Dynamic Ontology-Based Sensor Binding

Pascal Hirmer[1(✉)], Matthias Wieland[1], Uwe Breitenbücher[2], and Bernhard Mitschang[1]

[1] Institute of Parallel and Distributed Systems, University of Stuttgart, Universitätsstr. 38, Stuttgart, Germany
{pascal.hirmer,matthias.wieland, bernhard.mitschang}@informatik.uni-stuttgart.de

[2] Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstr. 38, Stuttgart, Germany
uwe.breitenbucher@informatik.uni-stuttgart.de

Abstract. In recent years, the Internet of Things gains more and more attention through cheap hardware devices and, consequently, an increased interconnection of them. These devices equipped with sensors and actuators form the foundation for so called smart environments that enable monitoring as well as self-organization. However, an efficient sensor registration, binding, and sensor data provisioning is still a major issue for the Internet of Things. Usually, these steps can take up to days or even weeks due to a manual configuration and binding by sensor experts that furthermore have to communicate with domain-experts that define the requirements, e.g. the types of sensors, for the smart environments. In previous work, we introduced a first vision of a method for automated sensor registration, binding, and sensor data provisioning. In this paper, we further detai l and extend this vision, e.g., by introducing optimization steps to enhance efficiency as well as effectiveness. Furthermore, the approach is evaluated through a prototypical implementation.

Keywords: Internet of Things · Sensors · Ontologies · Data provisioning

1 Introduction and Motivation

Today, the paradigm called *Internet of Things (IoT)* gains more and more importance in many different domains [16]. The IoT is generally based on the integration of sensors and actuators to allow monitoring and self-organization of what is called *smart environments*. For example, by an aggregation of raw sensor data, high level information – so called situations – can be derived, which enables automated adaptation of smart environments to occurring events. This enables new approaches such as advanced manufacturing – oftentimes referred to as Industry 4.0 [8] – smart homes or smart cities [16]. For example, automated exception recognition in a production environment as described in [9] can lead to severely reduced costs due to a faster repair and, as a consequence, a faster resumption of the production process.

J. Pokorný et al. (Eds.): ADBIS 2016, LNCS 9809, pp. 323–337, 2016.
DOI: 10.1007/978-3-319-44039-2_22

However, though there are many IoT applications, the registration and binding of sensors and actuators is still a great challenge. A manual binding is a complex and tedious task that requires technical knowledge about the sensors, actuators, and the environment. To realize a manual sensor binding, adapters have to be manually created and deployed for each sensor to extract its data and to provision it to sensor-driven applications. Furthermore, these steps are error-prone and can take hours or even days to be processed manually: a sensor expert has to configure the sensors, install a sensor gateway, bind the sensors, implement the sensor data provisioning, and establish interfaces to applications that intend to consume the sensor data. By doing so, he constantly has to communicate with domain-experts that define the requirements for the smart environment. Furthermore, nowadays environments are very dynamic, i.e., the contained sensors and actuators may change constantly, e.g., when a smart phone is carried into a smart home environment. To cope with these issues, we need a means for efficient, on-demand binding of sensors and actuators. In real-world scenarios, efficiency and accuracy are of vital importance. The drawbacks that come with a manual registration can lead to high costs due to occurring errors, a tedious, time-consuming registration process and, furthermore, omits building dynamic smart environments. In previous work [5], we worked on a first approach by introducing a vision of a method for on-demand automated sensor registration, binding, and sensor data provisioning. The goal of the method is to reduce the manual steps to the modeling of sensors and things using ontologies. All other steps (sensor binding, sensor data provisioning) can be processed automatically in milliseconds instead of hours or even days when conducting them manually. By doing so, we can reduce occurring errors that are more likely with manual processing and, as a consequence, save costs.

In this paper, we further enhance this method by introducing new optimization steps that can further improve the efficiency. Furthermore, we elaborate the details of this method, which was only described as a vision in our previous work, and we introduce a system architecture to realize the method. Finally, we provide a prototypical implementation that is the basis of a first detailed evaluation of our approach. This implementation is currently in productive use within the open source IoT project SitOPT[1]. In the context of this paper, *things* are physical devices containing an arbitrary amount of sensors. As a consequence, sensors cannot be things themselves.

Motivating Scenario: We present the motivating scenario as depicted in Fig. 1 to explain our approach: In a typical production environment, the machines on the shop floor are monitored in an ad-hoc manner by a sensor-driven application that consumes raw sensor data and derives high-level situations. These situations describe changing states of the machines. The following situations can occur: (i) *Running* indicates that the machine is running without any errors, (ii) *Critical* indicates an emerging error that could lead to the machine's failure, e.g., if a sensor measures an increasing temperature, and (iii) *Failed* indicates that the

[1] https://github.com/mormulms/SitOPT.

