Unequal Temperature Changes in City: A Case Study Using a Semantic IoT Platform

Maxim Kolchin, Nikolay Klimov, Ivan Shilin, Daniil Garayzuev, Alexey Andreev
ITMO University
Saint-Petersburg, Russia
{kolchinmax,nikolay.klimov}@niuitmo.ru, {shilinivan,garayzuev,a_andreev}@corp.ifmo.ru

***Abstract*—Number of connected devices grows much faster than their manufacturers and customers agree on standards which could realise addressing, identification, discovery, composition, security and privacy of these devices. A result of such situation is a lack of interoperability among different clusters of connected devices and systems. This is one of the known research problems in Internet of Things. At the same time knowledge engineering methods and Semantic Web technologies proofs that they could be a solution to this problem. In this paper we present an architecture of a platform built using aforementioned technologies and a case study demonstrating the middleware in analysing measurements of air temperature in city.**

I. Introduction

According to recent Gartner's report, "6.4 billion connected things will be in use worldwide in 2016, up 30 percent from 2015, and will reach 20.8 billion by 2020" [1]. Probably no one will have access to all connected devices in the world, however there are use cases where the number of connected devices may be relatively small, but they support different communication protocols and data representation models [2]. It causes lack of interoperability at *technical* (when systems can't communicate to each other), *syntactic* (when systems can't work with each others' data models) and *semantic* (when systems don't understand the meaning of each others' terms) levels. At the same time IoT standards mainly target concrete domains and systems which means that in the short run the vertical integration of sensors and business services will dominate IoT. However "interoperability between domains and systems is a key factor for sustainable success of IoT" [3].

Technical and syntactic interoperability can be achieved with a modular architecture of a gateway or a middleware which collect data from connected devices within a room or building and area or city respectively. And the use of ontologies is a possible approach to achieve semantic interoperability [4]. Ontologies and Semantic Web technologies proofs to be a suitable instrument for this task. Wache et al. [5] identifies three approaches to use ontologies for semantic interoperability:

- single ontology approach, this approach uses one global ontology providing a shared vocabulary for the specification of the semantics,
- multiple ontology approach, in this approach each information source is described by its own ontology,
- hybrid ontology approach, in this approach its own ontologies extends a shared ontology to use the integration.

In this paper we present an architecture of the SemIoT Platform which aims to provide a prototypical solution to interoperable access to the data of heterogeneous connected devices. The platform employee the OSGi approach [6] to a modular architecture and hybrid ontology approach to representation of device's data. In addition we describe two scenarios using the platform to collect and analyse data from different temperature sensors in Saint-Petersburg and Moscow and a web application to visualise the results.

A. Related work

Around ten years ago C. Thompson [7] envisioned Semantic Web as a key enabler of Internet of Things. Since then Semantic Web has grown from just an idea to a mature technology stack which is currently used in different domains. Semantic Web research targets two main areas related to IoT: ontologies and software solutions (namely gateways and middlewares) employing these ontologies for modeling and annotating device's data.

1) Ontologies: One of well-known ontologies related to Internet of Things domain is Semantic Sensor Networks (SSN) ontology [8] that covers the concepts of System, Sensor, Measurement capabilities, Observation, Deployment etc. It has been actively used in number of projects for publishing and analysing sensor observations [9], [10]. DogOnt ontology [11] includes concepts for description of house, its architectural elements, domestic devices, their states and functionalities. This ontology was primarily developed for an ontology-powered gateway [12]. These two ontologies only a small subset of existing ones, more detailed reviews of existing IoT ontologies can be found in [13], [14] papers. Number of IoT ontologies have been created already, but most of them were designed for specific projects, so they need to be modified to fit a new project, and the others are still under development and not ready for applying outside of research projects. In this work we use and extend the SSN ontology.

2) Gateways & middlewares: Gateways and middlewares are crucial parts of Internet of Things architecture since they provide access to the data of connected devices which don't have enough resources to serve clients over Internet or don't support respective protocols. They can cache and transform the data to a needed form, enable discovery and advanced querying. In the scope of OpenIoT project ended in 2015, a semantic sensor middleware was developed [15]. It has an extendable architecture which support configurable wrappers collecting data from heterogeneous devices. These wrappers supports various protocols and new ones can be added. The

observations collected from sensor are semantically annotated using the SSN ontology. In addition it provides visual and analytical tools for working with sensor data. The Dog gateway supports sensors as well as actuators in a domestic environment [12]. It uses the OSGi architecture to work over different communication protocols and OWL ontologies to model an environment and devices. The Smart-M3 Platform which implements "an information hub for agents of a given environment" [16] was applied to a smart room and e-tourism use cases.

