Enabling Query of Frequently Updated Data from Mobile Sensing Sources

Yuchao Zhou, Suparna De
Institute for Communication
Systems
University of Surrey
Guildford, United Kingdom
{yuchao.zhou, s.de}@surrey.ac.uk

Wei Wang
Department of Computer Science
and Software Engineering
Xi'an Jiaotong Liverpool
University, China
wei.wang03@xjtlu.edu.cn

Klaus Moessner
Institute for Communication
Systems
University of Surrey
Guildford, United Kingdom
k.moessner@surrey.ac.uk

Abstract—**The Internet of Things (IoT) paradigm connects everyday objects to the Internet and enables a multitude of applications with the real world data collected from those objects. In the city environment, real world data sources include fixed installations of sensor networks by city authorities as well as mobile sources, such as citizens' smartphones, taxis and buses equipped with sensors. This kind of data varies not only along the temporal but also the spatial axis. For handling such frequently updated, time-stamped and structured data from a large number of heterogeneous sources, this paper presents a data-centric framework that offers a structured substrate for abstracting heterogeneous sensing sources. More importantly, it enables the collection, storage and discovery of observation and measurement data from both static and mobile sensing sources.**

Keywords- Internet of Things; sensor data; time-series data; smart city; observation and measurement data discovery

I. INTRODUCTION

One of the objectives of the Internet of Things (IoT) research and development is to enable real world objects to be connected to the Web, so data generated by those objects can be discovered, collected, processed, shared and utilized to create intelligent and emergent applications and services in many domains such as smart cities, environment monitoring, health and energy [1]. From the perspective of data processing, one of the challenges in managing the IoT data is how to deal with the large number of heterogeneous sensing sources in a particular application domain. For instance, in the smart city applications, the real world data made available to the city applications is not only from sensor networks installed by city authorities at fixed locations, but also from mobile sources such as buses and taxis equipped with environment monitoring sensors and participatory sensing from citizens' smartphones.

The multitude of such data sources offers great opportunities to blend the observed data to detect events and to discover knowledge. However, the tremendous volume of observation and measurement (O&M) data from these heterogeneous sources tends to be complex to be processed and used. This kind of data has been characterized as being 'Frequently Updated, Timestamped and Structured' (FUTS) in the literature [1]. FUTS data contains states and updates of physical world things, such as in real-time traffic reports, air pollution detection, temperature monitoring etc. It can be classified into the category of 'acquisitional streams', where

"a tuple is measured at a predefined regular interval, e.g. readings from a sensor network" [2]. On the contrary, in an 'event stream', a tuple is generated every time an event occurs and can have variable, potentially very high data rates [2].

Most of the existing methods treat sensor observation data as time-series data, which is usually generated by the sensing sources installed at specific (fixed) locations (all observations have the same location metadata). FUTS data is different and varies not only along the temporal axis, but also the spatial axis. This issue arises when observations are made by mobile sensing sources such as citizens' smartphones and sensors installed for opportunistic sensing (e.g. sensors on public transportation systems measuring pollution levels at different parts of the city). As such, new techniques need to be developed by taking the spatial features of the sensing observations into account. FUTS data usually is defined using a structured format (e.g. JSON/CSV data), but the data models and schemas adopted by the various data sources are different and are not always compatible. Other identified issues with FUTS data include the lack of data management systems for data collection, storage, query and visualization [3].

This paper focuses on two problems: (i) how to design a common, structured sensing layer for the heterogeneous, mobile data sources and, (ii) how to query FUTS data from these sources. The contributions of our work include: (i) design and implementation of the virtualization component as a common substrate for FUTS data sources in IoT applications, (ii) design and implementation of a dynamic middleware to enhance the connectivity between data sources and applications. The middleware employs cloud-based storage to offer scalable capabilities to collect, store and retrieve both real-time and historical data, and (iii) discovery of FUTS data generated by both mobile and fixed sensing sources along both temporal and spatial axes.

The rest of the paper is organized as follows: section II reviews related work on modelling and querying O&M data; section III outlines the sensor O&M data management framework and presents our middleware for data collection and discovery; section IV elaborates different query techniques implemented, i.e., range query, distance query and time-windowing/aggregation based query. Section V concludes the paper and briefly points out some future research directions.

II. Related Work

In this section, we review the state-of-the-art from two aspects: schemas for modelling sensor O&M data and frameworks for annotating and querying sensor streaming data.

