Integration of WSNs into Enterprise Systems based on Semantic Physical Business Entities

Matthias Thoma*† Klaus Sperner*, Torsten Braun † Carsten Magerkurth*,
*SAP (Switzerland) Inc., Althardstrasse 80, 8105 Regensdorf, Switzerland
matthias.thoma@sap.com, klaus.sperner@sap.com, braun@iam.unibe.ch, carsten.magerkurth@sap.com
†Communication and Distributed Systems, University of Bern, Neubrückstrasse 10, 3012 Bern, Switzerland

Abstract—**Internet of Things based systems are anticipated to gain widespread use in industrial applications. Standardization efforts, like 6LoWPAN and the Constrained Application Protocol (CoAP) have made the integration of wireless sensor nodes possible using Internet technology and web-like access to data (RESTful service access). While there are still some open issues, the interoperability problem in the lower layers can now be considered solved from an enterprise software vendors' point of view. One possible next step towards integration of real-world objects into enterprise systems and solving the corresponding interoperability problems at higher levels is to use semantic web technologies. We introduce an abstraction of real-world objects, called Semantic Physical Business Entities (SPBE), using Linked Data principles. We show that this abstraction nicely fits into enterprise systems, as SPBEs allow a business object centric view on real-world objects, instead of a pure device centric view. The interdependencies between how currently services in an enterprise system are used and how this can be done in a semantic real-world aware enterprise system are outlined, arguing for the need of semantic services and semantic knowledge repositories. We introduce a lightweight query language, which we use to perform a quantitative analysis of our approach to demonstrate its feasibility.**

I. Introduction

In the enterprise community real-world aware business models and software innovations, e. g. as part of the so-called Sensing Enterprise, gained a lot of momentum recently. One of the main obstacles in enterprise integration is still the gap between the specialized knowledge needed to program the sensor nodes and add them to an enterprise backend system, and the (business) model driven way enterprise software is written and customized. In IoT research, one of the main observations is that people and thus many business processes are interested in managing the things and the properties of the physical objects. They are not at all interested in sensor readings. Our approach decouples the actual sensing devices (commonly referred to as motes) from the things in our business systems, which we call Semantic Physical Business Entities (SPBE). These SPBEs are described in a semantic service description language and change the programming and integration model from pure hardware centric to a service centric one. On the mote level two recent developments lead to more widestream adoption in industry: (i) 6LoWPAN [1], standardization towards IP-based protocols even on a sensor network level and (ii) development of lightweight application-level protocols like CoAP [2].

Our framework is based on the linked data principle, as suggested by Berners-Lee [3]. The general idea of linked data is to create typed links between data from different sources and make them accessible through standardized technologies. Linked data is described in a machine-readable way with an explicitly defined meaning.

Explicit meaning is achieved by using ontologies; for example for connecting the motes with the SPBEs we use the semantic sensor network ontology (SSN) or industry-specific ontologies. They are part of the enterprise systems' knowledge base and thus are, at least, reusable within one Enterprise Resouce Planning (ERP) domain. The real world objects, or to be more precise the properties sensed about them, will be described by a URI (endpoint). This decouples the real world objects from the sensing hardware, which is a big plus in an enterprise SOA environment. The endpoint itself is a RESTful interface and is described in a semantic service description language that makes it semantically exploitable by the SOA platform.

In previous work an entitiy notation for the Business Process Modelling Notation (BPMN) was introduced [4]. This work focuses on the conceptual embedding and the actual implementation on the lower levels. The SPBE abstraction can easily be used within enterprise systems with standard semantic web technologies. In the second part of the paper we concentrate on how SPBEs can be used to interact with motes. Our contributions in this context are: (i) Proposing SPBEs as an abstraction to represent physical objects in enterprise systems (ii) relate SPBEs to existing enterprise system and show a path for seamless integration (iii) proposing an integration platform taking several semantic repositories into account (iv) propose a simple SQL-like language to construct RESTful service endpoints that allow querying the properties of the SBPEs without having to take care of specific sensing devices. This can be seen as a semantic counterpart to sensor network database approaches.

The remainder of this paper is structured as follows: After relating our work to the literature in Section II, we introduce semantic services (section III). In section IV we introduce SPBEs. An integration strategy into existing enterprise systems is discussed in section V. We describe a custom query language for specifying SPBEs and their deployment to motes. Implementation details of our prototype solution are given in section VI. We then show the feasibility of our approach by presenting evaluation results.

II. Related Work

The idea of using services as an abstraction layer for physical objects, the so called entity services, is described in [5] as part of service taxonomy for the Internet of Things. We enhance these ideas, present an actual implementation and embed it into an enterprise framework. Similar concepts of abstracting physical objects and annotating them with semantic web technologies have been presented by Haseman et al. [6], following an approach where all the aggregation is done by the backend. Their system always has a complete view on the state of the system and aggregates the sensor data connected to their entities. In contrast we support processing on the actual sensor nodes beyond a simple sense and send approach. Furthermore, we show a path for actual enterprise integration of sensor devices. The use of linked open data for making sensor readings available has been proposed by the sense2web project [7], where the main objective was to make the sensor readings web-accessible.

The query language can be seen as a semantic version of sensor network databases: With TinyDB [8] we share the SQL-like syntax, but otherwise differ in many ways. Our language creates services as well as service endpoints and establishes connection between sensors and SPBEs, while sensor network databases mimic traditional databases and their main purpose is the direct retrieval of data from sensor networks.

