2015 3rd International Conference on Future Internet of Things and Cloud

Gatica: Linked Sensed Data Enrichment and Analytics Middleware for IoT Gateways

Soheil Qanbari*, Negar Behinaein†, Rabee Rahimzadeh† and Schahram Dustdar*
*Distributed Systems Group, Vienna University of Technology, Vienna, Austria
{qanbari, dustdar}@dsg.tuwien.ac.at
†Baha'i Institute for Higher Education (BIHE)
{negar.behinaein, rabee.rahimzadeh}@bihe.org

***Abstract*—Raw sensed data lacks semantics. This poses a challenge to apply analytics directly to raw IoT sensor data. Such operational data requires an intensive enrichment processes to drive value. Pragmatic use of naming conventions and taxonomies can increase the quality of data and make it more interpretable. In this paper, we incorporate semantic and linked data technologies and offer a middleware called Gatica, to dynamically inject semantics to make the raw streaming data of an IoT gateway "Rich" on the device layer. Gatica collects the real-time sensor data, enriches them using annotations then transforms and exposes them in RDF triples, and finally streams RDF objects to the analytic endpoint for querying the linked sensor streaming data. Various analytic applications can utilize our middleware by sending SPARQL requests over the sensor network to our query interface and retrieving the results. Our middleware offers the ability to discover hidden patterns of mutually correlated variables and uncover actionable information within raw data for more utility. This paper details Gatica's architecture together with its implementation.**

I. Introduction

A data stream is a massive sequence of sensed data. Sensing is a process of expressing true values in context. Context refers to an environment of interest in which a sensor is embedded. Such sensors might be a device or a service. They represent the context behavior by streaming the values of its computational objects attributes. To interpret the situation of a thing in a context, semantics of such raw data seems to be vital. An Internet of Things (IoT) incorporates cloud computing[1] and virtualization mechanisms to expose such data to analytic endpoints to drive context awareness. The W3C Semantic Sensor Network Incubator Group (SSN-XG)[1] has developed an ontology to elevate the quality of sensed data with semantic web technologies. Project Haystack[2] has developed naming conventions and tags for environmental equipment like buildings and lighting devices together with their operational data. Having these conventions in place, clients are able to consume Haystack REST APIs to discover, query, and tag objects and the data collected and stored by IoT frameworks like NiagaraAX[3]. IoT infrastructure is composed of resource-constrained gateways and a network of devices. The above-mentioned solutions are not lightweight and universal in terms of their provisioning and deployment model. They also lack analytic endpoints for reasoning purposes.

[1]http://www.w3.org/2005/Incubator/ssn/
[2]http://project-haystack.org
[3]http://www.niagaraax.com

To this end, our contribution is threefold: (i) A built-in solution for the semantic sensor data retrieval on the IoT device layer. We develop a semantic gateway middleware called Gatica, which delivers automated and on-demand semantic annotations, labels and taxonomies for sensor data acquisition at scale. (ii) In support of such linked sensor data which was collected and annotated by the wrappers, we provide a "manifest" as the meta-data dictionary of the sensor current readings. This contributes to real-time semantic sensor data retrieval. (iii) At streaming-time, the sensor annotated data object from the wrapper is injected into the mediator and the mediator virtually applies in lightweight transformations on raw data resulting RDF triples. Our Gatica middleware implements a layered approach to the interpretation of the sensor time-series data. The linked data RDFs are then streamed to the analytics endpoints for querying and reasoning purposes.

The paper continues with an initial analysis over the utility of the data source being studied in this research in section II. After the data utility mathematical model is detailed, sectionIII presents the basic concepts and preliminaries of IoT gateways. With some definitive clues on data source structure and its associated annotations, we propose a novel IoT gateway middleware to fulfill sensor data enrichment. Section IV is devoted to the core elements of Gatica layered architecture together with its interacting components. In support of our model, we deploy our middleware to production by processing the real-world medical data set. Subsequently, section V surveys related work. Finally, section VI concludes the paper and presents an outlook on future research directions.

II. The Utility of Sensed Data

There is a commendable survey[2] in which the authors explore the state of the art of how the health-care sensed data are utilized by applying analytics and mining algorithms. Along with such data use-cases, patients requiring intensive care need continuous observation and treatment. This is achieved via wearable or contextual sensors, which are connected to medical devices measuring physical attributes and producing a considerable amount of vital data on a per-patient basis. Medical institutions that are collecting big amounts of such data enable us to achieve a better understanding of the patient's current status and their recovery progress.

