Institute of Architecture of Application Systems

The OpenTOSCA Ecosystem – Concepts & Tools

Uwe Breitenbücher[1], Christian Endres[1], Kálmán Képes[1], Oliver Kopp[2], Frank Leymann[1], Sebastian Wagner[1], Johannes Wettinger[1], and Michael Zimmermann[1]

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

[2]Institute for Parallel and Distributed Systems, University of Stuttgart, Germany, oliver.kopp@ipvs.uni-stuttgart.de

BibTeX:

```
@article{Breitenbuecher2016_OpenTOSCAEcosystem,
  author     = {Uwe Breitenb{\"u}cher and Christian Endres and K{\'a}lm{\'a}n
                K{\'e}pes and Oliver Kopp and Frank Leymann and Sebastian Wagner
                and Johannes Wettinger and Michael Zimmermann},
  title      = {The OpenTOSCA Ecosystem – Concepts \& Tools},
  journal    = {European Space project on Smart Systems, Big Data, Future
                Internet - Towards Serving the Grand Societal Challenges -
                Volume 1: EPS Rome 2016},
  year       = {2016},
  pages      = {112--130},
  isbn       = {978-989-758-207-3},
  doi        = {10.5220/0007903201120130},
  publisher  = {SciTePress}
}
```

The paper has been published by SciTePress:
https://www.scitepress.org/PublicationsDetail.aspx?ID=2vC45wDfBew=&t=1

© 2016 SciTePress. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the SciTePress website.

Universität Stuttgart
Germany

The OpenTOSCA Ecosystem – Concepts & Tools

Uwe Breitenbücher[1], Christian Endres[1], Kálmán Képes[1], Oliver Kopp[2],
Frank Leymann[1], Sebastian Wagner[1], Johannes Wettinger[1], Michael Zimmermann[1]

[1]IAAS, [2]IPVS, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
{lastname}@informatik.uni-stuttgart.de

Abstract Automating the provisioning and management of Cloud applications is one of the most important issues in Cloud Computing. The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard for describing Cloud applications and their management in a portable and interoperable manner. TOSCA enables modeling the application's structure in the form of topology models and employs the concept of executable management plans to describe all required management functionality regarding the application. In this paper, we give an overview of TOSCA and the OpenTOSCA Ecosystem, which is an implementation of the TOSCA standard. The ecosystem consists of standard-compliant tools that enable modeling application topology models and automating the provisioning and management of the modeled applications.

1 Introduction

In recent years, Cloud Computing became highly market penetrating because of economical and technical benefits, for example, elasticity, outsourcing, pay-per-use pricing models, scalability, and self-service usage of services. Due the new mechanisms and technologies, companies are enabled to reduce their IT operations costs while achieving high automation and flexibility of IT systems [1]. But with high automation and flexibility, more complexity has been introduced. Modern applications that realize complex business functionality often require a wide and heterogeneous landscape of technologies. Especially, if application functionality is spread across multiple providers, the complexity of automation increases, for example, in Multi-Cloud and Hybrid-Cloud scenarios as well as in cyber-physical systems that combine Cloud technologies and physical entities. Due to the (i) immense complexity of provisioning, configuring, and managing such applications and (ii) the error-prone task of orchestrating the involved heterogeneous components, a manual execution of these provisioning and management tasks is not sufficient. Additionally, the manual provisioning and management of such applications is time-consuming and, therefore, not appropriate in the domain of Cloud Computing that requires automated management and on-demand features [2]. Thus, automating the provisioning, configuration, and management of *Complex Composite Cloud Applications* is a major issue in modern IT management [3].

However, automating the provisioning, configuration, and management of arbitrary complex applications is one of the most difficult challenges in current Cloud Computing research. This problem is impeded by the massive heterogeneity of Cloud services and respective service providers, introducing challenges such as the variety of API designs and data formats and the difficult integration of different components for composing new services [4]. Especially, the interoperability of Cloud services offered by different providers and their integration with existing middleware technology and configuration management tooling is a major issue [3]. Therefore, the Topology and Orchestration Specification for Cloud Applications (TOSCA) [5–7] has been standardized by OASIS to tackle these issues. TOSCA is a standard aiming for a portable description of Cloud application and automating their provisioning and management. The standard enables modeling the application's structure in the form of topology models and employs the concept of executable management plans to describe all required management functionality, e.g., how to migrate a component between different infrastructures.