III. Semantic Services

A. Introduction

Services, in particular Service Oriented Architectures (SOA), play a major role in nowadays enterprises. Traditionally, service descriptions are stored in repositories, which serve, for example WSDL descriptions. Inputs, outputs and effects or effected entities are not explicitly stated, only in datatypes and naming conventions. The objective of semantic services is to go one step further and make all this implicit knowledge explicit.

B. Linked USDL

Linked USDL (L-USDL: Linked Unified Service Description Language) is the semantic successor of the original USDL language, which was based on XML schemas. L-USDL is based on semantic web technologies (RDF/OWL) and follows Linked Data principles. L-USDL aims towards developing a standard vocabulary to represent services. As services can cover a wide range of different domains, explicit ontological links (which is the origin of the name Linked USDL) to other domain specific ontologies have to be created.

The objectives leading to the design of L-USDL were [9]:

1) Retain the necessary simplicity for computation as well as for modeling purposes.
2) Reuse existing vocabularies to maximize the compatibility of related systems, reusability of previously modeled data, and reduce engineering efforts.
3) Provide a simple yet effective means for publishing and interlinking distributed data for automatic computer processing.

In this work we are using the latest development version of L-USDL that is available. Figure 1 shows the essential parts of L-USDL.

The main entry point to L-USDL is usdl:Service. It describes a service in a way, so that it can act as an interface between the service provider and the service consumer. As a service is more than just a technical interface, the service description also describes functional as well as non-functional properties. The non-functional properties include qualitative and quantitative values, like the precision of an actor or the accuracy of a sensor. The pure functional/technical properties are described by the interaction protocol (interaction points).

An usdl:InteractionPoint is an actual step to be performed when accessing the service, like calling a CoAP based REST interface. Classes of services can be modeled with ServiceModel, sharing common characteristics and/or properties. Furthermore, usdl:ServiceOffering is shown as an example for a connection to the business side: Service offerings can define a price, terms and conditions, and service level agreements (SLAs).

C. Service Result Data Representation

While our approach is generally capable to deliver data in an arbitary format, we concentrate on the use of RDF for the results of the service calls in this work.

IV. From Business Objects to Semantic Physical Business Entities

A. Introduction

In this section we present the concept of Semantic Physical Business Entities (SPBE), which, together with Semantic Service Descriptions, are the foundation of our integration platform. The core idea is to decouple business objects, as tracked in enterprise systems, from the actual sensors monitoring them. For an enterprise SOA user this service based approach removes several layers of indirection. It enables direct access to the properties of the monitored business objects.

We first elaborate on the need for semantics in enterprise systems, and continue with introducing SPBEs in more detail. We conclude this section by arguing for the need of domain specific ontologies.

B. Semantics in Enterprise Systems

Semantics and even ontologies do already exist in nowadays enterprise IT systems, nonetheless they are often hidden and not explicitely stated as such, like the semantic web movement does. To enable interoperability between different components of an enterprise system, often a common vocabulary is used. The specific modules are then based on such common datatypes. For instance, SAP ERP systems use a Global Data Type (GDT) directory that represents business related content SAP wide. All elements of SOA services provided by SAP systems are described (typed) by GDTs. A GDT is more than just an integer or a float, but a GDT might be something rather generic like an invoice or something very domain-specific like AirCargoSecurityStatusCode. This implicit knowledge, the GDT directory, currently specifies more than 5100 different data types and is documented on more than 16000 pages.

C. Semantic Physical Business Entities

We now introduce the concept of Semantic Physical Business Entities, which in conjunction with ontologies will make

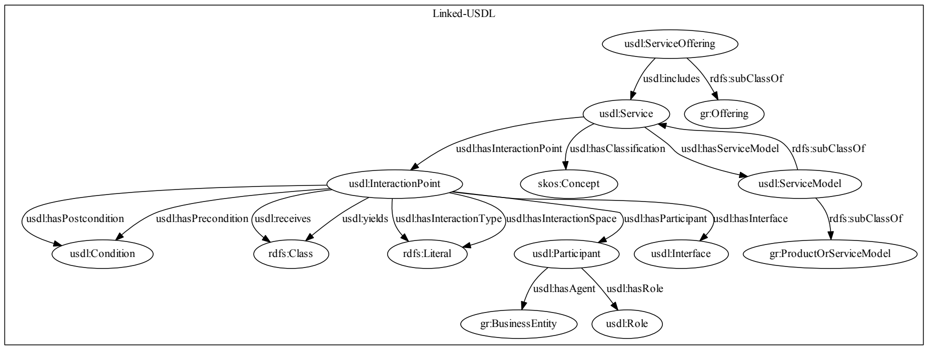


Fig. 1: Linked Unified Service Description Language (Linked USDL) (Excerpt)

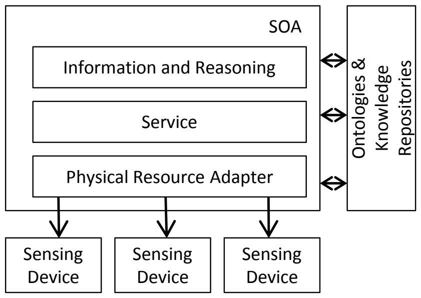

SOA
Information and Reasoning
Service
Physical Resource Adapter
Ontologies & Knowledge Repositories
Sensing Device
Sensing Device
Sensing Device

Fig. 2: Layered Architecture

this implicit knowledge explicit and directly accessible. First, we define the term Physical Business Entity (PBE):

> A *Physical Business Entity* is a conceptual representation of a real-world object processed by one or more enterprise IT systems.

