Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments

Ana C. Franco da Silva[1], Uwe Breitenbücher[2], Pascal Hirmer[1], Kálmán Képes[2], Oliver Kopp[1], Frank Leymann[2], Bernhard Mitschang[1] and Ronald Steinke[3]

[1]*Institute for Parallel and Distributed Systems, University of Stuttgart, Stuttgart, Germany*

[2]*Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany*

[3]*Next Generation Network Infrastructures, Fraunhofer FOKUS, Berlin, Germany*

Keywords: Internet of Things, TOSCA, Application Deployment, Device Software.

Abstract: The automated setup of Internet of Things environments is a major challenge due to the heterogeneous nature of the involved physical components (i.e., devices, sensors, actuators). In general, IoT environments consist of (i) physical hardware components, (ii) IoT middlewares that bind the hardware to the digital world, and (iii) IoT applications that interact with the physical devices through the middlewares (e.g., for monitoring). Setting up each of these requires sophisticated means for software deployment. In this paper, we enable such a means by introducing an approach for automated deployment of entire IoT environments using the Topology and Orchestration Specification for Cloud Applications standard. Based on topology models, all components involved in the IoT environment (devices, IoT middlewares, applications) can be set up automatically. Moreover, to enable interchangeability of IoT middlewares, we show how they can be used as a service to deploy them individually and on-demand for separate use cases. This enables provisioning whole IoT environments out-of-the-box. To evaluate the approach, we present three case studies giving insights in the technical details.

1 INTRODUCTION

The Internet of Things (IoT) paradigm has received great attention in the last years. It relies on the interconnection and cooperation of so-called *smart devices*, attached with sensors and actuators, that are able to sense the state of an environment and to adapt it through their actuators (Atzori et al., 2010; Gubbi et al., 2013). Such smart devices provide the foundation for the existence of a multitude of so-called *IoT applications*. An example of such an IoT application is a *smart home*, which is able to automatically regulate its temperature based on sensor data. Using such IoT applications requires setting up the whole IoT environment consisting of smart devices, middlewares to bind the devices to IoT applications, and the IoT applications themselves. However, the setup of such IoT environments comes with major challenges. First, physical hardware components (i.e., devices, sensors, actuators) are highly heterogeneous since they employ different technologies, protocols, and data models. Second, heterogeneity emerges among the many existing IoT middlewares, which are employed to bind the physical hardware to higher-level IoT applications, abstracting the complexity of the devices (Mineraud et al., 2016). Third, although the connection between an IoT application and the IoT middleware is typically hard wired, an IoT application might require different middlewares to fulfill its goals due to different use cases and business constraints. Therefore, an important aspect for the setup of IoT environment is also the interchangeability of different IoT middlewares. A high interchangeability enables building more flexible IoT applications, tailor-made for specific use cases. Furthermore, it is important that the setup of IoT environments is done in an automated manner because a manual setup is time-consuming and error-prone due to the high complexity and dynamic in the IoT.

In this paper, we show how the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) standard (OASIS, 2013) can be used to fully automate the setup of entire IoT environments including (i) physical hardware components, (ii) IoT middlewares that bind the hardware to the digital world, and (iii) IoT applications that interact with the physical devices through the middlewares. Using our approach, we can set up entire IoT environments *out-of-the-box*, i.e., once the IoT environment is modeled in TOSCA,

Silva, A., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B. and Steinke, R.
Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments.
DOI: 10.5220/0006243303580367
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 330-339
ISBN: 978-989-758-243-1

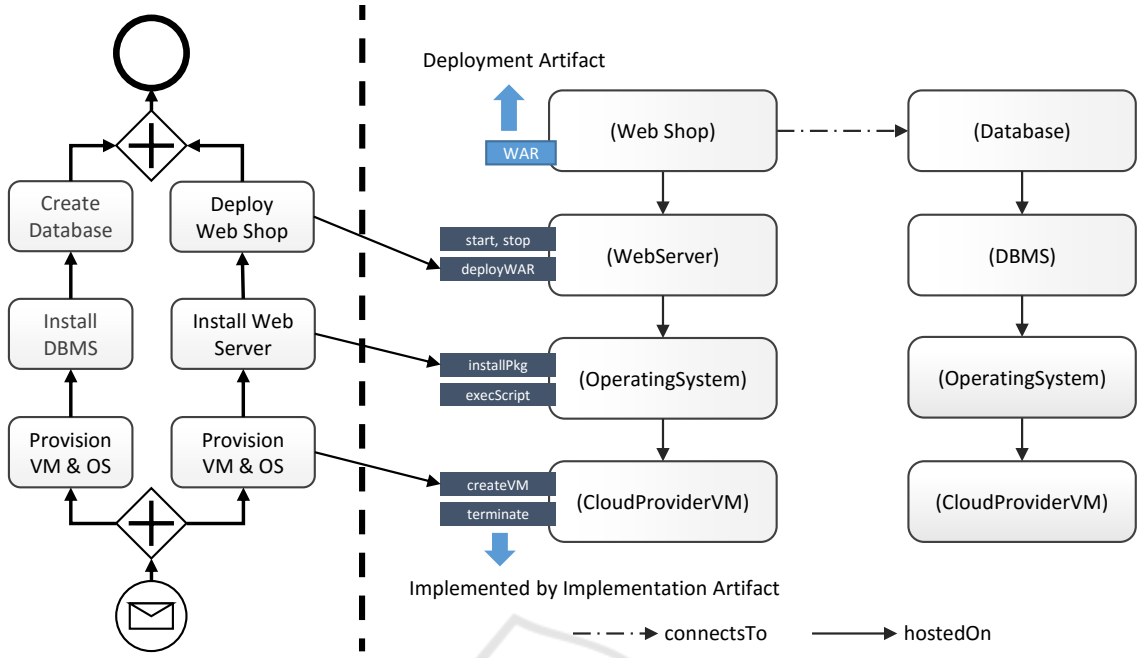

Figure 1: left: example of a TOSCA build plan, right: the corresponding TOSCA Topology Template to set up a web shop.