In many different research and industry projects, we developed the *OpenTOSCA Ecosystem*[1] that is an open-source implementation of the TOSCA standard that consists of an integrated end-to-end toolchain to model applications using TOSCA and to automatically provision and manage them. In this paper, we provide an overview of the OpenTOSCA Ecosystem, the innovative research results, the developed concepts and tools, and the planned future work in the domain of the Internet of Things (IoT). We show how the OpenTOSCA Ecosystem works and how the different tools work together. Moreover, we explain how the integration of declarative and imperative provisioning is supported by the toolchain and how management processes can be modeled easily based on the standard. As a result, the paper summarizes the results of the past years regarding OpenTOSCA and enables understanding the big picture.

2 The TOSCA Standard – The OASIS Topology and Orchestration Specification for Cloud Applications

In this section, we introduce the fundamentals of the Topology and Orchestration Specification for Cloud Applications (TOSCA). In Section 2.1, an overview of TOSCA is given. Section 2.2 describes the concepts and the metamodel of TOSCA. To realize automated management functionality, the utilized mechanisms can be categorized into imperative and declarative provisioning approaches that are described in Sections 2.3 and 2.4. For more details, we refer interested readers to the TOSCA Specification [5], the TOSCA Primer [6], and the TOSCA Simple Profile [7]. A compact overview of TOSCA is given by Binz et al. [8, 9].

2.1 An Overview of TOSCA

In current Cloud Computing, there is a plethora of services that realize cloud applications. These services need to be managed and orchestrated to realize complex business functionality. But without a model to describe the Cloud application to be provisioned,

