Generic semantic platform for the user-friendly development of intelligent IoT services

Pieter Bonte, Femke Ongenae, and Filip De Turck

Ghent University - iMinds, Gaston Crommenlaan 8, 9000 Ghent, Belgium
Pieter.Bonte@intec.ugent.be

Abstract The rising popularity of the Internet of Things (IoT) requires simple and user-friendly methods for designing and deploying intelligent IoT services. End-to-End IoT platforms should support the straightforward integration of various sensors and devices, provide user-friendly methods to define the complex processing that should be performed on the captured IoT data and allow easy deployment on Big Data frameworks to ensure scalability.
In this paper we present a generic and scalable platform that allows the automatic detection and integration of various sensors, provides intelligent processing of the captured data through semantic reasoning technologies and scalable and reliable deployment on a Big Data infrastructure. The configuration of the platform and the definition of its intelligent processes can be performed through a visual interface. The focus of the platform is to provide user-friendly support for defining complex IoT applications with limited to no need for coding.

1 Introduction

In the Internet of Things (IoT) paradigm, numerous devices are connected to the Internet [14]. Their sensor reading are captured and further processed to enable automated and intelligent decision making based on the sensed environment. To achieve this, understanding the raw sensor data is necessary. Collection, modeling, reasoning, and distribution of context in relation to sensor data plays a critical role in order to realize the IoT vision [16]. Semantic Web Technologies can ideally be used for this, since they have proven their use in data integration, knowledge extraction and reasoning [2, 6, 19, 20]. Semantics can aid in the integration of the heterogeneous IoT data, enabling interoperability between different sources and providing a uniform model. Semantic reasoning allows integrating the IoT data with available background knowledge, which facilitates the extraction of high-level knowledge to perform accurate and intelligent decision making.

Generic middleware platforms aid in various aspects of IoT integration and facilitate the development of IoT services. According to Perera, et al. [16], one of the important design principles for intelligent IoT systems is scalability and extensibility. Gartner expect 21 billion connected things to be in use by 2020[1]. Thus, integration of new sensors and devices should be straightforward. As new sensors get integrated and thus new types of data comes available, it should also be possible to add new intelligent processing services on the fly in a user-friendly manner to extract high-level knowledge.

[1] http://www.gartner.com/newsroom/id/3165317

As the number of supported devices and services increases, the platform should be able to scale accordingly to still offer adequate performance. However, currently none of the available semantic IoT platforms provides support for all these requirements [2].

In this paper, we present a three-layer generic platform that tackles the aforementioned hurdles. The first layer allows the integration of, discovery of and interaction with any kind of sensor or device in a transparent fashion. This layer hides the underlying APIs and protocols from the individuals sensors and devices from the application developers and maps the gathered heterogeneous IoT data on one uniform semantic model. The second layer facilitates intelligent processing and decision making by combining the captured sensor readings with static background knowledge, such as profile data, environmental info and house plans, through semantic reasoning. It also provides a user-friendly interface that allows application developers to easily define new processing services on the fly. The third layer allows the easy deployment of the platform on a Big Data scale.

The remainder of the paper is structured as follows. Section 2 describes the related work in the IoT field. Two illustrative use cases to motivate the need for our platform and that are used as examples throughout the paper are detailed in Section 3. Section 4 details the basic building blocks the generic platform is build upon. In Section 5, the details of the platform are described. Section 6 discusses the advantages of the platform and the further required enhancements. Finally, in Section 7 the prevalent conclusions are highlighted.

2 Related Work

Numerous semantic IoT frameworks exist. Platforms such as LinkSmart [13], XGSN [5], Sense2Web [8] and OpenIoT [18] focus on providing facilities for the integration of the devices and sensors. Gray et al. [11] propose a platform that focuses on the discovery and integration of data sources, both static as streaming data. Ali et al. [1] describe an IoT-enabled communication system that allows the annotation of sensory data through the use of XGSN and the continuous analysis of data streams through the use of a stream query processing module for the detections of events. OpenIoT and SOFIA2 [12] allow deployment on the cloud, with the focus on cloud-based storage of the IoT-data. Ryu et. al [17] describe a semantic IoT platform deployed in a smart office use case. The approach provides a solution to communicate with IoT devices and semantically process the IoT data. However, no support for expressive ontology reasoning, service collaboration or scalability tactics are provided. As can be seen, none of these platforms combine their approach with intelligent data processing of the IoT data through expressive ontology reasoning or the required scalability tactics needed to support such services [2]. Furthermore, user-friendliness was not the incentive of these platforms.

