ONTOLOGY API FOR WEB-ENABLED FDD SYSTEM

Marko Batić, Nikola Tomašević, Sanja Vraneš
University of Belgrade, The Mihailo Pupin Institute

Abstract – *This paper focuses on the development of an Application Program Interface (API) towards the ontology based Airport Data Model (ADAM), used as a metadata layer within new generation of airport energy management and maintenance systems. Moreover, several examples of ontology API application are presented, clearly depicting its ability to serve in both locally and remotely stored ontology use case scenario. Therefore, the details describing how information is extracted, modified and inserted to the ontology are also provided. Finally, a practical procedure revealing how this API can be used for reasoning upon the stored knowledge is given.*

1. INTRODUCTION

Existing airport energy management related systems in most cases rest on the utilization of the bulk energy consumption measurements, linked to the specific area, operated by one particular power sub-station. Thus, the recorded data suffer from very poor granularity, both when it comes to the temporal resolution (usually recorded on hourly level, or less) and, what is even more important, the breakdown of energy consumption of different sub-systems serviced by specific power sub-station. Therefore, without introduction of additional measuring points, advanced Fault Detection and Diagnosis (FDD) algorithms, operating within these systems, can hardly be applied to the specific airport Significant Energy Users (SEU). However, if this requirement could be achieved, it would lead to a new kind of energy management systems, being able to provide user oriented energy saving actions, thus resulting in notable savings from the Energy Efficiency (EE) perspective. Nevertheless, current facility management systems in general, and energy management and maintenance systems in particular, are characterized by complexity of communication and control of numerous, heterogeneous, devices coming from different vendors and using different protocols. Therefore, additional measurement points would only contribute to this complexity. To rectify the situation and provide smarter, more comprehensive, airport energy management systems, description of various information/data within airport infrastructure is necessary. The main objective of such intelligent airport energy management system would be to provide more flexible data interpretation and event management, as well as advanced communication and control system capabilities, both in case of regular/operational phase and in the exceptional situation, when the efficient faults detection and diagnosis is crucial. Therefore, utilization of standardized and comprehensive *Airport Data Model* (ADAM) which will support harmonization of this diversity, as well as interoperability between different proprietary systems and devices is necessary.

At the same time, the following step in the development of energy management systems is shaped with increased use of the state of the art Semantic Web technologies which enable wider utilization of both open-source and proprietary concepts for knowledge storage and presentation. The advantage of such technologies reflects in improving interoperability and reducing the diversity of the system components, higher adaptability of management strategies and better downward and upward compatibility of technical systems/equipment and corresponding software. Moreover, an ontology based approach, as the most widely adopted Semantic Web paradigm, can be used for formal representation of the aforementioned ADAM, thus storing the knowledge through the definition of a set of concepts within airport domain and describing the corresponding relationships between them.

This paper in fact addresses the implementation of the specially tailored interfaces for one possible implementation of ADAM, using ontology based approach as main paradigm for development of the airport knowledge base repository. More precisely, since the airport ontology is used to model static knowledge regarding all relevant energy consumption related devices (Such as manufacturer data related to HVAC, lightning, fire suppression systems etc. including, e.g. specific device efficiency, communication protocols, hard- and software interfaces etc.) and equipment necessary for the operation of a FDD system , custom ontology Application Program Interface (API) is needed in order to be able to read and store data, in real-time, within ADAM.

This original API was implemented in the framework of the FP7 CASCADE project [1] which is dealing with improvement of the Energy Efficiency (EE) within European airports through the implementation of FDD algorithms. More specifically, it enables the use of the knowledge stored within the CASCADE airport ontology, which hierarchy was previously elaborated, partly in [2] and [3].

The remainder of this paper starts with Chapter 2 which introduces the state of the art of the most advanced Semantic Web technologies, containing the overview of the major ontology languages and leading ontology development tools. Following is Chapter 3 which briefly discusses the decisions taken from previous analysis and explains a wider context in which the airport data model is utilized. Moreover, the overall system architecture is described. Chapter 4 thoroughly elaborates the developed

API used to integrate the airport ontology as a metadata layer of the overall CASCADE system. Finally, Chapter 5 concludes the paper.