Having such data in place, the health-care service providers are able to construct a wellness-function for the *normal* range of the vitals and produce alerts upon on deviating from the normal values. Through this vital range interpretation, various disease patterns can be discovered together with its severity.

In this paper we have used the Massachusetts General Hospital/Marquette Foundation (MGH/MF) Waveform Database[3] that represents a comprehensive collection of electronic recordings of hemodynamic and electrocardiographic waveforms of stable and unstable patients in critical care units, operating rooms, and cardiac categorization laboratories.

This data set is used as a real world motivation scenario in putting Gatica in production mode. At any given time t, Gatica provides enriched pieces of information about the medical sensor observations. Such observations can be represented as a column-vector $\mathbf{o}_t \equiv [v_{t,1}\ v_{t,2}\ ...\ v_{t,n}]^T \in R^n$ of sensor data stream values at time t. The stream data can be regarded as a frequently expanding $t\times n$ matrix $\mathbf{O}_t := [\mathbf{o}_1\ \mathbf{o}_2\ ...\ \mathbf{o}_t]^T \in R^{t\times n}$ where the new incoming streams are added as matrix rows at each time interval t in real-time. In our health-care example, $\mathbf{O}_t$ is the measurements column-vector at t over all the sensors, where n is the length of the vector and indicates the number of health-care sensors and t is the measurement time-stamp. These vectors represent the set of measurements obtained by the n sensor at a specific observation. In particular the rows of the matrix represent the different observations in a given period, while the columns the sample values detected from each sensor during the observations.

In our scenario, as shown in Equation 1, let $\mathbf{O}$ be a matrix representing the patient vital data measured by sensors in the hospital ICU room. For instance, vector ECG_i represents the electro activity of heart beats, vector ART_j observes the blood pressures and vector $CO2_k$ indicates the levels of blood carbon dioxide in a period of time.

$$\mathbf{O}_{t,n} = \begin{pmatrix} ECG_{1,1} & ART_{1,2} & \cdots & CO2_{1,n} \\ ECG_{2,1} & ART_{2,2} & \cdots & CO2_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ ECG_{t,1} & ART_{t,2} & \cdots & CO2_{t,n} \end{pmatrix} \quad (1)$$

Then we perform Principal Component Analysis (PCA)[4], $PCA(\mathbf{O}) \rightarrow \mathbf{O}'$, which divides the matrix $\mathbf{O}$ into components to extract the patients health behavior pattern. The PCA orthogonalizes the columns (take set of orthogonal vectors) of $\mathbf{O}_{t,n}$ with the *Gram-schmidt*[4] process as shown in Equations 2 and 3.

$$\mathbf{v}_1 = \mathbf{w}_1 = \begin{vmatrix} ECG_{1,1} \\ ECG_{2,1} \\ \vdots \\ ECG_{t,1} \end{vmatrix} \cdots \mathbf{w}_2 = \begin{vmatrix} ART_{1,2} \\ ART_{2,2} \\ \vdots \\ ART_{t,2} \end{vmatrix} \quad (2)$$

$$\mathbf{v}_2 = \mathbf{w}_2 - \frac{\langle v_1, w_2 \rangle}{\langle v_1, v_1 \rangle} v_1 \quad (3)$$

where $\langle v, w \rangle$ denotes the inner product of the vectors v and w. This recursive process generates the set of orthogonal principal vectors as generalized in Equation 4.

$$\mathbf{v}_n = \mathbf{w}_n - \sum_{i=1}^{n-1} \frac{\langle v_i, w_n \rangle}{\langle v_i, v_i \rangle} v_i \quad (4)$$

The result is matrix $\mathbf{O}'$ with orthogonal components with minimized dimension of f.

$$\mathbf{O}' = \begin{vmatrix} ECG'_{1,1} \\ ECG'_{2,1} \\ \vdots \\ ECG'_{t,1} \end{vmatrix} \begin{vmatrix} ART'_{1,2} \\ ART'_{2,2} \\ \vdots \\ ART'_{t,2} \end{vmatrix} \cdots \begin{vmatrix} CO2'_{1,f} \\ CO2'_{2,f} \\ \vdots \\ CO2'_{t,f} \end{vmatrix} \texttt{with}\ f \preceq n \quad (5)$$

After performing the PCA, we will have a set of components/vectors where each one represents observations of one header in the data set.