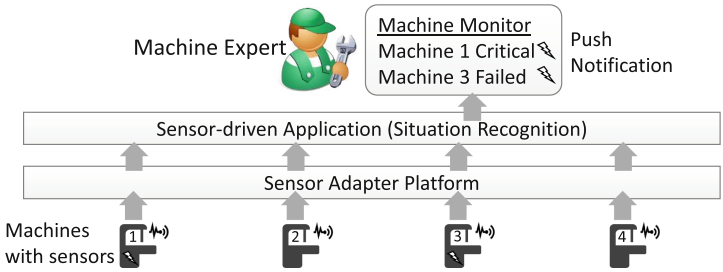

Fig. 1. Motivating scenario: monitoring machines on the shop floor

machine has failed due to an occurred error detected by one or more sensors. To enable such a situation recognition, all available sensors of a machine have to be monitored. To realize this, the sensors somehow have to be connected to the situation recognition system. This requires: (i) creating and deploying adapters to connect the recognition system to each individual physical sensor, and (ii) provisioning of the sensor data to the situation recognition system. Important aspects such as efficiency and sensor availability are of vital importance to enable a reliable recognition of occurring situations. However, even if all required adapters are available, e.g. by using integration technologies such as FIWARE[2] or OneM2M[3], connecting each physical sensor manually to the respective applications, in this case the situation recognition system, is – as described in Sect. 1 –, a tedious, time-consuming and error-prone task: different types of sensors have to be managed, adapters need to be selected, and physical endpoints of sensors must be configured in the respective application. Thus, to increase the efficiency of building sensor-driven applications, we need an automated means to dynamically bind applications to the required sensors by using software-defined specifications. The approach presented in this paper copes with these issues by enabling an automated sensor registration and a dynamic, automated sensor binding and provisioning based on an ontology model to enable scenarios such as situation recognition in smart environments.

The remainder of this paper is structured as follows: In Sects. 2 and 3, the main contribution of this paper is presented. In Sect. 4, we describe related work. After that, in Sect. 5, we evaluate the approach through a prototypical implementation. Finally, in Sect. 6, we give a summary of the paper and an outlook on future work.

2 Dynamic Ontology-Based Sensor Binding

This section and the following Sect. 3 present the main contribution of this paper by introducing a system architecture and a method for dynamic sensor binding

[2] https://www.fiware.org/.
[3] http://www.onem2m.org/.

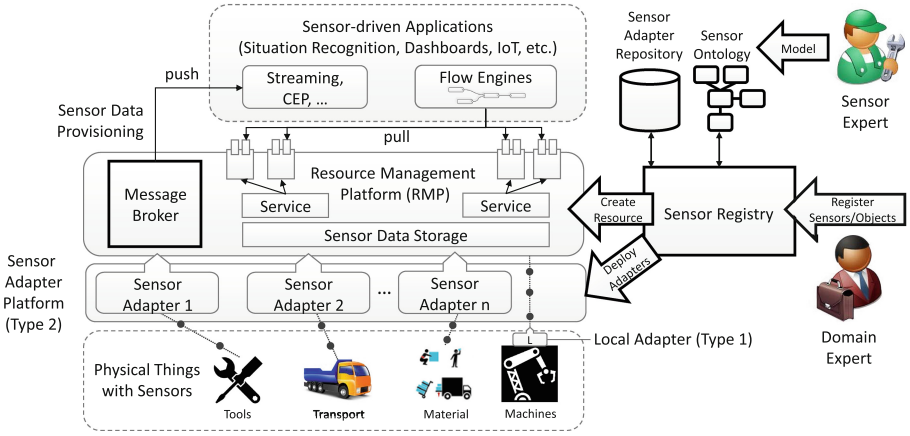

Fig. 2. Architecture for on-demand sensor binding and sensor data provisioning

and sensor data provisioning. In the context of this paper, sensor data provisioning means to enable sensor-driven applications retrieving the required sensor data, e.g., via REST interfaces or MQTT. Figure 2 depicts the overall architecture of our approach. The components and interaction steps marked in bold are newly added to the architecture introduced in previous work [6]. The architecture consists of the following main components: (i) the sensor registry, which stores meta-information about the physical things and sensors, (ii) the sensor ontology, containing sensor binding information, (iii) the sensor adapters – stored in the *sensor adapter repository* – that extract the data from the sensors and can be deployed directly on a thing or on an adapter platform, and (iv) the *Resource Management Platform* that provisions the sensor data as remotely accessible resources (pull) or via a publish-subscribe approach (push) using a *message broker*. The support of a pull and a push-based provisioning of sensor data is necessary due to different needs of sensor-driven applications. Some applications, e.g. streaming systems, require the data as soon as they occur because they are working directly on the sensor data stream. Other applications, e.g., flow-based applications, require the data *on-demand*, i.e. independent of the sensor's reaction, e.g., when a certain step in the flow is reached. This requires a means to store sensor data in the *sensor data storage* and provide them when needed. The components of our architecture are further described in the following. Security and privacy features are out of scope of this paper, however, they are part of our approach and system architecture.

