From relational databases to an ontology – practical considerations

Rafał Tkaczyk*§, Paweł Szmeja*, Maria Ganzha†*, Marcin Paprzycki‡*, Bartłomiej Solarz-Niesłuchowski*
*Department of Intelligent Systems
Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
Email: firstname.lastname@ibspan.waw.pl
†Department of Mathematics and Information Sciences
Warsaw University of Technology, Warsaw, Poland
Email: M.Ganzha@mini.pw.edu.pl
‡Department of Management and Technical Sciences
Warsaw Management Academy, Warsaw, Poland
§IT Systems Department, Vemco Co. Ltd., Sopot, Poland

***Abstract*—One of interesting problems, arising with deployment of large-scale systems, is integration of its nodes (systems / devices). In this work, we discus how to apply semantic technologies, as a mechanism to support node integration and facilitate interoperability within the developed ecosystem. We focus on pragmatic aspects of the proposed solution, discussed from the perspective of the Dependable Embedded Wireless Infrastructure (DEWI) EU project. In this context, a brief analysis of typical integration problems and reasons to apply solution based on semantic technologies is presented. Moreover, the integration procedure is outlined. Here, the key aspect that is discussed in considerable detail, is conversion from the DEWI nodes (based on a traditional relational database approach) towards universal cooperative nodes, which use semantic technologies.**

I. Introduction

One of key issues, concerning modern building automation management systems, is co-existence of multiple subsystems, introduced by various suppliers, and based on different technologies. As a result, applications, (sub-)systems, devices, sensors, etc., work "in parallel" (i.e. separately), without cooperation or, even, simple communication (e.g. data exchange). Since so many different entities (from different levels of hardware-software stack) may co-exist in a building management ecosystem and since, in large part of what follows, there is no actual need to distinguish between them, in the remaining part of this paper, unless a more specific term is needed, we use the name *node* to represent each and every one of them.

One of very common problems, arising during system integration, is considerable variety of technologies used for data storage. Here, different systems may use standard/popular, but different, databases. Moreover, within a single ecosystem, both SQL (MySQL, PosgtreSQL, MS SQL, Oracle, etc.) and NoSQL (MongoDB, Redis, CouchDB, Neo4j, etc.) databases can be found. Note that, even in the case of the same database model, e.g. relational, document or graph, different vendors provide solutions that are, for all practical purposes, incompatible. Separately, two systems, which use the same technology (e.g. MySQL databases), can use different data model and thus, again, remain incompatible. Even if the same relational model is used, its implementation might differ because of different SQL dialects and database capabilities. However, the goal of deployment of an integrated ecosystem is to allow its nodes to communicate. In this sense, from purely logical perspective, as a result of integration, all sources of data in the system should be "visible as a single distributed database". Henceforth, the real difficulty is to combine multiple sources into one, consistent and logical whole.

Separately, it is worth noting that while the complexity of an *existing* infrastructure plays an important role, complications brought about by adding new nodes to the working system (*modification* of the infrastructure) need to be considered as well. In this case, along with connecting yet another technology (node), logistical difficulties of expanding the infrastructure may materialize, e.g. the location of the next wire to be placed in the infrastructure, or wireless networks that start adversely affecting each other. Similarly, when wireless infrastructure is considered it can be realized using different technologies: Wi-Fi (IEEE 802.11), ZigBee, Bluetooth (IEEE 802.15.1), NFC (ISO 15408 and ISO 14443/ ISO 15693), 6LoWPAN/ IPv6 (PFC 4919)). Furthermore, different nodes may use different communication protocols (e.g. TCP/IP Socket, WebSocket, HTTP, etc.), thus requiring further work to establish some form of "gateways" between them. Nevertheless, these considerations are out of scope of our current work.

Issues raised thus far, illustrate multitude of aspects that must be taken into account during integration of nodes into and/or within an ecosystem, and when the size of the ecosystem is increasing due to addition of new nodes. Observe that, typically, such integration aims at allowing new/additional forms of cooperation within the nodes, and thus introduce new functionality into existing applications and/or development of new applications based on them. Multiple projects attempted at proposing solutions to the problem of integration of nodes within an ecosystem, in general, and within a building automation management system, in particular. One of them was the Dependable Embedded Wireless Infrastructure

(DEWI) project [1]. Here, actual experiences from DEWI provide the background for the proposed solution, and are the source of issues that had to be solved during DEWI building management system deployment.

In this context, we proceed as follows. Section II contains the general description of the DEWI project, including architecture and an example scenario realized during the pilot demonstration. Moreover, we present issues that arose during the development process, and approaches used to solve them. Next, we elaborate processes that were necessary to convert an SQL database model into the ontological one (Section III) and, later (in Section IV), use of this model to instantiate a graph database. This allows us, in Section V, to describe the integrated solution in action. Finally, in Section VI, we summarize the main contributions of this paper.

II. Need for data integration in the DEWI project

Let us start from a brief description of the Dependable Embedded Wireless Infrastructure (DEWI) project, which involved use of wireless embedded systems in: aerospace/space industry, car production, smart buildings and railways. In general, DEWI had five main objectives (as shown on the project website [2]):

- dependable, auto-configurable, optionally secure, short-range communication,
- local energy-management: efficiency, harvesting, storage,
- localization of sensors and mobile devices, and
- smart composability and integration of wireless sensor networks.