2. STATE OF THE ART

Ontology represents one of the frequently used Semantic Web technologies for implementation of knowledge base repositories and can be defined as a formal way of representing the knowledge as a set of concepts within a domain of interest, while also describing the corresponding relationships between those concepts. Furthermore, the ontology is utilized to describe a domain of interest by classifying and defining the related entities, but also to facilitate reasoning upon the entities. It enables definition of a common classification of all objects within a specific domain describing the basic entities, their attributes and the corresponding relationships between them. Having in mind all this, the concept of ontology is widely used in various contexts, thus enabling the sharing of common domain perception, storage of the corresponding knowledge and making it available on demand and, finally, using it for domain analysis.

The following is an overview of the state of the art ontology languages and tools that were taken into consideration for both ontology development and the corresponding API.

2.1. Major ontology languages

Different ontology languages have different expressiveness and inference mechanisms coming from diverse underlying knowledge representation paradigms. The following is an overview of the current specifications for ontology languages developed in the scope of the W3C Semantic Web Activity.

RDF and RDF Schema (Resource Description Framework) [4] - specified by the W3C as a standard for metadata describing web resources and its data model is equivalent to the semantic networks formalism, consisting of three object types: resources, properties and statements. Since the RDF data model does not have mechanisms for defining the relationships between properties and resources, the RDF Vocabulary Description language, widely known as RDF Schema, was introduced to support these functionalities.

OWL [5] - the language developed by the W3C Web Ontology Working Group based on the DAML+OIL and, as the previous languages, is intended for publishing and sharing ontologies in the Web. OWL is built upon RDF(S), has a layered structure and is divided into three sublanguages: OWL Lite, OWL DL and OWL Full.

SPARQL (SPARQL Protocol and RDF Query Language) [6] – Although it is not an ontology development language, the SPARQL supports querying these languages and therefore plays the key role in ontology API development. It allows performing queries over RDF and RDF-S data, as well as OWL ontologies. SPARQL can be used to express queries across diverse data sources and has syntax similar to SQL to facilitate its adoption.

2.2. Leading ontology tools

Wide range of tools, used for ontology management and its exploitation, cover all processes from ontology creation to their storage and/or visualization. More precisely, according to Garcia-Castro et al. [7], these tools can be divided into ontology management, querying and reasoning, ontology engineering, ontology processing and instance generation. However, in order to keep in the scope of this paper, a complete list of tools is omitted and only the ones relevant for the development of ontology API are mentioned.

Owing to its extensible architecture, user friendly environment and an open source license, for the purposes of the ontology engineering, more precisely ontology editing, Protégé™ [8] tool was utilized, allowing the creation and modification of ontologies, ontology elements and ontology documentation. When it comes to ontology management, various repositories are available and Virtuoso™ Universal Server [9] was employed to store and access ontology and its instances. Finally, in order to facilitate the feature of ontology reasoning both Pellet™ and HermiT™ were hired.

3. OVERALL SYSTEM ARCHITECTURE

In the previous section, a comparative analysis of major available ontology languages and development tools was performed in order to select the most appropriate ones for the implementation of both airport ontology and the corresponding API. Before further elaboration regarding the development of the ontology API, a brief overview of the general context in which the airport ontology is used as a meta-model and platform providing integration and interoperability between different subsystems is given.

To be able to cope with previously mentioned complex project requirements, various system architectures were examined with the main objective to enable the development of an ICT based energy management system that implements a general, standardized approach for the most typical sub-systems of the airport terminals (such as lighting, heating, cooling, ventilation and air conditioning), capable of detecting and diagnosing typical operational faults. In order of reaching these ambitious objectives, a comprehensive, multiparadigm, ICT system architecture is proposed, as given in Figure 1. This architecture provides the means for execution of all steps in the required workflow such as data acquisition, storage, processing and analysis. To summarize, all these steps can be divided into three following domains/phases:

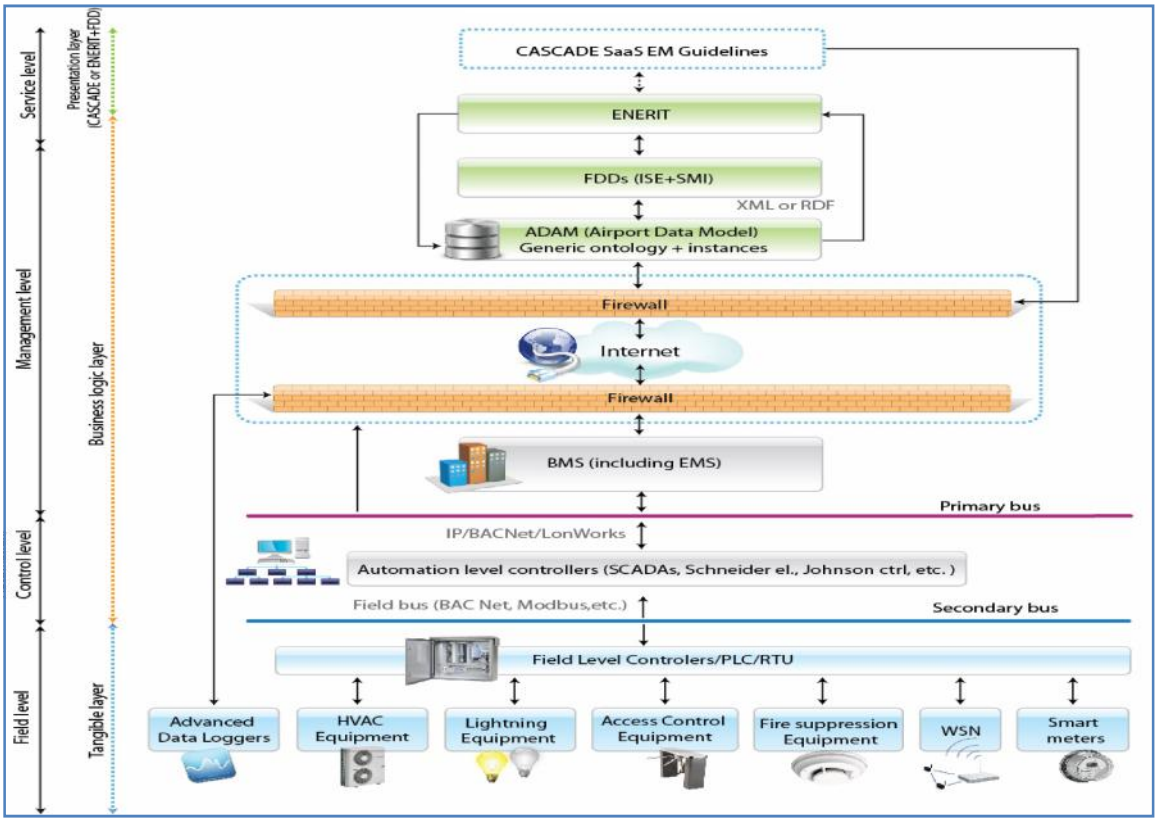

Figure 1 FDD system architecture

- **Airport Terminal systems**- Performance data, related to HVAC, lighting, cogeneration and grid exchange, is collected through the ICT framework: available BAS/BMS, ad hoc sensors and meters and installed data loggers;
- **FDD system** - Measured data is then processed and compared against specific benchmarks.
- **Action Management Module** - Whenever a fault is identified, action needs to be taken and the responsibility of this action is assigned to someone to have it solved quickly. This module represents the core and the main output of the CASCADE solution.

As depicted above, the overall system architecture is based on the ICT integration of different FDD software tools and ICT interoperation with BMS/BAS systems/sub systems that currently operate a broad range of open public spaces including airports. More precisely, the system should provide energy management actions through the use of the data extracted from existing BMS/BAS systems, as well as additional sensors and meters installed at the targeted areas of the airport. However, these data are scattered over various platforms and systems and in order to be able to extract and use them in a new software application (built atop of the existing BAS/BMSs), a metadata layer is needed to support integration and interoperability of various industrial open communication protocols such as BACnet, CAN, KNX/EIB, MODBUS and etc. This metadata layer is previously referred as Airport Data Model (ADAM) and its use is further elaborated in the following section.