In addition to the works done mainly by research community, there are also commercial frameworks for Internet of Things which was surveyed in [17].

B. Structure

The remainder of the paper is organised as follows. In Section II we describe two scenarios which demonstrate the main capabilities of the platform. We provide an overview of the architecture of the SemIoT Platform in Section III. And in Section IV we describe specific technical details of implementation of the given scenarios. In Section V we conclude and describe lessons learned from this case study and our future vision and work.

II. Scenarios

The whole IoT sphere provides huge variety of different use case scenarios, even in rather wide-scale fields, such as environment monitoring of particular cities. Most of that cases are available thanks to crowd-sourcing effect and particular enthusiasts, that deploy different types of sensors on their apartments and make them accessible through the Web. In this paper we work with two scenarios related to air temperature in two Russia cities: Moscow and Saint-Petersburg. There are a bunch of online-based services, that provide information about such crowd-sourced sensors, located in Russia, in real or near-real time, but the most popular are two of them. First, NarodMon (http://narodmon.ru), provides a set of observations from different types of sensors - from humidity and air pressure to luminosity. In the case of checking air temperature, as it was found, described a non-representative count of sensors (only about 15 in Saint-Petersburg). Second, Netatmo (http://netatmo.com) is an international manufacturer of different kinds of portable weather stations; through it's regular HTTP API we can retrieve temperature from around 400 in Moscow and 160 in Saint-Petersburg, that is quite enough to perform some regular analysis, that can be associated with the whole city or it's regions without huge affections by occasional fluctuations. Described below scenarios are a bit academic; the main reason of them and developed demo application is to demonstrate the abilities of the SemIoT Platform to perform data annotation, aggregation and real-time analysis, that, thanks to the OSGi architecture and abstraction level of device drivers, can be applied to more complex and narrow scenarios, that can be valuable for end-user.

Scenario A: Unequal temperature values

Thinking of a city in general, it is often desirable to observe some parameters in dynamics. E.g. in case of big cities the air temperature may not be the same among different city areas which depends on the size of a particular city. In this scenario we aim to visualise the air temperature values in city to show how they differ depending of an area. If an area is bordered by a sea or a big lake, then it's air temperature normally lower comparing to areas removed from a big water.

To provide such a visualisation we construct a map of "relativelly colder" and "relativelly warmer" areas. In addition it's possible to navigate through the history of the observations to create an understanding of "microtrends" based on particular city areas' temperature values.

Scenario B: Unequal temperature changes

In this scenario in addition to the current temperature value we include the previous value to calculate the difference. It may help to see in which areas the air temperature grows (or drops) faster and slower comparing to the growth (or drop) of the average temperature in the whole city. Such derivative of the parameter is demonstrated in this work by constructing the map of air temperature changed, indicating, in which area temperature changed the most from the analysed average difference for captured temperature snapshot.

III. Architecture of the platform

The architecture of the platform was developed to meet two main high-level requirements:

- support of different communication protocols and data models for collecting data from connected devices, and ease extension for a new protocol and model;
- access to the data of the devices through a unified interface, so the data could be accessed and queried regardless of the data model supported by a particular devices.

The first requirement was decided to meet by using an approach based the OSGi architecture where a system consists of a set of independent services (called *bundles*) having independent lifecycle, so they can be started or stopped without effecting the other services, but only if they don't directly depend on this service. This approach allows to implement an architecture where a different types of devices supported by a dedicated service, we call such services *device drivers*. A driver is an OSGi service implementing a special interface and using a module called *Driver Manager* to provide the data collected from devices.

The second requirement is met by using Semantic Web technologies such as RDF, OWL and SPARQL. RDF provides And in Section IV we describe specific technical details observations and OWL allows to create sharable ontologies for annotating the data and unifying its semantics. SPARQL provides an advanced querying language with extensions for geographical (GeoSPARQL extension [18]) and streaming (CQELS [19] and C-SPARQL [20] extensions) data.

The architecture of the Platform is presented on Fig. 1. It consists of 5 modules, 2 database and device drivers:

- *Device drivers* are OSGi services implementing an algorithm of data collection from connected devices over some communication protocol and data model.

E.g. HTTP API and JSON-based model. The drivers are installed, uninstalled and configured by *Configuration Manager*.