[1] https://www.github.com/OpenTOSCA

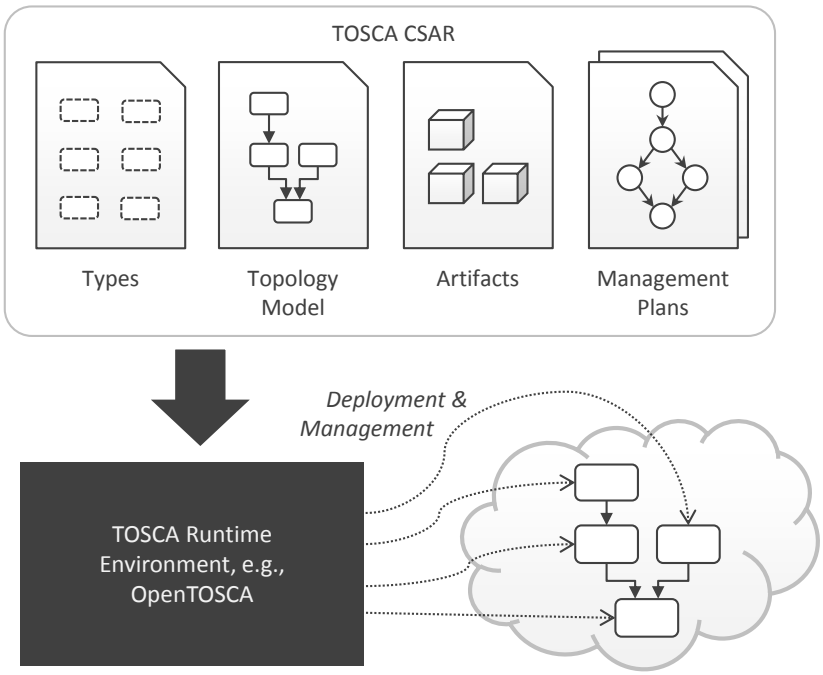

Figure 1: TOSCA Overview

there is no possibility to realize the management and orchestration in an automated manner. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [5] tackles these issues by providing a metamodel and packaging format. The main building blocks of a TOSCA model are shown in Figure 1 and can be summarized as follows: TOSCA enables (i) modeling *application topology models* that describe the structure of the Cloud application to be provisioned. The topology model is a directed, possibly cyclic graph consisting of typed nodes, which describe the application's components, and typed edges between these nodes, which, in turn, describe the relationships and dependencies between these components. To specify the semantics of components and their relationships, (ii) *type definitions* can be described in TOSCA. To provision and manage an application, (iii) different kinds of *artifacts* are required, for example, installation scripts or SQL files. Management functionality, for example, how to scale the application or how to migrate a component from one provider to another provider, can be described using so called (iv) *management plans*, which are executable process models, e.g., scripts or workflows [10]. To package all these different models, artifacts, and management plans, TOSCA defines a *Cloud Service Archive (CSAR)* that serves as portable packaging format. Standard-compliant CSARs can be consumed by any *TOSCA Runtime Environment* to deploy and manage the described application. Thus, portability is achieved by using a standardized metamodel and a standardized packaging format. A TOSCA Runtime Environment that can be used to consume such CSARs is provided by the OpenTOSCA Ecosystem presented in this paper.

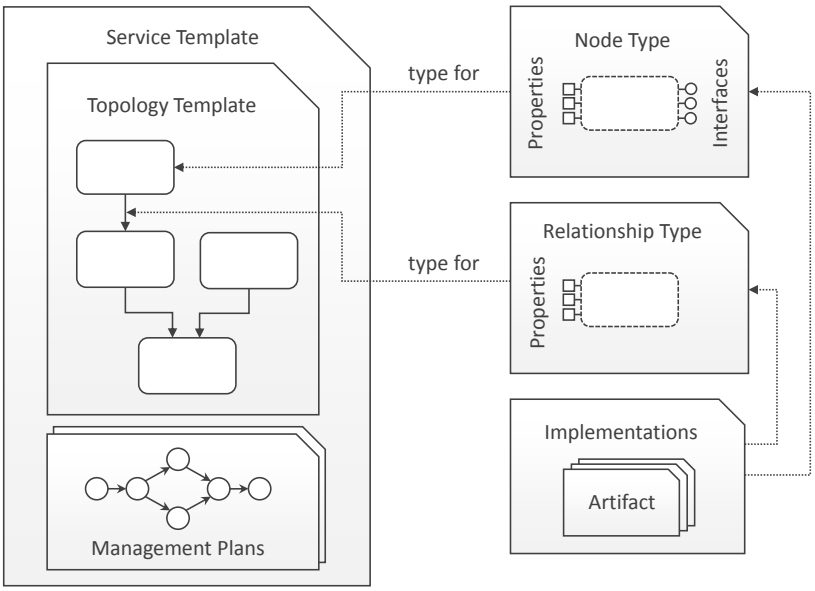

Figure 2: TOSCA Concepts

2.2 TOSCA Concepts & Metamodel

This section describes the underlying TOSCA concepts as well as its metamodel. Figure 2 shows the main building blocks of a TOSCA model. In TOSCA, components described by the topology model are called *Node Templates* while relationships are called *Relationship Templates*, the respective types are called *Node Types* and *Relationship Types*. These type definitions specify the semantics of the respective templates as well as their properties and interfaces. For example, a Node Template of Node Type *PHP* is connected via an Relationship Template of Relationship Type *SQL* with a Node Template of Node Type *MySQL database*. The topology model is called *Topology Template*.