To reach these objectives, a DEWI architecture was proposed (see [3] and [4], for details), which consisted of:

- DEWI Bubble (the core of the DEWI solution, a group of nodes, gateways and users),
- DEWI Gateway (device supporting communication between DEWI Bubbles and also Internal/External users),
- DEWI Node (entity acting within the DEWI Bubble, e.g. service, system, application, device, sensor, etc.),
- Internal User (user acting within the DEWI Bubble that has access to any DEWI Service via the DEWI Gateway), and
- External User (user acting outside the DEWI Bubble and communicating with it via the DEWI Gateway, with very limited access to DEWI services).

Each of these elements is described, in detail, in [5].

It was assumed that various nodes are to cooperate, within the DEWI architecture, to achieve measurable benefits, in the form of reduction of hardware infrastructure and wired connections, improvement of energy management, optimization of operations of (sub)systems, network flexibility, etc. Hence, one of subgoals of the project can be stated as: to integrate nodes running in the same environment (i.e. in a separate subnet, called DEWI Bubble; see Figure 1) that have to cooperate to achieve a "common goal". In the project, it was assumed that a single DEWI Bubble handles one domain, e.g. a single building. The purpose of the developed solution was to facilitate seamless cooperation of many systems (access control, positioning, emergency, security, lighting, CCTV, etc.) and devices (terminal, smoke sensor, camera, sensor, etc.), installed within a single Bubble (building).

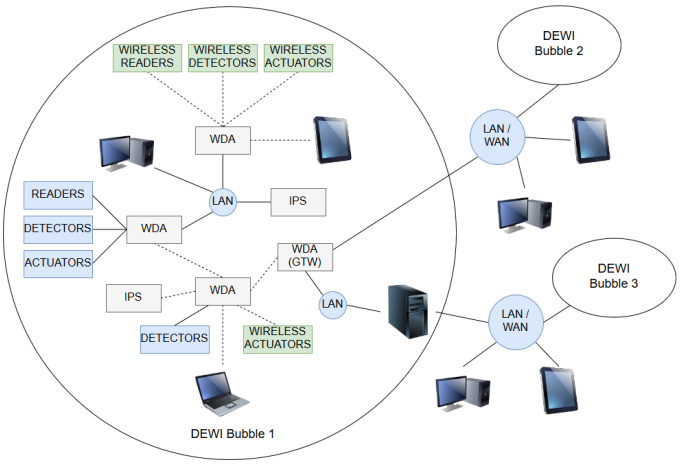

Fig. 1. Topology of DEWI wireless sensor subnetworks (Bubbles)

While the DEWI project pilots were instantiated in multiple domains (i.e. aeronautics, automotive, rail, building and interoperability), we will concentrate on building automation management. This is a very good example to describe project approaches, because of two reasons: (i) building domain is a very popular and well-known (involving many approaches to develop solutions, e.g. Internet of Things, Cyber Physical Systems, Multi Agent Systems, etc.), and (ii) the final result of the implementation of this part of the DEWI project was a successful pilot deployment in Gdańsk, and one of the authors participated directly in this pilot. Within the DEWI homepage [6] five main objectives concerning application of DEWI within building automation management can be found:

- to improve situation-awareness and access control in buildings,
- to decrease energy consumption and reduce emissions in buildings,
- to optimize facility operation and maintenance work,
- to easily deploy and maintain the WSN networks in buildings, and
- to increase the performance and scalability of building WSN solutions.

The pilot environment is a very good example of a DEWI Bubble. It illustrates cooperation of variety of nodes in the closed network (where the connection "from the outside" is possible only via the gateways). The pilot architecture consists of (see, Figure 2):

- Access Control System (ACS) – system managing the restricted areas,
- Wireless Data Aggregator (WDA) – specific type of gateway, developed for the DEWI project,
- Indoor Positioning System (IPS [7]) – objects and persons localization system, and
- a prototype mobile terminal device.

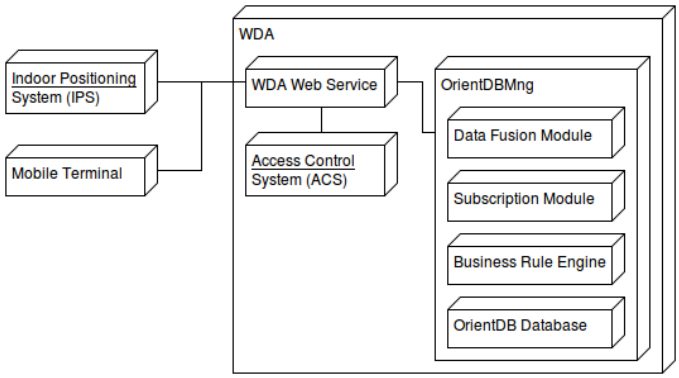

Fig. 2. Pilot architecture

The building automation pilot assumed two general scenarios (consisting of multiple requirements). The first scenario dealt with cooperation between the ACS and the IPS. The purpose of the ACS is to feature information about the entities' (person, thing or device) relation to the controlled areas. The role of the IPS, is to facilitate information about positioning of tracked objects. This information can be card read events from ACS devices; information provided by identification devices, like Jennic modules (a ZigBee [8] wireless microcontroller module with a built-in antenna) or RFID cards; or other data describing entities that appear within the monitored area.

