Towards a semantic edge processing of sensor data in a smart factory

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Paula-Georgiana Zălhan[a,*], Gheorghe Silaghi[a] and Robert Buchmannn[a]
[a]*Business Informatics Research Center, Babeș-Bolyai University of Cluj-Napoca, Teodor Mihali 58-60, 400591 Cluj-Napoca, Romania*

Abstract. The global world is experiencing a digital revolution with Industry 4.0 and Internet of Things (IoT) which enable enterprises to optimize their production processes using intelligent, interconnected equipment and sensing devices. By harnessing the data generated from sensor networks, smart factories can remotely monitor their assets, and can make decentralized decisions. However, handling massive amounts of data that are continuously generated with a fast rate within the industrial environment is a demanding task. Semantic technologies have shown great promise in achieving machine-interpretability of IoT data. This paper proposes a semantic sensor stream processing pipeline that employs Apache Kafka to annotate in a scalable way the sensor data using the Semantic Sensor Network ontology, then store the annotated output in a RDF triplestore for reasoning. We are investigating whether it is preferable to do this on the consumer or on the producer side, considering the publish-subscribe model of Apache Kafka. The Design Science approach was followed, motivated by a Smart Factory application scenario that involves sensors for heartbeat, proximity and location. In this setup, we are investigating where it is preferable to execute the semantic annotation in a distributed, Kafka-based configuration. The experimental evaluations show that the "semantic edge" approach fulfills the low-latency processing requirement.

Keywords: Semantic stream processing, semantic edge, sensor data, Apache Kafka, Semantic Sensor Network ontology

1. Introduction

Ubiquitous computing, intelligent robots, autonomous systems, remote monitoring, are just a few evidences that we are witnessing a revolutionary change. Billions of entities (computing devices, complex physical machines, objects, or people) are connected to digital networks, dramatically increasing the efficiency of organizations and the societal outcomes [41][40]. In this context, Industry 4.0 is becoming a top priority for researchers and manufacturers, since it provides many economic benefits and enables smart production of services and goods.

Industry 4.0 promises what has been referred to as a “smart factory”, where machinery and equipment improve the effectiveness of manufacturing processes. In this regard, Industrial Wireless Networks (IWNs) have the potential to connect and integrate smart entities (such as workmen, machines, sensing devices) with traditional industries to provide greater flexibility, intelligence and adaptation [28][27].

The main objective of Industry 4.0 is to facilitate the effective communication between the interconnected entities and ensure cooperation among differ-

*Corresponding author. E-mail: paula.zalhan@econ.ubbcluj.ro.

ent factories that may be located in different remote places.

However, the heterogeneity of these entities makes the interoperability a difficult task. These entities collect and transmit massive amounts of data regarding the working environment and machine status, which requires low latency processing capabilities to improve productivity and to enable real-time monitoring of the industrial site. The diversity, volatility and veracity of data streams impose significant challenges to existing industrial systems.

Thus, innovative solutions are demanded to overcome these issues – not only in terms of Big Data challenges, but also in terms of Smart Data. Quantity-oriented methods must converge with semantics-oriented techniques towards building processing pipelines that fulfil the needs of Industry 4.0 environments. This paper is part of a larger effort in addressing research challenges that derive from this convergence, approached through the methodological lens of Design Science research [48].

The motivating design problem context is to support a Smart Factory with a safer workspace allowing emergency interventions and incident management. Geolocation, heartbeat and proximity sensors become relevant for preventing workplace injuries or death of employees. The architectural core that streamlines the sensor data collection, semantic annotation and reasoning is built with the help of a tool pipeline that includes: the Apache Kafka distributed streaming platform [2], the GraphDB semantic database server [20] and the Semantic Sensor Network (SSN) ontology [42] extended with additional classes and properties through semantic reasoning to model sensor types and critical sensor observations. To build the best Semantic Stream Processing (SSP) pipeline that fulfills the low latency requirement of the smart factory application, an iterative development cycle of the pipeline is adopted, assessing two alternatives architectures of the SSP system: (i) semantic annotation after the sensor data is ingested into the real-time system, and (ii) semantic annotation on the "edge" – i.e., at the producer point. The experiments reported in this paper support the decision on which of the two approaches should be taken in order to advance our Smart Factory pipeline beyond the current, proof-of-concept status.

The remainder of this paper is organized as follows. Section 2 reviews the state of the art Semantic Stream Processing systems. Section 3 describes the adopted methodology when building the pipeline. Section 0 exposes the proposed semantic stream processing pipeline. Section 5 presents a real-world use case. Section 6 describes the implemented alternatives and reports the results in terms of performance of the evaluated pipeline architectures. Finally, Section 127 concludes the work.

2. Related work

In recent years, the Semantic Web community has worked on developing tools and techniques to represent, integrate and reason upon data [9]. Their adoption in real-time processing of data-streams ensures the communication between smart entities by providing machine-interpretable descriptions of the underlying data. Semantic enrichment of data coming from heterogeneous sources deployed in the industrial environment increases interoperability and provides contextual information which is essential for situational knowledge [27].

The streaming data generated by multiple heterogeneous data sources introduced several challenges in addressing their dynamic and time-dependent nature. This opened a new research trend in the Semantic Web community, called RDF Stream Processing (RSP) or Linked Data Stream Processing [24], dealing with fast changing, temporal data that can be modelled by means of the RDF model [37], the most popular standard for Linked Data representation. In the last decade, efforts have been spent to build RSP engines that represent streaming data using RDF. In recent years, several RSP engines such as C-SPARQL [8], CQELS [23], and $\text{SPARQL}_{\text{stream}}$ [13] have emerged which allow applying continuous queries over streams using extensions of SPARQL [44].

To overcome these discrepancies which reflect different semantics of existing RSP systems, various efforts were done to create a unifying and comprehensive query model that generalizes the existing SPARQL extensions for evaluating continuous queries over RDF streams. For example, a unifying query model named RSP-QL is proposed in [16] to formally define the semantics of a RSP system using a SPARQL-extended query language. Furthermore, these centralized RSP systems are not capable of handling massive amounts of data streams, as they do not benefit from task parallelism and the scalability offered by a cluster computing infrastructure. To remedy these limitations and improve the performance of existing RSP systems, distributed RDF streaming systems were designed to enable concurrent queries over the incoming data. For example, CQELS Cloud system [25] uses Apache Storm [5] and Strider [38] uses Apache Spark [4] to parallelize

the continuous execution of queries over RDF data streams in the Cloud.