On the right side of Figure 2, the reusability of TOSCA-based descriptions and artifacts is addressed. Node and Relationship Type definitions enable modeling common properties of components and dependencies. For example, an *Apache Web Server* provides the properties *HTTP-Port*, *username*, *password*, and others. Node Types and Relationship Types specify *Management Interfaces*, that provide a set of operations that can be invoked to manage the respective component. For example, an *Apache Web Server* Node Type may provide an interface that offers a *deployApplication* operation. This operation can be called by the TOSCA Runtime Environment or by Management Plans to manage a certain application instance. The type definitions support inheritance, so hierarchies of abstraction can be modeled. Both kinds of types need artifacts to enable its operation, for example, installation scripts or the implementation of the aforementioned *deployApplication* operation of the web server. Therefore, Node and Relationship Types are implemented using so called *Node Type-* and *Relationship Type Implementations*. These implementations contain all artifacts required to operate the corresponding type.

Artifacts are called *Artifact Templates* in TOSCA and can be classified as follows: (i) *Deployment Artifacts* implement business functionality, for example, the binary files of an Apache Web Server that are required to run this component. On the other hand, (ii) *Implementation Artifacts* implement management functionality that is required to execute the defined management operations. For example, an Implementation Artifact of a Web Server is an install script that installs the Web Server on an operating system. These latter artifacts are mostly used by TOSCA Runtime Environments and Management Plans to execute component-specific management logic.

In Figure 2 at the lower end, Management Plans are depicted. These plans implement management functionality for the application, for example, for provisioning the application (called *Build Plan*), to scale components or to migrate the application to another Cloud provider. Management Plans are typically implemented using the workflow technology [10], which provides certain standardized languages such as BPEL [11] or BPMN [12]. Leveraging these concepts, TOSCA standardized also a packaging format called *Cloud Service Archive (CSAR)*. This CSAR contains all aforementioned elements and, thereby, provides a self-contained package containing all files required for operating the application. Through the standardization, the format ensures portability of the contained models and artifacts. As a result, CSARs can be consumed by any standard-compliant TOSCA Runtime Environment to operate the described application.

2.3 Imperative Provisioning

For provisioning instances of the modeled application, TOSCA distinguishes between two approaches: (i) imperative provisioning and (ii) declarative provisioning [4, 6]. In this section, we explain the imperative approach that is based on executing a Management Plan that creates a new instance of the modeled application.

In general, Management Plans can be implemented to automate management functionality. They automate high-level management tasks, such as migrating a component, by orchestrating the low-level management operations provided by the Node Templates of the topology model. These low-level operations typically encompass small functionality, for example, a Linux operating system Node Template offers a management operation to execute a shell script or to install a package whose name is passed as input parameter of the operation. Such management operations can be implemented using various kinds of technologies. For example, using shell scripts as mentioned before or through executable programs. Management operations described by the topology model may also only wrap an existing publicly available service or API, for example, an Amazon EC2 Node Template may provide an operation to create a new virtual machine, whose implementation is just the management API of Amazon. To orchestrate these management operations, Management Plans can be implemented using process execution technologies like scripts or workflows [10]. However, due to features such as reliability, recovery, and compensation, the workflow technology provides a robust execution environment for such process models. In addition, especially the existence of standardized workflow modelling and execution languages such as BPEL [11] and BPMN [12] enable the creation of portable plans that can be executed on any standard-compliant workflow engine that supports the respective workflow language.