The Open Geographical Consortium's (OGC) Sensor Web Enablement (SWE) [4] initiative defined a number of schemas and standards for discovering Web accessible sensor networks and archived data. The Sensor Model Language (SensorML) [5] is an XML schema for describing sensor systems and processes. It provides information for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties. The O&M XML Schema [6] can be applied for encoding both real-time and archived O&M data from a sensor. The properties of an observation include 'feature of interest', 'observed property', 'sampling time', 'result' and 'procedure'. Feature of interest can include any real world entity such as coverage region, vehicle or weather storm. Procedure refers to a sensor or sensor system defined within a SensorML document. The syntactic XML OGC schemas are extended to RDF in a number of works to link the domain knowledge to external schemas and enable cross-domain query and reasoning. Annotation of observation data with external temporal and geographical concepts using the Linked Data principles is demonstrated in [7]. In this work, observations are annotated with time (at which they occurred) and location concepts published by DBpedia[1]. Rule-based reasoning is applied over the annotated data to derive additional knowledge, e.g. road conditions derived from weather readings, and to enable querying over the derived knowledge. This work does not address annotating a series of observations and its subsequent storage or querying over past data. In a similar initiative, the Linked Sensor Middleware [8], sensor descriptions and the resultant data are annotated with inter-links to existing datasets such as DBpedia and GeoNames [9], with the focus on sensing resource, data annotation and provisioning via common interfaces. An API for converting OGC O&M data to RDF is developed in [10]. The authors retrieve observation data from a number of weather stations in the US and develop a method to convert this raw textual data into RDF. Location information, encoded as WGS-84 [11] point data, is linked to concepts in GeoNames to enable answering queries related to location of sensors 'near' a place name. However, this work does not describe any mechanism to query O&M data based on both location and time.

Sensor Data Stream Management Systems (DSMS) include the GSN framework [12] that offers ad-hoc data access APIs for virtual sensors whose values are fed to the GSN system through some data adapter. The queries allow sliding windows through explicitly defined time parameters. Ontology-based querying of live sensor data is addressed in [2, 13]. In [13], the authors use R2RML (RDB-to-RDF mapping language) for mapping streams stored in relational databases to ontological schemas. The subsequent virtual RDF streams can then be queried using SPARQL extensions that support time windows. The authors validate their approach by providing query mapping for data retrieved from existing DSMSs such as ESPER[2], GSN and Pachube (now Xively) [14]. While the work provides a useful approach for ontology-based query of sensor O&M time-series data, the actual query processing and O&M data collection/storage is delegated to the DSMS.

An alternative approach for semantic annotation of sensor O&M data to generate streams (which are then assigned a unique identifier) is presented in [15]. The naming mechanism is based on the location and start time of the measurements. To deal with the large-scale annotated data, a K-means clustering algorithm is applied to distribute the data among different repositories. Finally, resolution is supported by using a prediction method based on the clustering model to identify repositories maintaining different parts of the data. However, queries supporting time windows or data aggregation functions are not supported in this work.

Moreover, all of the reviewed approaches assume fixed sensor locations, wherein the sensor location metadata (assumed to be static) is first queried to determine relevant sensors. Subsequently, a sliding time window is applied to either the selected sensors' stored observation data or the identified relevant storage (to retrieve the data).

III. FUTS Data Framework

This section presents the roles and functionalities of each component in the proposed FUTS data management framework and how each of these contributes to the overall objective of a common, structured sensing and discovery platform. Fig. 1 shows a schematic representation of the functional blocks in the framework and their subcomponents.

A. Virtualization of Data Sources

The virtualization component is designed to abstract the heterogeneous data sources into a common sensing substrate. Its aim is to virtualize IoT sensing objects in a way that their features, in conjunction with the associated external time-series data processing stores, can be used to support queries at different levels of detail. The framework abstracts the functionalities and O&M data (including its associated metadata) of heterogeneous sensing sources into 'virtual objects' (VOs). We first look at the proposed virtual object model before describing the supporting blocks.