Due to the different automation tasks and involved communication requirements BAS/BMS are typically divided into three functional levels - management, automation and field level. Starting from the field level, where the measurement data are collected via standardized protocols using both wired and/or wireless transmission, additional advanced smart meters are installed, to enable both parallel and supervisory data logging of the minimal data set required to process a FDD based energy management. Data is then transferred in real time via firewall interfaces and a secure channel to the data server (metadata layer) where they are imported, filtered, aggregated and reasoned upon, according to an original Airport Data Model (ADAM). The FDD software tools have huge benefits from the metadata layer, since it provides a semantic interpretation of the signals coming from the data logging system and shares the common perspective of the system towards the two instances of the FDD component. Furthermore, information and knowledge stored within the ADAM will also have a high impact on the web interface of the client oriented energy management system, responsible for graphical visualization of the system recommendations, since it will enrich the energy management action messages with high level information such as geographical location of underperforming devices, identification to which system/sub-system a specific device belongs to and etc. The following chapter discusses in more detail the functionalities of ontology API and its development utilizing the state of the art semantic web technologies.

4. ONTOLOGY API

As described previously, the ADAM layer consists of both knowledge base repository, i.e. the ontology itself, and the corresponding interface towards the components which have benefits from retrieving the information stored within. The common interface between the ontology and the rest of the system components is referred to as Ontology API. The Ontology API has several important functionalities, necessary for providing a transparent interface for the integration of the knowledge base repository and other software components within a CASCADE system, as listed:

- **SPARQL Query generation** - Enables generation of SPARQL queries based on the predefined argument, given in any desired format such as integer, floating-point numbers, characters and etc.
- **SPARQL Update generation** - Enables generation of SPARQL update commands, including MODIFY, INSERT and DELETE, based on the predefined argument, given in any desired format such as integer, floating-point numbers, characters and etc.
- **Running the Queries and Updates** - Enables appliance of the SPARQL query and update commands to the corresponding ontology.
- **Reasoning** - Enables setting of the generic rule engine capable of inference on the ontology based on the set of pre-defined rules.
- **Conversion of the results/output** - Enables export of the knowledge/information/data retrieved from the ontology in ant desired format such as integer, floating-point numbers, characters and etc.

From the technical perspective, the ontology API is exported in JARs and delivered as a single library which has all the functionalities implemented within, thus achieving the seamless integration with the rest of software components. Another important functionality of the Ontology API is that it provides all previously mentioned features both in the case when the ontology is stored locally as well as when it is stored on a remote server. The remaining of the chapter describes in detail the implementation of both use cases.

4.1. Local access

In the case when different components do not have to share the same instance of ontology, a solution of having the ontology stored "locally" (ontology stored at the side of the end-user) was chosen in order to avoid unnecessary communication, thus increasing the reliability and providing more robust prototype.

Since the relevant system components were implemented as Java applications, interface towards the ontology was also developed in Java. Therefore, for the purposes of the implementation of the Ontology API, a Java framework for building Semantic Web applications called Apache Jena™ [10], which completely matches the specification of the CASCADE Ontology API, was utilized.

Several practical examples describing the main features of the ontology API application in the case of the locally stored ontology, are given in the following along with the corresponding code snippets:

1) *Technical characterization and semantic interpretation of fault detection signals* – Enables device characterization based on the unique device ID (e.g. "P4.1") found in the incoming signal from sensors and extraction of its relevant technical characteristics such as device power, pressure and water flow (?power, ?pressure, ?flow), to which system it belongs (?sysID), to which devices and components it is connected to(?devID ?compID) and where it is located (?areaID).

```
// SPARQL Query
queryStr.append(
"SELECT ?power ?pressure ?flow ?areaID ?sysID ?devID
?compID "+
"FROM <http://localhost:8890/OntologyTesting> "+
"WHERE { "+
          "?srdf:typepref:waterPump. " +
          "?spref:device_id \"" + deviceID +
"\"^^xsd:string. " +
          "?spref:outputPower_kW ?power. " +
          "?spref:hydraulicPressure_kPa ?pressure. " +
          "?spref:waterFlow_l_s ?flow. " +
          "?spref:locatedAt_area ?area. " +
          "?areapref:area_id ?areaID. " +
          "?spref:partOf_system ?system. " +
          "?systempref:system_id ?sysID. " +
          "?spref:connectedTo ?connectedToDevice. " +
          "?connectedToDevicepref:device_id ?devID. " +
          "?spref:connectedTo ?connectedToComp. " +
          "?connectedToComppref:component_id ?compID. " +
          "}");
Query query=QueryFactory.create(queryStr.toString());
QueryExecution
qe=QueryExecutionFactory.create(query,model);
```

2) *Updating the knowledge base repository* – Enables update of specific entity property based on the received information/data from sensors. Update arguments are considered to be both unique device identifier and new property value.

```
// SPARQL Update - To replace a property first delete
current value and then insert new one
queryStr.append(
"DELETE { " +
"?spref:waterFlow_l_s ?o. "+ "} " +
"WHERE { " +
"?spref:device_id \"" + id + "\"^^xsd:string. " +
"?spref:waterFlow_l_s ?o. " + "} " +
"INSERT { " + "?spref:waterFlow_l_s \"" + flow +
"\"^^xsd:float. " + "} " +
"WHERE { " + "?srdf:typepref:waterPump. " +
"?spref:device_id \"" + id + "\"^^xsd:string. " + "}");
// Update execution
UpdateAction.parseExecute(queryStr.toString(),model);
```

3) *Processing of acquired data* – In case the FDD component detects that one AHU device is malfunctioning (e.g. ?seu1, AHU_Failure2) and another AHU device is transmitting irregular energy performance indications (e.g. ?seu2, AHU_EnPerfLow1), it could be reasoned whether these AHUs are installed within a same area (?area1), and it could be concluded that AHU device with irregular EnPIs is still operable since area is regulated by reduced number of AHUs.

```
// SPARQL Query
queryStr.append(
"SELECT ?devID1 ?devID2 ?areaID1 "+
"FROM <http://localhost:8890/OntologyTesting> "+
"WHERE {"+
"?signal1pref:signaled"+ AHU_Failure2+ "\"^^xsd:string.
" +
"?signal2pref:signalID" +AHU_EnPerfLow1+
\"^^xsd:string." +
"?devID1pref:readingSignal ?signal1. " +
"?devID2pref:readingSignal ?signal2. " +
"?devID1pref:locatedAt ?areaID1." +
"?devID2pref:locatedAt ?areaID2. " +
"FILTER (sameTerm(?areaID1, ?areaID2) &&
!sameTerm(?devID1, ? devID2)}");
// Update execution
UpdateAction.parseExecute(queryStr.toString(),model);
```

4) *Applying a generic inference engine* – In order to maintain consistent information within the ontology, a set of rules between entities and their corresponding properties can be developed. These rules are further transferred towards the inference engine which is responsible for reasoning upon the ontology instances. In this example, it is shown how to transfer the information about the system a specific component belongs to, if there is already such information about a device which this component constitutes. In this case, property value (partOf_system) for a specific device is used to create a new triplet which suggests which system a component belonging to this device (partOf_device) is part of.

```
PrintUtil.registerPrefix("pref",this.defaultNameSpace);

String ruleSrc="[rule1: (?xpref:partOf_device ?y) (?y
pref:partOf_system ?z) -> (?x pref:partOf_system ?z)]";

GenericRuleReasonerreasoner=
newGenericRuleReasoner(Rule.parseRules(ruleSrc));
InfModelinf=ModelFactory.createInfModel(reasoner,
model);
```

4.2. Remote access

On the other hand, if the ontology is accessed remotely, i.e. in the scenario when system components are not located at the same place but need to operate upon exactly the same ontology, there is a need for different approach and the corresponding technologies. For the purposes of storing the ontology, the Virtuoso™ Universal Server was utilized. Virtuoso is an innovative enterprise grade multi-model data server which delivers a platform agnostic solution for data management, access, and integration.

