Institute of Architecture of Application Systems

Declarative vs. Imperative: How to Model the Automated Deployment of IoT Applications?

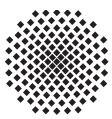
Uwe Breitenbücher, Kálmán Képes, Frank Leymann, Michael Wurster

Institute of Architecture of Application Systems,
University of Stuttgart, Germany,
[lastname]@iaas.uni-stuttgart.de

BibTeX

```
@inproceedings{Breitenbuecher2017,
    author    = {Breitenb\"{u}cher, Uwe and K\'{e}pes, K\'{a}lm \'{a}n and
                 Leymann, Frank and Wurster, Michael},
    booktitle = {Proceedings of the 11\textsuperscript{th} Advanced Summer
                 School on Service Oriented Computing},
    pages     = {18--27},
    publisher = {IBM Research Division},
    title     = {{Declarative vs. Imperative: How to Model the Automated
                 Deployment of IoT Applications?}},
    year      = {2017}
}
```

The full version of this publication has been presented as a poster at the Advanced Summer School on Service Oriented Computing (SummerSOC 2017).
http://www.summersoc.eu



Universität Stuttgart
Germany

Declarative vs. Imperative: How to Model the Automated Deployment of IoT Applications?

Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Michael Wurster

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
[firstname].[lastname]@iaas.uni-stuttgart.de

Abstract. The Internet of Things (IoT) has become an increasingly important domain, which more and more requires application deployment automation as manual deployment is time-consuming, error-prone, and costly. However, the variety of available deployment automation systems also increases the complexity of selecting the most appropriate technology. In this paper, we discuss how the deployment of complex composite IoT applications can be automated and discuss the conceptual strengths and weaknesses of declarative and imperative deployment modelling.

Keywords: Deployment Modelling, Declarative, Imperative, TOSCA

1 Introduction

The Internet of Things (IoT) has become an increasingly important domain as more and more IoT applications influence our life. IoT applications typically consist of physical devices that are connected to software, which is often deployed in cloud environments. Thus, IoT applications are *cyber-physical systems* that consist of one or more physical parts and virtual parts. However, especially the combination of physical devices and cloud-based software deployments quickly leads to complex architectures as multiple physical as well as virtual components have to be deployed, configured, and wired. As a result, the deployment of such *complex composite IoT applications* is a serious challenge that requires immense technical expertise [28]: Physical devices must be installed, scripts deployed, sensors configured, and backend software provisioned. Due to this complexity manually deploying IoT applications is time-consuming, error-prone, and costly. However, although various deployment approaches exist to automate the deployment of IoT applications, the variety of available technologies makes it very difficult to select the most appropriate technology for a certain use case.

In this paper, we discuss how the deployment of such complex composite IoT applications can be automated by analyzing the conceptual strengths and weaknesses of declarative and imperative deployment approaches based on our experiences we gained in the BMWi project SmartOrchestra[1]. Thus, we do not focus on individual technologies but on the general *deployment modelling concepts* and discuss their suitability for different IoT deployment use cases.

[1] http://www.smartorchestra.de

2 Fundamentals & Related Work

The declarative deployment modelling approach is based on *declarative deployment models* that describe the structure of the application to be deployed including all components, their configuration, and their relationships. A declarative deployment model is consumed by a declarative deployment system that interprets the model, derives all required technical tasks to deploy the described application, and executes these tasks [11]. Thus, declarative deployment models specify only *what* has to be deployed, while the actual deployment logic gets calculated by the deployment system and is, therefore, not contained in the model. There are multiple scientific works that support the declarative approach for modelling the deployment of applications, for example, by Eilam et al. [9], Maghraoui et al. [10], Hewson et al. [13], and Breitenbücher et al. [6, 5]. Moreover, configuration management technologies such as Puppet [27] often support declarative deployment modelling, too. The *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [22, 21, 24, 3] is a standard that enables automating the deployment of cloud applications. TOSCA also supports the declarative approach as it provides a metamodel for modelling the topology of the application to be deployed including all components, their configurations, and their relationships.