1) Virtual Object Model: The virtual object model is designed to overcome the problem of heterogeneity of different sensing objects and to potentially enable context awareness by exploiting more features of surrounding or related objects. The model includes constructs that allow linking to concepts in external domain models and creating linked IoT data. Since we need to differentiate between 'data about things' (i.e. things' metadata, e.g. identity, location etc.) and 'data generated by things' (i.e. O&M data), the central concepts in the VO model are the virtual

---

[1] http://dbpedia.org/About

[2] Esper and Event Processing Language EPL, http://esper.codehaus.org/

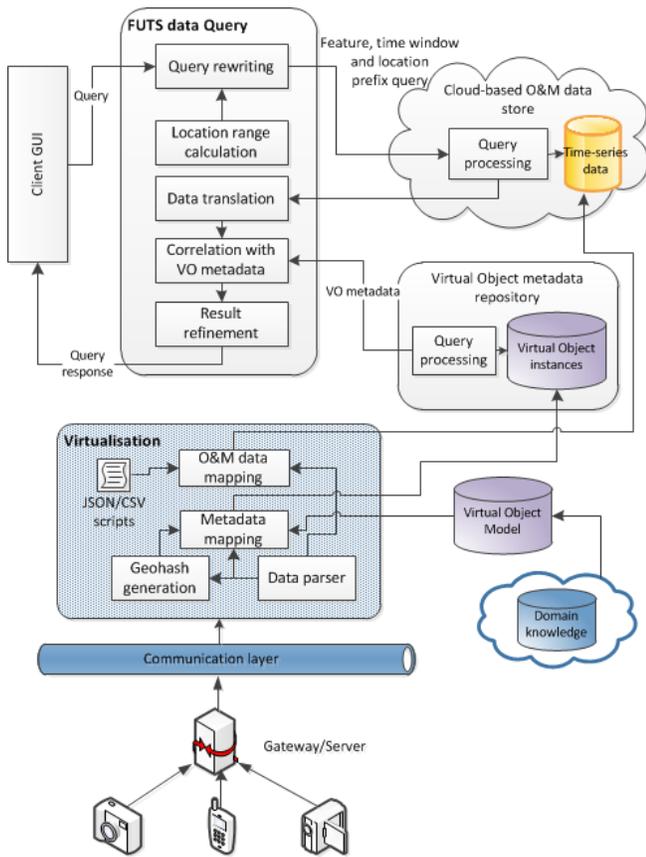object metadata and the information it provides. The overall structure of the VO model is shown in Fig. 2.

Fig. 1.Sensor measurement data management framework

The VO metadata includes an ID, name, type (expressed in terms of concepts from DBpedia) and a Boolean property indicating if the underlying real world object is mobile. The VO location is captured in a number of ways, e.g., latitude-longitude information modelled as WGS-84 concepts and a geohash[3] value.

O&M data and the associated observed real world features are modelled through multiple 'information' elements. Each information instance has a name and semanticURI that specifies the type of the observed feature (e.g. temperature) or links to an external domain model, for instance, the vocabulary of climate and forecast features – the CF[4] taxonomy. The actual O&M data is captured through a literal 'value', associated unit of measurement (UoM) drawn from UoM vocabularies such as Quantities and Kinds (QU) [16] taxonomy, the time of measurement and the location of measurement captured by a geohash string.
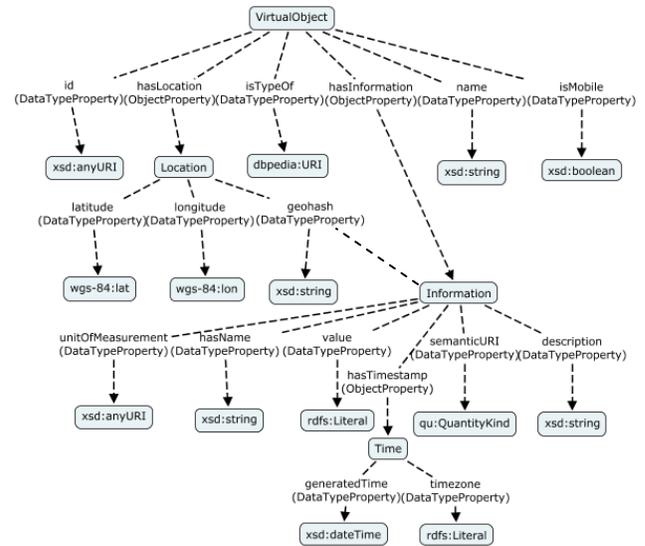

Fig. 2.Virtual Object Model

2) Virtualization Functional Blocks: The communication layer is the lowest layer for data collection in the framework and facilitates connection to FUTS data sources. Both polling (when data is exposed through Web Service interfaces and can be read through for instance, cURL [5] commands) and publish-subscribe mechanisms (through the MQTT protocol [17]) are supported. We assume that data sources either have communication capabilities to expose Web-level APIs or connect through intermediate gateway servers to the Web.