Having in mind the Virtuoso functionalities, querying the ontology can easily be performed in a web-service based manner (by sending the requests in order to query the ontology) and the extracted information is usually wrapped up into the XML based message, sent as a response to the end-user. However, in order to avoid building the corresponding XML schema and manually creating the XML messages each time the communication is requested, one could take advantage of the built-in SPARQL end point within Virtuoso Server. This would enable user to have transparent and utterly simple communication with the server using only the HTTP protocol. Nevertheless, in this case the response from query is received in the form of XML and requires further parsing in order to store the information appropriately. Therefore, further effort was put in finding a way of utilizing the technologies that would simplify this procedure.

Again, Jena framework was used in order to create queries and, more importantly, to initiate a communication with the Virtuoso server. However, during this research it was revealed that only the SPARQL Query language is supported in this way, whereas if the SPARQL Update commands are needed, an original solution using the HTTP protocol must be employed. The following code snippets describe in detail the implementation of Ontology API that provides access to the remotely stored ontology.

1) *SPARQL Query (Jena)* – In order to query a remotely stored ontology from Java application one option is to use Jena framework to access the Virtuoso SPARQL end-point, which is actually implemented as a web-service. Fortunately, Jena provides all the necessary communication, such as the creation of the XML messages that are actually running between the user and server, and makes it completely transparent for the user.

```
String service
="http://fraunhofer2.imp.bg.ac.rs/sparql";
// SPARQL Query
queryStr.append("SELECT ?power ?pressure ?flow ?areaID
?sysID ?devID ?compID " +
"FROM <http://localhost:8890/OntologyTesting> " +
"WHERE { " +
"?srdf:typepref:waterPump. " +
"?spref:device_id \"" + deviceID + "\"^^xsd:string. " +
"?spref:outputPower_kW ?power. " +
"?spref:hydraulicPressure_kPa ?pressure. " +
"?spref:waterFlow_l_s ?flow. " +
"?spref:locatedAt_area ?area. " +
"?areapref:area_id ?areaID. " +
"?spref:partOf_system ?system. " +
"?systempref:system_id ?sysID. " +
"?spref:connectedTo ?connectedToDevice. " +
"?connectedToDevicepref:device_id
?devID. "+"?spref:connectedTo ?connectedToComp. " +
"?connectedToComppref:component_id ?compID. " + "}");
String query =queryStr.toString();
QueryExecutionqe=QueryExecutionFactory.sparqlService(se
rvice, query);
```

2) *SPARQL Query (HTTP)* – A second ranked option for querying a remotely stored ontology is via HTTP protocol, in which case a POST request method is utilized and the query itself is embedded within it. The main disadvantage of this approach, comparing to the one which utilizes Jena framework, is that it requires parsing of the response which is received in XML format. Therefore, manual post processing is needed in order to extract and store the information appropriately.

```
// SPARQL Query
queryStr.append(
"DELETE FROM <http://localhost:8890/OntologyTesting> "
+ "{ ?s pref:waterFlow_l_s ?o. } " +
"WHERE { " + "?spref:device_id \"" + deviceID +
"\"^^xsd:string. " +
"?spref:waterFlow_l_s ?o. " +"} " +
"INSERT INTO <http://localhost:8890/OntologyTesting> "+
"{ ?s pref:waterFlow_l_s \"" + flow + "\"^^xsd:float.}"
+ "WHERE { " + "?srdf:typepref:waterPump. " +
"?spref:device_id \"" + deviceID + "\"^^xsd:string. ");
String queryStr="query="+URLEncoder.encode(query,"UTF-
8");
...
```

```
...

StringBuffersb=newStringBuffer();
try{String
queryString="query="+URLEncoder.encode(queryStr,"UTF-
8");
HttpURLConnection conn =null;
//String type = "text/plain";
URL url=
newURL("http://fraunhofer2.imp.bg.ac.rs/sparql");
conn=(HttpURLConnection)url.openConnection();
conn.setRequestMethod("POST");
conn.setDoOutput(true);
OutputStreamWriter
os=newOutputStreamWriter(conn.getOutputStream());
os.write(queryStr);
os.flush();
os.close();
// Get the response
BufferedReaderrd=newBufferedReader(newInputStreamReader
(conn.getInputStream()));
String line;
String NL =System.getProperty("line.separator");
while((line =rd.readLine())!=null){
sb.append(line + NL);
}
        rd.close();
}catch(Exception e){
        e.printStackTrace();
}
```

