Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 9, December 2016/January 2017, Pages 28–43
Published online by RTU Press, https://csimq-journals.rtu.lv
https://doi.org/10.7250/csimq.2016-9.02
ISSN: 2255-9922 online

Automating the Provisioning and Configuration of Devices in the Internet of Things

Pascal Hirmer[1]*, Uwe Breitenbücher[2], Ana Cristina Franco da Silva[1], Kálmán Képes[2], Bernhard Mitschang[1] and Matthias Wieland[1]

[1]Institute of Parallel and Distributed Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
[2]Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany

pascal.hirmer@informatik.uni-stuttgart.de (orcid.org/0000-0002-2656-0095),
uwe.breitenbuecher@informatik.uni-stuttgart.de (orcid.org/0000-0002-8816-5541),
francoaa@informatik.uni-stuttgart.de, kalman.kepes@informatik.uni-stuttgart.de,
bernhard.mitschang@informatik.uni-stuttgart.de,
matthias.wieland@informatik.uni-stuttgart.de (orcid.org/0000-0002-3388-483X)

Abstract. The Internet of Things benefits from an increasing number of interconnected technical devices. This has led to the existence of so-called smart environments, which encompass one or more devices sensing, acting, and automatically performing different tasks to enable their self-organization. Smart environments are divided into two parts: the physical environment and its digital representation, oftentimes referred to as digital twin. However, the automated binding and monitoring of devices of smart environments are still major issues. In this article we present a method and system architecture to cope with these challenges by enabling (i) easy modeling of sensors, actuators, devices, and their attributes, (ii) dynamic device binding based on their type, (iii) the access to devices using different paradigms, and (iv) the monitoring of smart environments in regard to failures or changes. We furthermore provide a prototypical implementation of the introduced approach.

Keywords: Internet of Things, sensors, actuators, digital twin, ontologies, TOSCA.

1 Introduction and Background

Nowadays, the integration of sensors and actuators becomes more and more important, especially for the emerging Internet of Things (IoT) [1] and Industry 4.0 [2]. Through the integration of raw sensor data, high-level information can be derived that leads to huge benefits, e.g., in advanced manufacturing [3], smart homes [4], or smart cities [1]. Additionally, the integration of

* Corresponding author

physical actuators controlling the real world enables not only the self-organization of such smart environments, but also allows IoT applications to control the smart environments themselves. Smart environments are divided into two parts: (i) the physical environment containing devices, sensors, and actuators, and (ii) its digital representation, which is referred to as *digital twin* [5], [6], [7] in the context of this article. In general, smart environments are very dynamic in the sense that devices might break down or are moved, added, or removed, e.g., when a smart phone enters or leaves the environment. Hence, the digital twin has to be in-sync with the physical environment at all times in order to be used for reliable monitoring. The synchronization of the physical environment and the digital twin in order to enable this monitoring is the goal of this article.

However, to realize such a monitoring, physical devices, sensors, and actuators have to be bound to the digital world to create the digital twin of their environment. In many IoT approaches, devices, sensors, and actuators are manually registered and bound for processing, which is a complex task that requires deep technical knowledge about them. Furthermore, appropriate device adapters have to be manually created and deployed for each sensor and actuator to connect them to an IoT middleware. The main purpose of such device adapters is to extract and provision sensor data to IoT applications, as well as to receive control commands from IoT applications. Deploying these adapters manually is error-prone and can take hours or even days to be processed: a hardware expert has to configure the devices, install necessary device adapters for them, bind them, and establish interfaces and gateways to IoT applications. All these steps are necessary for the *provisioning* of devices in the IoT. In real-world scenarios, e.g., for situation recognition [8], [9], [10], efficiency and accuracy are of vital importance. The drawbacks that come with a manual registration can lead to high costs due to occurring errors and a tedious, time-consuming binding process. In this context, *binding* means enabling IoT applications to access sensors and actuators of devices. Therefore, only the *automated* registration, configuration, and binding of devices enables efficiently monitoring a smart environment through its digital twin.

In this article, our goal is to reduce the required manual steps to the modeling of smart environments with their sensors, actuators, and devices. All other steps for the binding and monitoring of devices in smart environments shall be processed automatically. By doing so, we can reduce occurring errors that may occur in manual processing and, as a consequence, save costs. To achieve this goal, we present a method and system architecture for (i) easy modeling of sensors, actuators, devices, and their attributes, (ii) dynamic device binding based on their type, (iii) the access to IoT resources using different paradigms, and (iv) the monitoring of smart environments through their digital twins. Note that there are also objects in the world that are not observable by sensors, these are not covered here.

This article extends the paper *"Automated Sensor Registration, Binding, and Sensor Data Provisioning"* [11] presented at the *Forum of the 28th International Conference on Advanced Information Systems Engineering (CAiSE) 2016* with several new concepts, namely: (i) the modeling of IoT environments to enable registration and binding of whole environments instead of single devices, (ii) the monitoring of smart environments, and (iii) the binding of devices using topology models. Furthermore, we not only focus on devices equipped with sensors but also consider actuators.

The article is structured as follows: Section 2 presents the main contribution of the article: a system architecture and a method for the provisioning and configuration of devices for the Internet of Things. In Section 3 we describe how the binding of devices can be realized through topology models. In Section 4 we introduce our prototypical implementation. Section 5 describes related work, Section 6 evaluates the approach and describes limitations, and, finally, Section 7 gives a summary and describes future work.

2 Automated Provisioning and Configuration of Devices

This section introduces the main contribution of the article: a system architecture and a method for automated provisioning and configuration of devices. Due to the many existing terminologies in the IoT area, we use the following terms in the context of this article: *things* are physical devices embedded with or connected to sensors and actuators. A *sensor* collects specific data within physical environments, e.g., the temperature of a room. An *actuator* is a mechanism used to act upon the smart environment, e.g., the electric engine to close a window. Furthermore, a *digital twin* is a virtual representation of the physical environment that is kept in-sync with it to enable its effective monitoring and adaptation.

Our approach for the provisioning and configuration of devices for the Internet of Things supports three types of devices: (i) plug-and-play devices, (ii) configurable devices, and (iii) constrained configurable device. A *plug-and-play device* has embedded sensors and actuators. However, it does not allow configuration via a (remote) management system but rather only provides interfaces to access sensor data and to control it through its actuators. Examples for such devices are WiFi-enabled wearables, cameras, and audio systems. A *configurable device*, such as a Raspberry Pi[1], has sensors and actuators attached to it and offers a runtime, e.g., for the deployment of device adapters that extract and provision sensor data. A *constrained configurable device*, such as a microcontroller board Arduino[2], has sensors and actuators attached to it as well. However, such devices have limited processing and storage capabilities and are connected to more powerful hardware, in this article referred to as *gateway* or as *Device Adapter Platform*.

