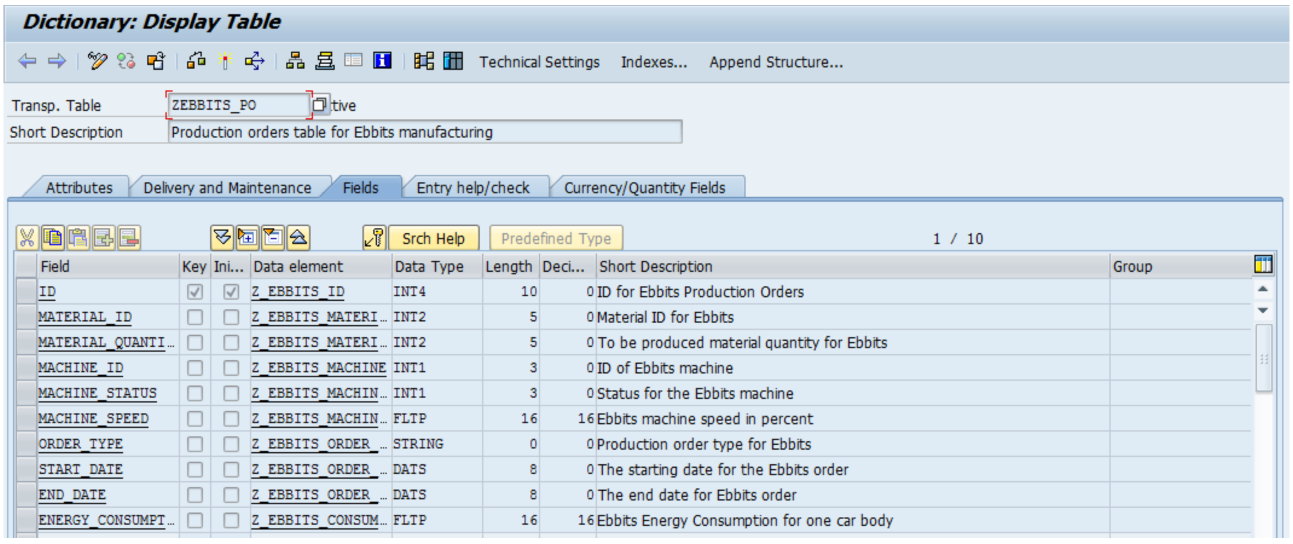
As an example for the contemplated manufacturing scenario, consider the meta-data of a Gateway service to exchange energy consumption data and error reporting depicted in Figure 16.


http://10.55.8.230:8042/sap/opu/odata/sap/EBBITSPOS/$metadata - Microsoft Internet Explorer
http://10.55.8.230:8042/sap/opu/odata/sap/EBBITSPOS/$metadata
Google Searc
Favorites
http://10.55.8.230:8042/sap/opu/odata/sap/EBBI...

```
  <?xml version="1.0" encoding="utf-8" ?>
- <edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns:sap="http://www.sap.com/Protocols/SAPData">
  - <edmx:DataServices m:DataServiceVersion="2.0">
    - <Schema Namespace="EBBITSPOS" xml:lang="en" xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
      - <EntityType Name="Production_Order" sap:content-version="1">
        - <Key>
            <PropertyRef Name="id" />
          </Key>
          <Property Name="id" Type="Edm.Int32" Nullable="false" />
          <Property Name="material_id" Type="Edm.Int16" />
          <Property Name="material_quantity" Type="Edm.Int16" />
          <Property Name="machine_id" Type="Edm.Byte" />
          <Property Name="machine_status" Type="Edm.Byte" />
          <Property Name="machine_speed" Type="Edm.Double" />
          <Property Name="order_type" Type="Edm.String" />
          <Property Name="start_date" Type="Edm.DateTime" />
          <Property Name="end_date" Type="Edm.DateTime" />
          <Property Name="energy_consumption" Type="Edm.Double" />
        </EntityType>
      - <EntityContainer Name="EBBITSPOS" m:IsDefaultEntityContainer="true">
          <EntitySet Name="Production_Orders" EntityType="EBBITSPOS.Production_Order" sap:content-version="1" />
        </EntityContainer>
        <atom:link rel="self" href="http://10.55.8.230:8042/sap/opu/odata/sap/EBBITSPOS/$metadata" xmlns:atom="http://www.w3.org/2005/Atom" />
        <atom:link rel="latest-version" href="http://10.55.8.230:8042/sap/opu/odata/sap/EBBITSPOS/$metadata" xmlns:atom="http://www.w3.org/2005/Atom" />
      </Schema>
    </edmx:DataServices>
  </edmx:Edmx>
```