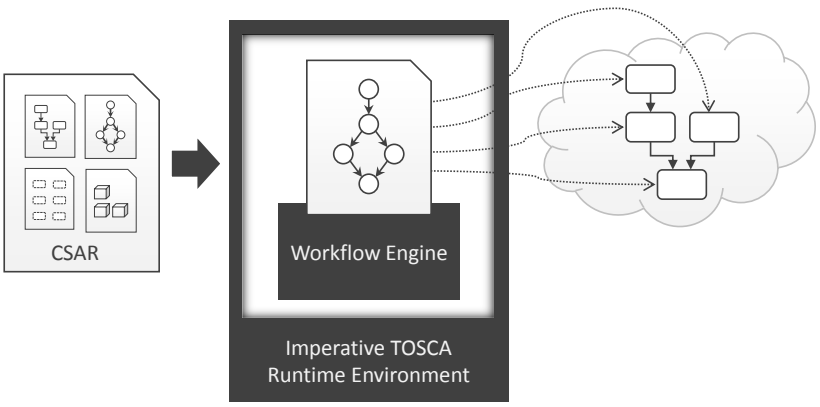

Figure 3: Imperative provisioning with TOSCA

Management Plans are contained in CSARs and are processed by TOSCA Runtime Environments. Following the distinction introduced above, runtime environments can be categorized by imperative and declarative environments. An *Imperative TOSCA Runtime Environment* supports executing Management Plans contained in CSARs, e.g., for provisioning an instance of the application by executing the contained Build Plan. Therefore, an imperative environment requires an internal process execution engine as shown in Figure 3. In this figure, a workflow engine is employed to execute Management Plans that are modeled as workflows. After consuming the CSAR, the environment extracts all plans and deploys them onto the engine. As a result, the plans can be invoked to execute the respective functionality, for example, provisioning the modeled application. Management Plans may implement arbitrary functionality by orchestrating all kinds of scripts, programs, APIs, etc. Therefore, using Management Plans enables the full customization of automated provisioning of even more complex applications.

2.4 Declarative Provisioning

In the previous section, we explained how TOSCA can be used to provision arbitrary complex applications using the imperative approach. This approach enables describing arbitrary management logic in the form of explicit, executable process models that are contained in the CSAR. However, creating Management Plans requires manual effort and technical expertise, especially, if the management task to be executed or the application is complex [3]. However, some simple management tasks, such as the provisioning of applications that mainly employ common components, can be executed without the need to model a Management Plan [4]. In this section, we describe how the provisioning of Cloud applications can be executed automatically only by interpreting the topology model, thus, without the need to create Build Plans. Following the TOSCA standard, we call this provisioning flavour *declarative provisioning* [6].

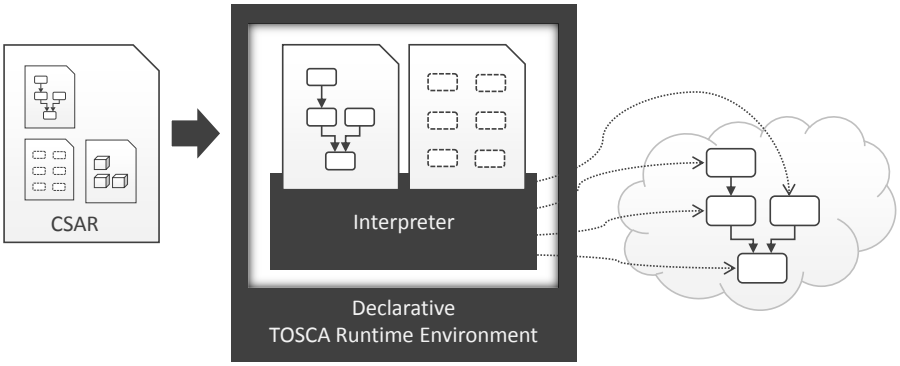

Figure 4: Declarative Provisioning with TOSCA

Contrary to the imperative approach, with the declarative provisioning approach the topology model gets interpreted by a *Declarative TOSCA Runtime Environment*, as shown in Figure 4. The runtime environment extracts the topology model including all type definitions and interprets the application's structure. Based on internal plugins, the provisioning of components can be executed following the semantically-defined dependencies between the components. For example, if component A has a hosted-on-dependency to component B, component B has to be provisioned first. Therefore, based on the semantics of type definitions, the order of provisioning activities can be calculated and executed using the management operations provided by Node Templates. To enable this, the TOSCA Primer [6] defines the so called *Lifecycle Interface*, which defines the semantics of the management operations *install*, *configure*, *start*, *stop*, and *terminate*. Based on Implementation Artifacts that implement these operations, imperative Management Plans as well as declarative TOSCA Runtime Environments are able to provision the individual components by executing these artifacts. How the provisioning order can be calculated can be found in Breitenbücher et al. [4]. As a result, a Declarative TOSCA Runtime Environment has to be able to interpret the application's structure for processing the CSAR in the declarative manner.