These extracted vectors together with the patients profile PP vector are then modelled as a matrix $\mathbf{D}$ in the Equation 6. This matrix will be used to detect and discover the *diagnoses* via some correlation pattern discovery.

$$\mathbf{D} = \begin{vmatrix} PP_{1,1} \\ PP_{2,1} \\ \vdots \\ PP_{t,1} \end{vmatrix} \begin{vmatrix} ECG'_{1,1} \\ ECG'_{2,1} \\ \vdots \\ ECG'_{t,1} \end{vmatrix} \begin{vmatrix} ART'_{1,2} \\ ART'_{2,2} \\ \vdots \\ ART'_{t,2} \end{vmatrix} \cdots \begin{vmatrix} CO2'_{1,f} \\ CO2'_{2,f} \\ \vdots \\ CO2'_{t,f} \end{vmatrix} \quad (6)$$

Using the above equation, we compute a correlation matrix $\mathbf{C}_{x,y}$ where x and y represent matrix indices to find the relationships among variables. The $\mathbf{C}_{x,y}$ is given by the correlation coefficient[5] $c_{x,y}$ between the $\mathbf{D}_x$ and $\mathbf{D}_y$.

$$\mathbf{C}_{x,y} = Corr(\mathbf{D}_x, \mathbf{D}_y) = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \quad (7)$$

The Equation 7 contains terms of correlation where σ_{xy} indicates the covariance between $\mathbf{D}_x$ and $\mathbf{D}_y$. The two variables of σ_x and σ_y represent the standard deviation of $\mathbf{D}_x$ and $\mathbf{D}_y$. This constructs various sub-matrices acquiring and utilizing cross-correlation patterns of actual health status observations over patient's hospitalization profile within a specific intensive care period. Such discovered patterns can be clustered using K-means[6] or Bond Energy Algorithm (BEA)[5] methods, for instance, into a similar behavior patterns to diagnose the disease based on their similarities to the centroids of the cluster. These centroids of our k-means can be defined as the attributes of a specific disease.

The major issue with this raw sensed data is the lack of semantics to detect early diagnosis. The interpretation of the sensor data into meaningful prescription requires a deep understanding of the medical information and should be driven by domain experts. Since the data is raw, the medical sensor records should be linked with the labels to ease the disease detection by doctors. This leads to improved delivery of care

[4]http://en.wikipedia.org/wiki/GramSchmidt_process

[5]Pearson's correlation coefficient between two variables is defined as the covariance of the two variables divided by the product of their standard deviations.

[6]http://en.wikipedia.org/wiki/K-means_clustering

by providing the health caring services to patients in an proactive fashion. With this motivation in mind, we proceed with some IoT gateway preliminary concepts.

III. IoT Gateways: Terms & Preliminaries

IoT gateways are resource-constraint devices (i.e, with limited compute, memory, and storage amounts) which expose connected sensors as cloud services to become addressable, discoverable and controllable. In order to operate our gateway, we use the size-optimized and tailored BusyBox[7] OS which is a combination of UNIX utilities into a single small executable. On top of the OS, a provisioning framework is required to deploy and manage the life-cycle of the lightweight execution units or IoT tiny applications. In our architecture, we used the Sedona[8] framework for such large-scale application deployments in very constrained embedded environments.

In our architecture, we have utilized the Sedona framework in the medical gateway to build a solution for aggregating device signals and performing some operations through lightweight execution units (Sedona apps) which have been deployed on each Gateway. Now, we elaborate a bit more into Sedona core concepts:

◇ *Classes*: they extend the *Component* class to perform the defined tasks. The *component* class includes *Slots* that specify how the component is exposed.

◇ *Kits*: Sedona language sources are compiled into fine-grained modularity archive files with the ".kit" extension. Kits include application's *Manifest* and *Intermediate Representation (IR)* file that is a non-executable compiled *Component* class in Assembly. All kits are compiled into a compact *SCode* binary image which will be executable on Sedona VM.

◇ *Manifest*: Each Sedona application has a manifest file which contains meta-data of the *Kit* like name, version, build host and dependencies.

◇ *Sedona Virtual Machine*: The SVM is a small interpreter written in C designed for portability. It allows *Kits* to be executed on any Sedona-enabled device.

Sedona is a component oriented language, so composing the application by assembling pre-defined components is a principle in this framework. By assembling the required components and compiling it, there will be a file with .sab extension. This file is deployable on the Sedona device what we call it lightweight execution unit in our gateway. In real world, there are different Sedona Gateway devices with different specifications, for instance, Raspberry Pi[9] device including Busybox linux 2.6.32 kernel. To simulate this device as our gateway, we used BusyBox hosting the Sedona VM. This image is running on a docker[10] container to mimic the physical gateway. Once the SVM is up and running, the device will be discoverable from our client. Next, we detail the Gatica's architecture in the following section.