- *Driver Manager* is an OSGi service responsible for collecting the data from device drivers and passing it to *Metadata Writer* and *Observation Writer*.
- *Metadata Writer* is a module that manages metadata of the devices, such as structure, measuring capabilities, location and etc. The metadata is stored in *Triplestore*.
- *Triplestore* is an RDF database which provides a SPARQL endpoint and implements the GeoSPARQL extension for querying devices by their location. In the current implementation we use Apache Fuseki.
- *Observation Writer* is a module responsible for managing observations of the devices. They are stored in *TSBD*.
- *TSBD* is a time series database optimised for storing time-based data such as sensor observations. In the current implementation we use OpenTSDB (http://opentsdb.net).
- *API Gateway* is a REST-like programming interface for accessing data of the devices and accessing *Configuration Manager*.

The source code of the platform is open source and published on Github (https://github.com/semiotproject/semiot-platform).

A. Data collection layer

As described above, to support different types of connected devices we a notion of device drivers which implements a data collection algorithm specific to a particular device type. Since each driver is an OSGi service, it's a JAR file containing:

- `MANIFEST.MF` file includes the service's metadata and its dependencies;
- `metatype.xml` describes the settings required to run a driver;
- RDF templates that are used to annotate the metadata and the observations by filling placeholders predefined by the developer of a driver;
- `Activator` class which is instantiated during the start;
- and other classes necessary for the implementation of the particular algorithm.

The drivers register itself with Device Manager and periodically or based on new data notify Device Manager about existence of new data which should be passed to the other modules of the platform. All modules except the drivers work already with annotated data, i.e. with RDF data.

B. Ontologies

In this work we follow the *hybrid ontology approach* which presumes to use a shared ontology with combination of own ontologies extending it. The SSN ontology [8] plays

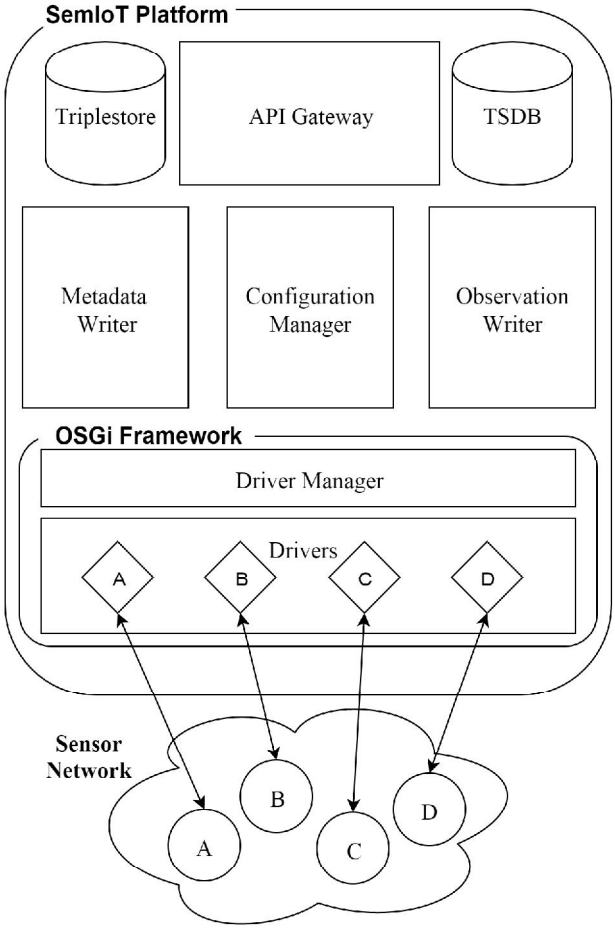


Fig. 1. An architecture of the SemIoT Platform

TABLE I. Prefixes of ontologies

Prefix	URI
ex	http://example.com/
qudt-quantity	http://qudt.org/vocab/quantity#
qudt	http://qudt.org/schema/qudt#
lgdo	http://linkedgeodata.org/ontology/
ssn	http://purl.oclc.org/NET/ssnx/ssn#
rdfs	http://www.w3.org/2000/01/rdf-schema#
geo	http://www.w3.org/2003/01/geo/wgs84_pos#
ssncom	http://purl.org/NET/ssnext/communication#
dul	http://www.loa-cnr.it/ontologies/DUL.owl#
schema	http://schema.org/

the role of a shared ontology, so all the modules expects the data of the devices to be annotated with this ontology concepts, such as `ssn:System`, `ssn:SensingDevice`, `ssn:Observation` and etc. In Fig.2 RDF data about a Thermometer device is presented. This RDF description says that `ex:Thermometer-1` is a device which has a label, an identifier, location and a sensor which measures a temperature in Celsius. In a similar way the observation of the devices are described.