The second scenario involved use of a mobile terminal (with an embedded Jennic module) inside the building. The main goal of this scenario was to support the visitor (one of the end users of the ACS), moving inside the controlled areas. With the ACS and a terminal, the user can register her/his visit to the restricted area (using the ACS) and obtain permission to enter (or be refused). Moreover, based on the data obtained from the ACS and the IPS, the terminal can locate the position of an object, which is sought by the user (e.g. person, thing, device) or a place (e.g. a specific room). The result of this search is not only the information about the object position, but also a map displayed on the screen, including a directions to the "target".

A. Data integration in the DEWI project

Considering the goals of both scenarios, it is easy to see the importance of data integration. In the project pilot, two popular databases have been used. PostgreSQL was utilized in the ACS, because it is a good solution for big database with sophisticated architecture, adapted to run in a cross platform environment. In the IPS, one of lighter database distributions has been used, namely MySQL. The role of these databases, in their respective applications, was as follows:

- PostrgreSQL, in the ACS, was used to support the access control server applications (to handle end user requests and device management), i.e. store information about application users and entities (person, vehicle, object) and their access rights, as well as information about areas of the building, device infrastructure, events, etc.
- MySQL, in the IPS was used to store data about the areas of the building and the Jennic modules (and their positions) within them. Moreover, it gathered data about the device infrastructure for the IPS visualization module.

Both ACS and IPS databases contain data that is relevant to establishing cooperation with other systems /devices. For instance, the ACS stores (among other) events generated within the system, and consumes the entities' positions, detected by the IPS. The IPS, on the other hand, stores data about entities position and uses data generated by the ACS (e.g. events).

It is worth noting that information, stored in the databases of individual nodes, is not always logically separated (especially in one domain). Very often it describes the same structures, e.g. both ACS and IPS store data about the same areas, devices, positions, etc. Here, the following problem materialized – "the same" data was stored in two databases, albeit using different structures. The building floor (level) is a simple example. In the IPS SQL structure, it was described as a *BuildingFloor* table (Figure 3) with rows: id, (primary key), id_building (unique id of a record), and additional fields to describe the specific building floor. More information, related with the floor geolocation is stored in the *BuildingFloorCalibration* table. The *BuildingFloor* table was linked with the *Building* table (defining the set of buildings in the system) to achieve complete building definition (i.e. building consists of building floors). In the ACS SQL structure, on the other hand, building floor was described as a local standard floor number (an integer). So, as we can see in Figure 4, when one wants to set a floor, for example for the *Area* table, one does not need to assign the foreign key of building floor. It is enough to set an integer of the floor level. These differences, arose because of the specific representation of the area structure in different components. In the ACS, the area is described in more detail, than in the IPS. Therefore, the ACS uses the *Area* table in the database, with a number of fields, describing it (it is necessary for management related functions). In the case of the IPS, the *BuildingFloor* table contains all the necessary information to describe the area, so the database does not contain any other structure to define it (like in the ACS).

The next issue was that both systems use the WGS84 standard to describe geolocation. In this standard, data can be represented in different ways: by separate fields (longitude and latitude), single field (where longitude and latitude are separated by a space), by GeoJson [9], etc. Both systems use the separate latitude longitude option, to describe the position within the area, but use different ways to define the area. The IPS defines the area as a quadrangle and stores its "opposite points" (vertices) in the database. The ACS uses the GeoJSON format, which means, it stores all points of the polygonal area.

The last example of mutual data structure is device representation. In both systems it is similar, but the most important difference is the device type description. The IPS uses the dictionary table *DeviceType* and many-to-many relation with the *Infrastructure* table (device). In the ACS, it is also the *DeviceType* table, but it is not just a dictionary. It contains other properties, related with the device configuration (it was