Figure 16: The Meta-Data for the Ebbits Manufacturing BI Data Exchange Service

In Figure 16 one can see the most important parameters for this simplified example in the manufacturing scenario expressed in the OData format. The service is constructed by the SAP NetWeaver Gateway connectivity framework on the basis of the internal data structure as shown in Figure 17. As they are many clients available to access an OData service for consumption (see also (D6.9, 2013)) and for data exchange, the ebbits distributed intelligence could, for example, determine the overall energy consumption for one car body through retrieving the data for every process step and then report the overall sum to the enterprise system for further business intelligence analysis. Of course, the exchange is bidirectional, so for example, after determine with predictive analytics that a machine will need to undergo maintenance very soon, the enterprise system could tell the middleware to operate this machine only at half speed.



Transp. Table	ZEBBITS_PO tive
Short Description	Production orders table for Ebbits manufacturing

1 / 10

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description	Group
ID	✓	✓	Z_EBBITS_ID	INT4	10	0	ID for Ebbits Production Orders	
MATERIAL_ID	☐	☐	Z_EBBITS_MATERI...	INT2	5	0	Material ID for Ebbits	
MATERIAL_QUANTI...	☐	☐	Z_EBBITS_MATERI...	INT2	5	0	To be produced material quantity for Ebbits	
MACHINE_ID	☐	☐	Z_EBBITS_MACHINE	INT1	3	0	ID of Ebbits machine	
MACHINE_STATUS	☐	☐	Z_EBBITS_MACHIN...	INT1	3	0	Status for the Ebbits machine	
MACHINE_SPEED	☐	☐	Z_EBBITS_MACHIN...	FLTP	16	16	Ebbits machine speed in percent	
ORDER_TYPE	☐	☐	Z_EBBITS_ORDER_...	STRING	0	0	Production order type for Ebbits	
START_DATE	☐	☐	Z_EBBITS_ORDER_...	DATS	8	0	The starting date for the Ebbits order	
END_DATE	☐	☐	Z_EBBITS_ORDER_...	DATS	8	0	The end date for Ebbits order	
ENERGY_CONSUMPT...	☐	☐	Z_EBBITS_CONSUM...	FLTP	16	16	Ebbits Energy Consumption for one car body	

Figure 17: A possible BI data structure for the ebbits manufacturing scenario

The above described business intelligence data exchange at pre-determined time points may not be enough. If the business intelligence analysis on either side (distributed or enterprise) leads to results that call for an immediate decision, we further propose a rule based system to be able to define and execute business rules and to perform complex event processing. One such system is the Drools Business Logic integration Platform. It contains the modules expert as a rule engine and fusion for complex event processing. Drools was already introduced in deliverable 6.1 and the reader is referred to (D6.1, 2011) for a detailed description. Using Drools one could setup another monitoring and execution service which receives events from the business intelligence analysis of the distributed and the traditional business intelligence mechanisms. If for example, one or more customers in the manufacturing scenario reduce their request for car bodies, this information will turn up in the enterprise system and then the following rule could be used to save energy at the shop floor through forwarding it to the middleware.

```
declare QuantityReduced
        @role(event)
        @timestamp(timestamp)
end

rule "Reduce Speed to Save Energy"
        when
                $e1 : QuantityReduced(Material.amount=1);
        then

        ReduceSpeed(Machine.status=”reduced”,Machine.speed=50,$e1.getTimest
amp());
        end
```

In the other direction, the distributed intelligence may deduce from sensor readings that a machine is now broken. This has consequences for the business and the following rule would catch this event and update the end date of the order in the business system.