The data retrieved from the communication layer is parsed in the 'Data Parser' block. The parser supports the plug-and-play mechanism for extracting data from heterogeneous data sources, each of which may annotate data according to different schemas.

The location data parsed by the parser is fed to the 'Geohash generation' block which generates the geohash string, for annotating both the VO metadata as well as the O&M value. Geohash is a geocoding algorithm that uses Base-32 encoding and bit interleaving to convert latitude and longitude pairs to a 1-dimensional string. The string denotes the spatial feature of an object and its length implies the precision of the representation. Geohashes with a common prefix with different lengths represent a hierarchy of locations. An interesting feature of geohash strings is that nearby locations share similar prefix.

The generated geohash, along with the rest of the parsed information, is used to create a VO instance and to annotate it according to the designed VO model by the 'Metadata mapping' block. The annotated VO instances are stored in the VO instance repository. The VO repository offers a query interface for searching VO instances in terms of the modelled concepts. The parsed O&M values are annotated according to the 'information' concepts of the VO model and stored in the O&M data store by the O&M data mapping

---

[3] Geohash, http://geohash.org/
[4] Climate and Forecast Features, http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-feature

[5] cURL, http://curl.haxx.se/

block. The O&M data store is a cloud-based time-series database (TSDB) that allows scalable storage and query of time-series data. The TSDB accepts O&M values annotated as JSON strings and identified by a URL. Hence, the mapping function utilizes JSON scripts to serialize the data values. The data embedded in the JSON string includes the feature of observation, the observation value, its UoM and timestamp. A number of features and associated values (including metadata) may be uploaded simultaneously as a bulk transfer with a 'write time series' HTTP call to the cloud TSDB. This is converted by the TSDB into distinct time series corresponding to each observed feature. The O&M data mapping block embeds the derived geohash string into the time series URL to include location information along with the O&M values (since only 'time-stamped' values are typically supported by time-series databases). The time series are organized in a hierarchy similar to a file system. The time-series URL thus has the following format: series/{dataseries}/{VO-ID}/{feature}/{geohash}. This also enables efficient and scalable retrieval of O&M data values by taking advantage of the inherent hierarchical properties of the geohash string.

Once a data source (with fixed location) is annotated and stored in the VO repository, future measurements from this source can be saved by just adding the additional measurement values and timestamps to the relevant time series in the TSDB. Data from mobile sources results in an 'update' to the existing VO metadata, with the location metadata updated to include the additional latitude/longitude and geohash values. The TSDB is also updated; however in this instance, a new time series is added for the same VO-ID and feature as the geohash part of the URL has changed (thus giving rise to a new URL). The O&M query mechanism however remains the same as reading from the time series allows wild cards at any level of the URL.

B. Querying FUTS Data

A Web-based user interface is designed to help users specify a number of criteria, e.g., location of interest, observed features, temporal extent and aggregation functions, for creating queries. The different filtering functions supported consist of the following:

- Range queries - users can specify a rectangular area on a map, along with the desired time window.
- Distance queries - users can define a circular area of interest by specifying a point on the map and a radius within which observations should have been stored. Time window is also supported.
- Time window and aggregation - in addition to specifying the time range for observations, users also can specify several aggregation functions: minimum, maximum or average.

A query also includes a geohash generated by the 'Location Range calculation' block. The geohash is computed based on the geographical information and other parameters provided by the users on the map. It tries to use the geohash with the maximum length to cover the query range. The geohash, together with the specified time window and the feature, are converted into a template, which serves

as a pattern to select the matching time series URLs from the underlying TSDB. The actual data retrieval based on the sliding time window is delegated to the TSDB. The retrieved data (value, UoM and timestamp) is processed by the 'Data translation' block which correlates the O&M data with the corresponding VO location metadata retrieved from the VO instance repository. The VO location metadata is used to further refine the set of retrieved results, i.e., transform them into a format appropriate for presentation to the users.