this model can be reused to set up the IoT environment without further configuration or modifications in the model. In particular, we present how the generic TOSCA concepts can be used for modeling IoT scenarios, their technologies and how heterogeneous IoT middleware systems can be deployed, wired, and exchanged by a standard-based TOSCA runtime environment. Moreover, the paper shows how IoT middleware can be used as a service and how it can be deployed individually and on-demand for separate use cases. We additionally present three case studies giving insights in the technical details of our approach.

The remainder of this paper is as follows: Sect. 2 introduces the background of our approach. Sect. 3 presents how to set up entire IoT environments out-of-the-box using TOSCA. In Sect. 4, we evaluate our approach based on three case studies. Finally, Sect. 5 describes related work and Sect. 6 gives a summary of the paper and an outlook on future work.

2 BACKGROUND

Cloud computing is a recently emerged paradigm for hosting and delivering services over the Internet (Leymann et al., 2016; Zhang et al., 2010), which has been increasingly employed together with the IoT paradigm in order to provide IoT environments with properties such as scalability and interoperability. Cloud computing can enable a rapid setup and integration of new physical components and IoT applications, while maintaining low-costs for the deployment of entire IoT environments (Botta et al., 2016).

The OASIS standard TOSCA enables modeling, provisioning, and management of cloud applications (Binz et al., 2014). As indicated by its name, the Topology and Orchestration Specification for Cloud Applications consists of two parts: (i) the topology and (ii) the orchestration of cloud applications. The topology describes the structure of the application, i.e., its software, platform, and infrastructure components. Consequently, TOSCA unifies the paradigms software-as-a-service, platform-as-a-service, and infrastructure-as-a-service. In these topologies – called *Topology Templates* in TOSCA – software components (e.g., databases, application servers) are represented as so-called *Node Templates*, and their connections, e.g., that one component is hosted on another component, as *Relationship Templates*. Each template has a type. Three Relationship Types are used in our approach: (i) the Relationship Type *hostedOn*, (ii) the Relationship Type *connectsTo*, describing a communication channel between two software components, e.g., to access a database, and (iii) the Relationship Type *dependsOn*, describing that one software component depends on another one (e.g., software packages, libraries), meaning that it cannot be operated without it. Node and Relationship Templates can be attached with properties (e.g., user credentials). *Implementation Artifacts* (IA) and *Deployment Artifacts* (DA) can be linked to Node Types and Relationship Types. Implementation

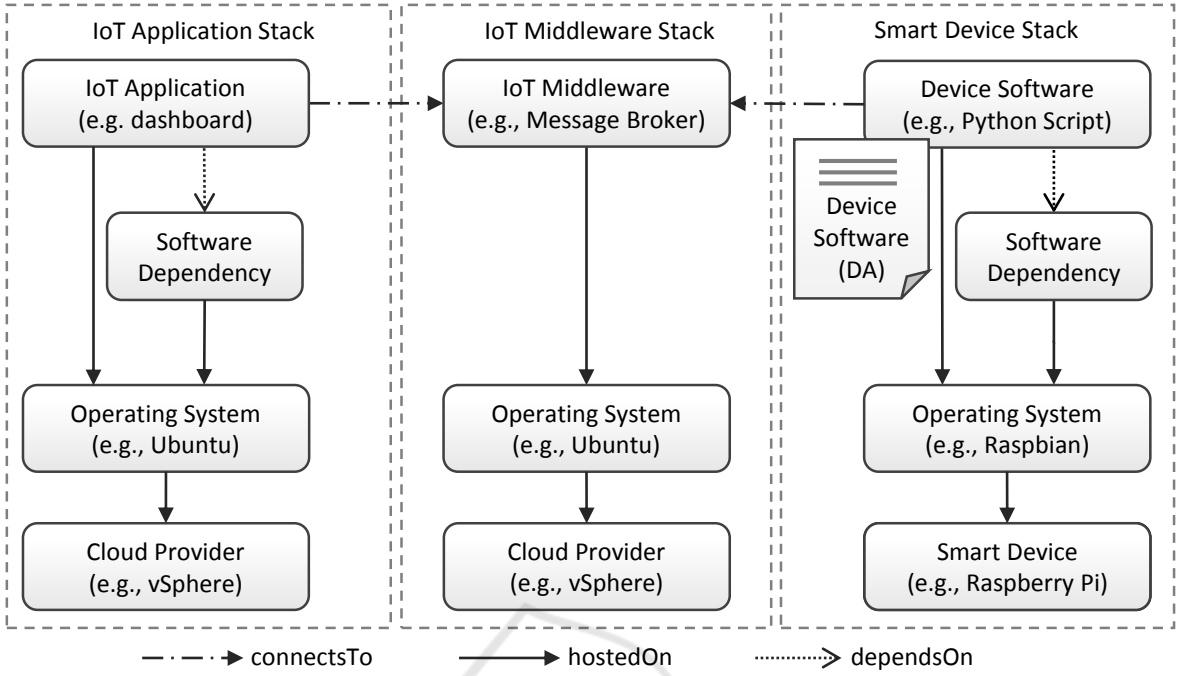

Figure 2: Abstract topology model for IoT environments.