At this point we limited ourselves to *physical objects* in the real-world, sometimes also called *entities of interest*. Nonetheless, our approach is generalizable to other kinds of entities, that can be observed by all kinds of sensors, e. g. including social networks.

In the literature there is no common agreement on the definition of the terms business object and business entity. Some authors use them interchangeably, others define business objects as entities with logic. As we want to emphasize the relationship with the entity concept as found in the IoT, we choose the term semantic physical business entity, which we define as follows:

> A *Semantic Physical Business Entity* is a conceptual representation of a real-world object processed by one or more enterprise IT systems. Information about it is discoverable. It is described through well defined vocabularies, that make internal and external relationships explicit.

The decoupling between the PBEs and the devices observing it is important: An enterprise IT system's user is usually not interested in the value of sensor no. 0815 or sensor no. 4711, but in the temperature of some given good or class of goods, which could be monitored by several sensors. This abstraction, moving away from the pure technical view concentrating on sensing devices, towards the "things" they monitor is one of the main ideas in the IoT community. In section V we show how our platform supports this abstraction.

D. Domain Specific Ontologies

It is important to note that only using semantic web technologies does not make a system *smart*. Data in RDF format is still machine-readable data only, which needs to be interpreted by reasoning algorithms. One very important prerequisite for smart systems are domain specific ontologies: This means the definition of common vocabularies and their relationships describing the actual domain. Even further, domain specific ontologies are not enough. For solving the semantic interoperability problem between different systems, we need *common* or *standardized* ontologies, or at least a mapping between ontologies of the same domain must be possible. For example we can have one ontology specifying: A banana is a fruit, and a fruit is a perishable good. A second one could specify banana as a fruto (Spanish for fruit) introducing an interoperability problem.

V. Integration of Semantic Services and Semantic Physical Business Entities into ERP Systems

A. Introduction

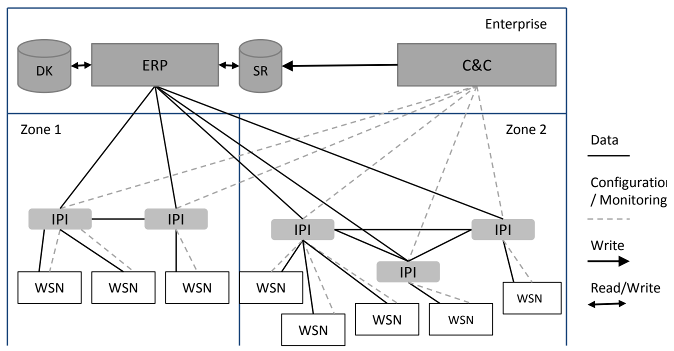
Considering interoperability as it is currently done in industry, one can observe that almost always a device-centric approach is used: Specific modules within the enterprise system interact with the sensing devices through proprietary protocols. Very recently the situation has changed towards IP-based protocols and standardized application level protocols.

One of the main challenges of innovation for an enterprise software vendor with a huge user base is to cope with the myriad of already existing code. Vendors aim towards innovation that has a clear integration path into already existing systems, even for more disruptive technologies. As most enterprise systems already use service repositories as an integrated part of their SOA environment, the integration of semantic service descriptions would not change the paradigm of how software is written today. Semantic Services thus could be added to enterprise software in an incremental manner without the need of disruptional changes. As explained in section IV-B there is already a lot of semantics in enterprise systems. The GDT in SAP systems can be seen as some kind of ontology. What is needed is to make the implicitly encoded knowledge more explicit. This would not introduce significant changes to nowadays systems.

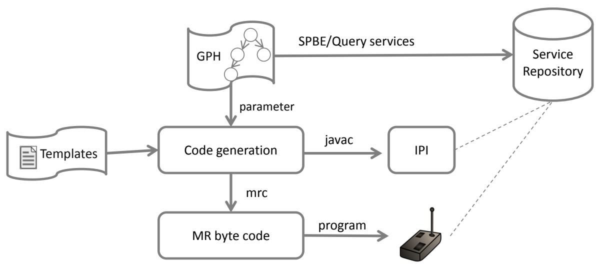
Business Objects are already stored in all kinds of data stores. Here the introduction of identifiers addressing these stores are necessary. Already existing data can be made available through service interfaces in a semantic web way. This would not introduce any changes to existing code. Access to all sorts of semantic entities is accomplished through semantically described services as introduced in this work.

B. Integration Platform

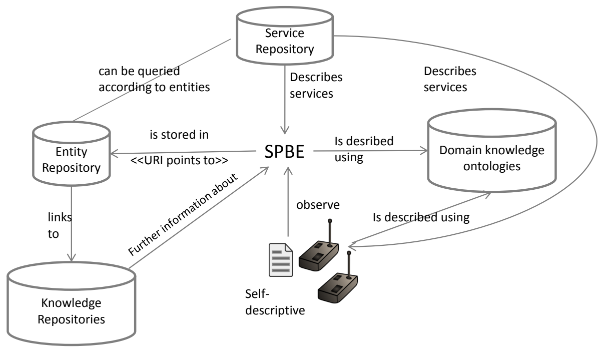
Our Integration Platform is shown in Figure 3. Specialized physical resource adapters are responsible for the communication between the motes (Figure 2) and an specific instance of the integration platform (IPI). The only requirement we have towards the motes is, that the IPI is notified when motes join or leave the WSN. Optionally, the motes can have a service description stored on them. This is necessary when third-party