In contrast, the imperative deployment modelling approach is based on *imperative deployment models* that describe the actual deployment logic to be executed in the form of a process [11]. Thus, imperative deployment models are process models that explicitly describe all technical deployment tasks to be executed, their order, and the data flow between these tasks. For application deployment and management automation, often the workflow technology [18] is used to describe the corresponding processes in an executable manner. For example, the approaches presented by Mietzner et al. [20], Bellavista et al. [1], Keller et al. [14], and Breitenbücher et al. [5, 8] are based on the workflow technology. Moreover, there are domain-specific extensions for workflow languages that focus on deployment, for example, BPMN4TOSCA [15, 17] or the approach presented by Weerasiri et al. [30]. In practice, low-level shell scripts are often used as well to describe software installation and configuration tasks in an imperative manner. Imperative deployment models are executed by a corresponding process engine, for example, a workflow engine, or by an imperative deployment system such as OpenTOSCA [2], which is an open-source runtime for TOSCA. Thus, TOSCA also supports the imperative deployment modelling approach by the concept of *Management Plans*, which are executable process models that can be used to automate the deployment of the modelled application.

We documented both deployment modelling approaches in our previous work [11] in the form of *Application Deployment Modelling Patterns*. This work also categorizes some available deployment automation technologies based on the two modelling approaches. In this paper, we discuss the general suitability of the two approaches for the deployment of complex composite IoT applications with respect to requirements of the IoT domain. We first describe the conceptual strengths and weaknesses of the declarative modelling approach in the next Section 3, which is followed by the imperative part discussed in Section 4.

3 Declarative IoT Deployment Modelling

In this section, we discuss the strengths and drawbacks of declarative deployment modelling with respect to the domain of IoT. To provide a conceptual overview, the discussion does not compare concrete technologies but discusses the general suitability of declarative IoT deployment modelling. Each following subsection discusses one certain strength or drawback. To support understanding and to better illustrate problems, we provide examples based on the TOSCA standard.

3.1 Creation and Comprehensibility of the Deployment Model

Declarative deployment models capture the structure of the system to be deployed as they describe the components that shall be deployed as well as their relationships, thus, the *topology* of the application. This directly reflects the application's structure developers have in mind during development, which provides an intuitive modelling approach and directly shows the final result. Moreover, especially for IoT deployments directly capturing the involved physical devices as well as their connections to software components eases the creation and understanding of the deployment model. In addition, for declarative deployment languages typically graphical modelling tools are available that enable a fast creation of the respective models. For example, the open-source TOSCA modelling tool *Winery* [16] supports graphically modelling TOSCA-based declarative deployment models based on the visual notation Vino4TOSCA [7].

3.2 Suitability of Declarative Technologies

Many IoT applications are composed of common components such as Raspberry Pis and IoT middlewares[2] such as the Mosquitto message broker. Moreover, typically standardized communication protocols, such as MQTT [23], are used that are explicitly defined regarding their technical details. As declarative deployment models are interpreted by the deployment system, the components to be provisioned and their relationships must be processable by the system. In a previous work [28], we have shown that automatically deploying IoT applications that are composed of such common components and protocols is possible based on declarative TOSCA models. However, if customization and application-specific deployment logic is required, the declarative approach reaches its limits as the system needs to interpret the declarative model that specifies only *what* has to be provisioned, but not *how*. To tackle this issue, declarative deployment technologies typically provide plug-points, which can be used to specify custom deployment logic for individual components. For example, the *TOSCA Lifecycle Interface* [21, 24] enables to provide an own install implementation for a certain type of component. However, this is typically limited to lifecycle operations and does not allow to customize the deployment arbitrarily [4]. Thus, the declarative modelling approach is mainly suited for common and non-complex deployments, but is limited regarding individual customizations and application-specific details.

[2] An overview of different IoT Integration Middlewares is provided by Guth et al. [12]

3.3 Required Technical Deployment Expertise

Declarative deployment models specify only *what* has to be deployed, but not *how* the deployment shall be executed. Therefore, modellers only need to specify the application's structure including the components, their wiring, and the desired component configurations. Thus, only little or even no technical expertise is required about the actual deployment execution: Neither scripting languages have to be understood nor API calls must be orchestrated, which is typically required for the deployment of complex systems [8]. Especially in the domain of IoT, this characteristic becomes of vital importance as deploying and configuring software on (remote) physical IoT devices is typically more complex than solely deploying software in cloud environments as more technical deployment and configuration tasks have to be executed [28]. The additional technical tasks range from, for example, connecting to (remote) physical devices for installing software to configuring gateways in order to establish the communication between devices and backend. The immense heterogeneity regarding IoT middleware systems [12] additionally increases this complexity. Therefore, only modelling the structure of the system to be deployed requires significantly less technical expertise than the imperative modelling approach, which has to specify all technical deployment tasks, remote communications with devices, API calls, script executions, etc.