3 Use case scenarios

To illustrate the need for our platform two office lab use cases are detailed. In a smart office, the environment is equipped with a plethora of IoT devices that constantly sense the state of the environment around them, e.g., movement and sound sensors that detect

activity in meeting rooms. Moreover, mobile applications and GPS tracking on the smart phones of the employees provide further information, e.g., travel time to work, location of parking spot, time to find a parking spot, etc. All this provided data can then be combined with background knowledge, e.g., meeting room reservation agenda or layout of the building, in order to derive important insights and anomalies. These insights can then be communicated to the employees to act upon these detected events and ideally support the employees in their daily activities. For example, meeting room reservation could be canceled when no activity is detected or employees can get updates about available parking spaces or current travel times to work.

For both use cases the employees at the office are provide with RFID-tags or wristbands (wearables) that allow to identify and track them, such that the office environment can automatically be adapted to their preferences. Visitors are given a temporal tag at the reception on their arrival and are directed to the correct floor of the building where their meeting takes place.

In the first use case, the office of the employees adapts automatically to their preferences when their presence is detected. Thus, when the employees arrive at their office, the doors will automatically unlock, the temperature will be adjusted and the windows are opened/closed based on the employee's personal preferences.

In a second use case, the building should interact with the visitors as well. Additional logic is defined to guide the visitors to the correct meeting room through the use of lightning strips on the floor, when their presence is detected.

4 Platform Preliminaries

To satisfy the need for an easy to setup and configurable IoT platform that facilitates complex processing, we have built our generic platform upon three existing frameworks, i.e., DYAMAND, MASSIF and Tengu. Each of these frameworks is specialized in its own field, namely sensors & devices integration, intelligent semantic processing and scalable big data processing respectively.

4.1 DYAMAND

The DYnamic Adaptive Management of Networks and Devices (DYAMAND)[2] [15] enables easy integration, interaction and discovery of various sensors using various technologies. It acts as a middleware layer between application developers and controllable devices by hiding the different protocols and interfaces of these devices. DYAMAND uses Service Discovery Protocols (SDP) to discover and utilize devices operating on various protocols. Application Services model the specific functionality of a particular type of sensor. As DYAMAND is plugin-based, new SDPs and Application Services can be added at run-time to communicate with and model new types of devices and sensors. Other types of plugins can also be added to DYAMAND to extend the platform or to add application logic.

[2] https://dyamand.ilabt.iminds.be/

4.2 MASSIF Platform

The MASSIF (ModulAr, Service, SemantIc & Flexible) Platform [4] facilitates the annotation of raw sensor data to semantic data and allows the development & deployment of modular semantic reasoning services which collaborate in order to allow scalable & performant processing of the annotated data. Each one of the services fulfill a distinct reasoning task and operate on their own ontology model. The Semantic Communication Bus (SCB) facilitates collaboration between services. Services indicate in which types of data they are interested in based on high-level ontology concepts by registering filter rules, i.e., OWL axioms, on the SCB. The annotated data from the sensors and the conclusions of the various services are pushed back on the SCB and forwarded to those services that have indicated interest in the published data. For example, a service might be interested in all sensory data. However, in the published annotated data, only the subclasses of the concept `Sensor` are used, such as `TemperatureSensor` and `RFIDSensor`. By subscribing to a high-level concept, such as `Sensor`, the service can receive any kind of sensor data. The SCB can coordinate the data on a high-level through the use of semantic reasoning. The SCB operates on its own ontology model and the registered OWL Axioms from the services are added to the ontology. Each time data gets pushed on the SCB, it is temporary added to the internal ontology and a reasoning cycle starts to determine whether the published data matches one of the registered OWL axioms. For each match, the data is forwarded to the service that registered that axiom. After the reasoning cycle, the published data is removed again from the ontology. By processing one published event at a time and thus minimizing the processed A-Box, the performance of the SCB can be guaranteed and it also allows duplication and scalability of the SCB. Moreover, the MASSIF Platform also provides several algorithms to optimize the reasoning performance in the individual semantic services [3].