IV. IMPLEMENTATION

A prototype based on the proposed framework has been implemented to search O&M data from mobile and static sensing sources. The prototype employs an open-source TSDB, Geras [6], to store the O&M data. Geras offers a scalable, cloud-based storage for streams of time-series sensor data and is optimized to store large amounts of data with frequent updates. The stored data is compressed for scalability but the original data is retained for maximum resolution. Geras also supports instant views at any time resolution. A relational database is employed to store the VO metadata and to enable fast updates on location metadata. An open-source reference implementation [7] of the geohash algorithm is used to compute the geohash tags.

Our framework is validated by applying data from the SmartSantander [18] testbed. SmartSantander provides a city-scale experimental research facility encompassing 20,000 sensors which expose a variety of services, such as static and mobile environmental monitoring, parks and gardens irrigation, outdoor parking area management and traffic intensity monitoring [1]. The mix of static and mobile monitoring sources allows us to validate the effectiveness of the proposed framework. We collected the O&M data from the SmartSantander testbed over a period of 3 months, from March 2014 to June 2014 at 2-hour intervals. The data can be retrieved through Web Service APIs and is available in JSON format. Each data item consists of a number of observed features (e.g., temperature, humidity, particle, CO and NO_2 levels, etc.) along with the associated UoM, time-stamp and latitude-longitude values denoting the location of observation. This type of data annotation also validates the FUTS data paradigm adopted in this paper.

A. Range Queries

Range queries are specified by an (rectangular) area of interest, feature and time range of the O&M data. The rectangle represents the scope of the geographical area that the user is interested in. It is defined with two geographical points on the map (the top-left and the bottom-right points of the rectangle) by the user. The latitude and longitude values of the two points are passed on to the 'Location range calculation' block which determines the most appropriate geohash that is able to cover the specified bounding rectangle. Since a geohash inherently covers rectangular grids, the calculated geohash prefix is accurate enough to capture the area specified in the query; therefore, it enables

---

[6] Geras TSDB, http://1248.io/geras.php
[7] Geohash in java, https://github.com/kungfoo/geohash-java

the retrieval of all O&M data within this range. The entire process of data annotation, storage and query is illustrated with an example below.

Geras accepts time series data in the Sensor Markup Language (SenML) [19] format, which enables transmission of simple sensor measurement data over HTTP or CoAp. It supports the JSON serialization of SenML, with both write and read operations on streaming JSON. Every SenML document can be seen as a single JSON object with an array of event objects. Each event object contains a name (n), a value (v), optional units (u) declaration and an optional timestamp (t) in seconds since the epoch (01/01/1970). Fig. 3 shows a snippet of an O&M data uploaded to Geras, using the URL: http://geras.1248.io/series/santander/bus3000. Values of the parameters n, v, t, and u for this example are also shown in the figure.

```
{
  "e":[
    {"n":"Humidity/e/z/t/r/3/2/4/g/f/p/f/e","t":1402077240,"v":22,"u":"%RH"},
    {"n":"CO/e/z/t/r/3/2/4/g/f/p/f/e","t":1402077240,"v":0.10,"u":"mg/m2"}]
}
```

Fig. 3.SenML event objects in JSON. The first event object represents a measurement of the Humidity and the second represents measurement of CO at a specific time and location

The geohash computed from the latitude-longitude of the parsed data, is appended to the feature name to form the n-field of the time series. It is transformed into a character array with every character forming a successive hierarchy of the n value. On upload, every n is interpreted as a path in a filesystem. If n is relative, the data point's destination is determined by combining the URL path with the n field. Thus, from Fig. 3, the two corresponding time series registered in Geras are: santander/bus3000/Humidity/e/z/t/r/3/2/4/g/f/p/f/e and santander/bus3000/CO/e/z/t/r/3/2/4/g/f/p/f/e.

Assuming that the user query is to determine the last updated humidity values in the area determined by the points (43.475700, -3.788650) and (43.464900, -3.833380), the first step is to determine the longest geohash prefix for the given area, which is calculated to be eztr. To read data simultaneously from multiple series, Geras provides API operations of the form /series?pattern=PATTERN, where the value of PATTERN is taken as an MQTT topic pattern. Considering the format of the stored time series and the query requirements, the pattern is derived to read all series that include humidity and the eztr prefix in the URL. This then computes as: /santander/+/Humidity/e/z/t/r/#. The '+' is a single level wildcard that matches only one level in the URL; in this case it matches any VO-ID. The # at the end of URL matches any number of levels, so in this case, it matches all time series that have a geohash beginning with eztr. The wildcards are HTTP URL encoded in the cURL command. The wildcard matching supported by Geras also illustrates the need for designing the time series URL in this particular format (i.e. series/{VO-ID}/{feature}/{geohash}) as the multi-level wildcard # must always be the last character within a pattern tree. The 'now' keyword returns the last

updated data point, so the URL sent to the read operation API is: http://geras.1248.io/now?pattern=/santander/+/Humidity/e/z/t/r/#.