Meanwhile, several middleware solutions were proposed to transform unstructured streaming data into RDF streams reusing the Semantic Web tools stack. For example, the DataTurbine engine [19] is a robust streaming data engine, delivering data in near real-time from various sources to a data center for later analysis using a buffered middleware. Another framework called Linked Stream Middleware (LSM) described in [26] provides a cloud-based infrastructure to annotate the collected sensor streams, using W3Cs Semantic Sensor Networks Incubator Group (SSN-XG) [43] ontology. The LSM system uses RabbitMQ [35] publish-subscribe messaging platform as Message Bus and Virtuoso [33] as triple storage. Later, Ztreamy middleware solution [18] was proposed for large scale publishing of semantically-annotated data streams on the Web. The authors show that Ztreamy outperforms existing middleware systems such as DataTurbine and LSM in terms of scalability and latency being able to process in real-time data streams gathered from an immense number of simultaneous data producers. A recent framework called SEASOR [31] includes features from both centralized and distributed RSP engines providing semantic annotation of the summarized sensor data streams using the SSN ontology and query processing of the arriving annotated streams using Cs-SPARQL, an extension of SPARQL that supports windows and parallel processing of data streams. The experiments regarding the performance evaluation of the framework reveal that SEASOR manages to reduce considerably the query execution time outperforming CQELS engine or LSM.

Another research topic called Stream Reasoning [17] has emerged from SSP in order to address the changing nature of streaming data using inference techniques by extending the traditional RSP engines with rule-based, logical, reasoning capabilities. LARS [9] framework formally expresses and analyses streams by extending Answer Set Programming foundations with reasoning capabilities. A logic-based framework [39] was introduced to perform stream reasoning over data streams based on temporal Datalog, a declarative rule-based language. A robust system called Streaming MASSIF [11] uses Cascading Reasoning approach to process massive amounts of heterogeneous events using expressive reasoning capabilities. Another approach based on stream reasoning models and techniques to process semantically-enriched data streams from a Smart City context was discussed in [15].

A remarkable extension to logic-based reasoning is presented in [7] where authors use a combined approach of deductive and inductive stream reasoning to support analysis of streaming data. A similar effort is highlighted in [14] where authors focus on supervised stream learning of semantically enriched traffic data using Description Logic reasoning in order to tackle the problem of concept drift. More recently, semantic reasoning techniques have also been applied to the field of robotics [28].

3. Methodology

The main goal of our research is to facilitate the streaming data analysis in an industrial environment by designing a novel semantic stream processing pipeline which includes data collection, semantic annotation, RDF data storage and query processing. In order to attain this goal, this paper focuses on the core pipeline design by answering the following design research question: "How to build the low-latency stream processing pipeline in order to sustain the semantic enrichment of the sensor data generated in a smart factory environment?" We are specifically interested in deciding the point in the pipeline where the semantic annotation should take place, considering the publish-subscribe model that is prevalent in IoT (and, specifically, in the Apache Kafka platform employed for our work).

When building this pipeline, the Design Science Methodology (DMS) was adopted, following an iterative treatment development cycle with the explicit intention of improving the system's performance which is empirically investigated taking into consideration a smart factory scenario. The stakeholders are emergency response teams which need to rapidly react for the rescue of workmen in case of critical observations have been sensed (and reasoned upon) within the factory environment.

The design process of the proposed pipeline iterates over two leading activities: 1) the scalable analytics pipeline design that enables rescue teams to take real-time actions, and 2) the subsequently investigation of the system's performance in terms of latency. Firstly, the semantic pipeline for sensor data processing is designed not only to support data analytics at scale following a Big Data reference architecture, but also to semantically enrich the underlying data using Semantic Web tool stack. Secondly, to investigate where is it preferable to execute the semantic annotation in order to deliver faster insights of the underlying sensor data, two alternative archi-

tectures were implemented: semantic annotation after the raw sensor data is ingested, and semantic annotation directly on the sensing devices output by means of semantic edge computing.

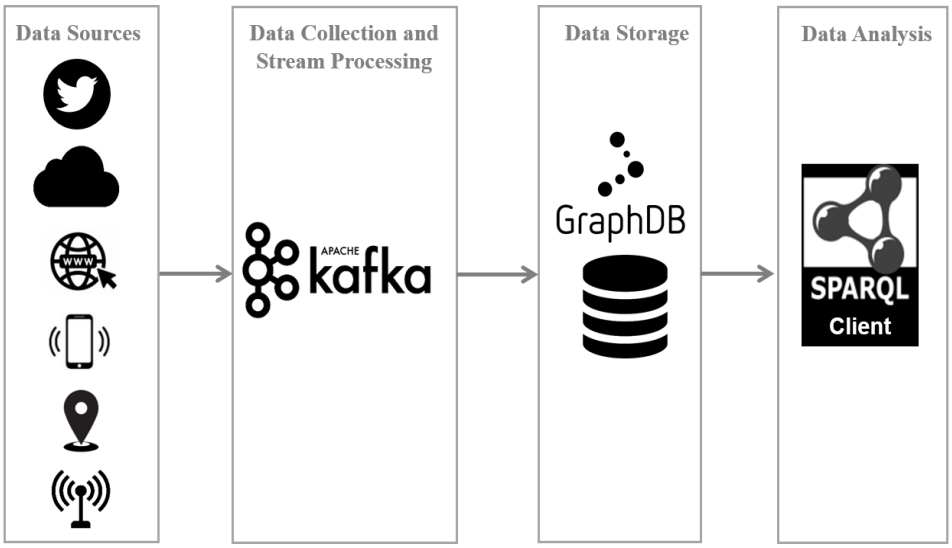

Fig. 1. Proposed semantic stream processing pipeline.

4. Real-time semantic stream processing pipeline

Implementing an effective real-time semantic stream processing pipeline architecture requires addressing several aspects including data collection, stream processing, data storage and analysis. The main components of the proposed pipeline are presented in Figure 1. This pipeline is based on Apache Kafka to collect and process the streaming data, GraphDB to store the annotated data streams and SPARQL query language to analyze the resulting graph.