3.4 Deployment Customization

Declarative deployment models are interpreted by the deployment system, which derives and executes the technical deployment tasks [11]. This interpretation is typically based on (i) known types of components and relationships and (ii) known management interfaces. For example, the OpenTOSCA deployment system provides a plug-in system for the deployment of different component types [5]. Many declarative technologies also support mechanisms to inject custom deployment logic into the model based on known management interfaces. For example, TOSCA not only provides a metamodel for declarative deployment models but also standardizes the *TOSCA Lifecycle Interface*, which defines the operations that are called during the deployment of a component, e.g., *install* and *start*. Moreover, TOSCA enables to provide own implementations for these operations on a per-component basis in the model. Thus, based on such interfaces, the deployment logic of a component can be influenced even if a declarative deployment technology is used. However, if more complex tasks must be executed that do not follow such predefined operations, the declarative approach reaches its limitations: The deployment can be customized only using such plug-points, but not in an arbitrary manner. For example, if two different components must be installed before both can be started, this cannot be realized using the TOSCA Lifecycle Interface. Especially in the domain of IoT this is a critical limitation as the configuration of physical devices often needs a special deployment execution order. For example, often two devices must be physically connected before software can be installed. Thus, the relationship between these two components must be established before software can be deployed on the devices, which breaks the typical deployment execution order of components and their relationships [5].

3.5 Integration of Human Tasks

IoT applications are cyber-physical systems and, thus, consist of one or more virtual parts and one or more physical parts. For automating the deployment of the virtual part, there are many declarative deployment technologies available that are capable of executing arbitrary deployment tasks, for example, creating virtual machines, installing Web-based applications, and instantiating a database on a Storage as a Service offering such as Amazon RDS[3]. All these deployment tasks have in common that they can be executed fully automatically without the need for human intervention as the components that have to be deployed as well as the components on which they have to be deployed can be accessed by software, e.g., via HTTP-based APIs of hypervisors and cloud service offerings or low-level communication protocols to access virtual machines. In contrast, deploying the physical part of IoT applications often requires humans, e.g., for installing devices, soldering sensors, etc. However, such *human tasks* are typically not natively supported by declarative modelling approaches and are, therefore, very hard to integrate into the available deployment systems and the corresponding models. For example, in TOSCA it would be possible to implement the *install* operation of a physical device component by a script that sends a message to a human to install the device. This means that typical workflow features such as staff resolution, work item management, and role management [18] would have to be re-implemented in such solutions as they are typically not supported natively by declarative deployment systems. However, the reliability and robustness of workflow management systems [18] cannot be achieved using such workarounds.

3.6 State-preserving and State-Changing Deployment Tasks

Deployment and management tasks can be abstractly classified into (i) *state-changing tasks* and (ii) *state-preserving tasks* [4]. State-changing task change the state of one or more components or relationships of the application, for example, modifying the HTTP port of a Webserver changes the state of the component. In contrast, state-preserving tasks do not change the state of one or more components or relationships, for example, exporting data from a database does not change the state or configuration of the database or of its stored data. Declarative models support state-changing deployment tasks natively: Components are transferred from state *uninstalled* to state *installed*, for example. However, this type of deployment models does not support state-preserving tasks very well as this kind of tasks cannot be modelled by specifying a component, a relationship, or a configuration [4]. Especially for deploying IoT applications this is a serious problem as state-preserving tasks are often required. For example, before connecting a physical device to the backend system, its physical functionality and also non-functional requirements such as its battery level may have to be checked. In particular, if a device has an actuator that triggers some physical action, a declarative model cannot specify that the physical functionality has to be verified manually after the successful installation of the device or its software.

[3] `https://aws.amazon.com/rds/`

4 Imperative IoT Deployment Modelling

In this section, we discuss the general strengths and drawbacks of imperative deployment modelling with respect to the domain of IoT. Where possible we refer to the declarative strengths and drawbacks to compare the both approaches.