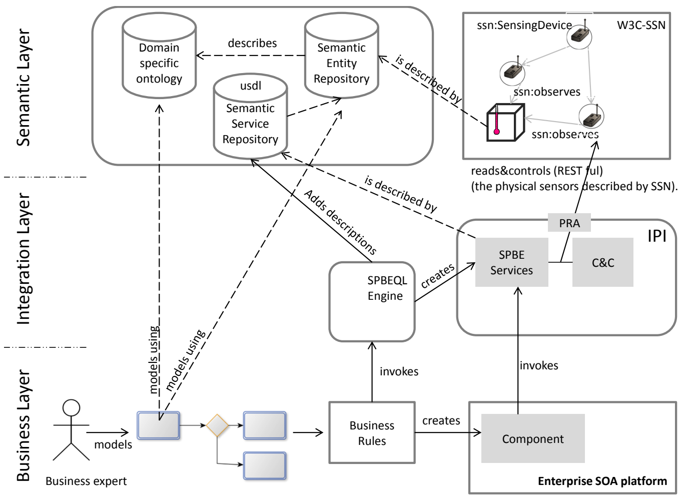
(a) Deployment view



(b) Code generation and service access



(c) Knowledge Repositories and Distribution



(d) SPBE in an enterprise framework

Fig. 3: Semantic Enterprise Integration Platform

smart objects (SPBE with a sensor attached) join the domain of an enterprise. Information from the motes is then used in the information and reasoning parts of the system, supported by ontologies and domain knowledge. The dependencies between the SPBE and the knowledge repositories are shown in more detail in Figure 3(c).

Access to services and provisioning of services is handled by the service layer. The service layer consists of the service repository and (if needed) proxies to access the services. Such a proxy could be a HTTP to CoAP conversion. Furthermore, utilizing the business information and reasoning layer, the system might provide additional higher level services based on SLAs, reasoning (e. g. ontology based virtual sensors) or the connection of entities and (virtual) sensor data.

The deployment view of our envisioned semantic enterprise integration platform is shown in Figure 3(a). The actual IPIs are deployed to different (geographical or logical) zones and control one ore more motes. They are responsible for reading data from them or setting their state in case of an actuator.

C. SPBEQL: SPBE Query Language

SPBEQL is a lightweight query language for establishing connections between SPBEs and the sensors measuring their properties. The creation of the language has been inspired by SQL and particularly by its subset GQL (Google Query Language) [10]. SPBEQL is currently tailored towards sensor data only. It supports a *create service* statement with the purpose to create a service endpoint that can be used to retrieve sensed data about one or more SPBEs:

```
CREATE [ALL] SERVICE [MODE] FOR
 [SELECT statement]

MODE :== PUSH | PULL

DELETE SERVICE URI;
```

The ALL keyword, if specified, causes the system to create distinct service endpoints for all SPBEs involved in the SELECT statement, otherwise only one service endpoint for the whole select is created.

The MODE property can either be push or pull. In push mode the sensors regularly send data to the endpoint (if applicable) thus making sure that data is available sooner, but it might be not the most recent one. In PULL mode the service values are fetched at the moment the request is executed. Additionally, the system is able to delete services.

The actual properties of interest are specified using an SQL-like SELECT statement with the following syntax:

```
SELECT ([AGGOPP] SENSED_PROPERTY)+ [FOR SPBE] WHERE
  <CONDITION> [HAVING] [WITH [CollectionInterval] [AND] [
       CollectionTime]];

AGGOPP :== AVG | COUNT | MIN | MAX | SUM
```

The SENSED_PROPERTY is derived from the W3C Semantic Sensor Network (SSN) ontology[1], the SPBEs are stored in a separate knowledge repository. Services are described in L-USDL[2] glueing SSN and the business domain specific ontologies together. It can be sensed from known sensing devices (motes) or originate from any other Linked Open Data source. We currently support the following aggregation functions: AVG (arithmetic mean), COUNT (number of values), MIN, MAX and SUM. We specify a keyword *CollectionTime* (ct), which determines for which time frame the data from the sensors should be collected, and *CollectionInterval* (ci), which determines how often new sensor readings should be acquired.

The following code demonstrates creating a service endpoint for getting the temperature of a given temperature zone:

```
CREATE SERVICE PUSH FOR
  SELECT AVG MIN MAX temperature FOR SPBE e WHERE
    e:hasLocation:city = "Zurich" AND
    e:is_an = "temperature zone"'
    WITH CollectionInterval = 30s AND CollectionTime = 60s
```

Mapping conditions to ontologies is done by pattern matching. Navigation in the semantic graph is therefore possible without prefixes (like ssn: or usdl:), as long as the identifiers are unique. In contrast to query languages that return a result set, the CREATE statement returns an URI allowing to access the specified service and retrieving a result set in RDF. The system also creates a service description for that particular service and adds it to the enterprise service repository.

We choose our own query language for complexity and experimentation reasons. It gives us full control over a language with medium complexity. We would not have that with a query language like SPARQL. Furthermore, we want to be open to extensions and new ideas, which are easier to implement in an SQL-like framework. In future work we will explore the possibility of using OData and SPARQL as further query languages and the use of moving aggregation functions.

D. Service Deployment Planning

There are currently two different ways of tackling the aggregation and sensemaking of data. On one hand, there is the growing big data community, that favours a sense, send and store approach, where aggregation is done later on the backend systems. On the other hand, influenced by the needs of the traditional embedded world, others favour data fusion on the mote to reduce the data to be transmitted. The actual to be preferred approach, of course, depends on the requirements and the environment of the application.