In this paper, various sensors such as heartbeat, proximity and location sensors are taken into consideration to support a Smart Factory scenario. The continuous data gathered from streaming data sources is collected and processed by a distributed data ingestion system for later analysis and storage. Multiple tools can be used as data ingestion systems in a stream processing system such as Apache Kafka, Apache Nifi [3] and Apache Flume [1].

In this paper, Apache Kafka is employed due to its ability to scale the load as data is ingested into the system and to use replication to guard against data loss [30].

Apache Kafka is a high throughput, distributed messaging system that supports millions of messages for both publishing and subscribing. Message senders or "producers" write streams of records in categories called topics and message receivers or "consumers" read streams of records from specific topics. Each topic can be partitioned and replicated over multiple machines working in parallel, thus ensuring data availability and load balancing over consumer instances in case of failure. Kafka runs as a cluster which connects multiple producers and consumers to one or more servers, called brokers. Apache Kafka uses internally Apache Zookeeper [6] to store metadata about the Kafka cluster, such as information about topics, brokers and consumers.

In the Smart Factory scenario the incoming data is collected into topics and processed on the fly using the Kafka core APIs. To provide machine-readable and machine-interpretable descriptions of the ingested data, the streaming data previously collected is continuously processed using semantic technology. The corresponding stream of records are turned into richer information via annotation with the SSN ontology. This ontology mainly focuses on describing physical sensor networks, such as sensors, observations that result from sensing, and deployments in which sensors are used. It adopts some concepts from the Sensor, Observation, Sample, and Actuator (SOSA) ontology [21].

For storing the annotated data and to subject it to reasoning mechanisms, we chose GraphDB due to

the results obtained when assessing the performance of currently available RDF stores [11]. Due to the fact that GraphDB executes inferences at load time, its number of indexed triples is bigger than those of the other semantic databases. Also, GraphDB performs better when a reduced number of query results are demanded.

5. The Smart Factory application case

In order to test the effectiveness of the proposed SSP pipeline, a real-world scenario has been considered: a smart factory infrastructure with multiple wearable sensors including heartbeat sensor and geolocation sensor attached to a worker's body to monitor its heart rate, and its location in order provide fast emergency interventions by saving human lives in case of accidents at workplace. In this way, every worker is under constant supervision to determine its heart condition by giving intervention teams' to-the-second information about workers' health status. Proximity sensors are deployed within the industrial environment to sense a potentially dangerous situation such as a worker walking too close to a piece of heavy machinery due to limited visibility. In this case, the system provides environmental protection from invisible risks, by preventing workmen from getting injured and alerting supervisors if their employees are in trouble.

To simulate the data streaming from the aforementioned sensors deployed in a smart factory environment, Kafka Producer APIs were used to support custom implementations that write and read streams of data in a Kafka cluster. Producer tasks send messages in JSON format to the Kafka cluster in three different topics: *WorkerPulse* topic contains datastreams from the heartbeat sensor, *WorkerLocation* topic stores streaming data gathered from location sensors and *MachineProximity* topic contains messages published by deployed proximity sensors. The stream of records from these topics have the following core schema:

- sensor_id: UUID,
- sensor_type: String in heartbeat sensor, location sensor, or proximity sensor
- station_no: int,
- stream_id: int,
- event_value: String,
- event_time: Timestamp.

For the aforementioned topics, the *sensor_id* field represents the Universal Unique Identifier (UUID) to uniquely identify the deployed sensors from the smart factory ecosystem. Raspberry Pi stations identified by a specific *station_no* are deployed in each building of the industrial site and host several heartbeat, location and proximity sensors. Every sensor stream is uniquely identified (*stream_id* field) in dependence of the type of sensor and the number of the events/observations generated by that sensor. The *event_value* field from the above record definition represents the value captured by a specific sensor and has different meanings, according to the type of sensor that generated the data stream. The event_value field of the WorkerPulse topic schema stores the heart rate of a worker, while the event_value field from the record definition of MachineProximity topic provides information about the proximity level, as detected by a heavy machine's proximity sensor. In case of WorkerLocation topic, the event_value field records the geospatial coordinates of the worker's current location in the factory.

The streams of records of the aforementioned topics are published with timestamps marking the time when a sensor observation value was generated. Afterwards, the collected sensor data is annotated using the SSN ontology.

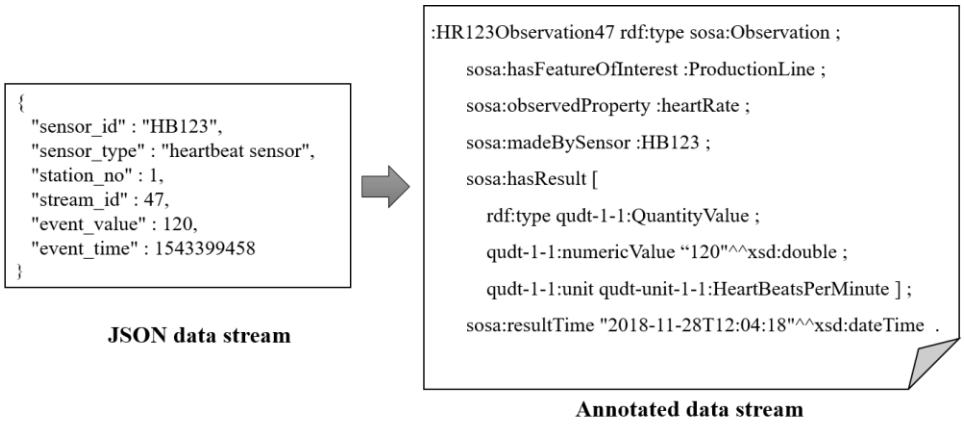

Fig. 3. Description of a heartbeat sensor observation.