All prefixes of the ontologies mentioned in the paper are listed in table I.

```
ex:Thermometer-1 a ssn:System ;
 rdfs:label "Temperature Meter #1"@en ;
 dcterms:identifier "1"^^xsd:string ;
 dul:hasLocation [
  a geo:Point ;
  geo:lat "60.023"^^xsd:string ;
  geo:long "34.4"^^xsd:string ;
  geo:alt "10"^^xsd:string
 ] ;
 ssn:hasSubSystem ex:Thermometer-1-Temperature .

ex:Thermometer-1-Sensor a ssn:SensingDevice ;
 dcterms:identifier "1-temperature"^^xsd:string ;
 ssn:observes qudt-quantity:ThermodynamicTemperature ;
 ssn:hasMeasurementCapability [
  a ssn:MeasurementCapability ;
  ssn:forProperty qudt-quantity:ThermodynamicTemperature ;
  ssn:hasMeasurementProperty [
  a qudt:Unit ;
  ssn:hasValue [
    a qudt:Quantity ;
    ssn:hasValue qudt-unit:DegreeCelsius ;
  ] ;
  ] ;
 ] .
```

Fig. 2. RDF description of a thermometer

C. Data analysis layer

The platform itself doesn't have any tool for analysis of the data collected from connected devices, but it exposes interfaces which are necessary for that. These interfaces allows to use the RDF Stream Processing (RSP) approach [19], [20] for querying the streaming and static data which is implemented in CQELS and C-SPARQL engines. The RSP engines use SPARQL extensions with operators that can handle the streaming and static data at the same which provides an advanced analytical tool.

For analysis of the device data it's possible to use the both engines, because each of them implements different query semantics. For instance, CQELS associates a named (time-varying) graph to each window in the query, and the window content is accessed with the STREAM clause, analogous to the GRAPH in SPARQL. However, it is not possible to declare the sliding window in such a way that its content is included in the default graph of the dataset. On the contrary, C-SPARQL does not allow to name the time-varying graphs computed by the sliding windows, but all the graphs computed by the sliding windows are merged and set as the default graph [21]. In other words, CQELS return result for every triple, which matched by query, but C-SPARQL return all triples, which matched by query, when time window is closed.

In this case studies we use the C-SPARQL engine, because we want to compute the results periodically for the data from all devices, but not each time new data arrived from a single device.

IV. Case study

As described in Section II in this case study we work with two similar scenarios which are used to show the capabilities of the platform. For this case study a web application was developed to visualise the data from temperature sensors deployed in Moscow and saint-Petersburg, on Fig. 3 an interface of this application is presented. The application has a map, controls to switch between the scenarios and the cities, and a button to start the visualisation the previous snapshots of the map.

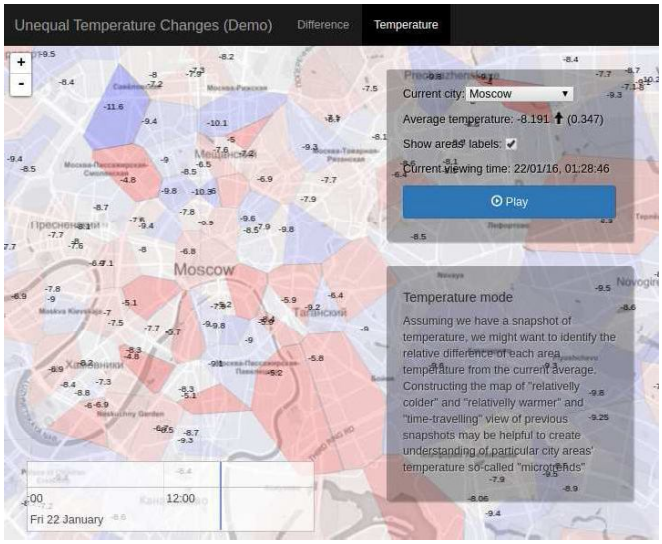


Fig. 3. Web interface of the visualisation tool

The city is divided into areas which are identified by sensors deployed in the city, we use the Voronoi diagram [22] to visualise these areas on the map. And area has a color which donates its temperature or change of its temperature within a period.

The web application is available online on http://w3id.org/semiot/demos/temperature-in-city.