In this work we suggest a heuristic that can either create a sense-and-send (S&S) style placement, or performs a greedy placement of aggregation functionality as close to the sensors as possible. A S&S approach, as done for example by Haseman et al. [6], has the advantage of an easy implementation. The authors of [6] did not perform an evaluation in terms of energy efficiency. Our experiments show that a S&S approach is sufficient when dealing with sensing devices that are not battery powered or have to send data very infrequently. Otherwise, the energy profile of an node aggregation is better.

[1]http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

[2]http://www.linked-usdl.org/

We distinguish three types of services: (i) Data Gathering Services (DGSs), (ii) Aggregation and Forwarding Services (AFSs) and (iii) SPBE Services. DGSs are responsible for the actual sensing of data. They are hosted on the mote and gather data. AFSs aggregate or forward data from one or more DGS. Finally, the SPBE service is the endpoint that gives access to the sensed properties of the actual physical business entity. These endpoints are described with Service Descriptions and stored in the Enterprise Service Repository. An actual service can have more than one of these roles. In some cases the SPBE services can be placed directly on one of the motes, sometimes this is even desired, as in the case of a moving intelligent container. Often the SPBE access point is on one IPI. There is no aggregation done over multiple IPIs. The IPI with most motes attached to it is selected as the endpoint and all other IPIs are sending their data to it.

The services are then placed on the motes following a resource-aware heuristic. The proposed heuristic expects tree routing, but is otherwise independent of the underlying routing as long as a tree can be build, for example by introducing an overlay routing based on shortest distances. We define:

A predecessor tree $P(\mathbb{M}, \mathbb{E})$, consisting of communication links $\mathbb{E}$ and motes $m \in \mathbb{M}$. The subset $\mathbb{M}_s \subseteq \mathbb{M}$ are the motes sensing data for the current query. Each mote has a set of available resources $\mathbb{C}$ consisting of tuples $\mathbb{R}$ (storage, processing). As the WSN might perform other tasks as well, there is an initial setting of these values. We define a mapping $\Phi(\mathbb{M}) \rightarrow \mathbb{S}$, that maps a mote m to a set of services $s \in \mathbb{S} = \mathbb{S}^{DGS} \cup \mathbb{S}^{AFS} \cup \mathbb{S}^{SPBE}$. For each service $s \in \mathbb{S}$ there is a function $\rho(\mathbb{S}) \rightarrow \mathbb{R}$ returning the needed resources.

As synchronization primitive we use epoch based barrier synchronization [11]. Without limiting the generality, we use beacon based time synchronization [12]: The used Moterunner (MR) [13] platform uses 6LoWPAN with tree routing and TDMA medium access, where the edge mote has a-priori knowledge of the send/receive frame timings.

We define a slack vector $\mathfrak{w}$ as follows:

$$\mathfrak{w} = \begin{bmatrix} t_1 & t_2 & \dots & t_n \end{bmatrix}^{\tau}, t_i > 0, \forall m_i : \mathfrak{w}(i) = t_i.$$

The slack vector $\mathfrak{w}$ is used for timing the aggregation points. Each t_i defines the time an aggregation point is waiting for data, until it closes an epoch. Later arriving data from a previous epoch is invalid. Applications, in which this later data has still a meaning, can turn off invalidation. The larger t_i is the more robust is this particular aggregation point against packet loss.

We try to aggregate data on the motes whenever possible, in order to decrease the energy needed (by reducing the number of transmissions) and the response time. The problem of placing services in the WSN corresponds to the task embedding problem, which is known to be NP-hard [14].

In the following we sketch the greedy placement heuristic (GPH) used for placing services on the tree P:

```
1: procedure CC($m \in \mathbb{M}, s \in \mathbb{S}$)                ▷ Check constraints
2:     $v \leftarrow \rho(s)$
3:     if $\forall v_i \leq \mathbb{C}(m)_i$ then
4:         update($\mathbb{C}(m)$); return true
5:     return False
6: end procedure
7: procedure LCA($a, b \in \mathbb{M}$)                ▷ Least common ancestor
8:     $c \leftarrow leastCommonAncestor(a, b)$                ▷ details omitted
```

```
9:       return c
10: end procedure
11: procedure GPH
12:     for all $m_i \in \mathbb{M}_s$ do                          ▷ Set DGS
13:         $s \leftarrow cs(m_i)$                                  ▷ Create service
14:         $\Phi(m_i)$ := s
15:         $\mathbb{S}^{DGS} \leftarrow \mathbb{S}^{DGS} \cup \{s\}$
16:         cc($m_i, s$)
17:     $\mathbb{M}_l \leftarrow sort(\mathbb{M}_s)$ by level desc
18:     for all $m_i \in \mathbb{M}_l$ do                          ▷ Set AFS
19:         for $j = 1..(|\mathbb{M}_l| - i)$ do
20:             $m \leftarrow LCA(m_i, m_{i+j})$
21:             $s \leftarrow cs(m)$                                ▷ Create service
22:             if !cc(m,s) then                                    ▷ Find slot
23:                 $f \leftarrow false$
24:                 while ((!f) & m!=EdgeMote) do
25:                     $p \leftarrow m.parent$
26:                     $f \leftarrow cc(p, s)$
27:                     $m \leftarrow p$
28:                 if (!f) then                                    ▷ No slot found
29:                     $m \leftarrow$ IPI
30:                     AggregationOnIPI($m_i, m_{i+j}$)
31:             $\Phi(m)$ := s
32:             $\mathbb{S}^{AFS} \leftarrow \mathbb{S}^{AFS} \cup \{s\}$
33: end procedure
```

In a nutshell, the algorithm first places all DGSs to the respective motes, then sorts the motes by level and places the AFS using the least common ancestor (LCA) algorithm [15] to the closest ancestor in adherence of the constraints. This can also be the IPI.