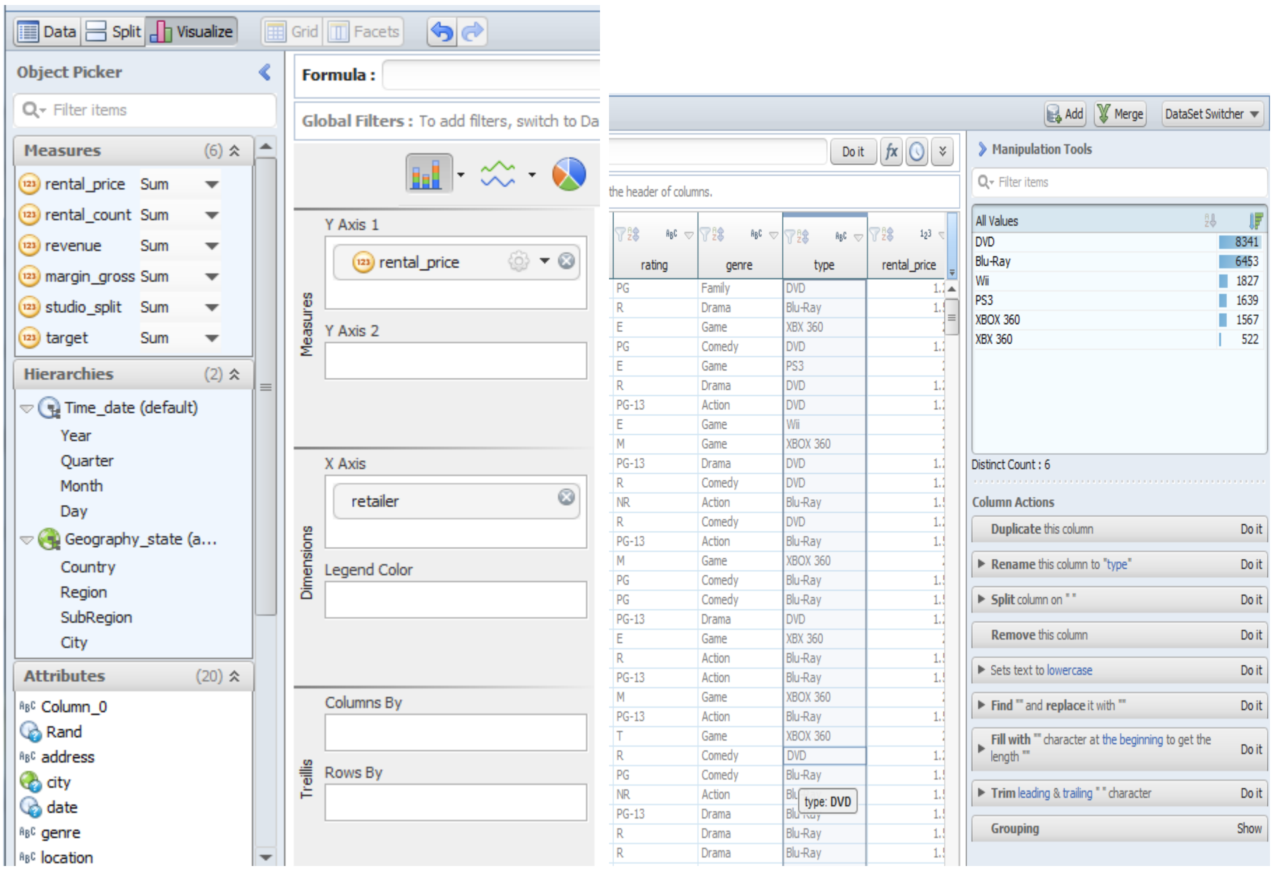
```
declare MachineBroken
        @role(event)
        @timestamp(timestamp)
end
```

```
rule "Postpone Order"
      when
            $e1 : MachineBroken(Machine.status=”broken”);
      then
            Order.enddate = Order.enddate + 2;
            PostponeOrder(Order.enddate,$e1.getTimestamp());
      end
```

4.2 Self-Service Business Intelligence with SAP Lumira

The ebbits middleware is envisioned to be deployed and used in a variety of scenarios and use cases and taking into account very different end-users. These end-users, for example, farmers in the traceability scenario, do not necessarily have the same technical knowledge as Business Analysts or can rely on an IT department to do scripting for them. Instead, they want an easy-to-use tool that delivers insight using a self-service approach without the need for IT to create predefined or custom queries and reports. A "point and click" solution to manipulate, organize and consolidate data and answer business intelligence questions in a visual way results in the quick discovery of valuable hidden insights. One such solution is SAP Lumira which has the following features to prepare and visualize/navigate business intelligence data (SAP Lumira Brief, 2013):