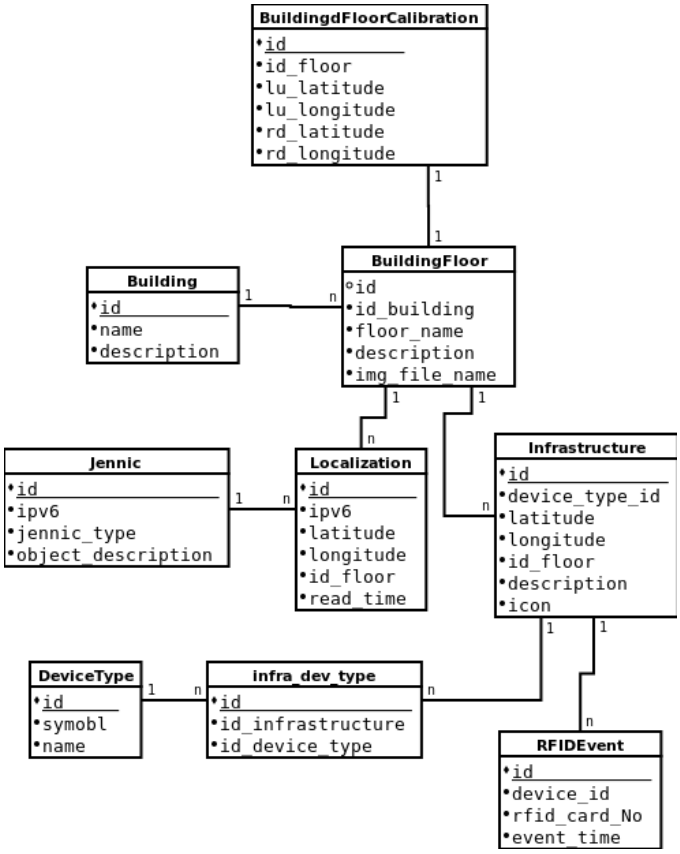

Fig. 3. IPS extracted data architecture

not depicted in the diagram, because it was not relevant for the pilot implementation, i.e. nodes cooperation). Moreover, the ACS *Device* table contains more properties, describing the device and is connected with *Entity* table, while in IPS, the entity is not relevant (the IPS system estimates the position of Jennic modules, the ACS links this device with the entity).

These examples describe how many aspects must be take into account during the data integration process. After literature research ([10], [11], [12], [13], [14]) and on the basis of the example of other EU projects ([15], [16], [17], [18], [19], etc.) we came to the conclusion that use of semantic technologies is worthy of exploration, as a way of integrating heterogeneous data sources and their divergent data models. Using an ontology (see, for instance, [20]) is a very convenient way of data unification, because (i) ontology is easy to expand, (ii) is easy and quick in search, (iii) it's easy to create a new relationships by describing object properties, (iv) allows capture data objectives to create new structure types, etc. Here, note that these features point out how useful an ontology can be not only in the case of *initial* system integration, but also in *subsequent* modifications to the "working ecosystem". Moreover, use of semantics provides tools for storing large amounts of data (triple stores) and run processing actions on them, (e.g. SPARQL [21] queries, reasoning, etc.). Since DEWI requirements assumed interoperability of nodes within the DEWI Bubbles (oriented towards specific domain) and also the cooperation across all the Bubbles, it was proposed to use a domain ontology in order to unify the data within the DEWI Bubble. Note that work on developing a meta-level ontology for cooperation between DEWI Bubbles is out of scope of this contribution.

III. Converting an SQL structure into an ontology (OWL)

Since it was decided that the main vehicle for data integration will be use of semantic technologies, the first problem that had to be addressed was lack of semantic data models in DEWI subsystems. Therefore, it was necessary to develop a common semantic representation (i.e. an OWL ontology) of data used across the system. Here, let us note that in [22] it was established, that lifting semantics of a database to an ontology faces the problem that the internal representation, within the database, is geared towards efficiency, rather than towards conceptual representation of the domain.

Therefore, the first step of establishing the ontological representation of data used across the DEWI Bubble was to pinpoint the pertinent elements of the pilot databases. Here, only the necessary tables were selected, i.e. data needed for cooperation / communication with other nodes. It was also important to predict, which data could be relevant for cooperating nodes in the future. The most laborious was the ACS data structure (Figure 4), because this system has the most complex database, containing variety of information, useful for many nodes. In general, the ACS part consisted of: Entity (three types: person, vehicle, thing), Event (a few types of access control events, e.g. door open, breaking door, door too long open, etc.), Area (restricted areas managed by ACS), Device (infrastructure of ACS), etc. Although, an IPS is a very complex system, the data it distributes is quite simple, i.e. it consists of information about infrastructure and (the most important) positions of objects (entities equipped with Jennic modules).

On the basis of the data structure, the entity relationship diagram (ERD; prepared using Dia Diagram Editor[1]) was obtained (see, Figure 4 and Figure 3). The next step was to use this diagram to instantiate an ontology. To design it, Protege (v5.0.0)[2] was used, while the final ontology file was in the OWL format. There were three main steps of creating an ontology graph.

- Creating class hierarchy, on the basis of the SQL tables names.
- Creating object properties, on the basis of relations (foreign keys) between tables.
- Creating data properties, on the basis column types and dictionary values in SQL tables.

During this stage, few problems were encountered. First, the most difficult of them, was the diverse representation of the same data in the systems. As described in Section II-A, the

[1](2017) Dia Diagram Editor Homepage. [Online]. Available: http://dia-installer.de

[2](2017) Protege Homepage. [Online]. Available: http://protege.stanford.edu/products.php