Figure 1 depicts the architecture of our approach. It consists of the following main components: (i) the *Device Registry*, which stores meta-information about the devices, sensors, and actuators, (ii) the *Device Ontology*, which describes sensor and actuator binding information, (iii) the sensor and actuator *device adapters*, stored in the *Device Adapter Repository*, which can be deployed directly on a device or on a *Device Adapter Platform*, and (iv) the *Resource Management Platform (RMP)*, which provisions the sensor data as remotely accessible resources (pull) or via a publish-subscribe approach (push). Moreover, the RMP provides the access to actuators. In addition, there are two components for the monitoring of the smart environment: the *digital twin* of the physical environment and the *Monitoring Dashboard*, which displays the state of the physical environment.

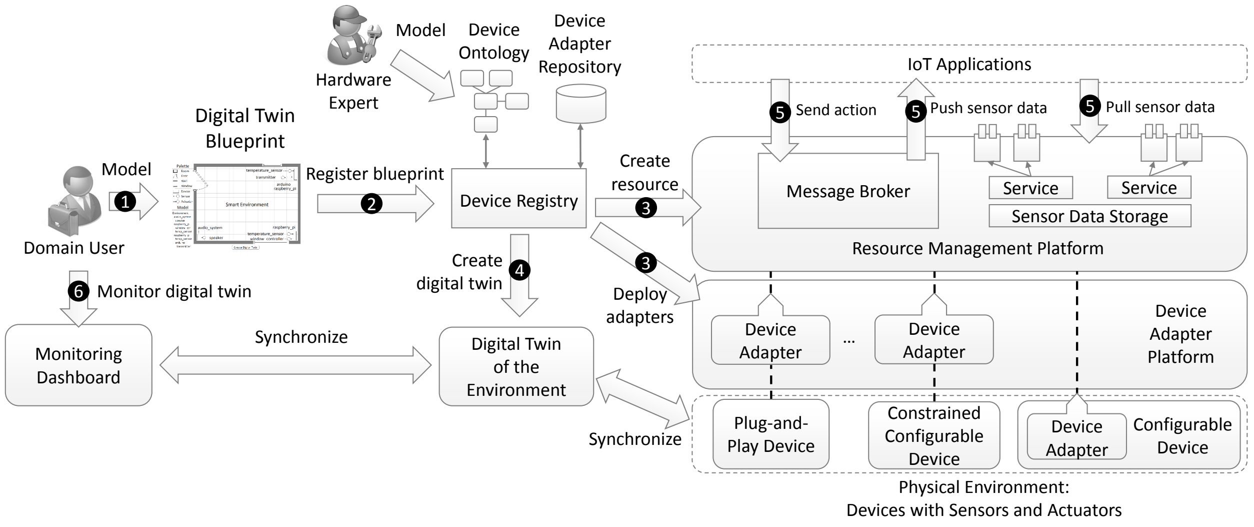

Figure 1. System architecture for the provisioning and configuration of devices

This architecture can be used as described by the steps depicted in Figure 1 (black circles). There are two kinds of actors: the *domain user* and the *hardware expert*. The domain user has knowledge

[1] https://www.raspberrypi.org/
[2] https://www.arduino.cc/

about the physical environment and thus can model smart environments with their devices, sensors, actuators, and corresponding locations. More precisely, he creates the *blueprint* for the digital twin. This blueprint is a model containing information about the smart environment, e.g., a room, and its contained devices equipped with sensors and actuators. It serves the basis for the binding of the devices and for the monitoring of the environment. The hardware expert, in contrast, hastechnical knowledge about the devices, sensors, and actuators in the physical environment. He is responsible for creating and maintaining the *Device Ontology*, which serves as the basis for the binding of the devices. The separation of technical experts and domain users leads to a clear *separation of concerns*.

The architecture represented in Figure 1 enables the use of the method for automated provisioning and configuration of devices in six consecutive steps: (1) creation of the blueprint of the digital twin, (2) blueprint registration, (3) automated device adapter deployment and device binding, (4) creation of the digital twin, (5) enabling IoT applications to access and control the devices i.e., their sensors and actuators, and (6) monitoring of the digital twin through a dashboard.

Step 1: Digital Twin Blueprint Modeling

In the first step of the method, the domain user creates the blueprint for the digital twin based on the physical environment, which is well-known to him/her. The structure of this blueprint model could be realized, e.g., similarly to the one introduced by [12]. The underlying data model for the digital twin blueprint can range from ontology-based to XML-based or JSON-based representations. The blueprint for small environments, such as homes and offices, could be modeled through a graphical modeling tool as depicted in Figure 2.

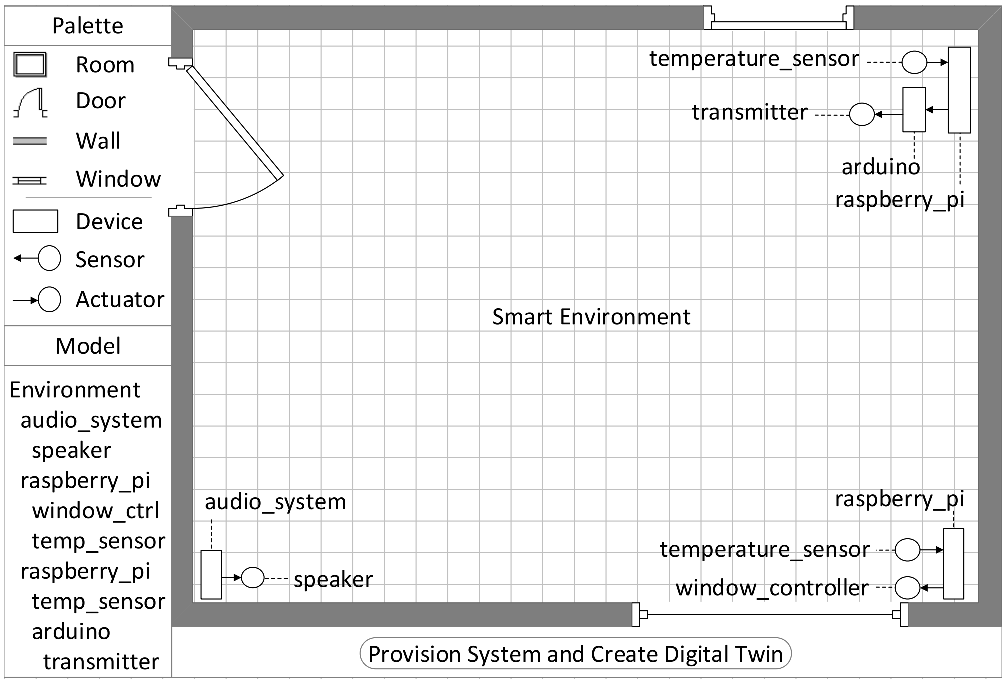

Figure 2. Mock-up of a digital twin blueprint modeling tool

In such a tool, the domain user can specify the size and the layout of the environment, e.g., one or several rooms in a smart home, and specify the hardware devices present in the environment with their corresponding properties. These properties describe specific information about the devices, such as the MAC-addresses, location, identifiers, and attached sensors and actuators. To simplify the modeling of devices with all these properties, they should already provide this information, e.g., through a QR code attached to them that can be scanned to automatically fill out the properties (e.g., the MAC address of the device). This leads to a reduced effort for blueprint modeling. However, the visualization has to cope with the modeling of very complex environments as well, such as a factory containing a huge number of machines and other elements. This problem is out of the scope of this article and, thus, part of our future work. In particular, we are working on a generic modeling language based on the TOSCA standard [13], [14], [15].