Artifacts contain the logic, e.g., a piece of code, to configure, install, start, or uninstall a component. Deployment Artifacts in contrast represent binary files, such as installers, images or JAR files, that are required to properly provision a component.

The orchestration part of TOSCA describes all actions necessary to provision, manage, and deprovision a cloud application based on its topology model. TOSCA supports two approaches for application provisioning: (i) an imperative approach, and (ii) a declarative approach. The imperative approach requires defining so-called *Build Plans*, which describe the concrete order of steps that need to be conducted to set up the components. The declarative approach only requires defining the topology model. The corresponding TOSCA runtime consequently provisions the application by itself. However, in this manner, only components can be set up that are known to the runtime environment (Breitenbücher et al., 2015). The TOSCA runtime *OpenTOSCA* (Binz et al., 2013) – used for our prototypical implementation – combines the declarative and imperative approaches through the generation of Build Plans (Breitenbücher et al., 2014). Figure 1 depicts an exemplary TOSCA Topology Template and its corresponding Build Plan.

Device software is defined as the logic responsible for extracting data from sensors and sending it to IoT middlewares, and for invoking attached actuators (Guth et al., 2016). Usually, this device software is implemented using lightweight scripts (e.g., Python-based) due to the devices' limited resources. These scripts are able to read sensor values and invoke actuators through the hardware interface (e.g., GPIO). The device software represents the bridge between the hardware components and the IoT middlewares. Through the middleware, IoT applications can receive sensor data as well as send control commands to actuators. Hence, the middleware serves as abstraction layer between the IoT applications and the device software.

3 IoT OUT OF THE BOX USING TOSCA

In this section, we present our approach for automated setup of entire IoT environments including the device software, the IoT middleware, and the IoT application. To explain our approach, we use an IoT scenario composed of: (i) a dashboard as the IoT application, which monitors the environment with all its smart devices, sensors, and actuators, (ii) a message broker as the IoT integration middleware, and (iii) a Raspberry Pi as a smart device. Figure 2 depicts the TOSCA topology model of this scenario. The topology for our scenario contains three different stacks, whereas each stack represents a different part of the IoT environment. The stacks are: (i) the IoT application stack, (ii) the IoT middleware stack, and (iii) the smart device stack. The setup of these stacks is very similar. At the bottom,

the hardware resources are modeled. These resources include physical hardware devices, such as a Raspberry Pis, personal computers, and virtual machines provided by a cloud provider. On the level above, the corresponding operating system is modeled and connected via a *hostedOn* relation. This means, that the operating system is hosted on the corresponding resource. On the top level of a stack, the actual components are modeled, i.e., the device software, which is responsible for reading sensor values and invoking actuators, the IoT middleware, which serves as bridge between devices and IoT applications, and, finally, the IoT application itself. Dependencies of Node Templates, such as required libraries, packages, or other software can also be modeled in TOSCA using the *dependsOn* relation. The logic to set up the software represented by the Node Templates is attached using Implementation Artifacts. Furthermore, monolithic elements, such as installers or scripts, are attached as Deployment Artifacts. After the topology of this IoT scenario is modeled, it can be used as input for a TOSCA runtime engine. The engine consequently sets up the IoT environment automatically. As a result, the Raspberry Pi, which is connected to sensors and actuators, is able to publish sensor values to the IoT middleware, which allows the dashboard application to access the data and to invoke the actuators.

In the following, we describe the modeling of the devices, the middlewares, and the IoT applications in detail. We assume the generic imperative provisioning approach, meaning that an explicit Build Plan needs to be provided to set the components up (cf. Section 2).

3.1 Modeling Devices and Device Software

This section describes how to model smart devices, their corresponding device software, and all corresponding dependencies using TOSCA topology models. As described above and depicted in Figure 2 on the right, the stack to deploy an smart device typically consists of three Node Templates, which correspond to three layers: the hardware resources (i.e., the smart device), the operating system, and the device software. To be able to model this stack in the topology model, the corresponding TOSCA Node Types need to be provided. The Node Type for the device's Node Template, modeled on the bottom layer of the stack, requires a set of properties. These properties comprise, e.g., the type of the device, its MAC address, and hardware capabilities such as the computing resources, available main memory, and so on. Implementation Artifacts and Deployment Artifacts are not required to create this Node Type because it represents the actual hardware. All involved software will be set up on the operating system and not on the device itself.

The Node Type for the operating system is more complex, because, additionally to a set of properties, it requires Implementation Artifacts and Deployment Artifacts. The properties comprise the type of the operating system, the IP address that was assigned in the network to access it, provided interfaces and ports (e.g., SSH), and capabilities regarding supported software (e.g., 32 bit software). Furthermore, the installer and all corresponding dependencies for the operating system need to be attached as Deployment Artifacts and Implementation Artifacts that invoke the installer and configure the operating system.

Finally, the Node Type for the device software is attached with a Deployment Artifact containing the logic to connect to the sensors and actuators as well as to the middleware, e.g., a script. Furthermore, a certain amount of properties needs to be provided that is required by this Deployment Artifact. For example, these properties comprise the pin set of sensors and actuators, or in which interval sensor values are gathered. Furthermore, an Implementation Artifact needs to be provided that starts the Deployment Artifact as a service of the operating system. Often, the device software additionally requires specific libraries, packages, or software, to be executable. These dependencies can also be modeled as Node Templates and need to be connected to the device software Node Template using the *dependsOn* Relationship Template (cf. Figure 2). This Relationship Template guarantees that all existing dependencies are set up first to avoid errors on execution of the device software. The Node Types of all dependencies need to be created accordingly with all their properties, IAs, and DAs.