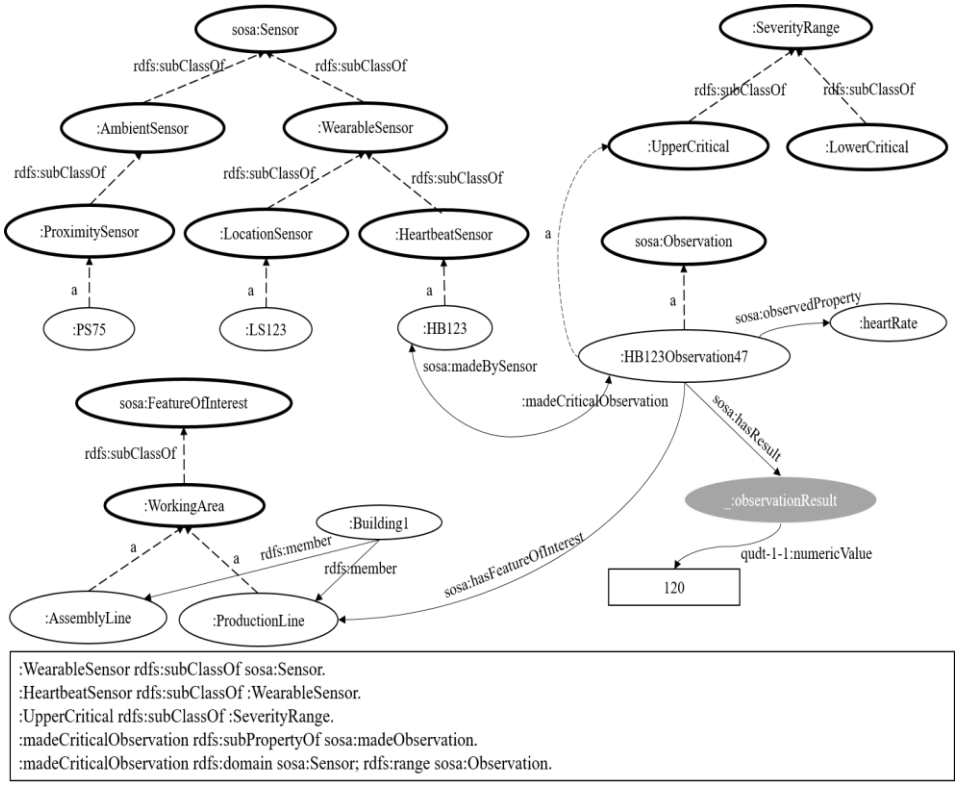

Fig. 2. Enriching SSN ontology with sensor type classes, critical observation classes and subproperties.

Understanding this context enables the SSP system to turn the raw sensor data into actionable knowledge using semantic annotation. Thus, when critical patterns are detected in the underlying data, the proposed system notifies intervention teams to take rapid actions in order to save still alive injured persons. Basically, the sensor streams are semantically annotated using SSN ontology by providing descriptions regarding to individual sensing devices, the relationship with their corresponding platform, their observation values and implied procedures, features of interest, and properties that were observed in the environment. Lastly, the resulted RDF descriptions are stored in a semantic database for later analysis that permits to extract machine-understandable meanings and valuable insights from the sensor data.

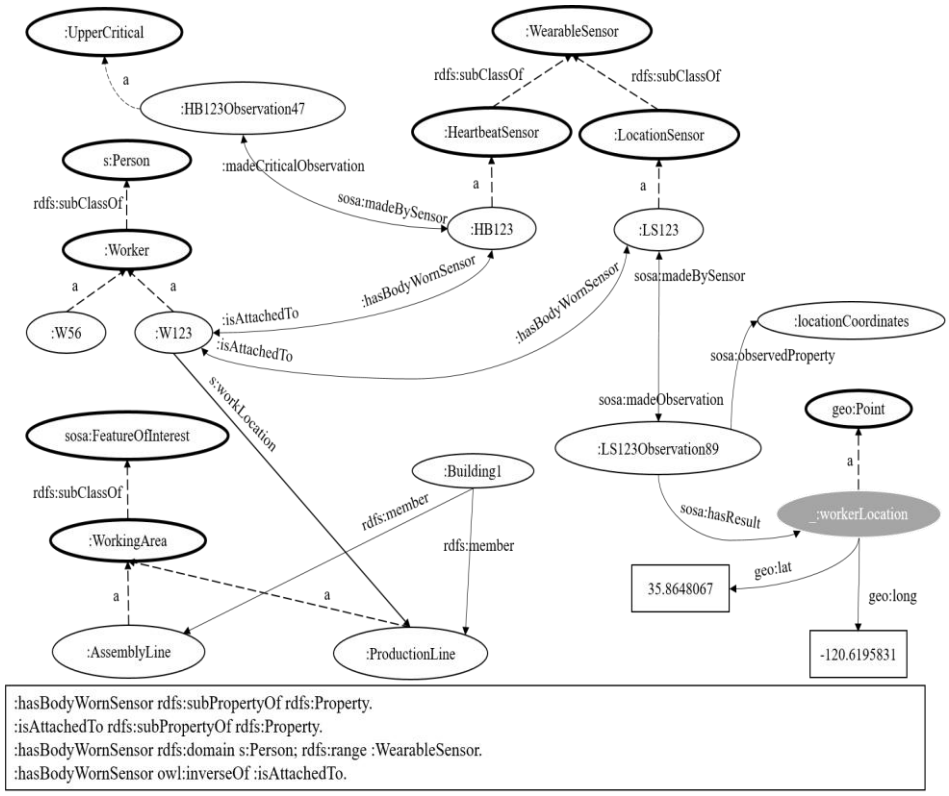

Fig. 4. Defining relations between factory personnel and wearable sensors.

The semantic annotation using the SSN ontology of a raw heartbeat sensor stream originally written in JSON format is shown in Figure 2. The *stream_id* field of this JSON data stream holds the observation counter value as an evidence of how many events were generated by a heartbeat sensor. The observation value gathered from the HB123 heartbeat sensor is stored in the *event_value* field and indicates a fast heart rate of a factory worker. The heartbeat sensor stream was generated at the production line of the first building (given by the *station_no* field value) of the industrial site where the Raspberry Pi station is deployed. To better understand the environment where the JSON sensor data was generated, the corresponding annotated data stream describes the observation made by the heartbeat sensor and explicitly link the property being analyzed (the heart rate) with the feature of interest (the production or assembly line where the heartbeat observation was made).

Due to the fact that some aspects such as detailed measure feature and units [22], location, mobility or other dynamic behaviors are not tackled by the existing SSN ontology, the RDF statements are constructed by combining the SSN ontology with external vocabularies such as geospatial vocabularies (WGS84 Geo Positioning) [47] to model the worker's location, and unit ontologies such as Quantities, Units, Dimensions and Data Types Ontologies (QUDT) [34] to provide machine readable access to unit and unit type information. For example, the heart beats per minute (bpm) standard unit from QUDT labeled as qudt-unit-1-1:HeartBeatsPerMinute is used to measure the worker's pulse. To encode the proximity level reported as the distance in centimeter from the sensor to a worker which is near to the heavy machine, the QUDT vocabulary offers the centimeter base unit of length labeled as "qudt-unit-1-1:CM". The resulted RDF descriptions are published into GraphDB for later analysis.