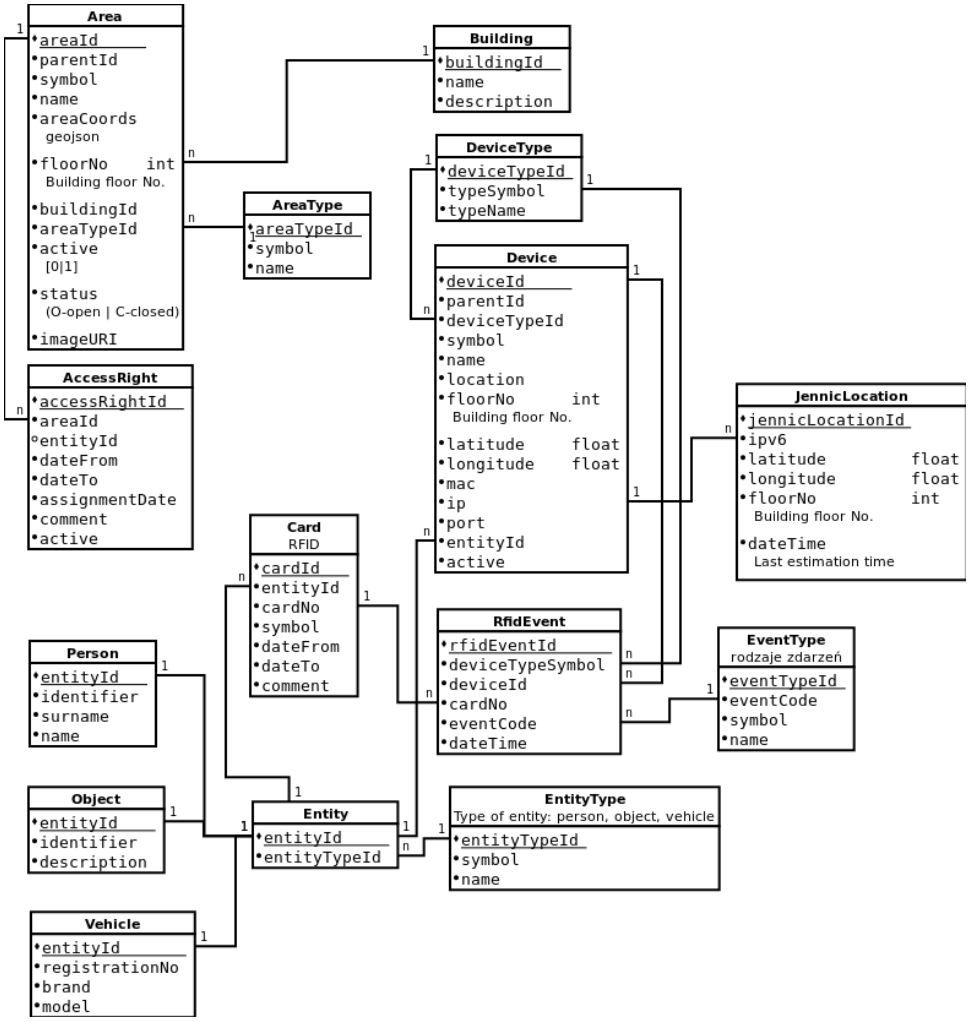

Fig. 4. ACS extracted data architecture

ACS and the IPS systems store data in a different way. Second problem was to convert the SQL dictionary tables into the ontology structure (e.g. describing events, or types of devices). Many dictionary symbols and names were described in a non unified way, depending on the internal nomenclature, e.g. Jennic module had symbols $Jennic$ and JEN, or RFID card reader has a symbol $READER\_V$ and $RFID\_R$.

Another aspect of problems with the dictionary was variety of possible methods for creating an ontological representation. During implementation, two options were considered. First, one can create object property hasType (Event hasType exactly 1 EventType). Next, instances of the EventType class for each type of event (e.g. http://DEWI.org/ACS.owl# AcknowledgehasType:EventType) are to be created. Finally, an $e1$ instance of the event type (i.e. $e1$ hasEventType: http: //DEWI.org/ACS.owl#Acknowledge) is assigned. The second option is to create subclasses of the Event class (e.g. AcknowledgeEvent subClassOf: Event). In the following step, $e1$ instance of the event type is assigned as a class (i.e. $e1$ hasType: AcknowledgeEvent). Finally, the second approach was chosen, because the queries are easier (due to the taxonomic structure). However, it should be noted that this is an example of a much broader issue related to possibility developing a uniform semantic representation (of any domain).

The next step to create the pilot ontology (with the help of the ERD diagram) was to search existing standardized ontologies, which could be used to describe key ontological entities, and serve as an upper, or domain, ontology. To complete the pilot scenarios, two basic areas had to be described: sensor/device and geolocation. For the sensor/device ontology three options were considered: SSN [23], IoT-Lite [24] and OpenIoT [25]. Upon further analysis, the IoT-Lite has been chosen, because it is relatively small and well-focused. Furthermore, it contains more device classes than, e.g. the SSN. Specifically, in the context of the project, the most important were the following classes: (i) *Device* (with subtypes *Sensor*, *Actuator* and *Tag_Device*) – to define the DEWI Bubble infrastructure, i.e. WDA, Jennic modules, area controllers, readers, cameras, detectors, etc., (ii) *Service* and *Attribute* – to define nodes' services and parameters; and

(iii) *Entity* – to define the entities managed by the platform, i.e. *Person, ThingEntity, Vehicle*. Of course, additional classes had to be added, e.g. *Event*, which can describe all kind of events generated by any node, e.g. ACS (*CardReadEvent, DoorsClosedEvent, DoorsOpenedEvent, BreakingDoorsEvent*, etc.), *Building* (to define buildings), *Card* (to define access cards, e.g. RFID card), *VehicleType* (as a dictionary to extend the *Vehicle* entity). Here, note that the choice of the IoT-Lite ontology rooted our model in the standardized IoT space.