Step 2: Blueprint Registration

In this step, the devices, sensors, and actuators of the environment are registered based on the information contained in the blueprint created in Step 1. Additional information about the devices, sensors, and actuators are retrieved from the Device Ontology by traversing it and searching for the corresponding entries of the sensors, actuators, or devices modeled in the blueprint. The ontology describes technical details that are necessary for an automated device registration, binding, and enabling the access to the devices; it can also be used as a meta-data source by IoT applications. The meta-data include sensor and actuator specifications (e.g., their accuracy and frequency). To enable an efficient storage and retrieval of this information, our ontology builds on the XML-based sensor markup language *SensorML* [16]. Ontologies offer the means for an automated editing and extension of information, using e.g., generated SPARQL queries. In IoT scenarios, the use of ontologies is commonly accepted [17], [18], [19] due to the heterogeneous, dynamic environments that have to be integrated.

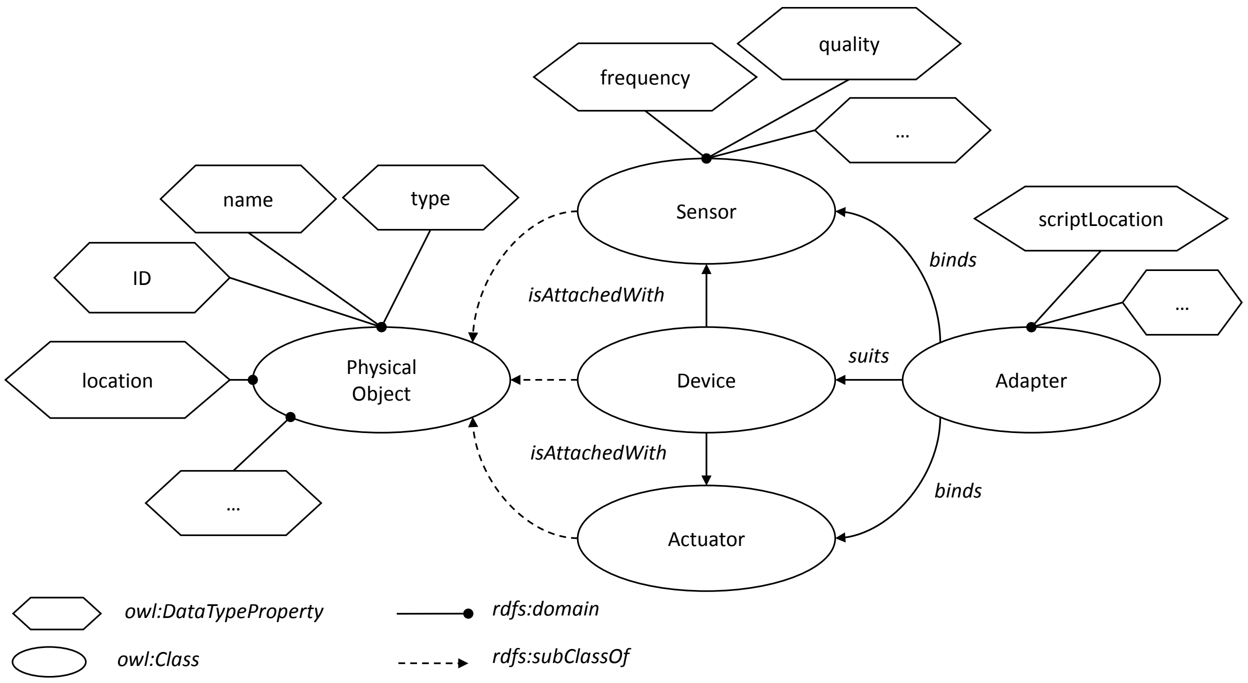

Figure 3. Condensed device ontology model (based on [11]; namespaces removed for clarity)

A condensed view of our ontology model is depicted in Figure 3. It contains the following elements: *Physical Object*, *Device*, *Sensor*, *Actuator*, and *Adapter*. The ontology describes, which sensors and actuators are embedded in or connected to a device, as well as the adapter that is used for binding them. Devices, sensors, and actuators are derived from the super class *Physical Object* because they share several properties. The runtime environment the adapter is deployed into, i.e., the Device Adapter Platform or the device itself, can be derived through its type as previously described. If a device, a sensor, or an actuator is not represented in the ontology, an ontology snippet describing their properties has to be provided with the blueprint, which contains the necessary binding information. In this case, binding information means the device adapter that is being deployed to bind the device and information about how to connect to the device, e.g., the Secure Shell (SSH) credentials. However, in the following, we assume that the ontology contains all sensors, actuators, and devices of the specific domain our approach is applied to and is modeled *correctly*, which means that there are no syntactical errors contained in it. Once the binding information of devices, sensors, and actuators is retrieved from the ontology, they are used for automated device binding, which is described in the next step.

Step 3: Automated Device Adapter Deployment and Device Binding

After the required information to bind the devices has been extracted from the ontology, the next step is the automated device binding to enable IoT applications to access them. To achieve this, we first have to have a means to extract the sensor data from the corresponding sensors and to control the corresponding actuators. For this purpose, *device adapters* are required. The concept of device adapters is well-known in the field of IoT as a common concept to bind devices. More precisely, a device adapter is a piece of code containing the logic to connect to a device, extract the sensor data from the serial interface, activate actuators, and provision sensor data, in our approach, to the RMP. The provisioning of sensor data is usually conducted by the protocol HTTP or the lightweight queuing protocol MQTT[3]. We assume that a corresponding device adapter is provided in the Device Adapter Repository suitable for each device represented in the ontology. The corresponding device adapters to bind the devices, sensors, and actuators modeled in the digital twin blueprint were extracted from the ontology in the previous step.

In order to deploy device adapters automatically, they are retrieved from the repository and are subsequently parameterized, e.g., with the pin placement of sensors or the location of the RMP. As already mentioned, we support three different types of device bindings: (i) plug-and-play device binding, (ii) configurable device binding, and (iii) constrained configurable device binding. A plug-and-play device has embedded sensors and actuators. Such devices do not allow the deployment of device adapters on them but rather provide interfaces (oftentimes based on HTTP REST or MQTT) to extract the sensor data. In this case, an adapter, which uses the provided interface to retrieve the sensor data, still has to be deployed somewhere else (e.g., on a gateway or on a virtual machine). A configurable device (e.g., a Raspberry Pi) has sensors and actuators attached to it. The device adapters are deployed and executed directly on this type of device. A constrained configurable device has sensors and actuators attached to it as well, however, due to its limited processing and storage capabilities, this type of device is generally connected to a gateway, which is more powerful and has network capabilities. In this case, the device adapters are not deployed directly on the device, but on the gateway that is capable of extracting the information from the constrained device.