While setting the slack vector is application specific and can take link and application specific characteristics into account, one possibility is to set it by the calculated time to collect data from the corresponding lower level services (DGS or AFS), multiplied by a factor β_i to compensate packet loss. The slack vectors typically increase in each level of the tree from leave to root, as the next level of AGS needs to wait for the results of their feeding AGS. In case of synchronous TDMA-based medium access control we can easily calculate the values for each AFS slack vector value t_i, with corresponding DGS with slack values $t_n...t_m$ (without limiting the generality), CollectionTime ct and maximum transmission time $tt_max(m_i, m_j)$ from mote m_i to mote m_j. This leads to the following values for the DGS and AFS service slack values:

$$t_n = ct * \beta_1, \beta_i \geq 1$$

$$t_i = max(t_n, .., t_m) +$$

$$(max(tt_{max}(m_n, m_i)..tt_{max}(m_m, m_i)) * \beta_i), \beta_i \geq 1$$

Motes can join or leave the WSN at any time. When a new mote joins, the system integrates it by deploying a DGS if necessary, and determines the next possible AFS by applying the LCA algorithm. In cases, in which motes, that still have children (according the the current routing) leave the network, the MR platform will try to create a new tree. From an IPI point of view, motes reattaching to the tree are like new motes. They will look for an existing AFS and feed their data to it. If an SPBE endpoint leaves the network the system will create a new one on the IPI and change the service repository accordingly. If a mote that has no direct path to a SPBE endpoint joins, the endpoint is also moved to the IPI and all components accessing the service will get a service redirection response.

E. Service Creation and Deployment

Once the deployment plan has been calculated, the platform creates the corresponding services. We create on-mote services in Java, compile them (see Figure 3(b)) and deploy them according to the mapping Φ, as calculated by the greedy placement heuristic. Currently we are using CoAP over 6LoWPAN for communication. Code generation is done with a parameterized template system. The services themselves are parameterized as well, so that the important parameters CollectionTime, the slack time and the corresponding AFS can be changed during run-time.

F. Service Access

An application can access the created service through the service URI. Furthermore, the service can be queried through the service repository. Each service is accessible through a CoAP RESTful interface and over HTTP via a CoAP proxy on the IPI. Both endpoints are stored in the service repository for consumption by service consumers. All calls are RESTful (GET) and return RDF triples. A DELETE to the service would remove the service, just as the delete service statement in SPBEQL. With a POST to special resources as /ct or /st the collection time (ct) and the slack vector (st) be changed.

VI. Protoype Implementation

We built a prototype implementation and performed quantitative evaluations. All code is written in Java. The platform is based on Java7. On the motes we use IBM Researchs' MR platform [13], with an improved version of their 6LoWPAN protocol. It uses a custom byte code and it has been shown that the platform has a very good tradeoff between using a VM and energy consumption [16].

We created a custom CoAP-14 implementation. We use the endpoint descriptions to generate code for specific REST services and add a small CoAP library with minimal overhead. CoAP uses a stop-and-wait congestion control algorithm, which needs only very limited resources for bookkeeping.

VII. Evaluation

In this section we perform a quantitative analysis of our SPBE-aware integration platform. In particular we evaluated the response times of services created by SPEQL as well as the energy consumption of the complete system. Furthermore, we experimented with external RDF data sources and their integration into the platform.

A. Evaluation Scenario

We assume a retail scenario, in which all relevant information (store, goods, properties of stores and goods) are kept in semantic repositories. We connect these to form SPBEs and create services for them. In the stores different perishable goods are stored in different temperature zones tz_i. The whole store is monitored by a WSN connected to one IPI. The WSN is monitoring this tz_i and has information about the goods in each particular temperature zones. The data sources used are as follows:

1) The tz_i are stored an repository. It is linked to the particular store, which has a specific location.
2) The goods in each tz_i are also stored in an enterprise repository as part of the retail system.
3) We have a dataset of locations, which allow to query for geolocations, e. g. for all stores of the chain in Zurich.

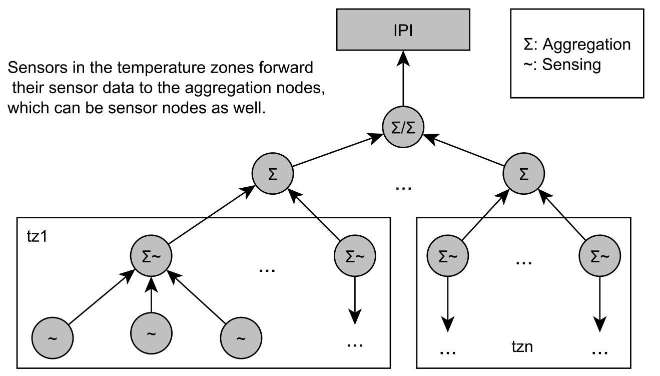

IPI
Σ: Aggregation
~: Sensing
Sensors in the temperature zones forward their sensor data to the aggregation nodes, which can be sensor nodes as well.
Σ/Σ
Σ
Σ~
~
tz1
tzn
...

Fig. 4: Scenario

TABLE I: Technical details of an IRIS Mote

CPU	ATmega1281 (8Mhz)	Serial Flash	512k Bytes	Program Flash	128k bytes
RAM	8k bytes	Current	8mA(act), 8μA(sleep)	RF power	3 dBm (typ)

As shown in Figure 3(d) the business experts formulates the process in a business modeling language, which uses the semantic repositories. Based on that the system creates business rules (BRs). These BRs then trigger the generation of SPBEQL statements, which in turn generates software to be deployed on the motes and on the IPIs. The generated services used by the business process are semantically described and stored in the semantic service repository. The modeler needs no knowledge about the WSN or any particular sensing device.