The W3C Basic Geospatial Vocabulary was used to describe the positions, because this ontology can be used, in an easy way, to depict many types of objects (from a single point to a complex polygon), and it uses the WGS84. It is important in the building domain, because it is necessary to describe variety of areas, and the device infrastructure (including mobile devices, e.g. Jennic module or terminal). In the project, geospatial ontology was extend by three important (for the pilot) elements. First one, is an *Area* to describe the areas managed by the systems. It uses the *polygon* class to define the position and, moreover, contains such data properties as *floor* (xsd:int), *hasName* (xsd:string), *hasSymbol* (xsd:string), *isInBuilding*. Next added element was the *GeoLocation*, to define the entity position (point), within the area. Moreover, the *GeoLocation* has a subclass for storing the history of entity positions (e.g. in the case of mobile device).

To sum up, the final pilot ontology (depicted in the Figure 5) imports two general ontologies IoT-Lite and W3C Geospatial Vocabulary, which were extended by structures more specific to the domains of operation of the ACS and the IPS.

IV. Creating ontological database

Following creation of OWL ontology, it was necessary to instantiate a database that uses it in its model. The first step to achieve this goal was the selection of the database model. There are a few possibilities of database models, which we can split into three main types. First, a typical semantic solution: RDF store, with embedded semantic mechanism, like SPARQL, reasoning, etc. (e.g. Jena [26], Algebraix [27], 4store [28], Redland [29], AllegroGraph [30], Stardog [31], Blazegraph [32], etc.), Second, Graph DBMS, sometimes with partial RDF support (e.g. OrientDB [33], Neo4j [34], Titan [35], ArangoDB [36], etc.). Because of graph based structure, this type is also natural to be used as an ontological store, but this approach requires additional data management modules (to handle the RDF structure). Finally, another model types, including SQL and NoSQL, but they also require extra supporting modules, more complex (and usually less efficient) than in the case of Graph DBMS.

Completed analysis revealed that graph database would be the best to realize all objectives, desired in the pilot, because there is no need to use very complex semantic processing (i.e. reasoning). Moreover, graph database seems to be a good fit for storage of ontological data, because of the universality of graph representation, with respect to RDF. To develop the pilot prototype, the graph database OrientDB has been selected. It is an open distribution, which contains many useful features (in many cases not offered by the competition). For instance, it offers multi-master replication and sharding (database partitioning), which is a big advantage in a distributed architecture. Other than the standard graph search methods, OrientDB allows to use an SQL-like language (OSQL). Next advantage is related to user and role and record level security. Moreover, it offers elastic scalability.

After choosing the database engine, the next step was to represent the previously generated ontology. To create an OrientDB database structure, special script in JavaSE was developed, which generated all the necessary OSQL scripts. It contained three main operations, for mapping OWL structures into the OrientDB structure: (i) mapping the *classes* into the *Vertex* class elements (while preserving class inheritance), (ii) mapping the *Data properties* into the vertex *Property* elements (including data types – *Ranges* in OWL), and (iii) mapping the *Object properties* into the *Edge* class elements. Moreover, the OWL file contained also an additional information (in *Annotations*, tagged with *OrientDB:validator*), which was taken into account during the mapping process. For instance, *abstract* annotation meant to create an abstract OrientdDB class (*Vertex* or *Edge*).

Utilization of ontologically based graph database, allows to use it, in a simple way, for data fusion and processing operations. Specifically, data that is important from the point of view of other (cooperating) nodes, or data analyzing modules (e.g. a Business Rule Engine) is gathered in this graph database. Data subscription module allows receiving the actual data on-the-fly.

Note that messages in DEWI are in the JSON-LD [37] format (see Figure 6). This has been decided because of two reasons. In general, JSON format is one of a DEWI communication standards ([38]). Moreover, using RDF (of which JSON-LD is a serialization) to describe data and sending messages is also a standard in the semantic communication approaches. The natural way of combining those two issues is JSON-LD. The message structure (Listing 1) is actually a batch of graph vertices, linked to each other by edges (object properties). The variables (in <> brackets) are as follows:

- *vertex_name* – unique name of an individual (vertex), valid only within the message content (vertices batch); it is necessary to recognize the vertex,
- *individual_class* – type of an individual (an ontological class name),
- *individual_property* – name of a data property (property in the graph database and a data property in the ontology),
- *property_value* – value of a property,
- *edge_name* – name of an object property, i.e. link/edge to another individual (an edge in the graph database and an object property in the ontology),
- *vertex_in_name* – name of an individual (vertex), which is linked with this individual by the object property.