The deployment and execution of device adapters can be conducted either through SSH connections or through more sophisticated deployment approaches. In [11], [20], e.g., we describe how an SSH based deployment of device adapters can be realized, assuming that necessary software to run the adapters is already provided on the devices or on the Device Adapter Platform. However, using SSH connections comes with the drawback that an operating system, as well as a suitable runtime for the device adapter already has to be provided by the environment. To cope with this issue, Section 3 describes how device adapters can be deployed based on topology models describing the whole eco system of the environment the device adapter is deployed on.

Once device adapters are deployed and started, they constantly extract sensor data and send them to the RMP, which serves as an interface to upper-level IoT applications. Device adapters send sensor data to the RMP using different protocols such as HTTP or MQTT. Furthermore, the RMP offers an interface to trigger actuators of devices by invoking the corresponding device adapter scripts. Consequently, through this approach, the RMP enables abstraction from the concrete sensors and actuators of the devices.

[3] http://mqtt.org/

Step 4: Creation of the Digital Twin

In this step, we create the digital representation of the physical environment – the digital twin – based on the registered blueprint of Step 2 and the bound devices, sensors, and actuators (Step 3). The digital twin is a model that is constantly synchronized with the physical environment. More precisely, it provides the logic to monitor the health of devices, their location, current sensor data, and other parameters. The corresponding data is provided through the RMP that is connected directly to the devices. Adaptations of the physical environment, e.g. when a device gets added or removed, change the digital twin model. This can be achieved fully automatically through model transformation approaches such as XSLT[4]. Changes in the physical environment can be either detected automatically through the RMP using heart beats, or manually by domain users through the Monitoring Dashboard (cf. Step 6 in Figure 1). The underlying model for the digital twin should be similar to the one used for the modeling of its blueprint in order to avoid tedious transformations. As mentioned earlier, ontologies or XML-based models could be used for this. We are aware that there are many challenges to synchronize the digital twin with real-time capabilities. This will be our future work.

Step 5: Enabling IoT Applications to Access Devices

In this step, we automatically enable IoT applications to access devices, i.e., IoT applications can get sensor data of devices and control their actuators. The devices have already been bound in Step 3 through the device adapters, which send sensor data to the RMP and receive commands to trigger actuators. We provide two approaches to access sensor data through the RMP, namely: pull-based and push-based approaches. In the pull-based approach, IoT applications can actively retrieve sensor data based on a send/request model. In the push-based approach, IoT applications are notified about sensor data through a publish/subscribe model. We enable this through (i) a sensor data storage that stores all the sensor data provided through the device adapters so they can be retrieved by the pull-based approach. Furthermore, we provide (ii) a message broker to enable the publish-subscribe approach. Actuators can also be controlled through the RMP that calls the corresponding operations provided by the device adapters. In this manner, we enable IoT applications to access sensor data, as well as to control actuators in order to act upon smart environments.

Step 6: Monitoring of the Environment

The physical environment is represented by its digital twin, which is constantly synchronized with the physical world. We provide a Monitoring Dashboard (cf. Figure 1) that displays this digital twin in a graphical representation. In this way, domain users can, e.g., get visual feedback about faulty hardware devices in the environment, which are highlighted in the digital representation. Due to the exact virtual representation of the physical environment, the model can provide information about the exact position of the defect device so it can be replaced easily. In this case, after replacing the hardware, the digital twin is synchronized automatically by binding the affected devices as described in this method. Faulty hardware devices can, e.g., be detected through heart beats. Furthermore, a newly added device can be recognized through the appearance of a new, unknown MAC address in the environment. However, on detection of new hardware devices, the domain user still has to provide the necessary properties of the new device, so it can be bound to the RMP and can be added to the digital twin of the environment.

[4] https://www.w3.org/TR/xslt

3 Automated Device Adapter Deployment and Device Binding Based on Topology Models

In Step 3 we introduced the deployment of device adapters, which can be realized through SSH connections or by employing more sophisticated approaches. In this section we describe how the deployment of such device adapters can be realized using topology models.

Topologies of sensors, actuators, and arbitrary hardware are inherently complex and heterogeneous in the domain of IoT. As such, the deployment and provisioning of software, especially device adapters that route data from the sensors and commands to the actuators, present a significant challenge. Device adapter software must be installed, configured, and started for a possibly large set of IoT resources. Our approach to cope with such challenges is introduced as follows.

For the deployment and provisioning of device adapter software – or software in general – we designed a service that is able to embed arbitrary software artifacts into a topology model that represents a suitable runtime environment for that particular artifact. This topology model – which we call *application topology model* – specifies what kind of components and resources are required to run a particular software artifact. Application topology models consist of a graph of nodes and edges, whereas the nodes represent components, such as Web Servers, Raspberry Pis, or other kind of devices and the edges specify the dependencies between these components, e.g., that one component is *hostedOn* another component. Metamodels that support such constructs are provided, e.g., by the Service Component Architecture (SCA) [21] and the Topology and Orchestration Specification for Cloud Applications (TOSCA) [15]. The components and dependencies in these topology models are typically typed, e.g., a component may be of type *ApacheWebServer* or *RaspberryPi3*.

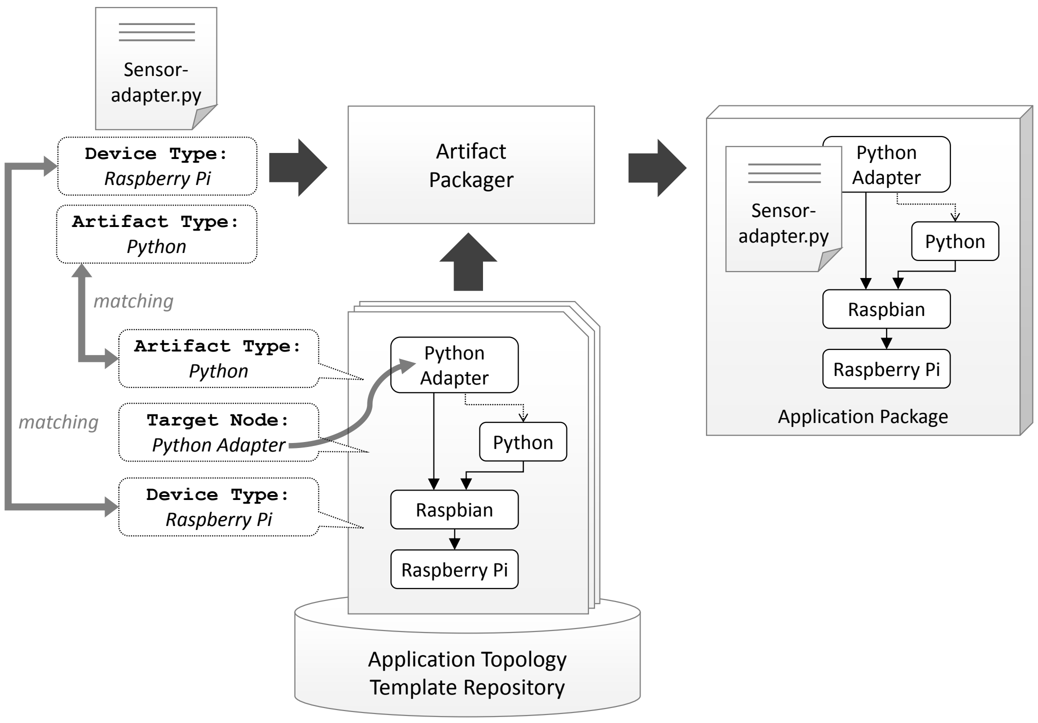

Figure 4. Artifact Packager with the Application Topology Template Repository for inserting software device adapters into suitable application topology models