Once all Node Types are provided, the above described topology stack can be modeled, e.g., through graphical modeling using the tool Winery (Kopp et al., 2013). Once the topology is handed over to the TOSCA runtime for execution, the operating system can be automatically installed, the device software is deployed on it and is being started.

3.2 Modeling IoT Middleware

In this section, we describe how to model IoT middleware using TOSCA. The middleware can be modeled in two different manners depending on whether it is provided as a service, e.g., by a cloud provider, or whether it is set up on an own infrastructure.

The more complicated way is setting up the IoT middleware on an own infrastructure because it requires modeling the infrastructure and platform components (e.g., virtual machines, web servers). As de-

picted in Figure 2 in the middle, the required stack to set up the middleware consists of three Node Templates. Similar to the modeling of smart devices, these Node Templates represent the hardware resources, the operating system, and the IoT middleware itself. However, there are differences because the underlying runtime can be very heterogeneous and is not dependent on a certain kind of device. More precisely, all kinds of hardware can be used to set up the IoT middleware. This hardware comprises for example, virtual machines hosted by a cloud provider, non-virtualized web servers, personal computers, or even smart devices. The great advantage of TOSCA is that these runtimes can be easily interchanged for different scenarios. For example, a cloud provider can be chosen based on different criteria such as costs, or how secure data is stored. Because of the high heterogeneity of the underlying hardware, the creation of the corresponding Node Types differs greatly. Because of that, we describe the creation of an exemplary Node Type which represents a virtual machine hosted on a cloud provider. This Node Type requires multiple properties defining the type of the virtual machine, and the virtual resources, such as CPUs, main memory, or disc space.

On top of this virtual machine, the operating system is hosted. As described in Sect. 3.1, the operating system Node Type is provided with the mentioned properties, IAs, and DAs. This Node Type is quite similar to the operating system Node Type used to model the smart devices and, thus, is not described again.

The IoT middleware itself is also represented by a Node Template, which is connected to the operating system via the *hostedOn* Relationship Template. IoT middlewares are also very heterogeneous, however, all corresponding Node Types require properties for their configuration, IAs that contain the logic to set up, and DAs, e.g., for the installers. An exemplary modeling of different IoT middlewares is described in Section 4.

If the IoT middleware is not hosted on an own infrastructure but is provided as an external service (e.g., hosted by an external cloud provider), the model differs greatly. In this case, the IoT middleware stack only consists of the IoT middleware Node Template, because its infrastructure is hosted externally. This means that only a single Node Type needs to be created, which can be kept very simple because no IAs and DAs are necessary due to the service being set up on a remote host. Usually, certain properties are required, such as credentials to authenticate at the cloud provider and specific properties required by the IoT middleware service, e.g., the payment plan to be used.

3.3 Modeling IoT Applications

In this section, we describe how IoT applications can be modeled using TOSCA. The Node Types for the underlying resource and the operating system of the stack are the same as the ones used for the IoT middleware. Consequently, they do not need to be described. Of course, the IoT application could also be hosted by an external service provider. In this case, modeling the infrastructure is not necessary, as described before. The Node Type for the IoT application itself highly depends on the kind of application. At least the IAs and DAs to set it up are required. Furthermore, depending on the application, a specific set of properties needs to be provided.

Once all three stacks are modeled as described in the previous sections, and as depicted in Figure 2, the whole IoT environment can be set up by using the resulting topology as input for a corresponding TOSCA engine. In the following, we will validate the described approach based on two case studies.

4 VALIDATION: THREE CASE STUDIES

In recent years, a large amount of IoT middlewares (Corici et al., 2012; Ramparany et al., 2014; Alaya et al., 2014; Mineraud et al., 2016) have emerged such as Eclipse Mosquitto[1], FIWARE[2], and OpenMTC[3]. However, these middlewares do not offer a homogeneous way of communication among smart devices and applications. Different protocols, such as MQTT, HTTP, and CoAP, are employed, as well as different interaction paradigms, such as publish/subscribe and request/response interaction models. In this section, we validate our approach by modeling and deploying an IoT environment based on three different IoT middlewares: (i) Eclipse Mosquitto, (ii) FIWARE Orion Context Broker, and (iii) the OpenMTC platform. In the following, we assume that the physical deployment of the involved devices has already taken place.

Eclipse Mosquitto is an open-source message broker implementing the OASIS standard MQTT[4], a lightweight publish/subscribe messaging protocol. Mosquitto is a topic-based message broker to which subscribers register their interest by subscribing to specific *topics* in order to get notified when publishers send messages to these topics. The *Orion Context*

[1] https://www.mosquitto.org/ [2] https://www.fiware.org/
[3] http://www.open-mtc.org/ [4] http://www.mqtt.org/