SSN does not model sensor types and models, representation of data, network data communication and topology. Consequently, some domain-specific extensions to the existing SSN ontology must be introduced targeting specific application cases. Firstly, the SSN ontology is extended with additional classes to model sensor types (see Figure 3) thus creating a hierarchy of sensor classes. Through multiple inheritance syntax, the sensor instances are categorized

into two main classes: :WearableSensor class corresponding to sensors types attached to human body, and :AmbientSensor class corresponding to sensor types deployed into the physical environment. For example, for the smart factory scenario, sosa:Sensor subclasses are defined to model the following sensor types: location, proximity and heartbeat sensors. Also, sosa:FeatureOfInterest class is extended to model specific working areas as (e.g., assembly line, production line) to have a clear evidence where the employee's labour activity is taking place in the factory. The rdfs:member super-property of the RDF Schema (RDFS) [36] vocabulary is used to specify the decomposition of a building from the industrial site into operative lines).

SPARQL-based reasoning is applied to categorize the heartbeat sensor observations into four main classes: LowerCritical (LCR), LowerNonCritical (LNC), UpperNonCritical (UNC) and UpperCritical (UCR) based on the observation value generated by the sensor. In this way, the SSP system provides a better understanding of the sensor readings by comparing them with threshold limits that define a normal resting heart rate. A heartbeat sensor reading below LCR or above UCR gathered in extreme cases of work incidents or natural disasters, indicate serious symptoms such as collapse, heart attack or sudden cardiac arrest. By monitoring workers' pulse in real-time, the SSP system can alert intervention teams to take rapid actions in case of critical patterns occurred.

The knowledge base is enriched with a set of axioms, defining RDF statements built with standard terms to establish machine-readable meaning for SSN properties and classes. In Figure 3, the axioms are meant to define a hierarchy of properties for the sosa:madeObservation superproperty that links a sensor instance with an upper/lower critical observation instance. The :madeCriticalObservation property defines the relation between a sensor and its upper critical observation (the observation was previously categorized as upper critical). To explicitly state that this relation connects sensors and observations, another axiom was declared using the rdfs:domain and rdfs:range terms from RDFS.

In Figure 4, simple reasoning based on RDFS and OWL is invoked, while adding new properties between worker instances and specific sensor type instances. The Schema.org [40] is also employed for established concepts (e.g., s:Person).

Once the axioms are persisted into the semantic database, the resulted knowledge base can further be inquired or reasoned upon through demonstrative client-initiated queries. The first query from Figure 5 defines a classification rule according to which if the result value of a heartbeat sensor observation exceeds a specified threshold then the sensor observation is

Query 1

```
insert { ?observation a :UpperCritical }
where
{
    ?observation sosa:hasResult/qudt-1-1:numericValue ?value
    filter (?value > 100)
}
```

Query 2

```
insert { ?sensor :madeCriticalObservation ?observation }
where
{
    ?observation a :UpperCritical;
          sosa:madeBySensor ?sensor
}
```

Query 2

```
select ?lat ?long
where
{
    ?observation a :UpperCritical; sosa:madeBySensor/:isAttachedTo ?worker .
    ?worker :hasBodyWornSensor ?sensor .
    ?sensor a :LocationSensor; sosa:madeObservation/sosa:hasResult/geo:lat ?lat .
    ?sensor a :LocationSensor; sosa:madeObservation/sosa:hasResult/geo:long ?long
}
```

Fig. 5. SPARQL-based queries to specify critical observations and relations in order to find the location coordinates of a worker if a critical observation was sensed.

considered upper critical. The direct connection between sensors and their critical observations is added through the :madeCriticalObservation property by executing the second query in Figure 5. To find the latitude and longitude coordinates of a worker whose body-worn heartbeat sensor has detected an upper critical observation, the existing RDF database is queried by following the chain of properties from the specific critical observation to the value of the location coordinates as it can be seen in the third query of Figure 5. This query is useful for emergency teams to know the location of a factory worker whose heartbeat sensor has sensed a severe heart condition or problem.

6. Performance evaluation and results

The performance of the deployed SSP pipeline from Section 0 depends heavily on the software and hardware settings. All the tests were carried out on Ubuntu 16.04.4 LTS x64-based PC with Intel(R) Core(TM) i7-7500U processor, 2.70 GHz CPU, 8 GB of RAM with Java version 1.8 and Java HotSpot(TM) 64-Bit Server VM. Confluent streaming platform version 4.0.0 was deployed with Apache Kafka version 1.0.0 and Apache Zookeeper version 3.4.10, which were the latest versions available at the time of building the SSP. For RDF parsing of the JSON streams was used RDFLlib 4.2.2 package with SPARQL 1.1 implementation. To store the annotated streams the GraphDB SE 8.7 version was used.

6.1. Experiment design

To identify where it is preferable to process the raw sensor data, two different architectures (see Figure 6) were implemented for experimental evaluation:

- Semantic annotation on consumer side (S1). The collected sensor streams are written to different topics in the Kafka cluster using producers. The published streams are then fetched using consumers that concurrently process the stream of records providing meaningful descriptions related to sensing devices, their observations and features of interest using semantic technology. Finally, the annotated sensor streams are persisted in the triple-store for later analysis.
- Semantic annotation on producer side (S2). The raw sensor data is processed near the sensing device that produced the observation value thus moving the semantic modelling to the *edge* of the network. In this case, the edge sensors collect and continuously annotate the incoming data transforming the sensor streams in RDF streams which are then categorized in different topics. Finally, consumers subscribe to these topics and store the RDF streams into the semantic database.