Our approach is built on the basic concept of reusing application topology models as templates for automatically deploying software artifacts, in our case, for device adapters on devices. An overview of the approach is given in Figure 4, which depicts a concrete example to embed a Python-based device adapter into a topology model consisting of a Raspberry Pi component as an IoT device and components to run a Python script on the Raspberry Pi. The main entities of the approach are the *Artifact Packager* and the *Application Topology Template Repository*. One of the responsibilities of the Packager is to receive requests containing software artifacts to be deployed, their corresponding application type and a device type which specifies the kind of device the artifact belongs to. With the given data the Packager determines a suitable *Application Topology Template* that can be used to deploy the artifact from the repository. After finding a suitable template, the Packager inserts the software artifact into the topology. For the Packager to be able to achieve this, the Application Topology Template Repository entails annotations for all stored templates. The minimal set of annotations consists of the *Artifact Type* and *Target Node*, where the Application Type depicts what kind of software artifact is expected to run inside the topology and the Target Node annotation specifies on which node the artifact must be inserted. After inserting the artifact to be deployed into an appropriate template, the resulting *Application Package* contains the topology model, as well as the inserted artifact. Thus, all information required to deploy this artifact is provided by this self-contained package. Instantiation of such topology models are, after packaging, achieved by using an application container that understands the used topology model, e.g., Apache Tuscany [22] in case of SCA models or OpenTOSCA [23] in case of TOSCA models. This allows to wrap software adapters into suitable topology models at runtime, which can be deployed fully automatically to retrieve data from sensors inside an IoT scenario or to control actuators. Thus, if a device adapter has to be deployed on a physical device, only an appropriate topology template has to be selected by the Packager depending on the device's type. Therefore, the packager also receives the type of the device on which the artifact has to be deployed. In our example, this is a *Raspberry Pi* as depicted in Figure 4. After inserting the artifact, the topology model in the resulting Application Package, only has to be adapted to the concrete device. This means, that the MAC or IP address of the device and other information are inserted into the respective device component element in the model. After this completion, the topology model can be deployed fully automatically by an appropriate runtime as all required information is contained in the completed package. As a result, the approach enables to reuse proven knowledge about deploying artifacts onto physical devices as this knowledge can be captured using topology models.

4 Prototypical Implementation

We implemented an open-source prototype of the introduced approach, which is available on GitHub[5]. The technologies used for implementation are described in the following. The Device Registry component is based on NodeJS and offers a REST-based program interface. The registry uses SPARQL requests to access the Device Ontology, SSH to deploy the device adapters, and HTTP to notify the RPM that a new device has been registered. Currently, the native file system is used as Device Adapter Repository. The ontology was defined using the Web Ontology Language (OWL) 1.1[6]. The access to the ontology is done by SPARQL requests through the Apache Jena[7] framework. The RMP is also implemented in NodeJS, which enables an easy definition of a RESTful interface. Furthermore, due to the lightweight platform, it offers high efficiency. The sensor data storage is implemented using the NoSQL database MongoDB[8], which allows high efficiency, scalability, and data replication. The direct push approach was realized using MQTT[9] and the Mosquitto[10] broker. A tool for modeling the digital twin blueprint, and a monitoring dashboard have not yet been implemented and are part of our future work.

[5] https://github.com/SitOPT/

[6] http://www.w3.org/Submission/owl11-overview/

[7] https://jena.apache.org/

[8] http://www.mongodb.org/

[9] http://mqtt.org/

[10] http://mosquitto.org/

The implementation of the Artifact Packager system resulted in a service exposing a RESTful API, which enables the packaging of software artifacts in TOSCA CSAR archives. Users are able to specify the artifact, its TOSCA ArtifactType, and a set of TOSCA NodeTypes that must be available in the TOSCA ServiceTemplate as TOSCA NodeTemplates. The implementation of the Artifact Packager itself was conducted in the language Java, using Jersey as REST framework. As an implementation for the Application Topology Template Repository inside the whole packaging system, we used the TOSCA Repository Eclipse Winery[11] [24], which allows modeling all relevant entities of a TOSCA document. The Artifact Packager is open-source and available at GitHub[12]. To enable the wrapping of software adapters (in our scenario we need the means to deploy a Python software adapter), we created an application topology in the language TOSCA. To deploy this application topology, we use the OpenTOSCA Runtime Container [23], which is a part of the OpenTOSCA Ecosystem [23], [24], [25], [26] and is able to provision the given models in a declarative manner [27]. The TOSCA topology itself is depicted in Figure 5 using the visual notation *Vino4TOSCA* [28] and contains nodes that allow to install a Python-based device adapter on a Raspberry Pi 3. To configure the adapter, the topology also specifies a Mosquitto broker node with its Remote Host node and the Topic node the adapter should publish to. For binding the topology against a specific Mosquitto broker, the properties of the corresponding TOSCA nodes must be set accordingly. For instance, the Remote Host node contains the address of the broker, while the Topic node contains a property specifying the name of it. With this information the Python Device Adapter can be configured automatically by the container to publish its received sensor data to the desired Mosquitto broker topic. The developed topology is available on GitHub[13].