- Enriching the data with geographic and time hierarchies
- Automatically creating useful measures based on existing data
- Formatting and cleaning existing data (for example, duplicating, grouping, and trimming)
- Merging data sets based on a common attribute or key
- Adding sophisticated formulas without any coding
- Surface patterns and relationships by selecting chart types suited for your analysis (for example, bar, radar, surface, tag cloud, bubble, and scatter charts)
- Identify trends with line and multiline charts
- Show the percentage of parts to a whole with pie charts and tree maps
- See the big picture at a glance and drill down into the details using filtering data in columns, facets, and charts



rating	genre	type	rental_price
PG	Family	DVD	1.
R	Drama	Blu-Ray	1.
E	Game	XBX 360	
PG	Comedy	DVD	1.
E	Game	PS3	
R	Drama	DVD	1.
PG-13	Action	DVD	1.
E	Game	Wii	
M	Game	XBOX 360	
PG-13	Drama	DVD	1.
R	Comedy	DVD	1.
NR	Action	Blu-Ray	1.
R	Comedy	DVD	1.
PG-13	Action	Blu-Ray	1.
M	Game	XBOX 360	
PG	Comedy	Blu-Ray	1.
PG	Comedy	Blu-Ray	1.
PG-13	Drama	DVD	1.
E	Game	XBX 360	
R	Action	Blu-Ray	1.
PG-13	Action	Blu-Ray	1.
M	Game	XBOX 360	
PG-13	Action	Blu-Ray	1.
T	Game	XBOX 360	
R	Comedy	DVD	1.
PG	Comedy	Blu-Ray	1.
NR	Action	Bl	1.
PG-13	Drama	Blu	1.
R	Drama	Blu-Ray	1.
R	Drama	Blu-Ray	1.

All Values	
DVD	8341
Blu-Ray	6453
Wii	1827
PS3	1639
XBOX 360	1567
XBX 360	522

Distinct Count : 6

Figure 18: Drag-and-Drop to Filter, Transform and Visualize BI Data, Source: (SAP Lumira, 2012)

In Figure 18 some of the capabilities of SAP Lumira are depicted. On the left side of this figure, it is shown how to simply use the mouse to select between different measures and various types of charts. On the right side, the different possibilities to pre-process the data are portrayed. As all these possibilities can be executed by a mere drag-and-drop and visualised instantaneously, SAP Lumira facilitates an iterative thought process with a quick navigation of the results of the analysis and adaption of the queries.

For the considered manufacturing scenario, one could imagine, for example, that the manager of the production plant wants to have a quick overview over the energy consumption in relation with the quantity produced and dependent on a specific machine out of a set of similar machines and for different order types. This business data is exposed by the enterprise system via the SAP NetWeaver Gateway connection framework and could be loaded into SAP Lumira. The manager could then do some data cleansing if necessary and a selection of the required parameters. The following visualization in SAP Lumira is now very easy. The manager just has to define energy consumption and quantity as a so-called measure and drag-and-drop them to the Y-axis for a bar chart. Furthermore, machine id and order type are added to the X-Axis. The result is shown in Figure 19 where for this analysis mock-up data was used. The visualisation may now be used by the manager to quickly determine which parts of the car consume the most energy and which machine is the most energy-efficient. This analysis could furthermore lead to the appropriate concentration of efforts for energy-savings with respect to car types and machines. More complex business intelligence analysis and drill-downs for different business intelligence questions could also be quickly visualised with SAP Lumira.