Scenario A: Unequal temperature values

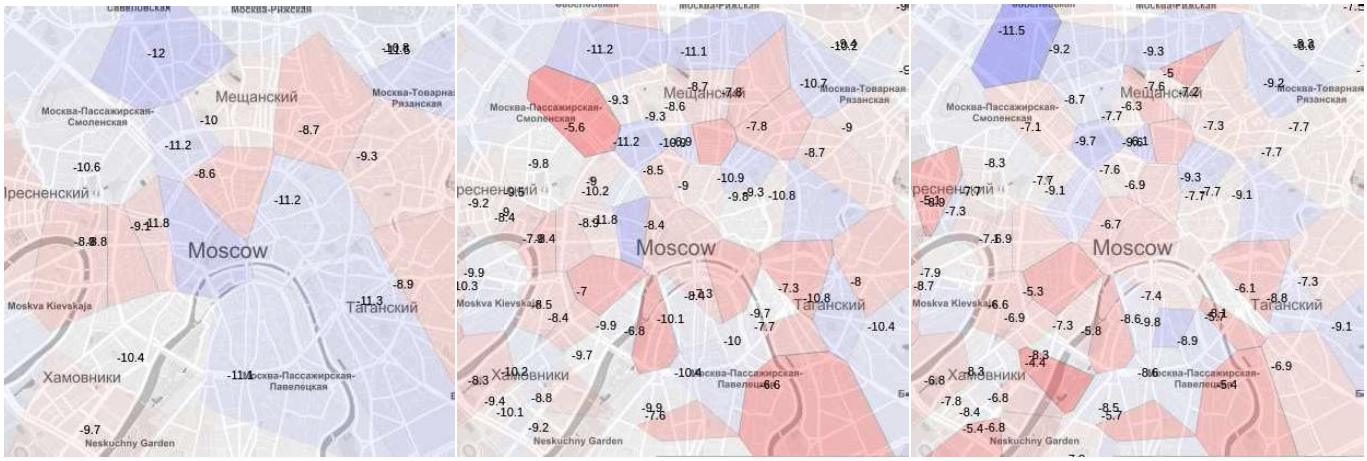
Comparing to Scenario B, this scenario is quite simple, the application requests the list of devices deployed in a city, requests their last observations and then visualise them on the map. To get the relevant devices we need to know the borders of a city, therefore we query an external SPARQL endpoint which has the necessary information. In Fig. 4 RDF descriptions of the both cities are presented, where we can see the geo-coordinates and the radius of circles which represent the borders.

```
ex:city/Moscow a lgdo:City ;
    rdfs:label "Moscow"@en ;
    schema:geoRadius "21" ;
    geo:lat "55.7516" ;
    geo:long "37.6187" .

ex:city/Saint-Petersburg a lgdo:City ;
    rdfs:label "Saint-Petersburg"@en ;
    schema:geoRadius  "16.5" ;
    geo:lat "59.9394" ;
    geo:long "30.3154" .
```

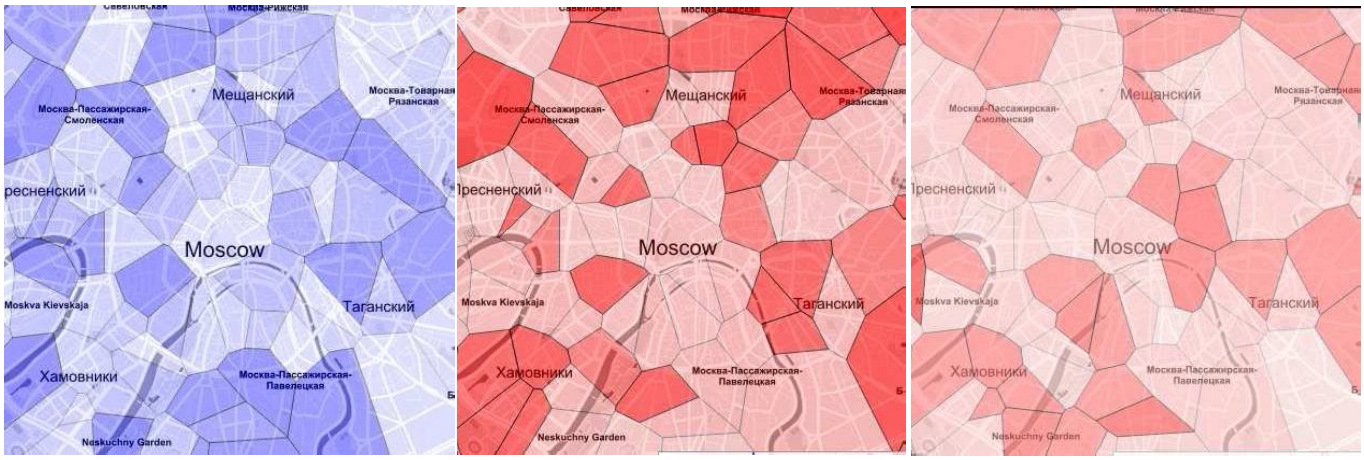
Fig. 4. RDF descriptions of Moscow and Saint-Peterburg cities