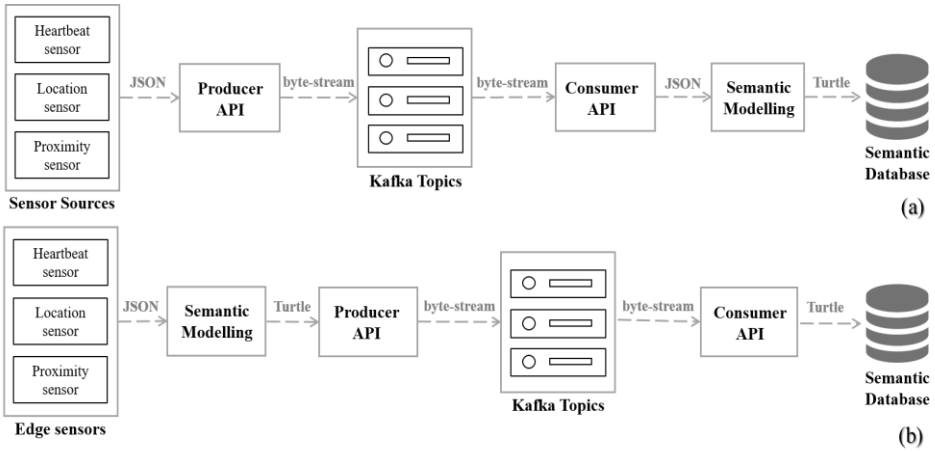

Fig. 6. Semantic modelling on consumer side (a) vs. semantic modelling on producer side (b).

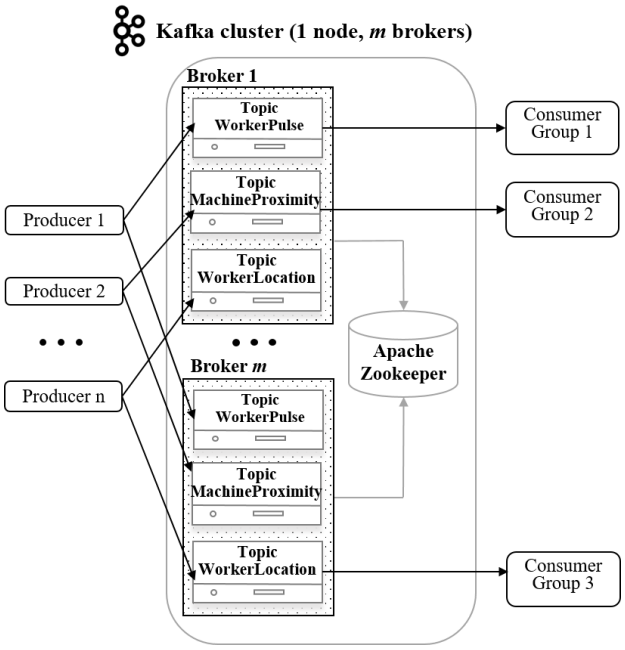

Fig. 7. Kafka cluster configuration consisting of a single node with multiple brokers and a Zookeeper instance.

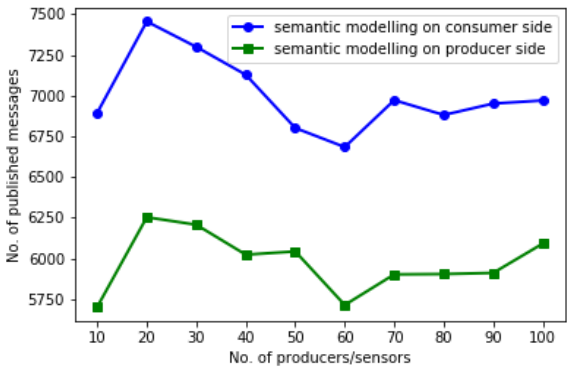

Fig. 8. Quantifying the published messages in both implemented architectures.

In both scenarios, the Kafka cluster was configured on a single machine, with multiple brokers and one Zookeeper instance as can be seen in Figure 7. The producers run in their own thread during a specific period of time and simultaneously publish the data streams to multi-partition topics. The timing of the produced data streams follows a Poisson process with a data rate that varies depending on the experiment. The Kafka cluster retains all the published stream of records, without consideration of their consumption. Using multiple partitions for each topic, Kafka guarantees write scalability across the servers and facilitates parallel data consumption within a consumer group.

6.2. Result analysis

The results shown in this paper are based on a three-broker Kafka cluster setup and take into consideration only heartbeat sensor streams that are published to a three-partition topic. A replication factor of three was used in both implemented architectures to store the data streams in a fault tolerant manner. The number of producers varies from 10 to 100, and each of them run for 600 seconds execution time.

A series of tests were carried out on producer side, and on consumer side, in order to compare the alternative architectures. To this end, the first experiment consisted in quantifying the published messages in both implementations. The results from Figure 8 show that in the edge computing architecture where the semantic modelling is done on the producer side there less messages are published to topics than in the architecture where the collected sensor data are semantically processed by consumers.

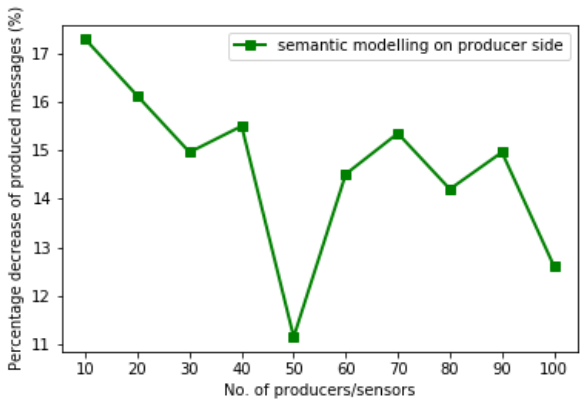

Fig. 9. Quantifying the percentage change of the written messages in the edge processing architecture.

Table 1 Generation time per message in semantic modelling on consumer side

No. of producers	No. of published messages	Generation time (ms) per message
10	6895	87.02
20	7455	80.48
30	7299	82.20
40	7128	84.18
50	6801	88.22
60	6684	89.77
70	6972	86.06
80	6881	87.20
90	6951	86.32
100	6970	86.08

Table 2 Generation time per message in semantic modelling on producer side

No. of producers	No. of published messages	Generation time (ms) per message
10	5703	105.21
20	6253	95.95
30	6207	96.67
40	6023	99.62
50	6043	99.29
60	5714	105.01
70	5902	101.66
80	5904	101.63
90	5911	101.51
100	6091	98.51

Processing the locally-collected sensor data by edge devices imposes a cost in terms of execution time necessary to annotate the sensor streams using the SSN ontology. The number of published messages from the implemented alternatives (see Figure 6) is compared in order to quantify the change of the amount of written stream of records in the edge architecture. The percentage decrease between the two number values is determined as a difference of published messages in both architectural designs. The results shown in Figure 9 reveal that when the semantic modelling is done on producer side, there were published with up to 17.29% less messages than when the semantic modelling is done on consumer side.