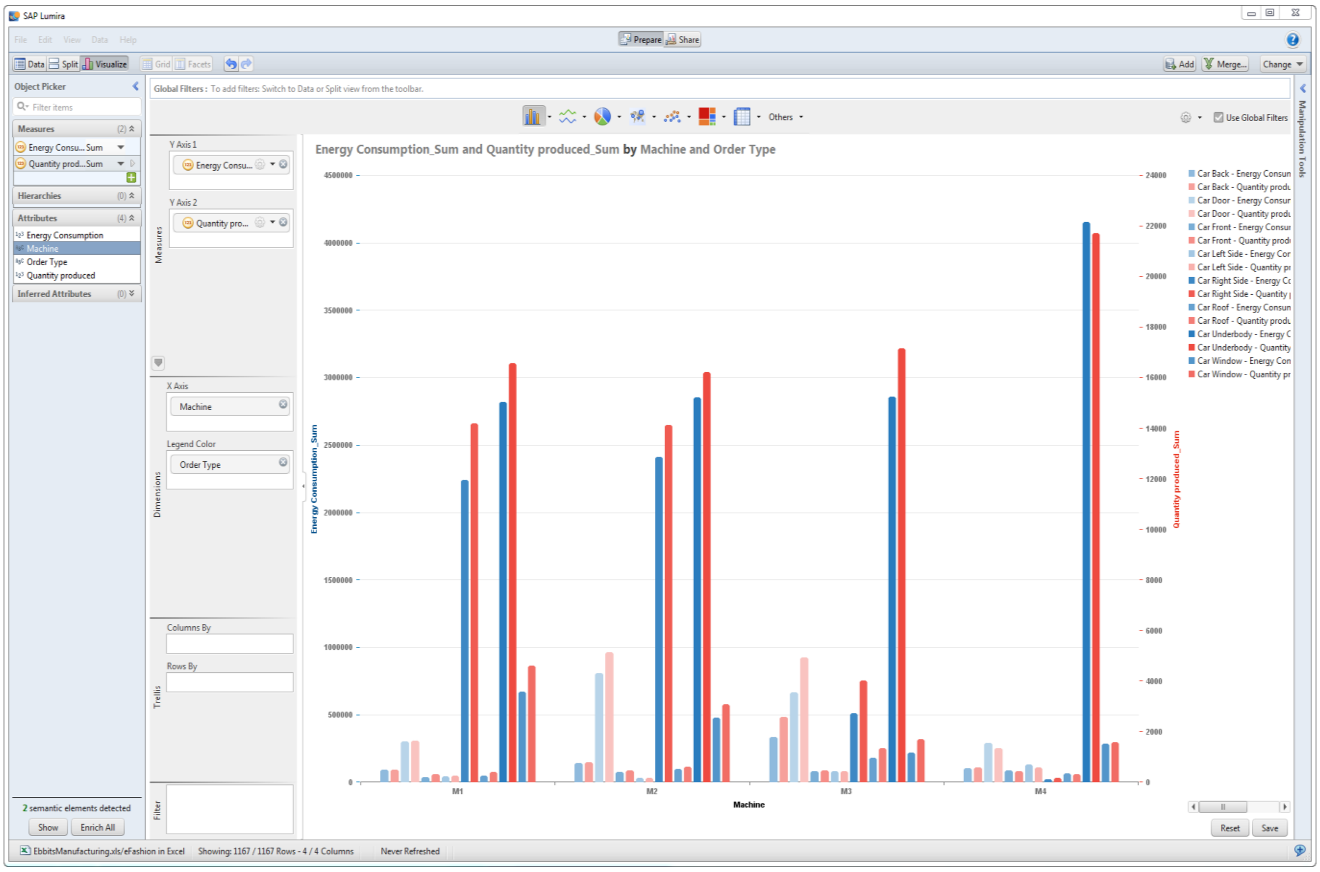


Figure 19: Energy Consumption and Quantity Produced in Dependence of Machine and Order Type

5. Conclusion

In this deliverable we have shown our approach on how to integrate sensors and devices in a distributed and centralized system. We use an interface layer to connect any kind of device to the ebbits platform and allow for semantically describe the entities involved. We also semantically enrich the gathered data in order to allow for complex event.

A domain model is the basis for processing events and generating a context from which additional information can be derived. We plan to implement a tool that supports the development of such IoT applications that combine a domain model with the actual sensors and therefore help developers.

An innovative aspect is the business-process-based monitoring and data gathering that allows for new ways to analyse data and in combination with ERP and MES systems to optimize production processes. We also plan to develop a tool for IoT developers to combine the business process steps within a BPMN model with the domain model and the sensors connected to this model. This would make it much easier to develop such complex applications. Also new ways of presenting the production monitoring data to various stakeholders are possible.

The previously mentioned integration of ERP and MES systems was also shown with SAP systems as an example. Such systems as SAP Lumira allow for non-technical persons to visualize the data gathered by the ebbits platform without the need for IT support. By the ability of the ebbits platform to export gathered data in standard formats like Atom or OData the connection to enterprise applications can easily be accomplished and further processing in such systems is possible. We plan to integrate such enterprise systems into the demonstrators for a distributed system within the traceability scenario.