[7]http://www.busybox.net
[8]http://www.sedonadev.org
[9]http://en.wikipedia.org/wiki/Raspberry_Pi
[10]https://www.docker.com

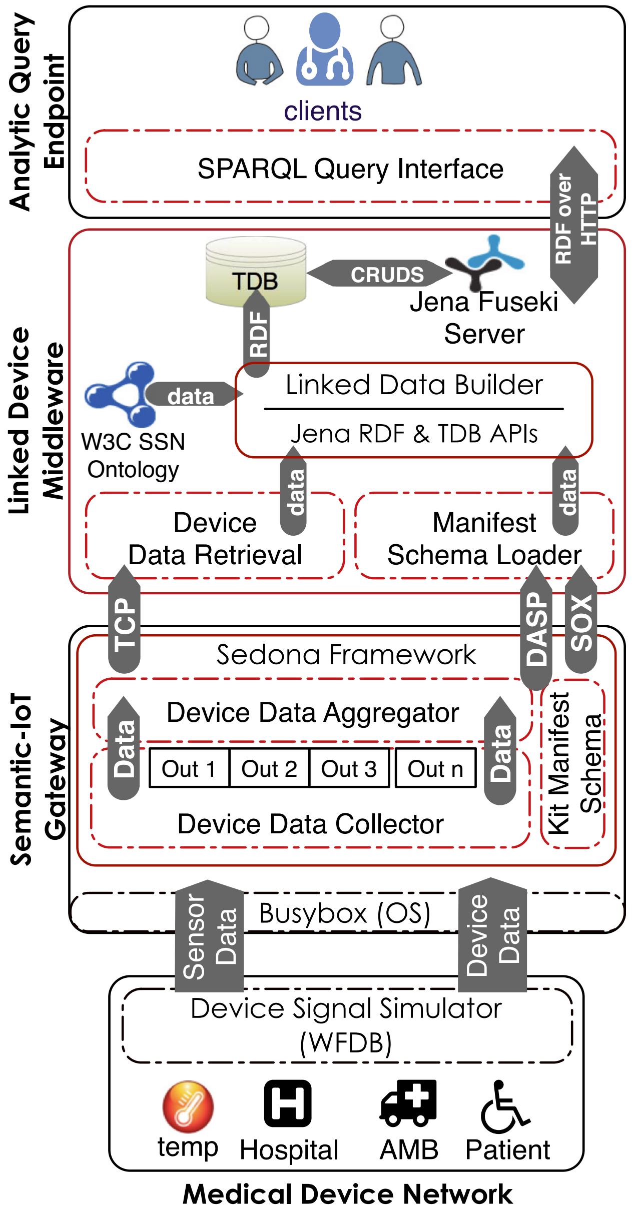


Fig. 1: Gatica Middleware layered architecture

IV. Gatica Middleware Architecture

Looking forward, Fig. 1. illustrates a schematic view on architecting Gatica's collaborating components. As a blueprint for designing the architecture, the Gatica is organized into four interconnected layers: (i) Medical Device Network, (ii) Semantic IoT Gateway, (iii) Linked Device Middleware and (iv) Analytic Query Endpoint. Here the whole architecture is implemented[11]. Next each layer's specification and capabilities together with its implementation details are described.

[11]https://github.com/soheil4TUWien/Gatica

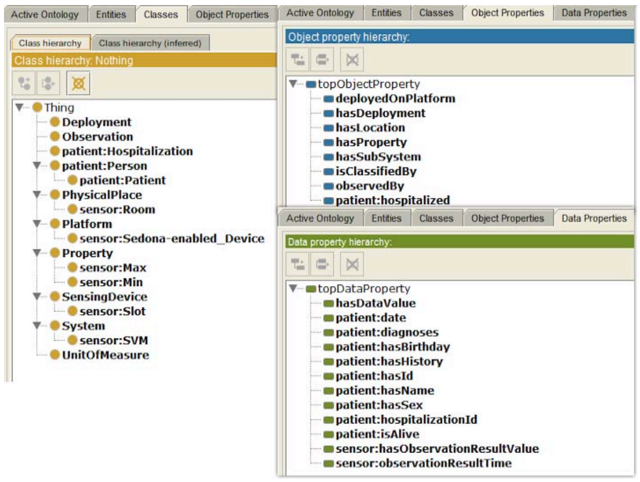


Fig. 2: Sensor ontology classes, objects and data properties

A. Sensor Data Retrieval

Now, we briefly review the data flow from the sensor/device network to the upper layers. Gatica is designed to cope with large amounts of real-time sensor data by transforming it into linked data. This includes operations involved in collecting data from external sensor data sources. Then, preprocessing operations like cleansing noisy data are applied to the data to prepare it for further analysis. The sensor data acquisition is performed by the deployed kits in the device layer that interfaces with sensors and feeds into the stream processing system. This data is a time series consisting of ordered sequences of `[key,value]` pairs of timestamps and data elements.