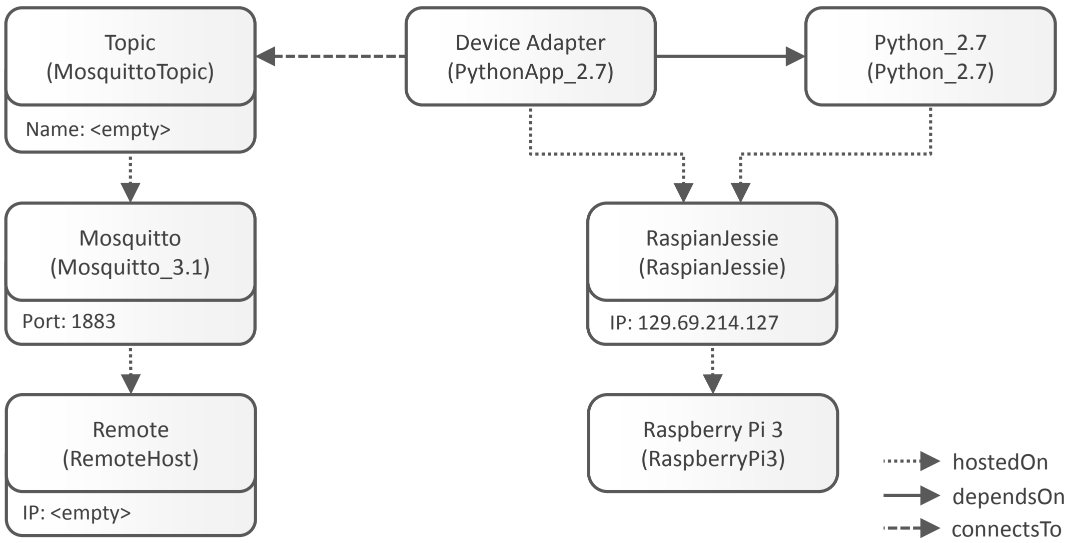

Figure 5. TOSCA topology for Python device adapters connecting to a MQTT Mosquitto broker topic

5 Related Work

Many similar approaches exist that either aim for the modeling of the smart environment or focus on the automated device binding. However, none of them combine both approaches in order to create and monitor digital twins of physical environments.

Mayer et al. [29] present an approach combining semantic metadata and reasoning with a visual modeling tool in order to enable the goal-driven configuration of smart environments. Users

[11] http://projects.eclipse.org/projects/soa.winery
[12] https://github.com/SitOPT/XaaSPackager
[13] https://github.com/SitOPT/CAiSE2016_MosquittoPythonDeviceAdapter

state which properties (i.e., goals) a smart environment should have. The system then determines whether goals can be reached and which actions are necessary based on the available services. This approach, however, assumes that devices are already deployed in the environment. In our approach, we model devices of the smart environment, which are automatically registered and bound in order to create a digital twin for their provisioning and configuration.

Nugent et al. [12] present *HomeML*, a XML-based open format for the exchange of data generated within smart environments. It aims to address problems caused by the heterogeneous data in such environments [12], [30]. With the proposed format, a home, existing rooms, and the sensors within a room can be described. Furthermore, information about the persons living in the home (e.g., patient details, healthcare plans) are specified as well. HomeML serves as basis for our approach in regard to its modeling of physical environments. However, in contrast to homeML, we provide a more generic and lightweight model, which contains only essential information (e.g., device type, location, and MAC address) for the creation of blueprints. Additional, specific information are provided in the Device Ontology.

Li et al. [31] propose the employment of the TOSCA standard [13] to specify the basic constructs of IoT applications (e.g., gateways, controller, etc.) and their configuration, in order to improve the reusability of service management processes and to automate IoT application deployment in heterogeneous environments. However, modeling the components of IoT applications directly in TOSCA still requires some efforts, since the available TOSCA tools are not yet completely integrated with IoT concepts. This work was extended by Vögler et al. [32], which present LEONORE, a scalable deployment framework to deploy and execute custom application logic directly on IoT gateways. However, in order to know which IoT gateways are available for the provisioning, the IoT gateways must have a pre-installed local provisioning agent. This agent registers the gateway with the framework by providing its unique identifier and gathered profile data (e.g., ID, MAC-address, instruction set, etc.). In our approach, no pre-installed components on the IoT-gateways are needed. The blueprint already contains the necessary information (e.g., MAC-address) to remotely connect to devices and bind them to the RMP.

In [33] the goal is similar to our approach. The authors present a middleware called Global Sensor Network (GSN), which enables binding data sources like sensors and data streams with zero programming effort. To realize that, a virtual sensor abstraction is provided in [33], which allows declarative specification of deployment descriptors and basic processing of the data using SQL-like queries. In our approach, we separate these steps strictly. First, the dynamic sensor binding is done using ontologies. Second, the data is provisioned to sensor-driven applications.

Sensor description and configuration is standardized in IEEE1451.2 defining Transducer Electronic Data Sheets (TEDS) [34] that enable the self-description of sensors. Furthermore, an interface for standardized dynamic plug and play binding of sensors to networks is provided. In our approach, the physical binding of sensors is not the focus. We concentrate on an easy provisioning of sensor data to IoT applications through the Internet. Note that standards, such as the IEEE1451.2, could be used for device binding in our approach.

In [35], a REST-based interface is built to access sensors and retrieve their data. By doing so, the article assumes an already in place sensor network bound to a gateway, which provides information of the sensors and manages their access for data retrieval. In contrast, our article does not necessarily assume the existence of such a gateway and, hence, manages the device binding itself. Only the provisioning of sensor data is similar to our approach in [35].

In recent years, a large amount of Machine-to-Machine (M2M) gateways [36], [37], [38], [39] have been created, such as FIWARE[14], OpenMTC[15], OpenIoT [40], or GSN [41], [42]. These gateways serve as a layer between physical sensors and “virtual” sensor data. It is important to note that the approach in this article does not try to compete with these approved platforms but

[14] https://www.fiware.org/

[15] http://www.open-mtc.org/

rather uses them, i.e., provides a more abstracted layer on top in order to enable an easy way to bind *devices* in contrast to specific sensors, and to automatically provision data of the contained sensors to IoT applications using Internet technologies. More precisely, the mentioned platforms can be used as gateways by our approach to realize the device binding.

Middleware between the physical and application layer gain more and more importance [43]. The main purpose of such middleware systems is to hide and abstract the physical details in order to allow the programmer to focus on the development of a specific sensor-driven application. Furthermore, it is important to abstract from the concrete environment and its contained sensors and actuators that will be used after deployment of the application, in order to avoid a cumbersome and time-consuming configuration in each new environment.

SStreaMWare [44] is a service-oriented middleware for heterogeneous sensor data. It uses a hybrid approach supporting both centralized data streams and distributed sensor networks. Furthermore, a generic schema for sensor data representation is proposed (measures, timestamps, and properties) and declarative queries can be executed on the sensor data streams. SStreaMWare has an approach similar to ours in managing the sensors and binding them based on the devices.

OntoSensor [17] is a sensor knowledge repository for modeling and management of sensors. It combines SensorML, IEEE SUMO, ISO 19115, OWL and GML. The goal of OntoSensor is to achieve a usage for description of sensors in different application domains. Through the combination of many different sensor definition languages, the ontologies become heavy-weight and complex. In our approach, we aim for a particularly lightweight ontology. Because of that, we decided to use only a subset of SensorML and omit using the whole OntoSensor ontology.

DCON [45] is an ontology for representation of user activity context. In [45], the authors combine many different OSCAF ontologies[16] to create a Personal Information Model: DDO (for Devices), DPO (for Presence), and DCON [45] for representation of user activity context. This is a specialized area and the ontologies are very detailed. In our approach the goal is to support any domain, so the concepts are more generic. Not everything is focused on the users, in our approach devices are the main focus and persons in contrast should not be monitored for privacy reasons.