The resulting visualisations for this scenario are presented on Fig. 5, these three figures are snapshots of the map at



(a) Screenshot in time T (b) Screenshot in time T + *2 hours* (c) Screenshot in time T + *4 hours*

Fig. 5. Snapshots of the map with the current air temperature values in Moscow at different time points



(a) Screenshot in time T (b) Screenshot in time T + *30 minutes* (c) Screenshot in time T + *1 hour*

Fig. 6. Snapshots of the map with the air temperature changes in Moscow at different time points

different time. As it can be seen, the areas on the maps have different colors which represent the difference from the average air temperature among the areas. More red areas are *warmer* than in the average and more blue ones are *cooler*. The map is updated each a new temperature value is received from a temperature sensor.

This scenario helps us on a quite simple example observe how the platform behaves with a relatively big number of connected devices.

Scenario B: Unequal temperature changes

In contrast to the previous scenario, in this scenario we use the RSP engine to compute the temperature changes within a time period. In Fig. 7 the C-SPARQL query which computes the change of the temperature for each sensor located in the given city, then computes the average change among the sensors to get the common ”trend” and groups these sensors by their change. The query has includes the following parts:

- the 1st *(SELECT)* subquery requests from the external SPARQL endpoint the border of the city for the given city name,
- the 2d *(SELECT)* subquery requests the sensors which are located within the circle which bounds the city,
- the 3d *(SELECT)* subquery computes the average change among the sensors,
- the next triple patterns select the change for each sensor,
- and at the end of the C-SPARQL query each sensor is assigned to one of the two groups.

The same query is used for Saint-Petersburg, the only change which needs to be done is at 16th line where we set the name of a city.

```
REGISTER QUERY test AS
CONSTRUCT {
 ?sensor a ex:Diff ;
  ex:hasDiff ?diff ;
  ex:hasAbsTemp ?v1 ;
  ex:hasAbsAvg ?absAvg ;
  ex:hasAvg ?avg ;
  ex:hasTime ?time1 ;
  ex:InGroup ?group .
}
FROM STREAM <urn:examples:test> [RANGE 4m STEP 2m]
WHERE {
{SERVICE <http://external.sparqlendpoint/sparql> {
 SELECT ?lat ?long ?rad WHERE {
  ?xza a lgdo:City ;
   rdfs:label "Moscow"@en ;
   geo:long ?long ;
   geo:lat ?lat ;
   schema:geoRadius ?rad .
 }
}
SERVICE <http://triplestore/sparql> {
 SELECT ?sensor ?lat ?long ?rad WHERE {
  ?loc spatial:nearby (?lat ?long ?rad 'km') .
  ?xaz dul:hasLocation ?loc ;
   ssn:hasSubSystem ?sensor .
 }
}
{SELECT (AVG(?d) AS ?avg) (AVG(?v2) AS ?absAvg) WHERE {
 ?o1 a ssn:Observation ;
  ssn:observedBy ?sensor ;
  ssn:observedProperty
   qudt-quantity:ThermodynamicTemperature ;
  ssn:observationResultTime ?t1 ;
  ssn:observationResult/ssn:hasValue/qudt:quantityValue ?v1 .
 ?o2 a ssn:Observation ;
  ssn:observedBy ?sensor ;
  ssn:observedProperty
   qudt-quantity:ThermodynamicTemperature ;
  ssn:observationResultTime ?t2 ;
  ssn:observationResult/ssn:hasValue/qudt:quantityValue ?v2 .
 FILTER(?o1 != ?o2 && ?t1 > ?t2)
 BIND (?v1 - ?v2 AS ?d)
}}
}

?obs1 a ssn:Observation ;
 ssn:observedBy ?sensor ;
 ssn:observedProperty
  qudt-quantity:ThermodynamicTemperature ;
 ssn:observationResultTime ?time1 ;
 ssn:observationResult/ssn:hasValue/qudt:quantityValue ?v1 .
?obs2 a ssn:Observation ;
 ssn:observedBy ?sensor ;
 ssn:observedProperty
  qudt-quantity:ThermodynamicTemperature ;
 ssn:observationResultTime ?time2 ;
 ssn:observationResult/ssn:hasValue/qudt:quantityValue ?v2 .

FILTER(?obs1 != ?obs2 && ?time1 > ?time2)
FILTER(?v1 < ?absAvg + 5 && ?v1 > ?absAvg -5)
BIND (?v1 - ?v2 AS ?diff)
BIND (IF(?diff > ?avg, 1, 2) AS ?group)
}
```

Fig. 7. C-SPARQL query used in Scenario B

The resulting visualisations for this scenario are presented on Fig. 6, these three figures are snapshots of the map at different time points. Fig. 6a shows that in average the temperature drops, but in the darker areas it drops faster, but in the lighter ones slower. In 30 minutes at Fig. 6b the situation is already changed, in average the temperature grows, but in the darker areas it grows faster, but in the lighter ones slower. And the same happens in 1 hour as depicted in Fig. 6c.