In the MGH/MF Waveform Database, three files describe each record. These files are .ari extension (beat and event annotation), .dat extension (digitalized signal (s)) and .hea extension (header file). In Device Signal Simulator, we have used the *mgh001*[12] waveform database. The headers for this waveform record include the following nine elements: 1. Timestamp 2. ECG lead I, 3. ECG lead II, 4. ECG lead V, 5. ART, 6. PAP, 7. CVP, 8. Resp. Imp, and 9. CO2 which are described in the header file of the data set. We have implemented a C program which simulates the sensor behavior by reading signal data from *mgh001*. In effect, this utility iterates over total records of database and will send each record including nine fields over a TCP socket connection to DDC. DDC will receive each record and tokenize the `value` into eight five-bits signal digits.

Each running SVM which is located in an ICU room has a unique deployment ID. This enables us to extract information for each SVM such as its IP address, location and the hospitalized patient ID. In our implementation we have defined a SVM_ID variable in DDA kit. At this moment we have assigned a default value to this variable to run the prototype. DDA aggregates sensor data and sub-joins the SVM_ID to the aggregated data message, then sends it to the LDM. But in a real world application, each SVM will ask the deployment server a unique ID at startup. Then the server sends a unique ID (SVM unique ID + patient ID) to the SVM device. This ID will be used in DDA kit.

In the Sedona framework, we implemented *Device Data Collector (DDC)* and *Device Data Aggregator (DDA)* applications which receives sensor data through its listeners via TCP Socket and aggregates the data for further processing. More specifically, DDA kit receives data from proper input slots which are linked to the outputs of DDC kit. The output-input link tags are defined in DDA sax file. This kit merges all inputs separated by semicolon and sends the aggregated byte stream via a TCP Socket object on an IP and port number which our next component, *Device Data Retrieval (DDR)* is hosted.

B. Artefact Manifest Schema

In our aggregator application, we define properties of WFDB fields as the meta-data for this SVM environment. These properties will be stored in the *manifest* XML file in compile-time. Then we use Sox protocol to retrieve each property of the manifest using the *Manifest Schema Loader (MSL)* in the next layer. To communicate with our Sedona device remotely and retrieve information about running Sedona VM, we used Datagram Authentication Session Protocol (DASP)[13]. In effect, we used DASP Socket in our middleware to send a multi-cast group request (DISCOVER request). Therefore each machine running Sedona VM (as server) will respond to the client. Notice that the discovery request will query a specific port address on destination machine which in our case is Sedona default port number 1876. As soon as each machine receive the discover request, it will response with the Platform ID and its IP address. In our case the DASP message received from server contain following SVM information: `Discovered SVM IP/Port: 192.168.1.3:1876` and `Platform ID: tridium-generic-unix-1.2.28`. Such information has two use-cases of (i) sending commands to control the device from the analytic endpoint side and, (ii) as a heartbeat to check if the running SVMs are still alive and healthy.

C. Sensor Linked Data Model

The detected events, which in our particular case are user health vital information, can be modelled in an ontology. The ontology model as depicted in 3, consists of two parts of sensor and patient parts. The patient part represents her personal and medical profile like hospitalization ID and the detected diagnosis, for instance. Every profile is linked via hospitalization ID to deployment ID in sensor ontology.

Sensor ontology constitute a network of sensors which are connected to a Sedona device and deployed in a Sedona Virtual Machine (SVM). Such sensors read vital data from the patient. For modeling sensor properties, capabilities and their observations we incorporated the W3C SSN ontology[14]. We utilize this ontology by representing our located Sedona device instance. The Sedona VM is mapped to the $system$ class which is deployed on a platform with a specific deployment ID. The sensors are considered as $subsystems$ and declared as $slots$. Slots have properties like ID, label, measurement unit