4.3 Tengu

Tengu[3] [21] is a Big Data framework that allows the easy and flexible deployment of multi-technology Big Data environments, including for example Hadoop, Spark or Kafka. It simplifies setting up Big Data platforms, and once configured, the platform can be deployed on various infrastructures. It takes reliability into account by logging messages and recovering failed instances.

5 Generic semantic IoT Platform

To provide an easy to use, end-to-end intelligent and scalable IoT platform, these three frameworks were seamlessly combined and adapted to upgrade the applicability within the IoT. This section describes how this collaboration was achieved. Figure 1 depicts the architecture of the generic platform and how the various components are linked together.

[3] http://tengu.intec.ugent.be/

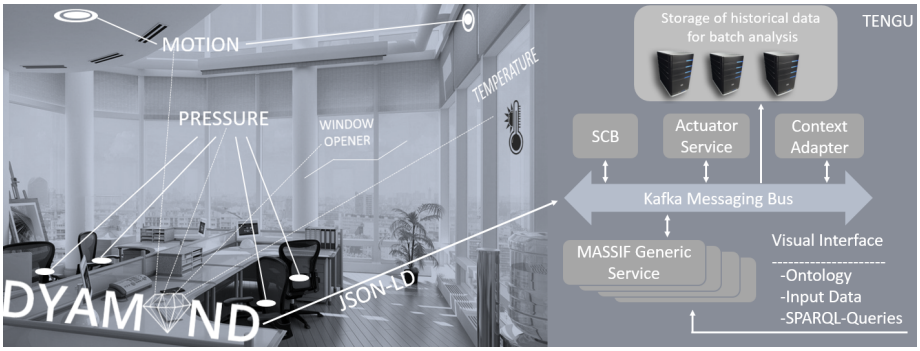

Figure 1. Architecture of the Generic Platform.

5.1 Converting the data captured by DYAMAND to semantic input for the MASSIF Platform

The DYAMAND platform uses an internal representation of the discovered sensors and devices, describing their capabilities and sensor readings. To integrate the DYAMAND framework seamlessly with the semantic MASSIF platform, we mapped this internal representation to the Semantic Sensor Network (SSN) [7] ontology. This enables a standardized representation of the sensor data and allows it to link it to available background knowledge about the sensor, e.g., types of data measured, accuracy and location information. An additional plugin was developed for DYAMAND that captures all its sensor discoveries and readings, maps this data to the SSN ontology and transmits data in the JSON-LD format to a Kafka Messaging Bus. By pushing this data on Kafka it can be dynamically picked up by the components of MASSIF or other applications that want to make use of this data. The MASSIF Platform was extended with a Context Adapter component that picks up the JSON-LD message from Kafka and converts them to OWL API Axioms, which are used internally in the MASSIF Platform and adhere to the ontology used by the SCB.

5.2 User-friendly design of new intelligent IoT Services

MASSIF has been adapted to allow the easy and user-friendly development of IoT-services. To limit the amount of code that needs to be written by application developers to deploy a new semantic service on the platform, a generic service was created. To design a new semantic service, only four things have to be defined, namely:

- a name and short description of the service,
- the used ontology by the service, which can be both A-Box and T-Box data,
- the OWL axiom, indicating the high-level input data the service is interested in, and
- the SPARQL-queries that define the application logic of the service

As illustrated in Figure 2, the creation of new services, by defining these four inputs, can be done through a visual interface. This visual interface also maintains and shows