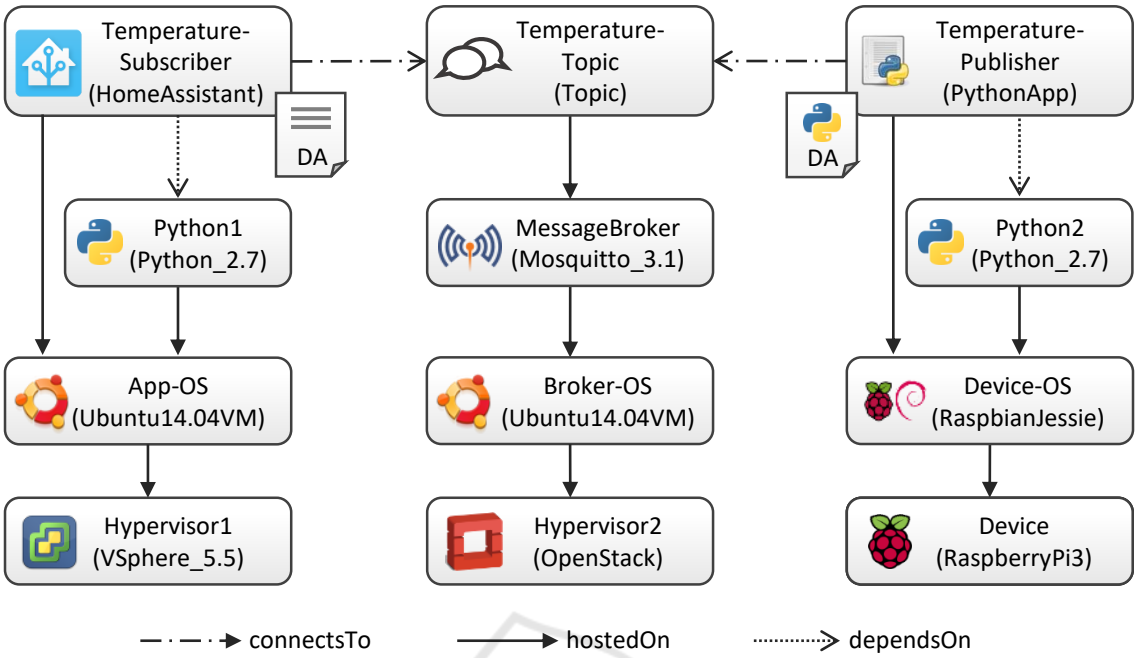

Figure 3: TOSCA topology model of an IoT Application and Eclipse Mosquitto.

Broker[5] is an implementation of the *Publish/Subscribe Context Broker Generic Enabler* developed as part of the FIWARE platform. Through its REST API, the Orion Context Broker allows the registration of so-called context elements, which can be updated by context producers. Furthermore, context consumers can either query these context elements or subscribe to them in order to be notified when they are updated. The *OpenMTC platform* is an implementation of the OneM2M standard[6], which intends to support machine-to-machine (M2M) communication for applications in a simplified way. It provides a REST API and uses the CRUD principle for the resources. It has a generic request/response model that can be mapped on different transport protocols, e.g., HTTP, CoAP and MQTT. The provided functionality includes registration of applications, discovery of resources, subscription to new data, simplified addressing of resources, scheduled communication and more.

In the following, we present the topology model for the IoT environment based on Eclipse Mosquitto. After that, we show the minimal changes necessary in the topology to exchange the middleware, i.e., to use the Orion Context Broker and the OpenMTC platform instead of Mosquitto.

[5] https://www.github.com/telefonicaid/fiware-orion/

[6] http://www.onem2m.org/

4.1 Topology Model based on Eclipse Mosquitto

Figure 3 depicts the TOSCA topology model, containing the necessary components and relationships for our IoT scenario based on Eclipse Mosquitto. The stack in the middle provides the infrastructure for Eclipse Mosquitto, i.e., the Node Types **OpenStack** and **Ubuntu14.04VM**. The **Mosquitto_3.1** Node Type contains shell scripts that can be executed in Linux as Implementation Artifacts. They are responsible for installing, configuring and starting a Mosquitto instance onto the Ubuntu Virtual Machine hosting it. As property, the **Topic** Node Type has the *topic name* to which the device software publishes sensor data and to which the IoT application subscribes. The configuration of topics differs in the many existing middlewares. In our approach, this issue is addressed by attaching the middleware-specific logic as an Implementation Artifact to the Topic Node Template. Mosquitto, however, does not require the topics to be pre-configured, i.e., topics are dynamically created by publishing. For this reason, the Topic Node Template in this case does not contain an Implementation Artifact and only has a single property.

The stack on the right models the infrastructure for a python-based application. The **RaspberryPi3** Node Type represents a physical Raspberry Pi 3, to which sensors and actuators are attached. The **RaspbianJessie** Node Type corresponds to the Raspbian