Tables 1 and 2 summarize the message generation time expressed in milliseconds when the semantic modeling is achieved in both implemented architectures (S1 and S2). It can be seen that the higher the number of published messages is, the lower the generation time of a message is, regardless where the sematic modelling is done. In this way, Kafka proves write scalability of the stream of records. When the semantic modelling is done on consumer side, the largest number of messages (7455, 7299 and 7128) are published by 20, 30 and 40 producers. In these cases, the generation time of a message is 80.48, 82.20 and 84.18, respectively. For other tests, with different number of producers, the number of published messages is below 6980 and the message generation time exceeds 86 milliseconds.

With respect of the semantic modelling at the edge of the network (see Table 2), the largest number of messages (6253, 6207, 6023, 6043 and 6091) are published by 20, 30, 40, 50 and 100 producers. In these cases, the message generation time varies from 95.95 to 99.62 and is lower than the generation time of a message published by 10, 60 to 90 producers. Comparing the message generation time in both implementations, it can be seen that publishing a message in S2 architecture requires more time due to the semantic modelling task that executes at the edge of the network. When the semantic annotation is accomplished on producer side, a message needs up to 18.19 milliseconds more to be generated, in dependence to the number of producers/sensors.

Another test that was carried out on consumer side is intended to see how many messages were consumed in the implemented alternatives over a certain period of time. Each consumer group consist of three consumer instances that subscribe to the Kafka topic and concurrently read the stream of records previously published during 600 seconds execution time. The chart in Figure 10 shows the evolution of consumed messages in both implementations. It can be seen that there are less consumed messages when the semantic annotation is accomplished by on consumer side. This is because the semantic modelling task imposes an IoT networking cost and a processing time for transforming the sensor streams into RDF streams.

Table 3 Annotated messages and corresponding RDF triples in both implemented architectures

No. of producers	Annotated messages (S1)	RDF triples (S1)	Annotated messages (S2)	RDF triples (S2)	Message difference (%)
10	529	8993	5703	96951	10.78
20	558	9486	6253	106301	11.21
30	696	11832	6207	105519	8.92
40	579	9843	6023	102391	10.40
50	357	6069	6043	102731	16.93
60	652	11084	5714	97138	8.76
70	707	12019	5902	100334	8.35
80	851	14467	5904	100368	6.94
90	751	12767	5911	100487	7.87
100	705	11985	6091	103547	8.64

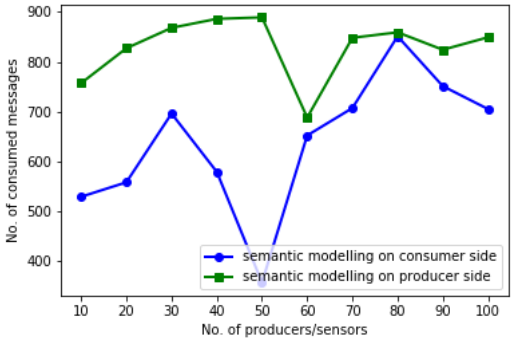

Fig. 10. Evolution of consumed messages during 600 seconds in both implementations.

Table 3 summarizes the number of annotated messages and their corresponding RDF triples, in the alternative architectures. It can be seen that when the semantic processing is handled locally by edge sensors, even ten times more annotated messages are generated. In the semantic edge architecture, the data does not waste time to travel through the pipeline. Thus, the edge computing architecture for semantic modelling of sensor streams provides advantages in terms of data processing power and network bandwidth reduction.

7. Conclusions and future work

To meet the low-latency demand of industrial systems, by sensing important parameters such as monitoring factory personnel, working environment and machine status, or delivering real-time insights of the underlying data, a semantic stream processing pipeline was proposed. When building the low-latency pipeline for streaming analysis of sensor data, the Design Science approach was adopted to investigate where is preferable to execute the semantic modelling of sensor data using the SSN ontology. Two alternative architectures of the sematic stream processing system have been implemented in the context of a smart factory to support real-time emergency interventions and workplace accident management. Experiments focus on assessing the performance of the alternative architectures of the semantic stream processing system and their results show that the semantic edge architecture is capable of producing an increased number of annotated sensor streams. The concurrent writes of the data producers facilitates the parallelization of the semantic enrichment of the sensor data. Processing the sensor data locally, not only reduces the physical distance of travelling data through the semantic pipeline, but has the potential to improve the Industrial Wireless Network's performance by combating latency.

Due to the fact that the SSN ontology does not model sensor types, data measurements units, location, mobility and other dynamic behaviors, the SSN ontology was extended with new domain-specific classes and relations. Axioms were constructed to define wearable sensor and ambient sensor classes, critical observations classes, relations between wearable sensors and their critical observations, as well as relations between factory personnel and their body-worn sensors.

As the tests were carried out on a Kafka cluster consisting of a single node, with multiple broker instances working in parallel and with Zookeeper as independent unit, the next goal is to setup a highly-distributed Kafka solution that consists of multiple nodes with multiple brokers. This configuration is desirable to advance the technological readiness level of our proposal and is on-going work that will employ a high-performance computing infrastructure.

References

[1] Apache Flume, https://ume.apache.org/.
[2] Apache Kafka, https://kafka.apache.org.