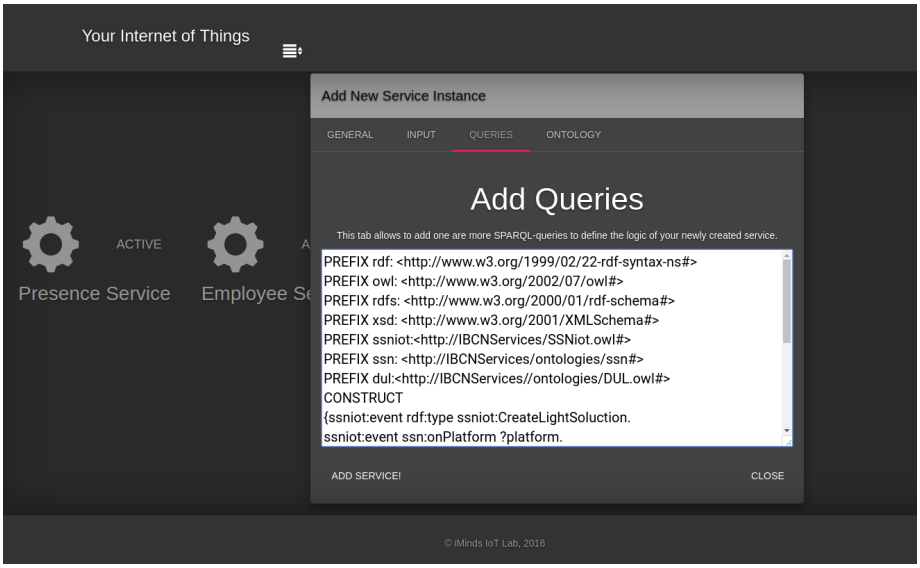

Figure 2. Example of the visual interface to create new generic service instances.

all the generic services that have been previously created such that they can be adapted if needed.

MASSIF internally uses OSGi[4], which provides an additional layer on top of the Java Virtual machine, to enable modularity. The use of OSGi allows to launch various services, even on distributed machines. Through the use of the Configuration Admin[5], new instances of the generic service can be launched, each with their own configuration. The configuration describes the defined properties, such as the input data, the used ontology and the SPARQL-queries. As such, the MASSIF Platform can be extended with new services on the fly, even at run-time.

The following paragraphs further detail how these inputted properties are used to configure a new instance of the generic instance within the MASSIF Platform.

The Ontology Since each service handles a distinct reasoning task, they can operate on their own ontology. This ontology should also import or extend the ontology used by the SCB in order to be able to adequately interpret the incoming data and output the conclusions in a uniform model. In this case, this means that the generic service ontology imports the SSN ontology. The ontology of the generic service instance can be defined by providing a local file where the ontology is described or an URL to a remote ontology location. Both A-Box and T-Box data can be defined in these files. The generic service instance will load the ontology definition and add it to its own ontology model.

[4] https://www.osgi.org/
[5] https://osgi.org/javadoc/r5/cmpn/index.html

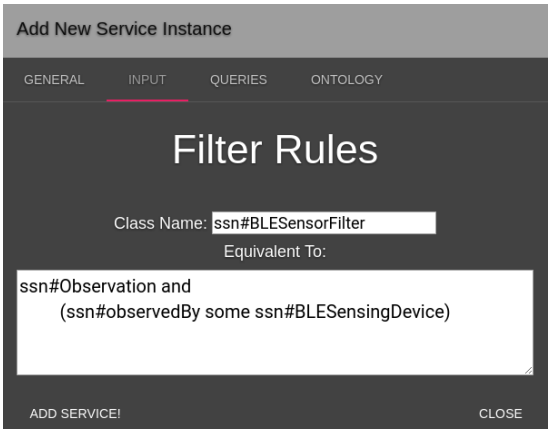

Figure 3. Example input definition in Manchester OWL Syntax defined in the visual interface.

The input data As mentioned before, a service can indicate in which types of data it is interested, by defining filter rules in the form of OWL Axioms. The visual interface allows to indicate one or more filter rules as text in various syntaxes. Currently, the supported syntaxes are those that can be parsed by the OWL API, such as JSON-LD, RDF XML, Turtle and Manchester OWL syntax. The generic service will parse the provided filter rules to OWL axioms and use them to subscribe the new service instance to the SCB. Later on, filter rules can be added or deactivated. For example, suppose we want to construct a service that detects the presence of people in particular areas to determine their location. This service requires as input data, amongst other things, all the data captured by a Bluetooth Low Energy (BLE) beacon. The beacon will capture the bluetooth tags of the nearby bleutooth devices and allows to detect the presence of the employees wearing a wristband for identification. Figure 3 shows the filter rule that filters this data for this presence service in Manchester OWL Syntax. When the service instances receives data based on the defined input, the data is added to the internal ontology of the service for further processing and decision making.