This scenario helps us to practise with the RSP engine, to identify existing shortcomings and capabilities. The issues which we faced during this case study are outlined in Section V-A.

V. Conclusion

In this paper we presented an architecture of a Semantic IoT Platform developed in SemIoT project (http://semiot.ru). And described a case study with two scenarios that is used to advertise the capabilities of the platform. In next subsections we outline lessons learned from thise case study and our future vision.

A. Lessons learned

We had some issues with the RSP engine. First of all, the static data can't be requested from SPARQL endpoint requiring an authorisation. The engine doesn't allow to provide authorisation credentials such as login/password, so we had to disable authorisation mechanism in *Triplestore*. Secondly, when the engine executes a query it has to look at each stream which is connected to it, but not all stream are relevant, e.g. if we don't know in advance which sensors are belong to the given city, we have to connect streams from each sensor connected to the platform that creates a unnecessary load to the engine. To eliminate this problem we added a pre-processing layer before the engine to dynamically connect relevant streams and disconnect currently irrelevant ones. This is achieved by creating a separate SPARQL query for filtering existing streams, in Fig. 8 we present such a query for Scenario B. The 1st *(SELECT)* subquery requests the border of the city from an external SPARQL endpoint and the next triple patterns request streams of the devices which are located within the border.

```
SELECT ?topic WHERE {
 SERVICE <http://external.sparqlendpoint/sparql> {
  SELECT ?long ?lat ?rad WHERE {
   ?uri a lgdo:City ;
    rdfs:label "Moscow"@en ;
    geo:long ?long ;
    geo:lat ?lat ;
    schema:getRadius ?rad .
  }
 }
 ?loc spatial:nearby (?lat ?long ?rad 'km') .
 ?device dul:hasLocation ?loc ;
  ssncom:hasCommunicationEndpoint [
   ssncom:protocol "WAMP" ;
   ssncom:topic ?topic
  ];
}
```

Fig. 8. An example of SPARQL query filtering irrelevant incoming streams

Another issue is filtering sensors by their relatedness to a given city. We didn't find an existing Linked Data dataset which would describe the border of cities. The well-know LinkedGeoData.org (http://linkedgeodata.org) dataset only has geo-coordinates of the centres of cities, so we couldn't directly use it. Therefore we deployed a separate SPARQL endpoint with data from LinkedGeoData.org and additional data (`schema:geoRadius`) which is used to query the borders.

Using SemIoT Platform as a semantic-powered middleware to work with data from different types of connected devices, we figured out that some of the problems come not from internal complexity of the data processing workflow, but rather from external data availability. During the work with the scenarios, that require large amount of data, you should anyway rely on such crowd-source services such as Netatmo or NarodMon with a full set of restrictions and limitations of them. For example, public APIs may have arbitrary limits on the number of queries per unit of time or on the observable area. Of course, not all of data sources, used in such services, are working as expected, and anomaly fluctuations should be considered during the analyse of such data in an appropriate way.

B. Vision and future work

As was shown in Section I so far there are quite lot of works in the area of Internet of Things: standards, technologies, systems and etc. But still there exist important issues such as interoperability among the system which have to be solved. We believe that Semantic Web technologies provide the necessary foundation to solve the interoperability problem.

In scope of SemIoT project we aims to develop a middleware and a gateway which as expected should contribute to the solution. The use of the ontologies and relevant technologies should be hidden from manufacturers of connected devices and end-users with appropriate tools. The manufacturers should be able to describe their devices with the ontologies and publish their RDF descriptions online. For the end-users it should be quite easy to connect a new device to a gateway or a cloud platform like we currently install mobile applications from mobile App Stores, e.g. Google Play.

The explicit description of connected devices' functionalities in RDF with shared ontologies should provide ways to develop new innovative applications and also enhance the security of the access to the device data by external agents by specifying in a high-level way to which data they have access.

ACKNOWLEDGEMENT

This work has been financially supported by the Ministry of Education and Science of the Russian Federation (Grant #RFMEFI57514X0101).

REFERENCES

[1] Gartner Inc. official website, Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015, Web: http://www.gartner.com/newsroom/id/3165317.