2.1 Sensor Registry

The sensor registry component provides a means to register sensors to the *Resource Management Platform* (RMP), which enables binding the sensors, receiving their data and providing them through a pull approach (e.g., by REST

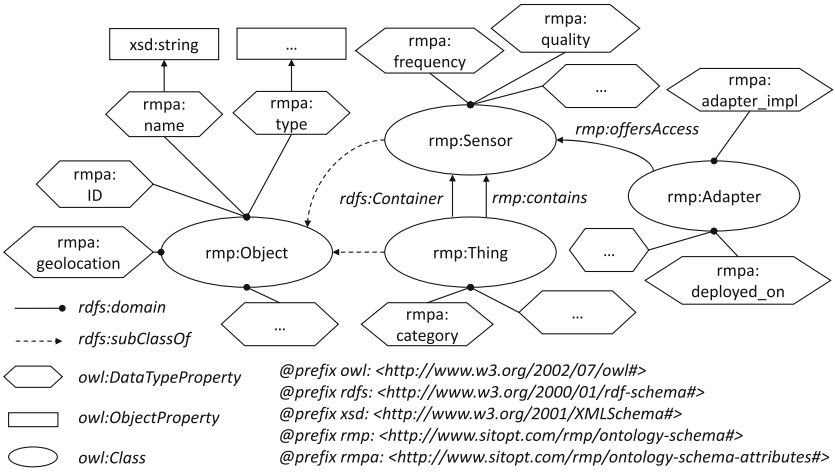

Fig. 3. Partial ontology of our approach based on SensorML

resources) or a push approach (e.g., by MQTT). To register a thing manually or automatically to the RMP, only a unique ID of a thing containing one or more sensors, e.g., in the motivating scenario a production machine, has to be provided. In this case, all sensors of the thing will be registered. In case only specific sensors of a thing should be registered, unique sensor IDs have to be provided as well. Providing such an easy-to-use registration entry point enables usage by domain users without any extensive knowledge of sensor technology. Although this is a simple registration step, if performed manually for hundreds of individual sensors, this becomes a time-consuming, error-prone task and is, therefore, not appropriate. Because of that, an automated registration is recommended and supported by our approach. The detailed sensor and thing binding information is stored in an ontology, which is described in the next section.

2.2 Sensor Ontology

The sensor ontology used in our approach to model things and sensors (cf. Fig. 3) – modeled by a sensor expert – is based on the Sensor Model Language[4] (SensorML), an XML-based model that enables defining things, sensors, their properties as well as their relations. In our adapted ontology, the following elements are contained: (i) the super type *Object* that is either inherited to the type (ii) *Sensor* or (iii) *Thing*, and (iv) *Adapters* that are attached to the sensors. *Objects* are defined as all things that are involved in sensor-driven applications and the sensors observing them. For example, a real world object like a machine with built-in sensors. However, there are also objects in the world that are not observable by sensors. These are not covered in this paper. Sensors and things have several specific attributes such as their quality, category, etc.

[4] http://www.opengeospatial.org/standards/sensorml.

Some attributes, e.g., their ID or geolocation, are defined in the *Object* they are derived from. Adapters provide a reference to a sensor-specific adapter implementation in the *sensor adapter repository*. This information is of vital importance to enable sensor binding and sensor data provisioning. In our approach, we decided to use ontologies to model and manage this information instead of SensorML, because SensorML is a complex and detailed language containing a large amount of elements and properties that are not needed in our lightweight approach. For our approach, we therefore only pick the core concepts of SensorML. However, we exploit the structure of SensorML to define our ontology using the Resource Description Framework (RDF) and the Web Ontology Language (OWL). This approach is similar to the one presented in [13]. Our ontology is depicted in an abstracted manner in Fig. 3. As default, we are using our lightweight ontology in this approach, because SensorML-based XML documents are cumbersome to process.