We created endpoints for a specific store in Zurich, with four distinct temperature zones, with 12 sensing sensors in each zone (see Figure 4). The endpoint delivers the maximum, minimum and the average temperature of all sensor readings. Based on this information the retail system does make decisions on the price of the specific goods (dynamic pricing). We used $ci = 5s$ and $ct = 20s$ and averaged the results over 100 runs for each experiment.

B. Evaluation Setup

We conducted our experiments in a close to real-world setup in a living lab and with simulation support. The evaluations were done using IRIS Motes with technical details as in Table I. We used the MR platform [13] with its simulation and profiling tools. The MR platform uses a 6LoWPAN TDMA MAC protocol that builds up a multi-hop tree. The whole network can be known, as the parent nodes send control messages to the edge mote as soon as they discover and add a new child or loose connectivity to one of their children.

C. Experiments

Table II shows the measured times needed to setup the system. The duration from issuing a SPBEQL statement to a complete system (without programming [3], but with compiling) is around 28s. The total time needed to set up the system depends mainly on the programming of the motes, as there is no multicast mechanism each mode has to be programmed individually (multicast). This is a limition of the current system. Unicast/Broadcast programming techniques, like the SNOMC protocol [17], exist. A multicast mechanism would speedup the overall system setup time. Configuration of the services on the motes can then be done via the REST interface.

[3]Programming in this case means transfering the bytecode to the mote and storing it on flash memory

In Table III we show the memory usage of the three service types. The actual memory needed depends on the data to be stored on mote. Its upper bound can be easily calculated per epoch and is used in the GPH.

After the initial set up, we measured the response times for a service endpoint on the gateway mote, and compared that to a service endpoint on the IPI. As shown in Figure 9, in pull mode the system needs around 20s for data gathering plus an overhead for communication. In push mode the data is immeadiately available after an initial data gathering round.

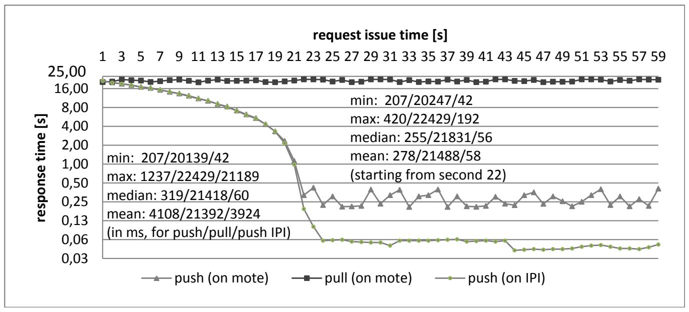


Fig. 9: Response times for request issued at request time t (log_2 scale)

The response times for a running system are shown in Figure 7 for a service endpoint on a mote, on a mote at level 2 in a tree, as well as on the IPI via CoAP and HTTP. We accessed the CoAP and HTTP interfaces with a machine on the same local subnet, which handled other traffic as well. The data we got from the WSN was relatively stable, while for the communication over Ethernet we observed some peaks.

The energy profile of our prototype is shown in Figure 6. We measured (by simulation) the system in pull mode, in push mode, and a S&S counterpart with all motes sending data to the IPI. The aggregation approach showed to be more energy efficient, as the computation time for the aggregation needed less energy than the additional transmission and receive cycles needed in the S&S approach. Pull is more energy efficient, for small request rates, for the price of longer response times.

As explained in section V-C the system allows to include external RDF data as sensor readings. We set up a second IPI with a single mote attached to it. Instead of adding the second IPI to the platform directly we publish the sensor readings as RDF data to a local server and to a server on the Internet. We added both servers as external RDF sensor sources to our system and measured the service response times (Figure 5) for the whole system and the external sources. In push and pull mode with an endpoint on the IPI, the influence of the two external sources on the system is negligable, both for local and remote data sources as long as ct is large enough to compensate the latency. As soon as the WSN responds faster than the external sensor source the picture changes. In cases were the freshness of data is most important the latency of the Internet connection becomes cruical and the latency (CollectionTime) of the whole system would have to be increased to get accurate data.

Finally, we tested how the system reacts towards change of the underlying sensing infrastructure: We removed one AFS

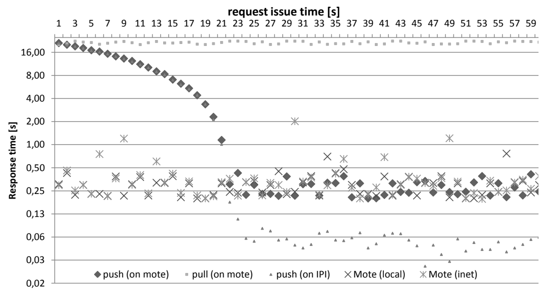


Fig. 5: Service Response times for external sensor sources

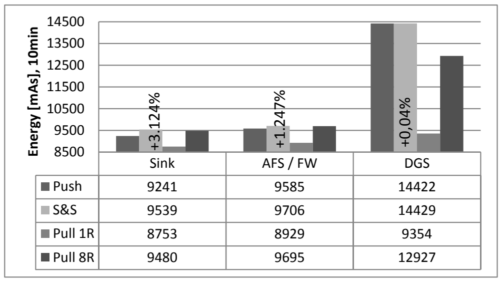


Fig. 6: Energy consumption (in mAs, 10min)