6. References

(Activity, 2013) Activiti website, http://www.activiti.org/ [Accessed August 2013]

(Ameling et al., 2010) Ameling, M., Schief, M., Nietzold, F., Kuhn, C., "Demand-side Energy Management Systems for Manufacturing", SAP Research, *GI Jahrestagung (1)*, pp. 467-472.

(Ammon et al. 2008) Ammon, R., Emmersberger, C., Springer, F., Wolff, C., "Event-Driven Business Process Management and its Practical Application Taking the Example of DHL", In *Future Internet Symposium, Vienna, 2008*

(Atom Syndication, 2005) Atom Syndication Format website, http://tools.ietf.org/html/rfc4287 [Accessed August 2013]

(BPMN 2.0, 2011) Business Process Model and Notation (BPMN), Specification Version 2.0, January 2011, http://www.omg.org/spec/BPMN/2.0

(D4.M36, 2013) D4.M36 Semantic Mechanisms and Models, EBBITS consortium, Aug. 2013

(D5.6.1, 2013) D5.6.1 Context awareness prototypes 3, EBBITS consortium, 2013.

(D6.1, 2011) D6.1 Design of business rules representation and execution, EBBITS consortium, 2011

(D6.9, 2013) D6.9 Analysis, design and implementation of interconnection and integration services, EBBITS consortium, 2013

(D7.3.2, 2012) D7.3.2 Implementation of Data and Event Management models, EBBITS consortium, 2012

(D7.6.1, 2013) D7.6.1 Technical description and prototypes of data and event management subsystems 1, EBBITS consortium, 2013

(D8.8.1, 2012) D8.8.1 ebbits network management & security framework 1, EBBITS consortium, June 2012

(D8.8.2, 2013) D8.8.2 ebbits network management & security framework 2, EBBITS consortium, June 2013

(D9.5, 2013) D9.5 Integrated platform prototype with main focus on traceability, EBBITS consortium, June 2013

(Estruch et al., 2012)Estruch, A., Heredia Álvaro, J. A., "Event-Driven Manufacturing Process Management Approach", Business Process Management, Proceedings of 10th International Conference, BPM 2012, pp.120-133

(Ghose et al., 2009) Ghose, A., Hoesch-Klohe, K., Hinsche, L. Le, L., "Green Business Process Management: A Research Agenda", Australasian Journal of Information Systems Volume 16 Number 2 2009, pp.103-117 Aditya Ghose, Konstantin, Lothar and Lam-Son Le

(IERC-AC2 D1, 2013) IERC Activity Chain 2 Deliverable D1 - Catalogue of IoT Naming, Addressing and Discovery Schemes in IERC Projects, IERC Activity Chain Naming, addressing, search, discovery, Jan. 2013

(IoT-A D2.1, 2012) Project Deliverable D2.1 – Resource Description Specification, The Internet of Things Architecture project, FP7 257521, March 2012.

(IoT-A D4.1, 2011) Project Deliverable D4.1 – Concepts and Solutions for Identification and Lookup of IoT Resources, The Internet of Things Architecture project, FP7 257521, Dec. 2011.

(Kestrel, 2013) Kestrel Message Queue, http://robey.github.io/kestrel/ [Accessed August 2013]

(Open Data, 2013) Open Data Protocol website, http://www.odata.org/ [Accessed August 2013]

(SAP Lumira Brief, 2013) SAP AG, "SAP Lumira Solution Brief," 2013. [Online]. Available: http://www.saphana.com/docs/DOC-3592 [Accessed August 2013].

(SAP PP, 2013) SAP AG, "PP - Production Orders," 2000. [Online]. Available: http://help.sap.com/printdocu/core/Print46b/Data/EN/PPSFC.pdf [Accessed August 2013]

(SAP Lumira, 2012) SAP AG, "SAP Lumira Presentation", 2012.

(Storm, 2013) Storm website, http://storm-project.net/ [Accessed August 2013]

(Tsai et al., 2011) Tsai, D., Yushi Jing ; Yi Liu ; Rowley, H.A. ; Ioffe, S. ; Rehg, J.M., "Large-scale image annotation using visual synset", Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE.

(WS-Discovery, 2009) http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01 [Accessed August 2013]

(Xively, 2013) Xively website, https://xively.com [Accessed August 2013]