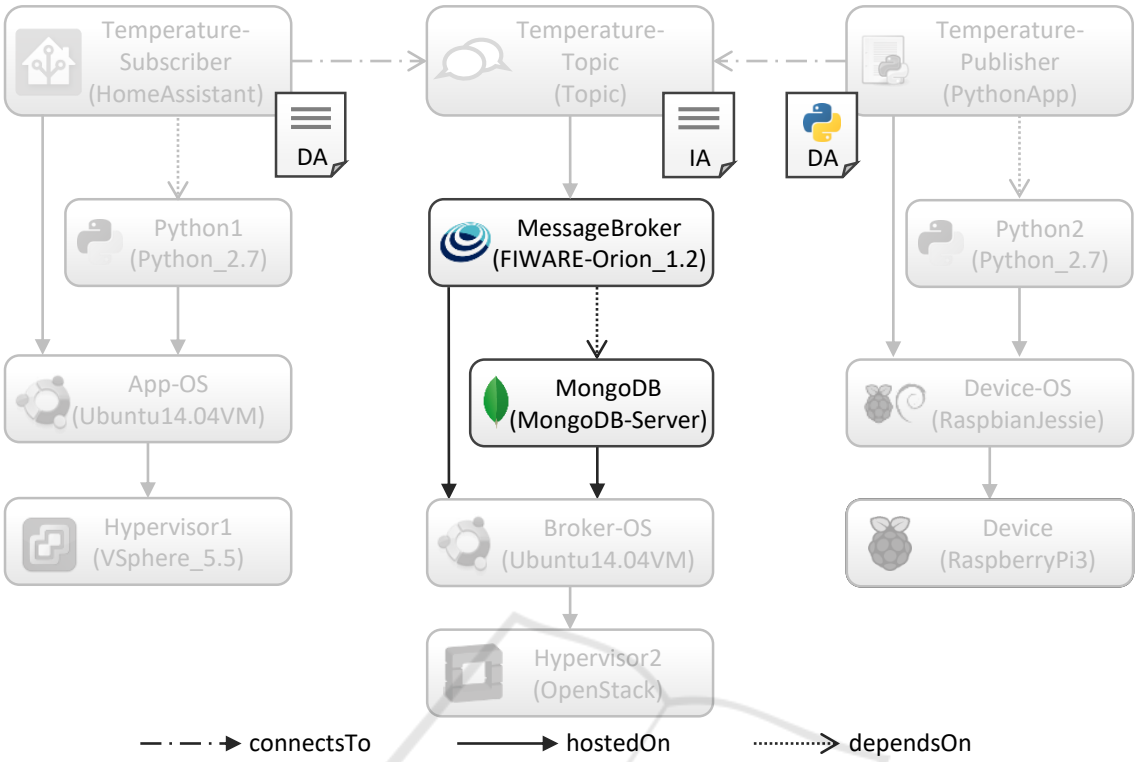

Figure 4: TOSCA topology model of an IoT Application and FIWARE Orion Context Broker.

operating system installed on the Raspberry Pi. The **PythonApp** Node Type models a generic Python-based application with its implementation, a Python script, attached as Deployment Artifact. This script periodically reads values of a temperature sensor and sends them to Mosquitto by using the open-source MQTT client library *Eclipse Paho*[7]. To connect to Mosquitto, the MQTT client needs (i) the *IP address* of the Virtual Machine hosting Mosquitto, and (ii) the *topic name* to which sensor data should be published. This information is provided as a configuration file, which is generated by the Implementation Artifact of the *connectsTo* relationship and copied to the Raspberry Pi during the deployment.

Finally, the stack on the left provides the infrastructure, i.e., the Node Types **VSphere_5.5** and **Ubuntu14.04VM**, for the open-source, Python-based IoT Application *HomeAssistant*[8]. The **HomeAssistant** Node Type contains shell scripts as Implementation Artifacts for installing, configuring and starting a HomeAssistant instance in the Ubuntu Virtual Machine hosting it. The HomeAssistant instance subscribes to Mosquitto in order to receive the temperature values and shows them in its graphical dashboard. For that, it needs to know the IP address of the Virtual Machine hosting Mosquitto and the topic name to subscribe to. During the deployment, this information is added to the HomeAssistant configuration file, which is attached to a HomeAssistant Node Template as a Deployment Artifact.

4.2 Topology Model based on FIWARE Orion Context Broker

Figure 4 depicts the TOSCA topology model of our IoT scenario based on the FIWARE Orion Context Broker. The highlighted components correspond to the changes in the topology model in Figure 3. Only the **Mosquitto_3.1** Node Template needs to be exchanged by the **FIWARE-Orion_1.2** Node Type, and its dependency, the **MongoDB-Server** Node Type. The **FIWARE-Orion_1.2** Node Type contains the shell scripts as Implementation Artifacts responsible for installing, configuring and starting the Orion Context Broker. In contrast to Mosquitto, Orion requires the configuration of topics. For that, we attach an Implementation Artifact to the **Topic** Node Template, which contains Orion-specific logic for the configuration of topics. Finally, since Orion does not use MQTT but instead provides a REST API to publish and query sensor values, the **Deployment Artifacts** of the **PythonApp** and **HomeAssistant** Node Templates are required to