Listing 1. WDA message structure

```
[{
  "@id" : "<vertex_name>",
```

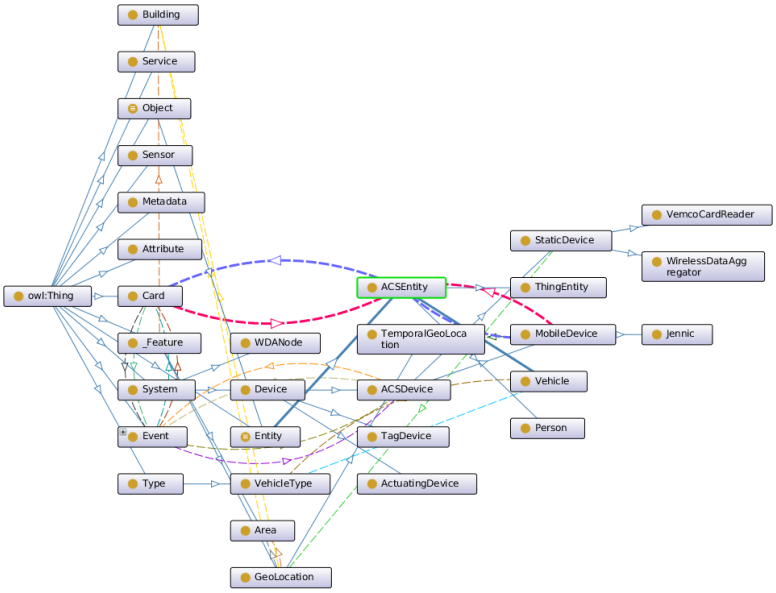

Fig. 5. DEWI pilot ontology class structure

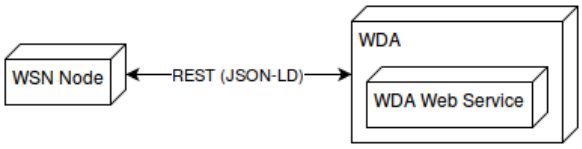

Fig. 6. Communication with WDA

```
  "@type" : [ "NamedIndividual",
                "<individual_class>" ]
  "<individual_property>" : [ {
    "@value" : "<property_value>"
  } ],
  "<edge_name>" : [ {
    "@id" : "<vertex_in_name>"
  } ]
}]
```

This solution offers easy graph database operations and also simple iteration and conversion to any data structure within the node side (i.e. node connecting with the WDA OrientDB management module).

V. Using the Ontological Database for DEWI Subsystems Integration

To illustrate how the developed solution works, we will describe how it was implemented in the pilot. The core of the idea is the Wireless Data Aggregator (WDA), which can work as a DEWI gateway. The WDA functionality comprises: access control management, area controller, etc. Furthermore, WDA is responsible for data fusion, aggregation and distribution. Logically, WDA is one of the nodes. It can be an independent device, or an element of a (sub-)system, e.g. the ACS. It enables direct communication between nodes (regardless of technology), since it supports multiple communication interfaces (wired and wireless, e.g. Wi-Fi, ZigBee, Bluetooth, etc.). The main reason for creation of the WDA is integration of different systems within a single domain (e.g. a building). The WDA device deals with:

- data fusion (considering all communication interfaces used in the project),
- distribution of data (relevant from the point of view of different subsystems),
- registration of nodes in the WSN network, and of services they provide,
- defining business rules associated with gathered data, fulfillment of which is to trigger specific actions (e.g. alert detection),
- as quickly and efficiently as possible, obtaining information (e.g. in the ACS), and
- running the internal software modules (e.g. system control, area controller, etc.).

The WDA data management mechanisms is based on the ontological database, described in Section IV. In order to use WDA modules (data fusion, distribution, subscription, etc.), a REST API was exposed. It contains three main methods: (i) /oriendb/aggregate (POST/PUT/DELETE/GET), to use the database management module, (ii) /oriendb/aggregate/subscription (POST/DELETE), to join the data subscription, (iii) /oriendb/aggregate/rules (POST/DELETE), to set a business rule.

The graph database is the core of the WDA. All scenarios, described in Section II, use this entity to cooperate. Every node can use WDA to store the data (in order to share it with other entities), or to read the data to "work with it" to achieve its purpose. Note that the second scenario shows that a device can also be a consumer of data gathered by other nodes, e.g. a mobile terminal imports data about entities (from the ACS) and

their positions (from the IPS) to present them to the user, or to bring about additional functionality (e.g. defines the transition path between two restricted areas, or to locate given device and show it on the map).

In the pilot, process of integration of new nodes was simplified, and is managed by network administrator, who has three responsibilities. First, is adding the node (system or device) to the network. Second, setting access rights in the WDA database, to allow (or prohibit) use of the data management module. Moreover, information describing a new element must be added to the database, e.g. the id (an individual symbol within the network), type (system or specific kind of device), address (IP, MAC, IPv6, type dependent), offered services, consumed services, etc. Third, if it is necessary to expand the pilot ontology, (s)he must add new classes and properties. In the future, implementation of more sophisticated automatic node registration mechanisms is planned. It will be the WDA node registration module that will receive the node's registration notification. This module will verify the node (its technical parameters and services) and will permit it to join the ecosystem (or deny it). However, the details of this process have not been finalized.

VI. Concluding remarks

Many projects considered integration of heterogeneous nodes to form an ecosystem. One of them is DEWI, which shows that one of the effective solutions to node interoperability is use of semantic technologies. However, in real life, most nodes use non-semantic data. Hence, it is necessary to convert their data models into ontological representations. In this paper, we have described in detail conversion of the (popular) relational model into the OWL format, and used it to instantiate a graph databases on gateway devices, in order to create an embedded cooperation support infrastructure. Showing the pilot demonstrator solutions, we proved that the semantic approach works. We are in the process of evaluating its efficiency.