The returned response consists of JSON event objects similar to the uploaded time series data. The returned JSON object is then parsed and the data is visualized on the client GUI.

B. Distance Queries

Distance queries accept a spatial point (denoted by its latitude and longitude values), a distance, the observed feature and the time parameter. The distance parameter is first used to define a circular area around the spatial point. However, since circular areas cannot be represented directly and accurately by a geohash, the minimum rectangle enclosing this circular area is first calculated and then the previous mechanism is employed to derive the geohash prefix. Fig. 4 shows all the relevant parameters and the query URL used to read data from Geras.

```
Latitude: 43.475700, longitude: -3.788650; distance: 1.7km
Geohash prefix: eztr
Derived pattern: /santander/+/Humidity/e/z/t/r/#
Read series URL: http://geras.1248.io/now?pattern=/santander/+/
Humidity/e/z/t/r/#
```

Fig. 4.Distance query parameters and query rewriting

The steps for reading the appropriate time series from Geras are the same as the ones for range queries described in the previous section. Since some O&M data points may fall within the rectangular area but outside of the circular area, they need to be filtered out. In this case, the actual WGS-84 encoded latitude-longitude values are first retrieved from the VO metadata store by correlating the observed feature, geohash and timestamp values stored in Geras. The distance to the geographical point in the query is then computed using the Haversine formula [8] and compared to the distance required by the query. The filtered results are then returned to the user.

C. Time-windowing and Aggregation

Requests for data can be limited to a time range with the time window function. These can be combined with pattern queries to read data from multiple series. Time windows are specified with a start and end date/time on the client GUI, which are converted to seconds since the epoch. Negative values are also allowed, in which case they are interpreted as historical values measured 'seconds ago'.

Aggregation applies a filter function across a period of time. They can be applied to a single series or across multiple series using the pattern query parameters. Supported filter functions include min, max, average and sum, denoted by the 'rollup' keyword in the URL. The time interval for applying the filter function can be defined in terms of intervals of seconds, minutes, hours, days, weeks, months or years.

---

[8] http://www.movable-type.co.uk/scripts/gis-faq-5.1.html

D. Query Interface

Fig. 5 shows the Web-based search interface for range queries. The interface facilitates the query formulation by allowing users to specify the searching criteria by filling up a form-like template.

Fig. 5.O&M data search user interface

The query area text box is linked to a Google maps pop-up window that allows users to specify the rectangular area to be searched by 'drag-and-drop'. The figure shows that the user query is to search for humidity values in the area determined by the points ($43.465021,-3.834145,43.468361,-3.827081$), between 14:15:10 and 19:15:10 on 6th June 2014. This results in the processing steps are depicted in Fig. 6.

Fig.6. Time windowing with range query

The resulting O&M values, retrieved as a JSON object, are parsed and displayed on the map with markers depicting the locations where the observations were made. The result on the map is shown in Fig. 7.
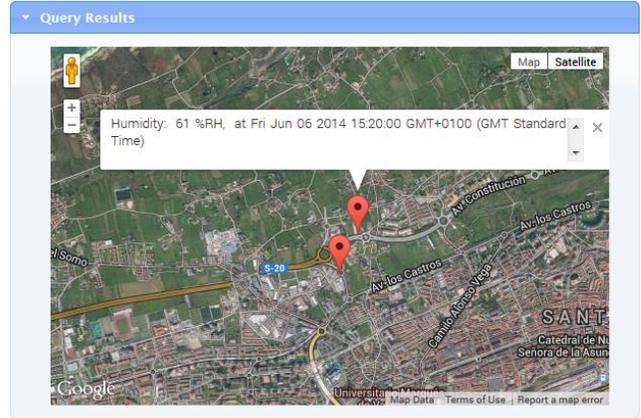