2.3 Resource Management Platform

The *Resource Management Platform* (RMP) combines two paradigms for provisioning sensor data: (i) a pull approach by providing sensor data as uniform REST resources, and (ii) a (e.g., queue-based) publish-subscribe (push) approach for enabling direct notification whenever a sensor value occurs. Which of the approaches is used depends on the sensor-driven application. The pull approach guarantees that a sensor value is present when needed, whereas the push approach provisions sensor data as soon as they occur but cannot deliver the latest sensor values *on-demand*. This enables usage by all kinds of sensor-driven applications, e.g., the one presented in the motivating scenario. Most importantly, it works without any additional software besides approved Internet technologies that are nowadays available in nearly all devices.

In the pull approach, REST-based resources can be accessed by sensor-driven applications using a (e.g., HTTP) GET request. To be able to provide sensor data on demand, which is necessary to support this pull-based approach, a persistent *sensor data storage* has to be provided, which is able to store the data to be available when it is needed. Additionally to the sensor data, a timestamp has to be provided describing when the data was produced because the quality of sensor data typically decreases with time passing. The sensor data is provided using REST resources accessible through the following URL schema: $< protocol >$: $// < RMP\_URL > / < thing\_id > / < sensor\_id >$ for a specific sensor value and $< protocol >$: $// < RMP\_URL > / < thing\_id > /$ for a list of all sensor values of a thing. The quality of a sensor value is at least dependent on its accuracy, staleness, as well as on the maturity of the value. In addition to this pull-based approach, we further enable a push-based approach to provision the sensor data. By using approved publish-subscribe queuing technologies such as MQTT, we are able to allow *queue registration* on certain sensors so the sensor-driven applications can be automatically notified once sensor data occur and are able to process them immediately. The information that is sent to the sensor-driven application is the same as in the pull approach.

2.4 Sensor Adapter Platform

Sensor adapters provide access to the sensors. That is, they connect to the sensors' physical interfaces (e.g., serial interfaces) and extract the values that are produced. For example, these sensor adapters could be lightweight scripts deployed directly on the things or on external platforms to retrieve the sensor values from a serial interface, or more sophisticated platforms (FIWARE, OneM2M, OpenMTC, etc.) using approved Machine-to-Machine standards such as ETSI[5]. With respect to our approach, the sensor values are passed to the Resource Management Platform including a timestamp, the sensor ID, the type of the sensor, the corresponding thing, and the *quality* [12] of the sensor value. There are two types of quality regarding sensors: (i) the sensor quality, which is specific to a certain sensor type and influences the quality of all values produced (e.g., the average deviation), and (ii) the quality of a sensor value (e.g., its specific staleness). The sensor quality information is stored in the ontology and does not have to be provided by the adapters. However, the adapter has to compute a single quality measure for each value that is passed. This requires knowledge about the definition of quality in the context of the sensor, but enables a better quality-aware usage and further processing of sensor values.

In general, there are two types of adapters as depicted in Fig. 2:

(i) Local Adapters (Type 1, Fig. 2**):** Local adapters are running on the same thing that contains the sensors. Usually, some kind of runtime environment or operating system is provided to deploy the sensor adapters onto the thing. This makes it easy to receive and pass sensor values to the RMP, preconditioned that the thing is connected to a network. The passing of the values can be conducted using approved protocols such as HTTP or MQTT.

(ii) Remote Adapters (Type 2, Fig. 2**):** Remote adapters are the regular case. If the corresponding thing does not offer any means to deploy an adapter or if a single sensor does not offer a means for direct access and is deployed without a corresponding thing, e.g., a temperature sensor attached to a wall, remote sensor adapter platforms are used for binding. Remote sensor adapters can, e.g., be deployed on micro controllers and are able to connect to the sensors, receive their values and pass them to the RMP. We recommend using approved M2M platforms such as FIWARE, OneM2M or OpenMTC supporting a wide range of sensor types and M2M communication standards to deploy remote adapters.

3 Method for Dynamic Ontology-Based Sensor Binding

The architecture described in the previous section is applied through the method depicted in Fig. 4. It covers the whole sensor lifecycle, from the registration to its deactivation. Based on our previous work that presented the vision for this method, we add several optimizations to the method steps and provide more

[5] http://www.etsi.org/technologies-clusters/technologies/m2m.