The Queries To define the application logic of the new generic service instance itself, one can define one or more SPARQL-queries in the visual interface. Each service instance maintains a list of queries. When the service is active, these queries are automatically executed every time new data is received through the SCB. By using SPARQL CONSTRUCT queries, the query result is a RDF graph that models the conclusions of the service. The results of the CONSTRUCT query are pushed back on the SCB where they can be picked up by other services. For example, the query in Listing 1.1 encodes part of the application logic of the presence service that was also used as example in the previous paragraph. Other services can subscribe to the constructed data that is pushed on the SCB if they want to be notified about the location updates of people.

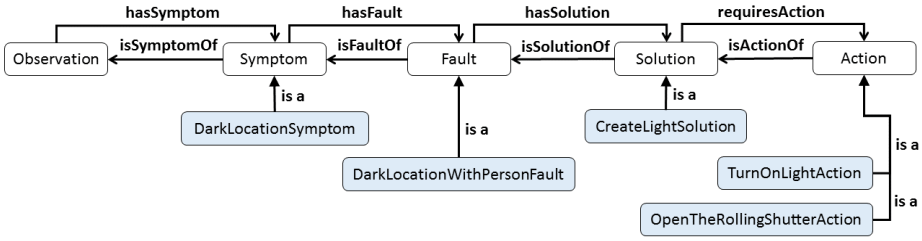

Figure 4. Observation pattern example

Listing 1.1. Example SPARQL query to identify the location of a person. For conciseness the IRIs 'http://IBCNServices.github.io/Accio-Ontology/' have been shortened to 'http://IBCNServices/'.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ssniot: <http://IBCNServices/SSNiot.owl#>
PREFIX ssn: <http://IBCNServices/ontologies/ssn#>
PREFIX dul:<http://IBCNServices/ontologies/DUL.owl#>
CONSTRUCT{?detectedPerson dul:hasLocation ?location}
WHERE {?observation rdf:type ssniot:BLEObservation.
?observation ssn:observationResult ?sensorOutput.
?sensorOutput ssn:hasValue ?sensorValue.
?sensorValue dul:hasDataValue ?bleTag.
?detectedPerson rdf:type dul:Person.
?detectedPerson ssniot:identifiedBy ?pesonIdent.
?pesonIdent dul:hasDataValue ?bleTag.
?observation ssn:observedBy ?sensingDevice.
?sensingDevice ssn:onPlatform ?platform.
?platform dul:hasLocation ?location.
}
```

5.3 Converting conclusions of the MASSIF Platform to DYAMAND actions

The IoT vision of course encompasses more then just capturing and processing sensor data. It also involves the interaction with various devices in an autonomous way in order to make the environment truly smart, e.g., turning on the lights to the preferred level when the presence of a particular person is detected.

First, the SSN ontology was extended with an observation pattern [22]. An example of this pattern is visualized in Figure 4. The `Observation` class of SSN models observations made by devices and sensors. We added four classes, namely `Symptom`, `Fault`, `Solution` and `Action`. A `Symptom` models specific phenomena that are detected in the `Observations`, e.g., when the light intensity in a certain location is below a particular threshold, a `DarkLocationSymptom` is detected. Queries or axioms

can then be defined that detect undesirable combinations of `Symptoms` and classify them as `Faults`, e.g., when the presence of a person is detected in a dark room, a `DarkLocationWithPersonFault` is detected. These detected `Faults` can then be coupled to `Solutions` that resolve them, e.g., `CreateLightSolution`. Finally, this `Solution` can be mapped on one or more `Actions` that need to be performed to reach this solution, e.g., `OpenTheRollingShutterAction` or `TurnOnLightAction`.

Next, a special service was created, i.e., the Actuator Service, that subscribes to all `Solutions` on the SCB. These Solutions can be outputted by other services as the result of a CONSTRUCT query. For example, Listing 1.2 visualizes the SPARQL query of a service that outputs a `Solution` to turn on the lights when a presence is detected in a location and the lights are off. The Actuator Service interprets the RDF data it receives from the SCB and converts it to actions that can be performed on the DYAMAND platform. The DYAMAND platform is then responsible for executing these actions in the physical environment. This way, interactions with the physical devices can be defined on a high level, without the need for any code implementation.

Listing 1.2. Example SPARQL-query describing a Solution to turn on the lights when a presence is detected and the room is dark. For conciseness the IRIs 'http://IBCNServices.github.io/Accio-Ontology/' have been shortened to 'http://IBCNServices/'.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ssniot:<http://IBCNServices/SSNiot.owl#>
PREFIX ssn: <http://IBCNServices/ontologies/ssn#>
PREFIX dul:<http://IBCNServices//ontologies/DUL.owl#>
CONSTRUCT
{ssniot:event rdf:type ssniot:CreateLightSoluction.
ssniot:event ssn:onPlatform ?platform.
ssniot:fault rdf:type DarkLocationWithPersonFault.
ssniot:event ssniot:isSolutionOf ssniot:faul.
ssniot:faul dul:hasLocation ?location.
}
WHERE { ?location rdf:type dul:Place.
?location ssniot:hasSymptom ?symptom.
?symptom rdf:type ssniot:DarkLocationSymptom.
?symptom rdf:type ssniot:PresenceDetectedSymptom.
?platform dul:hasLocation ?location.
}
```