[7] https://www.eclipse.org/paho/
[8] https://www.home-assistant.io/

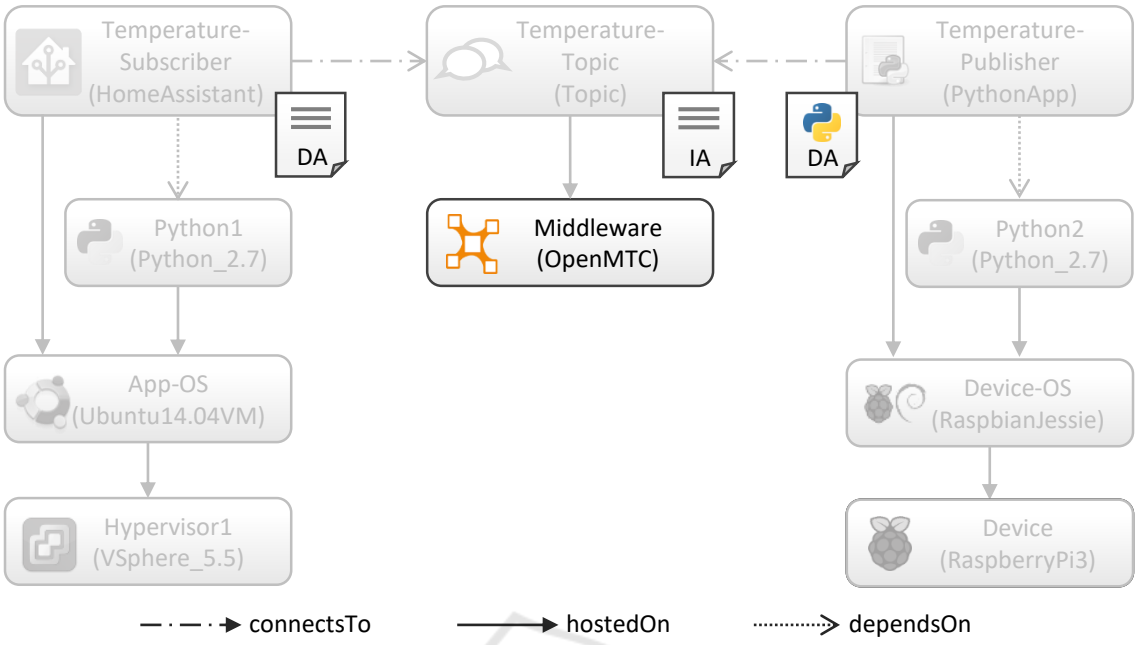

Figure 5: TOSCA topology model of an IoT Application and the OpenMTC platform as a service.

be exchanged.

4.3 Topology Model based on the OpenMTC Platform

Besides deploying an IoT middleware on an own infrastructure (cf. Sect. 4.1 and 4.2), it is possible to model a middleware, which is available on an external infrastructure (e.g., an external cloud), as a service. Figure 5 depicts the TOSCA topology model, containing the necessary components and relationships for our IoT scenario based on the OpenMTC platform, which is modeled as a service. The highlighted components correspond to the changes in the topology model in Figure 3. Only the **Mosquitto_3.1** Node Template needs to be exchanged by the **OpenMTC** Node Template. In this case, we do not need to provide the infrastructure Node Types for OpenMTC. The OpenMTC Node Type solely contains the properties indicating the IP address of the service's host and the required credentials. An IA containing OpenMTC-specific logic for the configuration of topic is attached to the **Topic** Node Template. Finally, the **Deployment Artifacts** of the **PythonApp** and **HomeAssistant** Node Templates are also exchanged to be able to publish and query sensor values.

We create the described Node Types and model the TOSCA topologies of Figs. 3 to 5 using the graphical modeling tool Winery. These topologies are handed over to the TOSCA runtime environment OpenTOSCA, which automatically sets up the modeled IoT environments out of the box, i.e., once we have these topologies, they can be reused to set up these environments without further configuration or modifications in the models.

5 RELATED WORK

This section describes the related work regarding software provisioning. Li et al. (2013) propose to employ the TOSCA standard to specify the basic components of IoT applications (e.g., gateways, controllers, etc.) and their configuration, in order to automate IoT application deployment in heterogeneous environments. Extensions of this work were presented by Vögler et al. (2015, 2016). The authors propose the deployment framework LEONORE for the deployment and execution of custom application logic directly on IoT gateways. However, for the framework to know the available IoT gateways for the provisioning, the IoT gateways must have a pre-installed local provisioning agent. This agent registers itself to the framework by providing its unique identifier and profile data of the gateway (e.g., MAC-address, instruction set, memory consumption). In contrast, our approach does not require any pre-installed components on the IoT-gateways (i.e., on the devices) beside the operating system because all the other necessary components are set up automatically.

Hur et al. (2015) propose a Semantic Service Description (SSD) ontology and a system architecture to automatically deploy smart devices to heterogeneous IoT integration middlewares, aiming to solve interoperability problems between them. Our paper also aims to solve these problems, however we propose a standard-based approach to automatically deploy smart devices to the IoT integration middlewares, and to deploy these middlewares as well.

Hirmer et al. (2016b,1016c) introduce an approach for automated binding of smart devices using a middleware called *Resource Management Platform* (RMP). The RMP enables an easy registration of smart devices and their binding through adapters. An adapter is a piece of code containing the logic to read sensor values of smart devices, send the sensor values to the RMP, and to invoke the devices' actuators. For the binding, adapter scripts are automatically deployed onto the devices or – if the device does not provide the necessary resources – on remote runtimes, e.g., web servers. An extension of the RMP, which uses TOSCA to deploy the adapters, is described by Hirmer et al. (2016a). In contrast to our work, Hirmer et al. concentrate on the binding of hardware devices whereas our approach focuses on the whole IoT environment including the middleware and the IoT applications themselves. Furthermore, in the approach by Hirmer et al., the deployment of the adapters is conducted based on predefined packages that already contain pre-modeled TOSCA Topology Templates. In this work, we show how these models can be created tailor-made for each use case scenario.

Clearly it is possible to employ other approaches (e.g., Ansible, Vagrant, Docker) instead of TOSCA for the automated deployment of IoT applications. However, in contrast to these approaches, TOSCA enables a generic approach based on topology models and a corresponding graphical notation (Breitenbücher et al., 2012). These topology models are highly adaptable in a way that software components can be easily interchanged. In other approaches, this requires a large adaptation effort, e.g., when editing Vagrant scripts. In addition, through the concepts of Node and Relationship Types, TOSCA offers a high reusability for the deployment of software.

We (Franco da Silva et al., 2016) implemented a prototype that realizes the first case scenario – a topology model based on Eclipse Mosquitto – and can be seen as a proof-of-concept for the approach introduced in this paper.