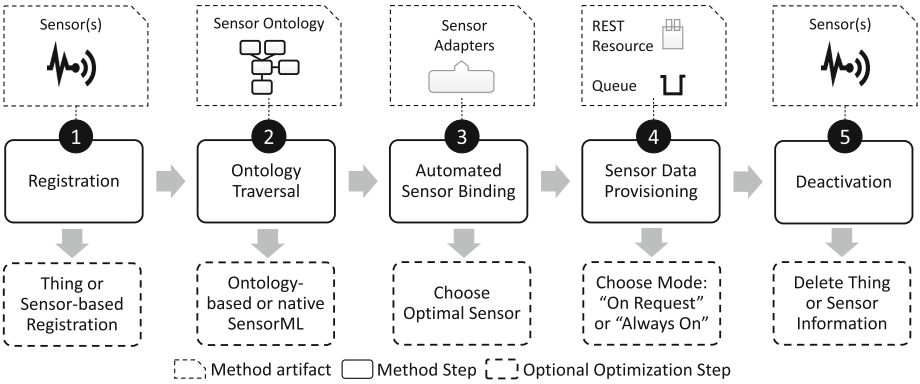

Fig. 4. Optimized method for dynamic sensor binding

details. The purpose of these optimizations is providing concepts for improving the method throughout the whole sensor lifecycle. The optimizations are not yet fully detailed, this, and further optimizations, will be part of our future work. We show that a full automation of this method is possible, which is necessary to achieve our goal to minimize human interaction during this process.

Step 1: Registration of Sensors
In the first step of the method, sensors are registered to the Resource Management Platform. By doing so, an unique identifier of the thing to be registered and, if specific associated sensors should be registered, also unique identifiers of the sensors have to be specified. Detailed information of sensors and things are contained in the sensor ontology (cf. Step 2). In case a thing or a sensor is not known, i.e., is not represented in the ontology, an ontology snippet describing their properties has to be added to the registration, which will be processed in the next Step 2. In the following, we assume that the ontology is modeled correctly and contains all sensors and things of the specific domain our approach is applied to, in the motivating scenario e.g. the shop floor.

Optimization: Registration of Things
The registration can contain either a "whole" thing (e.g., a production machine) or specific sensors of a thing. Registering a whole thing makes sense if all sensors of this thing should be registered and, as a consequence, are relevant for further processing. If only some of the sensors are relevant for sensor data provisioning, it makes sense to register them individually. This can save costs due to a more energy efficient solution.

Step 2: Ontology Traversal
Based on the information provided by the registration of Step 1, additional, specific information about things and sensors are retrieved from the ontology in Step 2. The ontology describes technical sensor information that are necessary for

an automated registration, binding, and sensor data provisioning. Furthermore, it can also be used as meta-data source by sensor-driven applications. These information include sensor specifications (accuracy, frequency, ...), information about sensor access, i.e., about sensor binding in terms of the corresponding adapter in the sensor adapter repository, and about the contained sensors of a thing. The sensor ontology is partly depicted in Fig. 3. On sensor registration, we traverse the ontology and search for the corresponding entry of the sensor or thing. Once the relevant sensor information is found, it can be used for automated sensor binding, which is described next.

Optimization: Use Native SensorML or Ontology
Due to the fact that these concepts are of vital importance in our approach, we decided to use ontologies as default option. However, in case of small, clear scenarios, e.g., describing a closed, non-extendable environment, an XML-based representation, such as native SensorML, is also supported by our approach.

Step 3: Automated Sensor Binding / Sensor Adapter Deployment
The next step is the automated sensor binding and, furthermore, the provisioning of the sensor data, which is based on the sensor information that was extracted from the ontology in Step 2. To enable sensor binding, we need a means to extract the sensor data from the corresponding sensors. This requires adapters, as described in Sect. 2.4, which are connecting to the sensors' serial interfaces, extract the data, and send it to the Resource Management Platform (e.g., using HTTP or MQTT). The great advantage of our approach is that the adapters do not have to care about sensor data provisioning to sensor-driven applications, because they send the data directly to the centralized RMP that manages the provisioning for them. Note that the data being produced by the sensors is non-stopping, i.e., the adapter has to be up and running. Techniques for guaranteeing such a high availability shows high complexity and is out of scope in this paper.

As described before, the sensor adapters are deployed automatically. First, the adapters are retrieved from the *sensor adapter repository*, then they are parameterized (e.g., with the RMP's URL). The information which adapter is needed to bind the sensor(s) defined by the registration was extracted from the ontology in Step 2. There are several possibilities how an adapter deployment can be realized: if the sensor is connected to a thing that is containing a powerful runtime environment such as, e.g., a Raspberry Pi, the adapter can be deployed directly using, e.g., SSH connections or more sophisticated approaches such as TOSCA [2]. However, in most cases this is not possible. Because of that, the adapters have to be deployed on external platforms, either self-implemented or using approved middleware, such as FIWARE, OneM2M or OpenMTC, that are capable to connect to the sensors, even if, e.g., they are embedded into a production machine, using Machine-to-Machine standards. The information how to bind a sensor is stored entirely in the ontology and has been extracted in Step 2.