[12]http://physionet.org/physiobank/database/mghdb/mgh001.hea

[13]http://www.sedonadev.org/doc/dasp.html

[14]http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

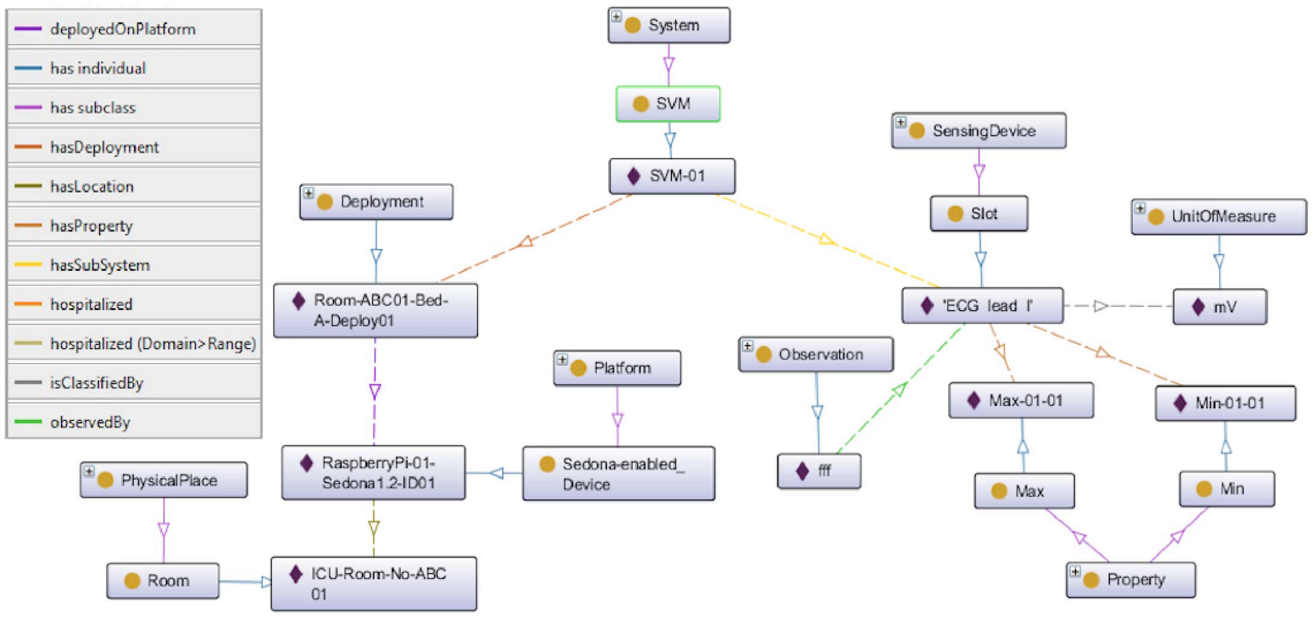


Fig. 3: Sensor ontology model representing the classes, relations and instances

together with their ranges like min or max values. They expose their located context observation results.

As shown in Fig. 2. in addition to using object and data properties of SSN ontology, we defined $hasID$, $hasObservationResultValue$ and $observationResultTime$ properties. The $hasId$ is defined for assigning the related IDs to SVM, slot and deployment. As such, the $hasObservationResultValue$ is defined for assigning sensor readings value to observation of each slot. And finally, the $observationResultTime$ for assigning an integer as a timestamp of reading observation of each slot. The ontology is built and loaded into the TDB RDF database[15] using the Jena TDB APIs to store and retrieve semantic sensor data in RDF graphs. We developed our sensor ontology in Protege[16]. Next, the manifest schema and observed values are written to TDB. We have used its APIs to store and expose our enriched sensor data as Triple RDF statements. Meanwhile some sample patient hospitilization data is stored in the database for future retrieval.

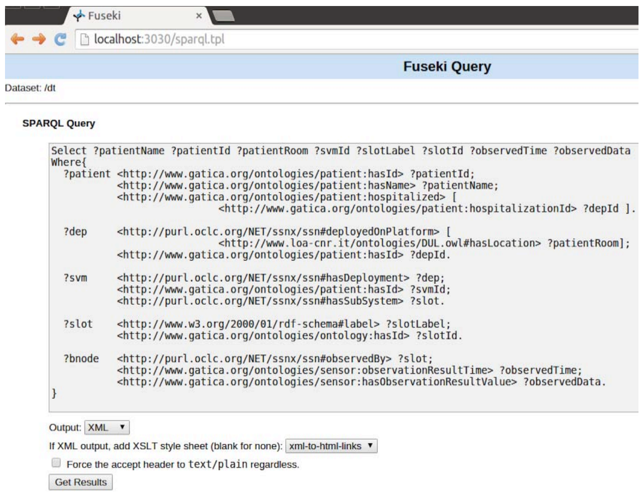
D. Manifest Schema Loader

We need to fetch the meta data of the deployed sensors (slots) on the Sedona device. The manifest file contains annotation data related to each slot such as min, max or a measurement unit. We use Sedona APIs to read such data that exists in the manifest file. A simple sox client is implemented which communicates, authenticates and listens on a sox port. This thin client requests slot information like names, values, flags, and annotations from the manifest schema. The response holds the manifest data which will be written to the TDB afterwards.