3) *SPARQL Update* – In case of the SPARQL Update commands, Jena framework does not provide support for access of remote web-service, and the only option is to use HTTP protocol. In this case, the procedure is almost identical as sending the SPARQL Query (HTTP) requests except that, because of security reasons, the URL has to include access credentials since the Virtuoso server requires authorization when any of the SPARQL Update commands are issued and they are, consequently, affecting the stored ontology.

```
URL url=
new URL("http://user:pass@fraunhofer2.imp.bg.ac.rs/
sparql-auth");
```

5. CONCLUSION

A new kind of airport energy management systems, able to provide user oriented energy saving actions utilizing advance Fault Detection and Diagnosis (FDD) algorithms to separate energy consumption systems, were addressed. Owing to the complexity of communication and control of numerous, heterogeneous devices coming from different vendors and using different protocols these systems require special metadata layer which will enable transparent integration and interoperability of different ICT systems operating in the airport environment. However, the previously developed Airport Data Model (ADAM) which serves as an ontology based metadata layer requires a corresponding Application Program Interface (API) towards existing software platforms in order to be able to utilize to stored knowledge. Therefore, having developed various important functionalities such as SPARQL Query generation, SPARQL Update generation, executing the queries and updates and reasoning over the ontology instances, hereby presented ontology API fully matches the specified requirements. Moreover, the ontology API provides all these features in case when the ontology is stored locally as well as it is stored on a remote server. The future work will be focused on the more generalized approach expanding the current data model and corresponding API beyond the framework of airport facilities.

ACKNOWLEDGEMENTS

The research presented in this paper is partly funded by the European Commission within the 7th Framework Programme (CASCADE project, Contract No. : 284920), and partly by the Serbian Ministry of Science and Technological Development (SOFIA project, Contract No.: TR-32010).

REFERENCES

[1] CASCADE project - Description of Work, EU Commission 7th Framework Research Project, Available: http://www.cascade-eu.org/

[2] Nikola Tomašević, Marko Batić, Gordan Konečni, Sanja Vraneš, *Ontology-based Airport Data Model*, ICIST 2012, 29.02-03.03.2012, Kopaonik, Serbia

[3] Marko Batić, Nikola Tomašević, Dejan Paunović, Sanja Vraneš, *A Novel Approach to Microgrid Data Modelling*, ICIST 2012, 29.02-03.03.2012, Kopaonik, Serbia

[4] Klyne G., and Carrol J. (eds.) (2004) "Resource Description Framework (RDF) Concepts and Abstract Syntax". W3C Recommendation 10 February 2004, Available: http://www.w3.org/TR/rdf-concepts/

[5] Dean, M. and Schreiber, G. (eds.) (2004) "OWL Web Ontology Language Reference". W3C Recommendation 10 February 2004, Available: http://www.w3.org/TR/owl-ref/

[6] Prud'hommeaux, E. and Seaborne, A. (eds.) (2008) "*SPARQL Query Language for RDF*" W3C Recommendation 15 January 2008. ,http://www.w3.org/TR/rdf-sparql-query/

[7] García-Castro, R., Muñoz-García, O., Gómez-Pérez, A., Nixon, L. (2009) "Towards a component-based framework for developing Semantic Web applications". *3rd Asian Semantic Web Conference (ASWC 2008)*. 2-5 February, 2009. Bangkok, Thailand

[8] Protégé tool, Available: http://protege.stanford.edu/

[9] Virtuoso™ Universal Server, Available: http://virtuoso.openlinksw.com/

[10] Apache Jena, Available: http://jena.apache.org/index.html