Acknowledgment

A part of the research leading to results presented in this paper has been conducted within the DEWI project (FP7/ARTEMIS, 2014-2017, grant no. 621353). This research was also partially supported by the European Union's "Horizon 2020" research and innovation programme as part of the "Interoperability of Heterogeneous IoT Platforms" (INTER-IoT) project under Grant Agreement No. 687283.

References

[1] (2017) DEWI Project Homepage. [Online]. Available: http://www.dewiproject.eu

[2] (2017) DEWI Project Homepage - Objectives. [Online]. Available: http://www.dewiproject.eu/the-project-2/objectives

[3] (2017) DEWI Deliverable - D405.002 Requirements analysis/technology evaluation. [Online]. Available: http://www.dewiproject.eu/download/d405-002-requirements-analysis-technology-evaluation

[4] (2017) DEWI Use Cases. [Online]. Available: http://www.dewiproject.eu/wp-content/uploads/2016/06/DEWI_A5_final_web.pdf

[5] (2017) DEWI Deliverable - Global Glossary. [Online]. Available: http://www.dewiproject.eu/download/d102-005-dewi-gobal-glossary

[6] (2017) DEWI Project Homepage - Building Domain description. [Online]. Available: http://www.dewiproject.eu/domains/building

[7] K. Górski, M. Groth, and L. Kulas, "A multi-building WiFi-based indoor positioning system", 2014 20th International Conference on Microwaves, Radar and Wireless Communications (MIKON).

[8] (2017) ZigBee Homepage. [Online]. Available: http://www.zigbee.org

[9] (2017) GeoJSON homepage. [Online]. Available: http://geojson.org

[10] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, and K. Wasielewska, "Semantic Technologies for the IoT - An Inter-IoT Perspective", 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), Berlin, 2016, pp. 271–276.

[11] J. Angele and M. Gesmann, "Data Integration using Semantic Technology: A use case", 2006 Second International Conference RuleML'06, Athens, GA, 2006, pp. 58–66.

[12] C. Batini, M. Lenzerini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computing Surveys (CSUR), Volume 18 Issue 4, Dec. 1986, Pages 323–364.

[13] M. Kifer and E. L. Lozinskii, "A framework for an efficient implementation of deductive databases", In Proceedings of the 6th Advanced Data base Symposium, Tokyo, August (1986) 109–116.

[14] R. Ramakrishnan and D. Ullman, "A survey of deductive database systems", The Journal of Logic Programming, Volume 23, Issue 2, May 1995, Pages 125–149.

[15] (2017) ACCUS Project Homepage. [Online]. Available: https://artemis-ia.eu/project/50-accus.html

[16] (2017) OpenIoT Open Source cloud solution for the Internet of Things. [Online]. Available: http://www.openiot.eu

[17] (2017) Vital — The future of Smart Cities. [Online]. Available: http://vital-iot.eu/

[18] (2017) FIESTA-IOT – Federated Interoperable Semantic IoT Testbeds and Applications. [Online]. Available: fiesta-iot.eu/

[19] (2017) symbIoTe Symbiosis of smart objects across IoT environments. [Online]. Available: https://www.symbiote-h2020.eu/

[20] S. Staab and R. Studer, "Handbook on Ontologies", 2nd ed. Springer–Verlag, 2009.

[21] (2017) SPARQL Query Language for RDF. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query

[22] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, K. Wasielewska, and G. Fortino, "Tools for Ontology Matching - Practical Considerations from INTER–IoT Perspective", Internet and Distributed Computing Systems: 9th International Conference, IDCS 2016, Wuhan, China, September 28-30, 2016, Proceedings, Pages 296–307.

[23] (2017) Semantic Sensor Network Ontology Homepage. [Online]. https://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[24] (2017) IoT-Lite Ontology Homepage. [Online]. Available: https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126

[25] (2017) OpenIoT Ontology. [Online]. Available: http://openiot.eu/ontology/ns

[26] (2017) JENA Homepage. [Online]. Available: http://jena.apache.org/index.html

[27] (2017) Algebraix Data – Optimizing Spark for Big Data Analytics. [Online]. Available: https://algebraixdata.com

[28] (2017) 4store – an efficient, scalable and stable RDF database. [Online]. Available: https://github.com/4store/4store

[29] (2017) Redland RDF Libraries. [Online]. Available: http://librdf.org

[30] (2017) AllegroGraph – Semantic Graph Database. [Online]. Available: https://allegrograph.com/allegrograph

[31] (2017) Stardog: the Enterprise Knowledge Graph. [Online]. Available: http://www.stardog.com

[32] (2017) Blazegraph – Graph Database. [Online]. Available: https://www.blazegraph.com

[33] (2017) OrientDB Homepage. [Online]. Available: http://orientdb.com

[34] (2017) Neo4j – Graph Database. [Online]. Available: https://neo4j.com

[35] (2017) Titan: Distributed Graph Database. [Online]. Available: http://titan.thinkaurelius.com

[36] (2017) ArangoDB – highly available multi–model NoSQL database. [Online]. Available: https://www.arangodb.com

[37] (2017) JSON–LD – JSON for Linking Data. [Online]. Available: https://json-ld.org

[38] A. Iivari and J. Koivusaari, "A RESTful Sensor Data Back-end for the Internet of Things", INFOCOMP 2016, The Sixth International Conference on Advanced Communications and Computation, pp.51–55.