5.4 Deployment using Tengu

Tengu uses juju charms[6] to deploy the various components in the cloud. To allow easy set up of our Big Data enabled IoT platform, various charms were written to easily

[6] https://jujucharms.com/

set up the MASSIF platform and a Kafka[7] message bus that takes the communication across various nodes into account. Tengu ensures that the various charms are deployed on the correct machine automatically. Through the use of Tengu, the end user only has to indicate which components, i.e., which services, he/she wants to deploy.

Through the use of a generic big data platform, all data passing over the Kafka message bus can be logged for offline batch processing. Traditional machine learning and data mining techniques can be utilized on the historical data. JSON-LD is used to facilitate this generic transition between semantic web data and traditional data.

6 Discussion

We presented a generic IoT platform that simplifies the deployment and definition of intelligent IoT applications. The use case scenarios described in Section 3 can easily be realized by using this platform. We assume that all the used sensors and devices are already known by the DYAMAND platform and mappings to SSN are thus already available in the created plug-in. The following services can then be designed using the visual interface:

- **Presence Service**: This service defines filter rules as input that capture all location, BLE and RFID observation that are made in the office environment. The loaded ontology contains, amongst other things, a model of the floor plan of the building, information about the sensors and their locations and a whether this device, e.g., the RFID or BLE tag, is mapped to a particular employee. When new data arrives in the service from the SCB, the service links it to the correct person, updates the location of this person and pushes this knowledge back on the SCB.
- **Employee Service**: This service captures all the temperature, light intensity and lock observations made in the offices and the location updates from the Presence Service that pertain to employees. The loaded ontology models the office environment and the employees' preferences. The defined logic in the SPARQL queries automatically adjusts the office environment to the detected employee by, e.g., unlocking the door or adjusting the temperature and light level to his/her preference.
- **Visitor Service**: This service takes as input the location updates from the Presence Service that pertain to visitors. The loaded ontology integrates available knowledge on the visitor and the calendar of the employees. The SPARQL queries check with whom and where this person has a meeting and outputs which light should be turned on in which color such that the visitor can easily find the meeting location. The DYAMAND platform is then responsible for executing the required actions in the office environment.

In our future work, we will incorporate an extension of the RML [9] framework to annotate the captured sensor stream in a standardized manner. Currently RML cannot annotate and map streams of data on the fly. We will also extend our visual interface. We want to foresee additional options when a service is created, such as which query execution policies should be employed (e.g., everytime an event arrives or after a predefined amount of time), whether reasoning should be turned on, which reasoner should

[7] http://kafka.apache.org/

be used and which reasoning optimizations should be facilitated (e.g., subsetting [3] or materialization). Furthermore, additional aids to simplify the definition of the input data and the queries will be researched. Currently, it might be hard to specify the input data and the queries, since their is no interaction with the loaded ontology, e.g., a drop-down list with the possible concepts and relations. We are also looking at incorporating existing research to provide the queries and filter rules as natural language sentences [10].