	$\bar{t}$	σ
Generate mote services	2520	350
Compile mote services	8410	2123
Compile IPI service	4120	210
Program mote	39200	5213
Query Service Repository	123	51
Query SPSE	661	94
Query geolocations	472	53
SPBEQL to System	28320	4476

TABLE II: Average system setup time $\bar{t}$ and std. deviation σ in ms

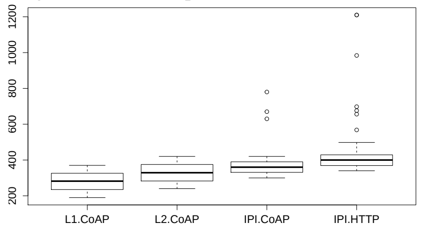


Fig. 7: Service Response Time (in ms)

	Flash	Stack	Heap
DGS	923	256	2348
AFS	800	234	2205
SPBE	810	245	2218

TABLE III: Memory consumption (in byte)

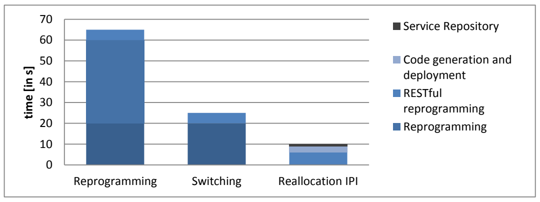


Fig. 8: Service reallocation time (in s)

below the edge mote. The system then recreates a tree. The new AFS node, which has only a DGS service on it, now needs to be reprogrammed. This takes almost 40s, as shown in Figure 8. If the AGS is already installed on the mote aggregation infrastructure, changes can be performed more easily: In such cases only a few REST calls have to be issued by the Command and Control (C&C) module. Whenever possible programming of motes is to be avoided. Here the system is available after around 10s ($\sigma = 3.3s$). In a second experiment, we forced a service reallocation from the edge mote to the IPI. The C&C issues a command to the current endpoint to respond with service temporarily unavailable. At the same time it will compile and deploy a new endpoint to the IPI and updates the service repository. The former endpoint from now on will respond with a service redirected code.

VIII. Summary

We presented Semantic Physical Business Entities as an abstraction for gathering data about business entities. It decouples the entities from the actual sensing devices and enables writing applications without any knowledge about the underlying hardware. Interoperability and machine-readability is achieved through ontologies and common vocabularies. Data is gathered and aggregated by an integration platform using a query language. The platform compiles the services depending on the query. These services are described by semantic service descriptions and are fully controllable through a RESTful interface. Feasibility experiments demonstrated that SPBEs can be integrated into enterprise systems with reasonable energy consumption, set up and response times.

Acknowledgment

The research on this topic received funding from the European Commission under grant 257521 (IOT-A) and grant 285248 (FI-WARE). We would like to thank the Moterunner Team at IBM Research, especially Marcus Oestreicher, as well as our students Theano Mintsi and Michael Gede for their valuable support.

References

[1] Z. Shelby and C. Bormann, *6LoWPAN: the wireless embedded internet*. Wiley, 2011, vol. 43.
[2] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP), Internet-Draft," 2011.
[3] T. Berners-Lee, "Linked Data - Design Issues," 2006.
[4] K. Sperner, S. Meyer, and C. Magerkurth, "Introducing entity-based concepts to business process modeling," in *BPMN*. Springer, 2011.
[5] M. Thoma, S. Meyer, K. Sperner, S. Meissner, and T. Braun, "On IoT-services: Survey, Classification and Enterprise Integration," *2012 IEEE International Conference on the Internet of Things*, vol. 0, 2012.
[6] H. Hasemann, O. Kleine, A. Kröller, M. Leggieri, and D. Pfisterer, "Annotating real-world objects using semantic entities," in *Wireless Sensor Networks*. Springer, 2013, pp. 67–82.
[7] M. Iqbal, H. B. Lim, W. Wang, and Y. Yao, "A service oriented model for semantics-based data management in wireless sensor networks," in *Advanced Information Networking and Applications*. IEEE, 2009.
[8] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, 2005.
[9] J. Cardoso, C. Pedrinaci, T. Leidig, P. Rupino, and P. De Leenheer, "Open semantic service networks," 2012.
[10] A. Zahariev, "Google app engine," *Helsinki University of Technology*, 2009.
[11] S. Shang and K. Hwang, "Distributed hardwired barrier synchronization for scalable multiprocessor clusters," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 6, no. 6, 1995.
[12] H. Dai and R. Han, "Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, 2004.
[13] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov, "Mote runner: A multi-language virtual machine for small embedded devices," in *SENSORCOMM'09*. IEEE, 2009.
[14] Z. Abrams and J. Liu, "Greedy is good: On service tree placement for in-network stream processing," in *ICDCS 2006*. IEEE, 2006.
[15] M. A. Bender and M. Farach-Colton, "The lca problem revisited," in *LATIN 2000: Theoretical Informatics*. Springer, 2000, pp. 88–94.
[16] A. Caracas, C. Lombriser, Y. Pignolet, T. Kramp, T. Eirich, R. Adelsberger, and U. Hunkeler, "Energy-efficiency through micro-managing communication and optimizing sleep," in *8th International Conference on Sensor, Mesh and Ad Hoc Comm. and Networks*. IEEE, 2011.
[17] G. Wagenknecht, M. Anwander, and T. Braun, "Snomc: an overlay multicast protocol for wireless sensor networks," in *Wireless On-demand Network Systems and Services (WONS), 2012 9th Annual Conference on*. IEEE, 2012, pp. 75–78.