Optimization: Choose Optimal Sensor
In many cases, things contain more than one sensor of a certain type and it makes sense to choose the most suitable one. The dynamic sensor binding of our approach enables such an optimization by enabling binding of sensors that are most suitable for a specific scenario, e.g., in regard to energy efficiency or accuracy. Note that this step is highly dependent on the use case scenario and, furthermore, on its non-functional requirements.

Step 4: Sensor Data Provisioning
After the sensor adapters are deployed and activated, they start sending data to the RMP. However, the data can only be accessed by the sensor-driven applications after the fourth step is processed, the sensor data provisioning. In this step, the interfaces to the sensor-driven applications are established. The sensor data provisioning step represents the integration of all components, from the sensor adapters to the sensor data provisioning through the RMP. After the automated adapter provisioning (Step 3), the RMP is informed that the registered sensors have started sending their data. By doing so, entries in the sensor data storage as well as corresponding REST resources are created for each sensor to provision its data to enable the pull approach. Furthermore, we create topics in a queue for each sensor and publish these topics to the sensor-driven applications that can subscribe to them to enable the push approach. After this step, the sensor data are available to sensor-driven applications.

Optimization: Choose Sensor Mode "On Request" or "Always On"
Sensors can be operated in two different modes. In the *on request* mode, the sensor is inactive only requiring minimal energy consumption. Sensor values are requested *on-demand* by the sensor adapters, which leads to the sensors to change to an active state, send the value and then return to an inactive state again. The main advantage is a reduced energy consumption, which makes sense when using battery-powered sensors, however, receiving sensor values will be less efficient. The second mode *always on* is the regular case, e.g., if sensors are built into things such as production machines as described in the motivating scenario. In this case the sensor is always in an active state. Of course, this behavior costs more energy, however, receiving sensor data can be realized more efficiently.

Step 5: Deactivation
The last step is the deactivation of sensors and/or things once they are not needed anymore. To do so, the thing and the type of the sensor have to be provided to the sensor registry. Based on this information, the sensor registry finds running sensors of a thing with the corresponding type, connects to the adapters to terminate it, clears the values from the sensor data storage, and removes the REST resources and the topics in the queue. Deactivation of sensors saves energy and costs.

Optimization: Delete Thing or Sensor Information (Partially)
The deactivation of sensors and things can be conducted in two manners: (i) completely deleting the stored information, or (ii) partially deleting it. When choosing the complete deletion, the entry in the registry is removed, the sensor

adapter is undeployed, and the corresponding part in the ontology is deleted. By doing so, the space and costs needed for executing the sensor adapter and storing these information can be reduced. When selecting the partial deletion, the user can select which parts should be deleted. For example, if the sensor will be re-registered in the near future, it makes sense to keep the adapter deployed.

4 Related Work

The related work can be separated into the following areas: (i) automated sensor binding, (ii) middleware to access sensor data, and (iii) ontologies for sensor modeling.

Automated Sensor Binding: Hauswirth et al. [4] present a similar approach by the introduction of the Global Sensor Network (GSN) to bind stream-based data sources such as sensors without any programming effort. By doing so, sensors are abstracted by a virtual representation to allow processing of the data using SQL-like queries. In contrast, our approach separates these steps strictly. After the ontology-based dynamic sensor binding is finished, the data is provisioned to sensor-driven applications. The standard IEEE1451.2 defines so called Transducer Electronic Data Sheets (TEDS) [10] to enable self-description of sensors. In addition, dynamic plug and play binding of sensors to networks is enabled through a standardized interface. In contrast to this standard, we do not focus on the physical binding of sensors. Our goal is an easy provisioning of sensor data to sensor-driven applications. However, in our approach, standards such as the IEEE1451.2 could be used for physical sensor binding. Li et al. and Vögler et al. [11,17] introduce an approach for IoT application deployment using TOSCA. However, they do not cope with the direct binding of sensors and actuators, i.e., they assume that the binding is done through specific sensor gateways. In contrast, we propose a generic approach that does not depend on sensor gateways. Furthermore, although the authors claim that no pre-configuration is necessary, the papers show that a configuration of the sensor gateways is needed in order for the approach to work. In our approach, no pre-configuration of devices is necessary at all.