6 SUMMARY AND FUTURE WORK

In this paper, we present an approach that employs the *Topology and Orchestration Specification for Cloud Applications* (TOSCA) standard to set up entire IoT environments automatically. Using our approach, it is possible to set up entire Internet of Things environments *out-of-the-box*, i.e., once the IoT environment is modeled using TOSCA, this model can be reused without further configuration or modifications in the model. Furthermore, we give an overview of the generic TOSCA concepts, and how they can be used to model concrete IoT scenarios based on different technologies. We also show how heterogeneous IoT middleware systems can be deployed, wired, and exchanged by a standard-based TOSCA runtime environment. Moreover, the paper describes how an IoT middleware can be used as a service available on an external infrastructure, and how it can be deployed individually and on-demand for separate use cases. Finally, we present technical details of our approach by providing three detailed case studies based on the IoT middlewares Eclipse Mosquitto, FIWARE Orion Context Broker and OpenMTC platform. As future work, we aim to develop more TOSCA Node Types related to IoT environments, in order to support the automated deployment of a wider range of IoT scenarios.

ACKNOWLEDGMENTS

This work was funded by the BMWi project SmartOrchestra (01MD16001F).

REFERENCES

Alaya, M. B., Banouar, Y., Monteil, T., Chassot, C., and Drira, K. (2014). OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32:1079–1086.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *11th International Conference on Service-Oriented Computing*, LNCS. Springer.

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, pages 527–549. Advanced Web Services. Springer.

Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 56:684–700.

Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., and Wettinger, J. (2014). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *Proceedings of the IEEE International Conference on Cloud Engineering*, pages 87–96. IEEE.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Schumm, D. (2012). *Vino4TOSCA: A Visual Notation for Application Topologies Based on TOSCA*, pages 416–424. Springer Berlin Heidelberg, Berlin, Heidelberg.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2015). A Modelling Concept to Integrate Declarative and Imperative Cloud Application Provisioning Technologies. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, pages 487–496. SciTePress.

Corici, M., Coskun, H., Elmangoush, A., Kurniawan, A., Mao, T., Magedanz, T., and Wahle, S. (2012). OpenMTC: Prototyping Machine Type communication in carrier grade operator networks. In *2012 IEEE Globecom Workshops*, pages 1735–1740. IEEE.

Franco da Silva, A. C., Breitenbücher, U., Képes, K., Kopp, O., and Leymann, F. (2016). OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker. In *Proceedings of the 6th International Conference on the Internet of Things (IoT)*, pages 181–182, Stuttgart. ACM.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., and Reinfurt, L. (2016). Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In *CLOSER*.

Hirmer, P., Franco da Silva, A. C., Wieland, M., Breitenbücher, U., Kálmán, K., and Mitschang, B. (2016a). Automating the Provisioning and Configuration of Devices in the Internet of Things. *Complex Systems Informatics and Modeling Quarterly*. to appear.

Hirmer, P., Wieland, M., Breitenbücher, U., and Mitschang, B. (2016b). Automated Sensor Registration, Binding and Sensor Data Provisioning. In *Proceedings of 28th International Conference on Advanced Information Systems Engineering*, volume 1612 of *CEUR Workshop Proceedings*, pages 81–88. CEUR-WS.org.

Hirmer, P., Wieland, M., Breitenbücher, U., and Mitschang, B. (2016c). Dynamic Ontology-based Sensor Binding. In *Proceedings of 20th East European Conference on Advances in Databases and Information Systems*, volume 9809 of *Information Systems and Applications, incl. Internet/Web, and HCI*, pages 323–337. Springer.

Hur, K., Chun, S., Jin, X., and Lee, K.-H. (2015). Towards a semantic model for automated deployment of iot services across platforms. In *Proceedings of the 2015 IEEE World Congress on Services*, SERVICES '15, pages 17–20. IEEE.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery - A Modeling Tool for TOSCA-based Cloud Applications. In *Proceedings of 11th International Conference on Service-Oriented Computing*, volume 8274 of *LNCS*, pages 700–704. Springer Berlin Heidelberg.

Leymann, F., Fehling, C., Wagner, S., and Wettinger, J. (2016). Native Cloud Applications: Why Virtual Machines, Images and Containers Miss the Point! In *Proceedings of the 6th International Conference on Cloud Computing and Service Science*, pages 7–15. SciTePress.

Li, F., Vögler, M., Claeßens, M., and Dustdar, S. (2013). Towards Automated IoT Application Deployment by a Cloud-Based Approach. In *Proceedings of the 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, SOCA '13, pages 61–68. IEEE.

Mineraud, J., Mazhelis, O., Su, X., and Tarkoma, S. (2016). A gap analysis of Internet-of-Things platforms. *Computer Communications*, 89 - 90:5–16.

OASIS (2013). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*.

Ramparany, F., Galan Marquez, F., Soriano, J., and Elsaleh, T. (2014). Handling smart environment devices, data and services at the semantic level with the FI-WARE core platform. In *2014 IEEE International Conference on Big Data*, pages 14–20. IEEE.

Vögler, M., Schleicher, J. M., Inzinger, C., and Dustdar, S. (2016). A Scalable Framework for Provisioning Large-Scale IoT Deployments. *ACM Transactions on Internet Technology (TOIT)*, 16(2):11:1–11:20.

Vögler, M., Schleicher, J. M., Inzinger, C., Nastic, S., Sehic, S., and Dustdar, S. (2015). LEONORE–Large-Scale Provisioning of Resource-Constrained IoT Deployments. In *Proceedings of the 2015 IEEE Symposium on Service-Oriented System Engineering*, pages 78–87. IEEE.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.