7 Conclusion

In this paper we presented an user-friendly, generic IoT platform that allows the discovery of devices and the capturing of their data, intelligent processing of this data and scalable deployment on Big Data infrastructures. To improve user-friendliness, a visual interface was designed that allows to develop intelligent IoT services, based on background knowledge captured in ontologies, by only indicating the input data and SPARQL-queries that define the IoT service.

8 Acknowledgment

This research was made possible by the DiSSeCt Strategic Fundamental Research (SBO), funded by VLAIO.

References

1. Ali, M.I., et al.: A Semantic Processing Framework for IoT-Enabled Communication Systems. In: The Semantic Web-ISWC 2015, pp. 241–258. Springer (2015)
2. Barnaghi, P., et al.: Semantics for the Internet of Things: Early Progress and Back to the Future. Int. J. Semant. Web Inf. Syst. 8, 1–21 (Jan 2012)
3. Bonte, P., et al.: Evaluation and optimized usage of owl 2 reasoners in an event-based ehealth context. In: 4th OWL Reasoner Evaluation (ORE) Workshop. Athens, Greece (June 6 2015)
4. Bonte, P., et al.: The MASSIF Platform: a Modular & Semantic Platform for the Development of Flexible IoT Services . Knowledge and Information Systems pp. 1–38 (2016), http://dx.doi.org/10.1007/s10115-016-0969-1
5. Calbimonte, J.P., et al.: XGSN: An Open-source Semantic Sensing. Middleware for the Web of Things. . Terra Cognita and Semantic Sensor Networks p. 51 (2014)
6. Compton, M., et al.: A survey of the semantic specification of sensors. In: 2nd International Workshop on Semantic Sensor Networks at the 8th International Semantic Web Conference (ISWC). Washington DC, USA (25-29 October 2009)
7. Compton, M., et al.: The ssn ontology of the w3c semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web 17, 25–32 (2012)
8. De, S., et al.: An Internet of Things Platform for Real-World and Digital Objects. Scalable Computing: Practice and Experience 13(1), 45–58 (2012)
9. Dimou, A., et al.: Rml: A generic language for integrated rdf mappings of heterogeneous data. In: LDOW (2014)
10. Dubey, M., et al.: Asknow: A framework for natural language query formalization in sparql. In: 13th Extended Semantic Web Conference (ESWC). Heraklion, Greece (29 May - 2 June 2016)

11. Gray, A.J., et al.: A Semantically Enabled Service Architecture for Mashups over Streaming and Stored Data. In: The Semanic Web: Research and Applications, pp. 300–314. Springer (2011)
12. Indra: IoT Interoperability Platform with a Big Data approach (Mar 2016), sofia2.com
13. Kostelnik, P., et al.: The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain. Scalable Computing: Practice and Experience 12(3), 307–316 (2011)
14. L., A., et al.: The Internet of Things: A survey. Computer Networks 54, 2787–2805 (2010)
15. Nelis, J., et al.: Dyamand: dynamic, adaptive management of networks and devices. In: Local Computer Networks (LCN), 2012 IEEE 37th Conference on. pp. 192–195. IEEE (2012)
16. Perera, C., et al.: Context aware computing for the internet of things: A survey. IEEE Communications Surveys & Tutorials 16(1), 414–454 (2014)
17. Ryu, M., et al.: Integrated semantics service platform for the internet of things: A case study of a smart office. Sensors 15(1), 2137–2160 (2015)
18. Soldatos, J., et al.: OpenIoT: Open Source Internet-of-Things in the Cloud. In: Interoperability and Open-Source Solutions for the Internet of Things, pp. 13–25. Springer (2015)
19. Song, Z., et al.: Semantic middleware for the internet of things. In: Internet of Things (IOT). pp. 1–8. Tokyo, Japan (29 November - 1 December 2010)
20. Strang, T., et al.: A context modeling survey. In: Workshop on Advanced Context Modelling, Reasoning and Management at 6th International Conference on Ubiquitous Computing (UbiComp). Nottingham, UK (7-10 September 2004)
21. Vanhove, T., et al.: Tengu: An experimentation platform for big data applications. In: 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops. pp. 42–47. IEEE (2015)
22. Verstichel, S., et al.: Distributed ontology-based monitoring on the ibbt wilab.t infrastructure. In: 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom). pp. 509–525. Berlin, Germany (18-20 May 2010)