Middleware to Access Sensor Data: Similar to our approach, Ishaq et al. [7] introduce an approach for sensor access through a REST interface. To realize this, Ishaq et al. assume a sensor network bound to a gateway, which allows accessing the sensors. In our approach, we do not necessarily assume such a gateway because we manage the sensor binding ourselves. However, the REST-based provisioning of sensor data is similar to our approach. Machine-to-Machine (M2M) gateways such as FIWARE, OneM2M, OpenMTC[6], OpenIoT [15], or GSN [1] have gained a lot of attention recently. These gateways serve as layer between physical sensors and "virtual" sensor data. Our approach in this paper does not try to compete with these approved platforms but rather uses them, i.e., we provide a more abstracted layer on top. This layer enables

[6] www.open-mtc.org/.

binding *things* and not specific sensors. Furthermore, it enables sensor data provisioning to sensor-driven applications exclusively using Internet technologies. Note that these approved gateways can be used to realize the sensor binding in our approach (cf. Sect. 2.4). The service-oriented middleware SStreaMWare [3] enables managing heterogeneous sensor data. SStreaMWare can both handle data streams and distributed sensor networks. To access stream-based data, SStreaMWare provides a schema for sensor data representation, which enables execution of queries based on the data streams. The management of sensors is based on the things observed, which is similar in our approach.

Ontologies for Sensor Modeling: The knowledge repository OntoSensor [13] enables modeling and management of sensors. It combines SensorML, IEEE SUMO, ISO 19115, OWL and GML. By combining these approved description languages, a wide range of sensors can be modeled. However, OntoSensor descriptions can become heavy-weight and complex. In contrast, our goal is designing a lightweight ontology for sensor modeling and management. To realize this, we use a subset of SensorML in contrast to the heavy-weight OntoSensor model. The ontology DCON [14] enables modeling of activity context. To enable this, different OSCAF ontologies[7] are combined to create a so called Personal Information Model: DDO (for Devices), DPO (for Presence), and DCON [14] for representation of user activity context. In contrast to the specialized DCON ontology, the goal of our approach is to be more generic, i.e., to support many different domains. Furthermore, we do not focus on the user, i.e., the things are the main focus. In general, persons should not be monitored for privacy reasons. In summary, this related work focuses on specific aspects like the access of sensors using gateways, or the execution of queries on sensor data streams or in a sensor network. The goal of our approach is to provide an easy-to-use ontology for the Internet of Things that combines sensor registration, binding of the sensors, and sensor data provisioning. Whereat the binding of a concrete sensor is done indirectly based on the things that are monitored by the sensors. Furthermore, our approach allows a *separation of concerns*, since the sensor data processing is specified separately, e.g., in the situation recognition as described in [6], based on situation templates that can be mapped onto different execution systems. Additionally, our approach allows the integration of heterogeneous sensor types in a standard way as REST resources or through a *publish-subscribe* model so that they can be accessed by multiple clients and in parallel.

5 Prototypical Evaluation

The presented system architecture has been implemented as a prototype and is currently applied to the project SitOPT (cf. acknowledgments). In SitOPT, the prototype has been integrated to provide a situation recognition system with sensor data. Based on this data, situations are derived that lead to adaptations of workflows. The following technologies have been used in this prototype (Fig. 5):

[7] http://www.semanticdesktop.org/ontologies/.

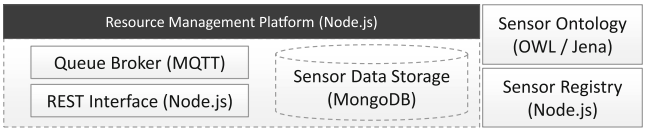

Fig. 5. Technologies used for the prototypical implementation

The sensor registry was implemented using NodeJS, offering an user interface as well as a programmatic interface, accessible through HTTP requests. The sensor ontology is accessed using SPARQL requests, furthermore, we use SSH to deploy sensor adapters. As the adapter repository, we use the native file system. The ontology is in the Web Ontology Language (OWL) 1.1[8] format, which is accessed through SPARQL requests. To support this, we used the Apache Jena[9] framework. Furthermore, to enable an easier access that does not require SPARQL, we also implemented a REST-based interface that abstracts from it. Similar to the Sensor Registry, the Resource Management Platform is implemented in NodeJS[10], which offers high efficiency. The limitations of such a lightweight solution regarding robustness are of minor importance in our approach, because in most IoT use case scenarios, e.g., lost sensor values are not critical to the sensor-driven applications and efficiency is much more important. As sensor data storage, we use the schemaless NoSQL database mongodb[11], which allows high efficiency, scalability, and data replication. The direct push approach was realized using MQTT[12] and the Mosquitto[13] MQTT broker. As also mentioned in [5], we conducted runtime measurements of our prototype for evaluation purposes using a machine with a Core i5-3750K @3.4 GHz and 8 GB RAM. We measured the average runtime of the method's steps based on 10 measurements: (i) the sensor registration took 1,91 ms, (ii) the ontology traversal 6,73 ms, and (iii) the adapter deployment 139,63 ms. The measurements show that we could achieve the efficiency goals of this paper (cf. Sect. 1). In the future, we will conduct several load tests to evaluate how the implementation can cope with a large amount of sensors.