[2] L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey", *Computer Networks*, vol. 54, 2010, pp. 2787–2805.

[3] IoT European Research Cluster official website, Position Paper on Standardization for IoT technologies, Web: http://www.internet-of-things-research.eu/pdf/IERC_Position_Paper_IoT_Standardization_Final.pdf.

[4] M. Uschold, M. Gruninger, "Ontologies and semantics for seamless connectivity", *ACM SIGMOD Record*, vol. 33, 2004, pp. 58–64.

[5] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner, 'Ontology-Based Information Integration: A Survey of Existing Approaches', *In Proc. of IJCAI-01 Workshop: Ontologies and Information Sharing*, 2001, pp. 108–117.

[6] OSGi Alliance official website, Architecture, Web: https://www.osgi.org/developer/architecture/.

[7] C. W. Thompson, 'Smart Devices and Soft Controllers', *IEEE Internet Computing*, vol. 9, February 2005, pp. 82–85.

[8] M. Compton, P. Barnaghi, L. Bermudez, R. Garca-Castro, O. Corcho, S. Cox, et al., 'The SSN ontology of the W3C semantic sensor network incubator group', *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, December 2012, pp. 25-32.

[9] P. Wetz, T.D. Trinh, B.L. Do, A. Anjomshoaa, E. Kiesling, A. Min Tjoa, 'Towards an Environmental Decision-Making System: A Vocabulary to Enrich Stream Data', *In Proceedings of the 28th International Conference on Informatics for Environmental Protection*, 2016, pp. 317–335.

[10] M.I. Ali, N. Ono, M. Kaysar, K. Griffin, A. Mileo, 'EA Semantic Processing Framework for IoT-Enabled Communication Systems', *In Proceedings of the 14th International Semantic Web Conference*, 2015, pp. 241–258.

[11] D. Bonino, F. Corno, 'DogOnt - Ontology Modeling for Intelligent Domotic Environments', *In Proceedings of 7th International Semantic Web Conference*, 2008, pp. 790–803.

[12] D. Bonino, E. Castellina, F. Corno, 'DOG: An Ontology-Powered OSGi Domotic Gateway', *In Proceedings of IEEE 24th International Conference on Tools with Artificial Intelligence*, 2008, pp. 157–160.

[13] SAREF project website, 'Study on Semantic Assets for Smart Appliances Interoperability', Web: https://sites.google.com/site/smartappliancesproject/deliverables.

[14] M. Kolchin, N. Klimov, A. Andreev, I. Shilin, D. Garayzuev, et al., 'Ontologies for Web of Things: A Pragmatic Review', *In Proc. of 6th Knowledge Engineering and Semantic Web Conference*, 2015, pp. 102–116.

[15] J. Soldatos, N. Kefalakis, M. Hauswirth, et al., 'OpenIoT: Open Source Internet-of-Things in the Cloud', *Lecture Notes in Computer Science*, vol. 9001, 2015, pp. 353–363.

[16] D. Korzun, A. Kashevnik, S. Balandin, A. Smirnov, 'The Smart-M3 Platform: Experience of Smart Space Application Development for Internet of Things', *In Proceedings of the 15th International Conference NEW2AN 2015 and the 8th Conference ruSMART 2015*, 2015, pp. 56–67.

[17] H. Derhamy, J. Eliasson, J. Delsing, P. Priller, 'A Survey of Commercial Frameworks for the Internet of Things', *In Proceedings of IEEE 20th Conference on Emerging Technologies & Factory Automation*, 2015, pp. 1–8.

[18] Open Geospatial Consortium, OGC GeoSPARQL - A Geographic Query Language for RDF Data, Web: http://www.opengis.net/doc/IS/geosparql/1.0.

[19] D. Le-Phuoc, M. Dao-Tran, J. Xavier Parreira, M. Hauswirth, 'A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data', *Lecture Notes in Computer Science*, vol. 7031, 2011, pp. 370–388.

[20] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, 'C-SPARQL: A continuous query language for RDF data streams', *International Journal of Semantic Computing*, vol. 4, 2010, pp. 3–25.

[21] D. DellAglio, J.P. Calbimonte, E. D. Valle, O. Corcho, 'Towards a Unified Language for RDF Stream Query Processing', *The Semantic Web: ESWC 2015 Satellite Events*, vol. 9341, 2015, pp. 13–25.

[22] F. Aurenhammer, 'Voronoi diagramsa survey of a fundamental geometric data structure', *ACM Computing Surveys*, vol. 32, n. 3, 1991, pp. 345–405.