[3] Apache Nifi, https://nifi.apache.org/.
[4] Apache Spark, https://spark.apache.org/.
[5] Apache Storm, storm.apache.org/.
[6] Apache Zookeeper, https://zookeeper.apache.org/.
[7] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, Y. Huang, V. Tresp, A. Rettinger and H. Wermser, Deductive and inductive stream reasoning for semantic social media analytics, IEEE Intelligent Systems, 25 (2010), 32-41.
[8] D.F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus, An execution environment for C-SPARQL queries, in: Proceedings of the 13th International Conference on Extending Database Technology, 2010, pp. 441-452.
[9] P. Barnaghi, W. Wang, C. Henson and K. Taylor, Semantics for the Internet of Things: early progress and back to the future, International Journal on Semantic Web and Information Systems, 8 (2012), 1-21.
[10] H. Beck, M. Dao-Tran, T. Eiter and M. Fink, LARS: A logic-based framework for analyzing reasoning over streamsm, in: AAAI'15 Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI Press, 2015, pp. 1431–1438.
[11] P. Bellini and P. Nesi, Performance assessment of RDF graph databases for smart city services, Journal of Visual Languages & Computing, 45 (2018), 24-38.
[12] P. Bonte, R. Tommasini, E. Della Valle, F. De Turck and F. Ongenae, Streaming MASSIF: Cascading Reasoning for Efficient Processing of IoT Data Streams, Sensors 18 (2018), 3832.
[13] J.-P. Calbimonte, O. Corcho and A.J. Gray, Enabling ontology-based access to streaming data sources, in: Proceedings of the 9th International Semantic Web Conference, Springer, Berlin, Heidelberg, 2010, pp. 96-111.
[14] J. Chen, F. Lécué, J. Pan and H. Chen, Learning from ontology streams with semantic concept drift, in: Proceedings of 26th International Joint Conference on Artificial Intelligence, Sierra C, 2017, pp. 957-963.
[15] G. D'Aniello, M. Gaeta and F. Orciuoli, An approach based on semantic stream reasoning to support decision processes in smart cities, Telematics and Informatics 35 (2018), 68-81.
[16] D. Dell'Aglio, E. Della Valle, J.P. Calbimonte and O. Corcho, RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems, Semantic Web and Information Systems 10 (2014), 17-44.
[17] D. Dell'Aglio, E. Della Valle, F. van Harmelen and A. Bernstein, Stream reasoning: A survey and outlook, Data Science 1 (2017), 59-83.
[18] J.A. Fisteus, N.F. García, L.S. Fernández and D. Fuentes-Lorenzo, Ztreamy : A middleware for publishing semantic streams on the Web, Web Semantics: Science, Services Agents on the World Wide Web 25 (2014), 16–23.
[19] T. Fountain, S. Tilak, P. Shin and M. Nekrasov, The open source dataturbine initiative: empowering the scientific community with streaming data middleware, The Bulletin of the Ecological Society of America 93 (2012), 242-252.
[20] GraphDB, http://graphdb.ontotext.com/.
[21] K. Janowicz, A. Haller, S.J. Cox, D. Le Phuoc and M. Lefrançois, SOSA: A lightweight ontology for sensors, observations, samples, and actuators, Journal of Web Semantics (2018).
[22] C. Lai, A. Pintus and A. Serra, Using the Web of Data in Semantic Sensor Networks, in: CISIS 2017. Advances in Intelligent Systems and Computing, ed., Springer, Cham, 2018, vol. 611, pp. 106-116.
[23] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira and M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: Proceedings of 10th International Semantic Web Conference, Springer, Berlin, Heidelberg, 2011, pp. 370-388.
[24] D. Le-Phuoc, J.X. Parreira and M. Hauswirth, Linked Stream Data Processing, in: Reasoning Web. Semantic Technologies for Advanced Query Answering. Reasoning Web 2012, Lecture Notes in Computer Science, T. Eiter and T. Krennwallner, ed., Springer, Berlin, Heidelberg, 2012, vol. 7487, pp. 245-289.
[25] D. Le-Phuoc, H. Nguyen Mau Quoc, C. Le Van, M. Hauswirth, Elastic and Scalable Processing of Linked Stream Data in the Cloud, in: The Semantic Web-ISWC 2013. ISWC 2013. Lecture Notes in Computer Science, H. Alani et al., ed., Springer, Berlin, Heidelberg, 2013, vol. 8218, pp. 280-297.
[26] 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 for the World Wide Web, 16 (2012), 42-51.
[27] D. Le Phuoc and M. Hauswirth, Semantic Stream Processing, Encyclopedia of Big Data Technologies, 2019.
[28] X. Li, D. Li, J. Wan, A.V. Vasilakos, C.F. Lai, C.F. and S. Wang, A review of industrial wireless networks in the context of Industry 4.0, Wireless networks, 23 (2017), 23-41.
[29] T. Morita, K. Nakamura, H. Komatsushiro and T. Yamaguchi, PRINTEPS : An Integrated Intelligent Application Development Platform based on Stream Reasoning, The Review of Socionetwork Strategies 12 (2018), 71–96.
[30] N. Narkhede, G. Shapira and T. Palino, Kafka: The Definitive Guide: Real-time Data and Stream Processing at Scale, ed., O'Reilly Media, Inc., 2017.
[31] S. Pacha, S. Ramalingam and R. Sethukarasi, Semantic annotation of summarized sensor data stream for effective query processing, The Journal of Supercomputing (2017), 1-23.
[32] N. Prat, I. Comyn-Wattiau and J. Akoka, Artifact evaluation in information systems design science research a holistic view, in: Proceedings of the 19th Pacific Asia Conference on Information Systems (PACIS 2014), 2014, pp. 23.
[33] OpenLink Virtuoso, https://virtuoso.openlinksw.com/
[34] QUDT, http://www.qudt.org/.
[35] RabbitMQ, https://www.rabbitmq.com/.
[36] RDF Schema 1.1, https://www.w3.org/TR/rdf-schema/.
[37] Resource Description Framework, https://www.w3.org/RDF/.
[38] X. Ren and O. Cur, Strider: A hybrid adaptive distributed RDF stream processing engine, in: International Semantic Web Conference, Springer, 2017, pp. 559-576.
[39] A. Ronca, M. Kaminski, B.C. Grau, B. Motik and I. Horrocks, Stream reasoning in temporal Datalog, in: Proceedings of the 32th Conference on Artificial Intelligence (AAAI 2018), AAAI Press, New Orleans, LA, USA, 2018.
[40] Schema.org, https://schema.org/.
[41] K. Schwab, The Fourth Industrial Revolution, World Economic Forum, Geneva, 2016.
[42] Semantic Sensor Network Ontology, https://www.w3.org/TR/ /vocab-ssn/.
[43] Semantic Sensor Network XG, http://www.w3.org/2005/ /Incubator/ssn/.
[44] SPARQL 1.1 Query Language, https://www.w3.org/TR/ /sparql11-query/.
[45] Stardog, https://www.stardog.com/.
[46] Web Ontology Language (OWL), https://www.w3.org/OWL/.
[47] WGS84 Geo Positioning: an RDF vocabulary, https://www.w3.org/2003/01/geo/wgs84_pos
[48] R. J. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, Berlin, Heidelberg, 2014.