Fig. 7.Query results overlay on Google maps

V. CONCLUSIONS

Modelling sensor O&M data as time-series is not sufficient for many IoT applications since the spatial and temporal features of the (mobile) sensing data constantly change. We consider the FUTS data model as a more appropriate apparatus to characterize the IoT data and aim to design a common substrate for the heterogeneous FUTS data sources to facilitate query and discovery. Directly storing spatial information into the time-series database is convenient, however, query and retrieval of the sensing data is computationally expensive because of the lack of spatial query support. Based on the proposed cloud-based middleware, we design methods for the proximity-based discovery of FUTS data generated by both mobile and fixed sensing devices in the smart city applications. Our approach hides the complexity of accessing the large number of heterogeneous sensing devices with virtualization of the data sources and provides easy-to-use data services for applications that need to perform range, distance and time-window based queries frequently. In our experiments, we implement the query and discovery functions by using geohashing as a spatial feature matching method and the URL pattern operations provided by the cloud-based time-series database. The functions can also be realized by the use of spatial indexing on the geographical values of the O&M data. However, a separate database for storing and updating the spatial index is needed. Moreover, a large number of update operations would be required for the spatial index, especially for mobile sensing sources. All of these might significantly degrade the query and discovery performance. Our future work is to evaluate the performance of the proposed method and compare it with the ones that combine spatial indexing and time-series database using large-scale datasets.

REFERENCES

[1]   Y. Qin, Q. Z. Sheng, N. J. G. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos, "When Things Matter: A Data-Centric View of the Internet of Things,"in CoRR, 2014.

[2]   J.-P. Calbimonte, O. Corcho, and A. J. G. Gray, "Enabling ontology-based access to streaming data sources," in Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I, Shanghai, China, 2010, pp. 96-111.

[3]   A. E. James, J. Cooper, K. G. Jeffery, and G. Saake, "Research Directions in Database Architectures for the Internet of Things: A Communication of the First International Workshop on Database Architectures for the Internet of Things (DAIT 2009)," in Proceedings of the 26th British National Conference on Databases (BNCOD), Birmingham, UK, 2009, pp. 225–233.

[4]   OGC, "Open Geospatial Consortium (OGC) Sensor Web Enablement: Overview and High Level Architecture," OGC white paper, 2007.

[5]   OGC, "OpenGIS® Sensor Model Language (SensorML) Implementation Specification," Open Geospatial Consortium, Inc. 2007.

[6]   OGC. (2011). OpenGIS® Observations and Measurements - XML Implementation. Available: http://www.opengeospatial.org/standards/om

[7]   W. Wang and P. Barnaghi, "Semantic annotation and reasoning for sensor data," in Proceedings of the 4th European conference on Smart sensing and context (EuroSSC2009), Guildford, UK, 2009.

[8]   D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth, "A middleware framework for scalable management of linked streams," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 16, pp. 42-51, 2012.

[9]   GeoNames ontology (2011). Available: http://www.geonames.org/ontology/documentation.html

[10]   H. Patni, C. Henson, and A. Sheth, "Linked Sensor Data," in Proc. International Symposium on Collaborative Technologies and Systems (CTS), 2010, pp. 362–370.

[11]   "World Geodetic System 1984 (WGS-84)," National Geospatial-Intelligence Agency 2005.

[12]   K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in Proc. 32nd International Conference on Very Large Databases, 2006, pp. 1199–1202.

[13]   O. Corcho, J.-P. Calbimonte, H. Jeung, and K. Aberer, "Enabling Query Technologies for the Semantic Sensor Web," Int. J. Semant. Web Inf. Syst., vol. 8, no. 1, pp. 43-63, 2012.

[14]   W. Wang, S. De, G. Cassar, and K. Moessner, "Knowledge Representation in the Internet of Things: Semantic Modelling and its Applications," Automatika - Journal for Control, Measurement, Electronics, Computing and Communications, vol. 54, no. 4, October 2013.

[15]   P. Barnaghi, W. Wei, D. Lijun, and W. Chonggang, "A Linked-Data Model for Semantic Sensor Streams," in Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, 2013, pp. 468-475.

[16]   OMG SySML. Library for Quantity Kinds and Units: schema, based on QUDV model. 1.2.

[17]   OASIS, "MQTT Version 3.1.1," Candidate OASIS Standard 01, 2014.

[18]   Smart Santander. (2011). contract number FP7-257992. Available: http://www.smartsantander.eu/

[19]   IETF, "Media Types for Sensor Markup Language (SENML)," in Internet-Draft, ed, 2013.