[15]https://jena.apache.org/documentation/tdb
[16]http://protege.stanford.edu

E. Device Data Retrieval

Next is to store the actual sensor observation values in the DB. As described before, from each sensor the stream of data (observations) are loaded in the Sedona device. Our Sedona main application merges the SVM ID, slot ID, observed value and its time-stamp, then sends them via TCP to a specific port. At last, the DDR receives the data periodically and stores them in the TDB.



Dataset: /dt

SPARQL Query

```
Select ?patientName ?patientId ?patientRoom ?svmId ?slotLabel ?slotId ?observedTime ?observedData
Where{
  ?patient <http://www.gatica.org/ontologies/patient:hasId> ?patientId;
           <http://www.gatica.org/ontologies/patient:hasName> ?patientName;
           <http://www.gatica.org/ontologies/patient:hospitalized> [
                           <http://www.gatica.org/ontologies/patient:hospitalizationId> ?depId ].

  ?dep     <http://purl.oclc.org/NET/ssnx/ssn#deployedOnPlatform> [
                           <http://www.loa-cnr.it/ontologies/DUL.owl#hasLocation> ?patientRoom];
           <http://www.gatica.org/ontologies/patient:hasId> ?depId.

  ?svm     <http://purl.oclc.org/NET/ssnx/ssn#hasDeployment> ?dep;
           <http://www.gatica.org/ontologies/patient:hasId> ?svmId;
           <http://purl.oclc.org/NET/ssnx/ssn#hasSubSystem> ?slot.

  ?slot    <http://www.w3.org/2000/01/rdf-schema#label> ?slotLabel;
           <http://www.gatica.org/ontologies/ontology:hasId> ?slotId.

  ?bnode   <http://purl.oclc.org/NET/ssnx/ssn#observedBy> ?slot;
           <http://www.gatica.org/ontologies/sensor:observationResultTime> ?observedTime;
           <http://www.gatica.org/ontologies/sensor:hasObservationResultValue> ?observedData.
}
```

Output: XML

If XML output, add XSLT style sheet (blank for none): xml-to-html-links

☐ Force the accept header to text/plain regardless.

Get Results

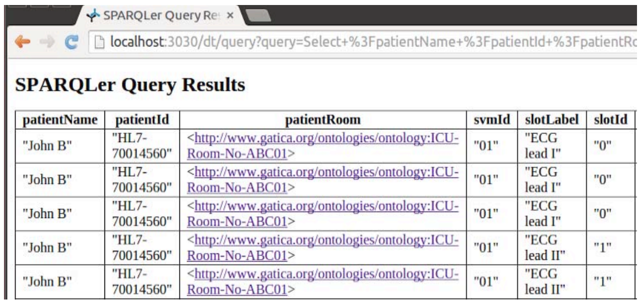
Fig. 4: Gatica middleware analytic query endpoint

F. Analytic Query Endpoint

As shown in Fig. 4. we implement SPARQL querying interface on the TDB by using Fuseki[17] to serve RDF data

[17]http://jena.apache.org/documentation/serving_data

over HTTP. After the Fuseki server is running, a SPARQL endpoint is provided to respond to various SPARQL requests. A sample response is presented in Fig. 5. Various analytic applications can send SPARQL request to our Fuseki server and retrieve the results.


SPARQLer Query Re ×

localhost:3030/dt/query?query=Select+%3FpatientName+%3FpatientId+%3FpatientRc

SPARQLer Query Results

patientName	patientId	patientRoom	svmId	slotLabel	slotId
"John B"	"HL7-70014560"	<http://www.gatica.org/ontologies/ontology:ICU-Room-No-ABC01>	"01"	"ECG lead I"	"0"
"John B"	"HL7-70014560"	<http://www.gatica.org/ontologies/ontology:ICU-Room-No-ABC01>	"01"	"ECG lead I"	"0"
"John B"	"HL7-70014560"	<http://www.gatica.org/ontologies/ontology:ICU-Room-No-ABC01>	"01"	"ECG lead I"	"0"
"John B"	"HL7-70014560"	<http://www.gatica.org/ontologies/ontology:ICU-Room-No-ABC01>	"01"	"ECG lead II"	"1"
"John B"	"HL7-70014560"	<http://www.gatica.org/ontologies/ontology:ICU-Room-No-ABC01>	"01"	"ECG lead II"	"1"