In summary, the presented related work is mainly focusing on specific aspects like the modeling of smart environments, the access to devices, sensors and actuators using gateways, or the execution of queries on sensor data streams or in a sensor network. It shows that either the modeling of smart environments or the binding of devices was achieved. However, the integration of these components has not yet been conducted. This is an important task to enable end-to-end modeling and deployment of devices. By realizing this, we enable an easy provisioning and configuration of devices of smart environments even for domain users that are not familiar with all the technical details about, e.g., device binding and network communication. The domain user only has to provide information about the physical environment, which is well-known to him/her. Based on the created digital twin, domain users can monitor and manage the devices of smart environments themselves.

6 Evaluation and Limitations

In this section, we discuss the strengths and current limitations of our approach for automating the provisioning and configuration of devices in the Internet of Things.

As already shown in the previous work this article builds on [11], [20], we can achieve the automated binding of devices using SSH connections within milliseconds instead of hours or even days. In the extended, more robust binding approach based on TOSCA, this is still the case. However, a slight overhead is expected due to the TOSCA runtime environment used for deployment, which in return offers many advantages such as robustness and error handling. This overhead is still very small (seconds) in contrast to a manual binding. Consequently, the evaluation

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

as discussed in [11] is still valid for our extended approach. The corresponding implementation to achieve this is described in Section 4. Furthermore, another strength of our approach lies in the separation of concerns between the domain user, interested in binding of devices without the need to know how this is achieved, and the hardware expert providing the expertise to realize the binding. Through this clear separation, non-IT domain users are able to bind devices without any necessary technical expertise about them.

There are still some limitations that have to be investigated in the future. First, the modeling concept for IoT environments, that serves as basis for the device binding, has to be detailed. More precisely, complex IoT environments, such as shop floor environments, currently cannot be modeled in the proposed graphical modeling tool due to the large amount of machines, devices, sensors, and actuators to be represented in the model. Consequently, an automated model creation concept needs to be developed that is able to discover devices and to automatically build the model. Second, our approach does not yet support existing standardized IoT environment models such as IoT-Lite[17]. To enable applicability in a larger variety of scenarios, support of such standards is mandatory and will be provided in the future.

7 Summary and Future Work

In this article, we present an approach for automating the provisioning and configuration of devices, and for monitoring of smart environments through digital twins. We introduced a system architecture and a method to make our approach applicable for a wide range of Internet of Things applications. After registration of a device, a device adapter is deployed automatically that reads the sensor data and passes them to the Resource Management Platform, or invokes actuators. By doing so, we create an easy-to use solution for IoT applications to bind devices, access their sensors and actuators, and monitor their state. Our goal was enabling this in an automated manner within milliseconds in contrast to a manual processing of these steps that can take up to hours or even days. This goal was achieved through the introduced automated approach. In the future, we will concentrate on the creation of digital twins by developing a corresponding model. Furthermore, we will enable monitoring through an implementation of the monitoring dashboard.

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

References

[1] O. Vermesan and P. Friess, Eds., “Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems,” River Publishers, pp. 363, 2013.

[2] N. Jazdi, “Cyber Physical Systems in the Context of Industry 4.0,” in IEEE International Conference on Automation, Quality and Testing, Robotics, pp. 1–4, May 2014. [Online]. Available: https://doi.org/10.1109/aqtr.2014.6857843

[3] F. Tao, Y. Cheng, L. Zhang and A.Y.C. Nee, “Advanced Manufacturing Systems: Socialization Characteristics and Trends,” Journal of Intelligent Manufacturing, pp. 1–16, 2015. [Online]. Available: http://doi.org/10.1007/s10845-015-1042-8

[4] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions,” Future Generation Computer Systems, vol. 29, no. 7, pp. 1645–1660, 2013. [Online]. Available: https://doi.org/10.1016/j.future.2013.01.010

[5] S. Boschert and R. Rosen, “Digital Twin – The Simulation Aspect,” Cham: Springer International Publishing, pp. 59–74, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-32156-1_5

[17] https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/

[6] R. Rosen, G. von Wichert, G. Lo and K.D. Bettenhausen, "About The Importance of Autonomy and Digital Twins for the Future of Manufacturing," IFAC-PapersOnLine, vol. 48, no. 3, pp. 567–572, 2015. [Online]. Available: https://doi.org/10.1016/j.ifacol.2015.06.141

[7] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I.S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer and P. Doody, "Internet of Things Strategic Research Roadmap," Internet of Things-Global Technological and Societal Trends, pp. 9–52, 2011. [Online]. Available: http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2011.pdf

[8] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher and F. Leymann, "SitRS-A Situation Recognition Service based on Modeling and Executing Situation Templates," in Proceedings of the 9th Symposium and Summer School on Service-Oriented Computing, pp. 247–258, 2015.

[9] A.C. Franco da Silva, P. Hirmer, M. Wieland and B. Mitschang, "SitRS XT – Towards Near Real Time Situation Recognition," Journal of Information and Data Management, vol. 7, no. 1, pp. 4–17, 2016. [Online]. Available: https://seer.ufmg.br/index.php/jidm/article/view/2109/2644

[10] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann and M. Wieland, "A Situation-Aware Workflow Modelling Extension," in Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2015), ACM, pp. 478–484, 2015. [Online]. Available: https://doi.org/10.1145/2837185.2837248

[11] P. Hirmer, M. Wieland, U. Breitenbücher and B. Mitschang, "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, CEUR, pp. 81–88, 2016.

[12] C.D. Nugent, D.D. Finlay, R.J. Davies, H.Y. Wang, H. Zheng, J. Hallberg, K. Synnes and M.D. Mulvenna, "homeML – An Open Standard for the Exchange of Data Within Smart Environments," ch. Pervasive Computing for Quality of Life Enhancement, Springer, pp. 121–129, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-73035-4_13

[13] T. Binz, Breitenbücher, O. Kopp and F. Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications," Springer, pp. 527–549, 2014. [Online]. Available: https://doi.org/10.1007/978-1-4614-7535-4_22

[14] OASIS, "TOSCA Primer," Nov. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf

[15] OASIS, "OASIS Topology and Orchestration Specification for Cloud Applications," Nov. 2013. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

[16] M. Botts, G. Percivall, C. Reed and J. Davidson, "OGC Sensor Web Enablement: Overview and High Level Architecture," in GeoSensor Networks, Lecture Notes in Computer Science, Springer Berlin Heidelberg, vol. 4540, pp. 175–190, 2008. [Online]. Available: https://doi.org/10.1007/978-3-540-79996-2_10

[17] D.J. Russomanno, C.R. Kothari and O.A. Thomas, "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models," in IC-AI, pp. 637–643, 2005.

[18] F. Probst, A. Gordon and I. Dornelas, "OGC Discussion Paper: Ontology-Based Representation of the OGC Observations and Measurements Model," Institute for Geoinformatics (IFIGI), pp. 26, 2006.