4.1 Deployment Customization

Imperative deployment models are process models that specify a set of activities to be executed as well as their order and the control flow between them. Thus, they specify exactly *how* a deployment has to be executed [11]. This enables influencing the deployment execution arbitrarily as each detail can be described and customized in the process model, for example, the deployment order of components can be changed or application-specific customizations can be implemented by additional tasks. Thus, in contrast to the declarative approach, imperative deployment models support state-preserving as well as state-changing tasks. In particular, for deploying complex composite IoT applications also tasks that are hard to describe declaratively can be realized, for example, installing a physical device at a certain place or testing a device before connecting it to the backend. TOSCA also supports the imperative deployment modelling approach in the form of so-called *Management Plans*, which are executable process models that automate the execution of a certain management function for the application, e.g., its deployment. Thus, the TOSCA standard supports the declarative as well as the imperative approach for deploying applications, which therefore provides a suitable basis to choose the right modelling approach for a certain IoT deployment use case. In particular, there are previous works that show how the TOSCA standard can be used for IoT deployment automation [19, 28, 29].

4.2 Integration of Human Tasks

The integration of human tasks in the automated declarative deployment of an IoT application is hard to realize and may misuse concepts provided by the deployment technology (cf. Section 3.5). Using imperative deployment models this is much easier as especially many workflow languages and workflow management systems support the integration of human tasks in automatically executed processes [18]. For example, the Business Process Model and Notation (BPMN) [25] defines a task type for integrating manual human task executions in an overall automated workflow. Thus, this kind of deployment model is suited for IoT application deployments in which manual task executions by humans are required, for example, to install or configure physical devices at a certain place. However, the available standards-based domain-specific workflow extensions for application deployment such as BPMN4TOSCA [15, 17] currently do not support the integration of human IoT deployment tasks, which is therefore part of our future work.

4.3 Required Technical Deployment Expertise

Imperative deployment models describe each detail about the deployment tasks to be executed, for example, technical details of script invocations, API calls, and file transfers to virtual machines. In addition, also the control flow of these tasks must be specified as well as the data flow between them. Moreover, when deployment tasks shall be executed in parallel, this quickly leads to complex process models that must be developed and maintained carefully. Thus, manually creating imperative deployment models is a complex and technically error-prone challenge [8]. Especially when multiple deployment technologies must be orchestrated, for example, for multi cloud or hybrid cloud deployments, different API designs, data formats, invocation mechanisms, and security concepts of the different deployment technologies and provider APIs also increase the complexity of imperative deployment models [8]. IoT applications additionally increase this complexity as also physical devices must be considered in the deployment process, which often requires establishing connections to devices, transferring files, and executing scripts—all these tasks must be reflected in the imperative process model. As a result, the development and maintenance of complex imperative IoT deployment models requires immense technical deployment expertise of possibly multiple different deployment systems and APIs that have to be combined. Therefore, the imperative approach is complex, error-prone, and time-consuming [8]. The TOSCA standard enables to reduce the complexity of imperative deployment models as deployment logic can be hidden by the lifecycle operation implementations of the components (cf. Section 3.4). Thus, a deployment plan can invoke these operations, which wrap technical details. Moreover, in previous work [31], we also developed a TOSCA-based *management bus* that encapsulates deployment technologies such as Chef [26] from plans. As a result, plans only invoke this bus in a standardized manner to execute a deploying task using a certain technology without the need to care about the technical invocation details.

4.4 Comprehensibility of the Deployment Model

Declarative deployment models specify the structure of the application including all components and relationships. Especially when graphical, graph-based representations are used, declarative models provide a comprehensive overview on the application to be deployed (cf. Section 3.1). In contrast to this, imperative deployment models specify the process of the deployment. Thus, the final result is not immediately visible in these imperative models and must be derived by analyzing the modelled tasks, their semantics, and their order. This quickly gets complex if an IoT application is complex and consists of various physical as well as virtual components. Moreover, in IoT applications often multiple devices of the same kind are involved, for example, devices with temperature sensors in a smart home. Thus, in such scenarios not only the deployment tasks and possibly their labels in the process model must be analyzed, but also their parameters must be understood to recognize which component is affected by a certain task.