6 Summary and Future Work

This paper presents an approach for optimized ontology-based sensor registration and sensor data provisioning. In this paper, we provide details and optimizations for the introduced system architecture and method. By doing so, we created an easy-to use solution for sensor-driven applications to bind sensors and access

[8] www.w3.org/2004/OWL/.
[9] https://jena.apache.org/.
[10] http://nodejs.org/.
[11] http://www.mongodb.org/.
[12] http://mqtt.org/.
[13] http://mosquitto.org/.

their data within milliseconds in contrast to a manual processing of these steps that can take up to hours or even days. This goal was achieved as described in our evaluation. Furthermore, we offer a flexible means to provision sensor data to sensor-driven applications. By providing two means for provisioning, a pull and a push based approach, we enable usage by a wide range of applications, both stream-based or static.

In the future, we will extend our prototypical implementation with data level security, privacy and robustness features and, furthermore, we will work on performance and scalability issues. In addition, we will concentrate on interfacing sensor-driven applications.

Acknowledgment. This work is partially funded by the DFG project SitOPT (610872) and by the BMWi project SmartOrchestra (01MD16001F).

References

1. Aberer, K., Hauswirth, M., Salehi, A.: Zero-programming sensor network deployment. In: Proceedings of the Service Platforms for Future Mobile Systems (SAINT 2007) (2007)
2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: Bouguettaya, A., Sheng, Q.Z., Daniel, F. (eds.) Advanced Web Services, pp. 527–549. Springer, New York (2014)
3. Gurgen, L., Roncancio, C., Labbé, C., Bottaro, A., Olive, V.: SStreaMWare: a service oriented middleware for heterogeneous sensor data management. In: International Conference on Pervasive Services (2008)
4. Hauswirth, M., Aberer, K.: Middleware support for the "Internet of Things". In: 5th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (2006)
5. Hirmer, P., Wieland, M., Breitenbücher, U., Mitschang, B.: Automated sensor registration, binding and sensor data provisioning. In: Proceedings of the CAiSE 2016 Forum at the 28th International Conference on Advanced Information Systems Engineering (2016). Accepted for publication
6. Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Leymann, F.: SitRS - a situation recognition service based on modeling and executing situation templates. IBM Research Report (2015)
7. Ishaq, I., Hoebeke, J., Rossey, J., De Poorter, E., Moerman, I., Demeester, P.: Facilitating sensor deployment, discovery and resource access using embedded web services. In: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 717–724, July 2012
8. Jazdi, N.: Cyber physical systems in the context of Industry 4.0. In: 2014 IEEE International Conference on Automation, Quality and Testing, Robotics (2014)
9. Kassner, L.B., Mitschang, B.: MaXCept-Decision Support in exception handling through unstructured data integration in the production context. an integral part of the smart factory. In: Proceedings of the 48th Hawaii International Conference on System Sciences (2015)
10. Lee, K.: IEEE 1451: a standard in support of smart transducer networking. In: Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference, IMTC 2000 (2000)

11. Li, F., Vögler, M., ClaeSSens, M., Dustdar, S.: Towards automated IoT application deployment by a cloud-based approach. In: 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, pp. 61–68, December 2013
12. Reiter, M., et al.: Quality of data driven simulation workflows. In: 2012 8th IEEE International Conference on e-Science (2012)
13. Russomanno, D.J., Kothari, C.R., Thomas, O.A.: Building a sensor ontology: a practical approach leveraging ISO and OGC models. In: IC-AI (2005)
14. Scerri, S., Attard, J., Rivera, I., Valla, M.: DCON: interoperable context representation for pervasive environments. In: AAAI Workshops (2012)
15. Saldatos, J., et al.: OpenIoT: open source Internet-of-Things in the cloud. In: Podnar Žarko, I., Pripužić, K., Serrano, M. (eds.) FP7 OpenIoT Project Workshop 2014. LNCS, vol. 9001, pp. 13–25. Springer, Heidelberg (2015)
16. Vermesan, O., Friess, P.: Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. River Publishers, Aalborg (2013)
17. Vögler, M., Schleicher, J., Inzinger, C., Dustdar, S.: A scalable framework for provisioning large-scale IoT deployments. ACM Trans. Internet Technol. **16**(2), 11:1–11:20 (2016). http://doi.acm.org/10.1145/2850416