[19] J. Attard, S. Scerri, I. Rivera and S. Handschuh, "Ontology-Based Situation Recognition for Context-Aware Systems," in Proceedings of the 9th International Conference on Semantic Systems. ACM, pp. 113–120, 2013. [Online]. Available: https://doi.org/10.1145/2506182.2506197

[20] P. Hirmer, M. Wieland, U. Breitenbücher and B. Mitschang, "Dynamic Ontology-Based Sensor Binding," in Proceedings of the 20th East-European Conference on Advances in Databases and Information Systems (ADBIS), Springer, pp. 323–337, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-44039-2_22

[21] OASIS, "Service Component Architecture Assembly Model Specification Version 1.1," Organization for the Advancement of Structured Information Standards (OASIS), Sep. 2011. [Online]. Available: http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.pdf

[22] Apache Foundation, "Apache Tuscany SCA Java Architecture Guide," 2016. [Online]. Available: http://tuscany.apache.org/sca-java-architecture-guide.html

[23] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak and S. Wagner, "OpenTOSCA - A Runtime for TOSCA-Based Cloud Applications," in Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013), Springer, pp. 692–695, Dec. 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-45005-1_62

[24] O. Kopp, T. Binz, U. Breitenbücher and F. Leymann, "Winery – A Modeling Tool for TOSCA-Based Cloud Applications," in Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013), Springer, pp. 700–704, Dec. 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-45005-1_64

[25] U. Breitenbücher, T. Binz, O. Kopp and F. Leymann, "Vinothek - A Self-Service Portal for TOSCA," in Proceedings of the 6th Central-European Workshop on Services and their Composition (ZEUS 2014) CEUR-WS.org, Demonstration, pp. 69–72, Feb. 2014.

[26] J. Wettinger, T. Binz, U. Breitenbücher, O. Kopp, F. Leymann and M. Zimmermann, "Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on Tosca," in Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014), SciTePress, pp. 559–568, Apr. 2014. [Online]. Available: https://doi.org/10.5220/0004859005590568

[27] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann and J. Wettinger, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in International Conference on Cloud Engineering (IC2E 2014), IEEE, pp. 87–96, Mar. 2014. [Online]. Available: https://doi.org/10.1109/ic2e.2014.56

[28] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann and D. Schumm, "Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA," in On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012), Springer, pp. 416–424, Sep. 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-33606-5_25

[29] S. Mayer, N. Inhelder, R. Verborgh, R. Van de Walle and F. Mattern, "Configuration of Smart Environments Made Simple: Combining Visual Modeling With Semantic Metadata and Reasoning," in Internet of Things (IOT), 2014 International Conference on the. IEEE, pp. 61–66, 2014. [Online]. Available: https://doi.org/10.1109/iot.2014.7030116

[30] H. McDonald, C. Nugent, J. Hallberg, D. Finlay, G. Moore and K. Synnes, "The homeML Suite: Shareable Datasets for Smart Home Environments," Health and Technology, vol. 3, no. 2, pp. 177–193, 2013. [Online]. Available: https://doi.org/10.1007/s12553-013-0046-7

[31] F. Li, M. Vogler, M. Claeßens and S. Dustdar, "Towards Automated IoT Application Deployment by a Cloud-Based Approach," in Proceedings of the IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA). IEEE, pp. 61–68, 2013. [Online]. Available: https://doi.org/10.1109/soca.2013.12

[32] M. Vögler, J. M. Schleicher, C. Inzinger and S. Dustdar, "A Scalable Framework for Provisioning Large-Scale iot Deployments," ACM Transactions on Internet Technology (TOIT), article no. 11, vol. 16, no. 2, 2016. [Online]. Available: https://doi.org/10.1145/2850416

[33] M. Hauswirth and K. Aberer, "Middleware Support for the "Internet of Things"," 5th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze", pp. 5, 2006.

[34] K. Lee, "IEEE 1451: A Standard in Support of Smart Transducer Networking," in Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference (IMTC), University of Stuttgart, 2000. [Online]. Available: https://doi.org/10.1109/imtc.2000.848791

[35] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman and P. Demeester, "Facilitating Sensor Deployment, Discovery and Resource Access Using Embedded Web Services," in 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 717–724, July 2012. [Online]. Available: https://doi.org/10.1109/imis.2012.48

[36] M. Corici, H. Coskun, A. Elmangoush, A. Kurniawan, T. Mao, T. Magedanz and S. Wahle, "OpenMTC: Prototyping Machine Type communication in Carrier Grade Operator Networks," in Globecom Workshops. IEEE, pp. 1735–1740, 2012. [Online]. Available: https://doi.org/10.1109/glocomw.2012.6477847

[37] F. Ramparany, F. Galan Marquez, J. Soriano and T. Elsaleh, "Handling Smart Environment Devices, Data and Services at the Semantic Level With the FI-WARE Core Platform," in Proceedings of the International Conference on Big Data (Big Data), IEEE, pp. 14–20, 2014. [Online]. Available: https://doi.org/10.1109/bigdata.2014.7004417

[38] M.B. Alaya, Y. Banouar, T. Monteil, C. Chassot and K. Drira, "OM2M: Extensible ETSI-Compliant M2M Service Platform With Self-Configuration Capability," Procedia Computer Science, vol. 32, pp. 1079–1086, 2014. [Online]. Available: https://doi.org/10.1016/j.procs.2014.05.536

[39] J. Mineraud, O. Mazhelis, X. Su and S. Tarkoma, "A Gap Analysis of Internet-of-Things Platforms," Computer Communications, vol. 89 - 90, pp. 5–16, 2016, internet of Things: Research challenges and Solutions. [Online]. Available: https://doi.org/10.1016/j.comcom.2016.03.015

[40] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P.P. Jayaraman, A. Zaslavsky, I.P. Žarko, L. Skorin-Kapov and R. Herzog, "OpenIoT: Open Source Internet-of-Things in the Cloud," in Interoperability and Open-Source Solutions for the Internet of Things, Springer International Publishing, 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-16546-2_3

[41] K. Aberer, M. Hauswirth and A. Salehi, "Zero-Programming Sensor Network Deployment," in Proceedings of the Service Platforms for Future Mobile Systems, Workshop at SAINT 2007, Hiroshima, Japan, IEEE, 2007. [Online]. Available: https://doi.org/10.1109/SAINT-W.2007.57

[42] K. Aberer, M. Hauswirth and A. Salehi, "A Middleware for Fast and Flexible Sensor Network Deployment," in Proceedings of the International Conference on Very Large Data Bases (VLDB 2006), ACM, 2006.

[43] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A Survey," Computer Networks, 2010. [Online]. Available: http://doi.org/10.1016/j.comnet.2010.05.010

[44] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro and V. Olive, "SStreaMWare: a Service Oriented Middleware for Heterogeneous Sensor Data Management," in Proceedings of the International Conference on Pervasive Services, 2008. [Online]. Available: https://doi.org/10.1145/1387269.1387290

[45] S. Scerri, J. Attard, I. Rivera and M. Valla, "DCON: Interoperable Context Representation for Pervasive Environments," in Proceedings of the AAAI Workshops, AAAI Technical Report WS-12-05, pp. 90–97, 2012.