5 Conclusion and Future Work

In this paper, we discussed the suitability of the declarative and the imperative deployment modelling approaches for automating the deployment of IoT applications. We analyzed the major conceptual strengths and weaknesses of both approaches and mainly compared them in terms of complexity for modellers and the required technical expertise. The paper shows that the major drawbacks of each approach is solved by the other one, which leads to the conclusion, that a hybrid IoT deployment modelling approach is required: A declarative deployment model eases the specification of the desired deployment but is limited to common and non-complex deployments. Thus, transforming declarative models into imperative deployment models enables customizing the IoT application deployment arbitrarily as any additional task can be added and even human tasks and state-preserving tasks can be included in the generated process model. It has been already shown that this transformation is possible for software-based application deployments [10, 9, 5] and also that simple declarative IoT application deployment models can be transformed into imperative workflows [29]. Therefore, in future work, we focus on this transformation—especially on the integration of human tasks into the automated deployment process. This requires a detailed analysis of existing imperative deployment modelling languages such as BPMN4TOSCA and new concepts that support humans in executing such manual IoT deployment tasks. For example, to install a certain software via USB stick on a device that cannot be managed remotely due to missing connectivity to the network.

Acknowledgments. This work was partially funded by the project SmartOrchestra (01MD16001F) of the BMWi program Smart Service World.

References

1. Bellavista, P., Corradi, A., Foschini, L., Pernafini, A.: Towards an Automated BPEL-based SaaS Provisioning Support for OpenStack IaaS. Scalable Computing 14(4), 235–247 (2013)
2. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 692–695. Springer (Dec 2013)
3. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549. Advanced Web Services, Springer (Jan 2014)
4. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik (2016)
5. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: International Conference on Cloud Engineering (IC2E 2014). pp. 87–96. IEEE (Mar 2014)

6. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Pattern-based Runtime Management of Composite Cloud Applications. In: Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013). pp. 475–482. SciTePress (May 2013)
7. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012). pp. 416–424. Springer (Sep 2012)
8. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013). pp. 130–148. Springer (Sep 2013)
9. Eilam, T., Elder, M., Konstantinou, A.V., Snible, E.: Pattern-based Composite Application Deployment. In: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011). pp. 217–224. IEEE (May 2011)
10. El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., Konstantinou, A.: Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools. In: Proceedings of the 7th International Middleware Conference (Middleware 2006). pp. 404–423. Springer (Nov 2006)
11. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS). pp. 22–27. Xpert Publishing Services (Feb 2017)
12. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In: Cloudification of the Internet of Things (CIoT). IEEE (Nov 2016)
13. Hewson, J.A., Anderson, P., Gordon, A.D.: A Declarative Approach to Automated Configuration. In: Proceedings of the 26th Large Installation System Administration Conference (LISA). pp. 51–66. USENIX (Dec 2012)
14. Keller, A., Badonnel, R.: Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. In: Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004). pp. 15–27. Springer (Nov 2004)
15. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012). pp. 38–52. Springer (Sep 2012)
16. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 700–704. Springer (Dec 2013)
17. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F., Michelbach, T.: A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications. In: Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015. pp. 51–54. CEUR Workshop Proceedings (May 2015)
18. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)

19. Li, F., Vögler, M., Claeßens, M., Dustdar, S.: Towards automated iot application deployment by a cloud-based approach. In: 6th International Conference on Service-Oriented Computing and Applications (SOCA). pp. 61–68 (Dec 2013)
20. Mietzner, R., Leymann, F.: Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. In: Proceedings of the International Congress on Services (SERVICES 2008). pp. 3–10. IEEE (Jul 2008)
21. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
22. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
23. OASIS: Message Queuing Telemetry Transport (MQTT) Version 3.1.1. Organization for the Advancement of Structured Information Standards (OASIS) (2014)
24. OASIS: TOSCA Simple Profile in YAML Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2015)
25. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (OMG) (2011)
26. Opscode, Inc.: Chef Official Site, `http://www.opscode.com/chef`
27. Puppet Labs: Puppet Official Site, `http://puppetlabs.com/puppet/what-is-puppet`
28. da Silva, A.C.F., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B., Steinke, R.: Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER). pp. 358–367. SciTePress Digital Library (Jun 2017)
29. da Silva, A.C.F., Breitenbücher, U., Képes, K., Kopp, O., Leymann, F.: 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). pp. 181–182. ACM (Nov 2016)
30. Weerasiri, D., Benatallah, B., Barukh, M.: Process-driven Configuration of Federated Cloud Resources. In: Database Systems for Advanced Applications, pp. 334–350. Springer (2015)
31. Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., Zimmermann, M.: 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). pp. 559–568. SciTePress (Apr 2014)

All links were last followed on 28.07.2017.