Fig. 5: Sample response from the query interface

V. Related Work

There is some valuable research regarding the IoT semantic sensor streaming which elevates the semantic technologies to Internet of Things domain. Banerjee et al.[6] proposed an architecture for a semantic search engine on sensor data where the sensed data is represented in form of triples (RDF), concepts and relations in form of ontologies (OWL) and the corresponding query language is SPARQL. In relation to our approach, Le-Phuoc et al.[7], [8] survey the state of the art Linked Stream Data processing systems, and highlights a comparison among them regarding the design choices. Authors in [9], propose a model and take a linked data approach to annotate the streams. These papers do not provide any implementation or evaluation of the proposed architecture. They also proposed a Linked Sensor Middleware (LSM) [10] that provides real-time data collection mechanisms using a cloud-based infrastructure. It also features a web interface for annotating and visualizing linked sensor data accompanied with a SPARQL endpoint for querying streaming data. Their focus is on annotating and provisioning of the data using unified interfaces.

None of these solutions enable a lightweight IoT deployment model and is not applicable to resource-constraint IoT devices like an IoT gateway environment. In contrast to all the noted related work, our solution addresses universal and large-scale application deployments in very constrained embedded environments. The Sedona framework is deployable on very resource limited devices with small memory, even less than 100 KB. Our solution is also based on a micro-service architecture as each Sedona application is highly coherent and decoupled from others. Last but not least, each container in the gateway is accompanied with the manifest schema which enables dynamic configurations of the annotations.

VI. Conclusion

In this paper, the authors offered and implemented a resource-constrained middleware to enable the semantic IoT linked data analytics. It dynamically injects semantics to make the IoT raw sensor data enriched and meaningful. For modeling a sensor's properties, capabilities and their observations, we extended the SSN ontology which represents our gateway containers. Gatica collects the real-time sensor data via gateways, enrich them using annotations then transforms and exposes them in RDF triples. Using principle component analysis we are able to cluster the data and run queries over the streaming sensor data to discover hidden patterns via analytic interfaces. We have evaluated our model with a real-world healthcare dataset to demonstrate the utility of our framework.

As an outlook, our future work includes further extension to the Gatica middleware to support large scale distribution of data processing capabilities over the increasing streaming linked sensor data.

Acknowledgment

The research leading to these results is sponsored by the Doctoral College of Adaptive Distributed Systems (ADSys)[18] at the Vienna University of Technology.

References

[1] T. G. P. M. Mell;, vol. The NIST Definition of Cloud Computing. NIST SP - 800-145, September 28, 2011.

[2] D. Sow, D. Turaga, and M. Schmidt, "Mining of sensor data in healthcare: A survey," in *Managing and Mining Sensor Data*, C. C. Aggarwal, Ed. Springer US, 2013, pp. 459–504. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-6309-2_14

[3] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000. [Online]. Available: http://circ.ahajournals.org/content/101/23/e215.abstract

[4] B. D. Abdi, Herve, *Principal Component and Correspondence Analyses Using R*, ser. SpringerBriefs in Statistics, 2015. [Online]. Available: http://www.springer.com/gp/book/9783319092553

[5] R. Watanabe, E. Morett, and E. Vallejo, "Inferring modules of functionally interacting proteins using the bond energy algorithm," *BMC Bioinformatics*, vol. 9, no. 1, 2008. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-9-285

[6] S. Banerjee, A. Mishra, and R. Dasgupta, "Semantic exploration of sensor data," in *Proceedings of the 5th International Workshop on Web-scale Knowledge Representation Retrieval &#38; Reasoning*, ser. Web-KR '14. New York, NY, USA: ACM, 2014, pp. 55–58. [Online]. Available: http://doi.acm.org/10.1145/2663792.2663800

[7] D. Le-Phuoc, J. Xavier Parreira, and M. Hauswirth, "Linked stream data processing," in *Reasoning Web. Semantic Technologies for Advanced Query Answering*, ser. Lecture Notes in Computer Science, T. Eiter and T. Krennwallner, Eds. Springer Berlin Heidelberg, 2012, vol. 7487, pp. 245–289. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33158-9_7

[8] D. Le-Phuoc, M. Dao-Tran, M. . Pham, P. Boncz, T. Eiter, and M. Fink, *Linked stream data processing engines: Facts and figures*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2012, vol. 7650 LNCS, no. PART 2. [Online]. Available: www.scopus.com

[9] P. Barnaghi, W. Wang, L. Dong, and C. Wang, "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*, Aug 2013, pp. 468–475.

[10] D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth, "A middleware framework for scalable management of linked streams," *Web Semant.*, vol. 16, pp. 42–51, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2012.06.003

[18] http://www.big.tuwien.ac.at/adaptive