The declarative approach enables modelers to focus on the application's structure without the need of creating Build Plans that orchestrate the management operations offered by Node Templates. Thus, this approach is much easier [4]. However, the approach is limited in terms of the application's complexity and works only for simple applications that employ common, semantically-defined component types. If a complex application consisting of custom components has to be provisioned, the imperative approach has to be used since arbitrary management logic can be modeled.

However, both approaches do not mutually exclude each other. There are concepts and technologies that are able to generate imperative Management Plans out of declarative models, e.g., [13–15]. In the scope of the OpenTOSCA Ecosystem, we developed a *Build Plan Generator* that consumes topology models and generates Build Plans [4]. Thus, this forms a *hybrid approach* that combines the benefits of both worlds.

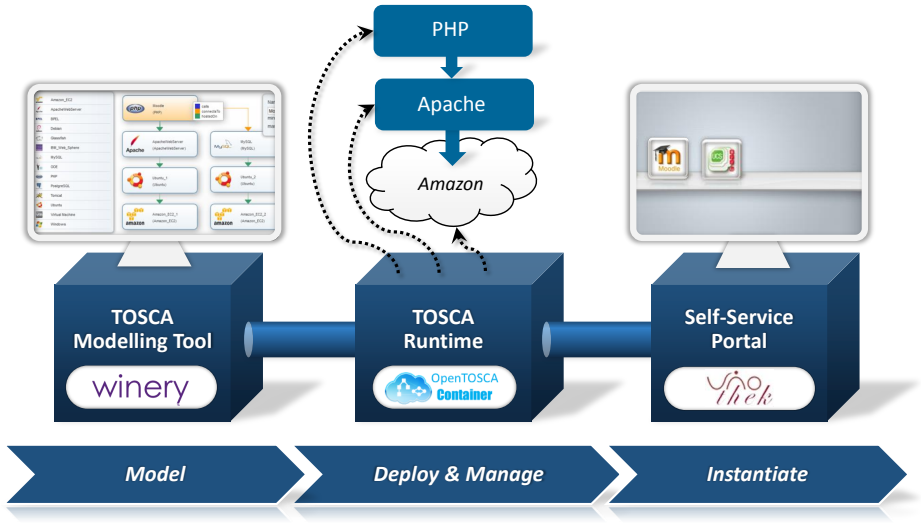

Figure 5: An Overview of the OpenTOSCA Ecosystem

3 The OpenTOSCA Ecosystem

In this section, we present an overview of the OpenTOSCA Ecosystem that mainly consists of the following tools: The TOSCA modeling tool *Winery* [16], the TOSCA Runtime Environment *OpenTOSCA Container* [17], and the TOSCA Self-Service Portal *Vinothek* [18]. Figure 5 shows how these three tools work together and how they build an ecosystem to (i) model, (ii) deploy, (iii) manage, and (iv) instantiate TOSCA-based applications. The entire OpenTOSCA Ecosystem[2] including all tools is an open-source implementation and publicly available on GitHub[3] and Eclipse[4].

The interplay of the three main tools is as follows. Winery is a standard-compliant TOSCA modeling tool that enables creating topology models using a graphical web-based editor. Moreover, Winery provides a backend system to create and maintain Node Types, Relationship Types, and other entities defined by the TOSCA metamodel. Winery supports exporting CSARs that contain all required files to deploy and manage the corresponding application using the OpenTOSCA Container.

The OpenTOSCA Container is a standard-compliant TOSCA Runtime Environment that is able to consume CSARs for deploying the therein modeled application. The runtime environment supports the imperative provisioning and management approach (see Section 2.3) by enabling the execution of BPEL-based workflow models [11]. Beside Build Plans, the runtime is able to execute arbitrary Management Plans to manage a certain application instance. Moreover, the OpenTOSCA Container also supports the

[2] http://www.iaas.uni-stuttgart.de/OpenTOSCA/
[3] https://www.github.com/OpenTOSCA
[4] https://projects.eclipse.org/projects/soa.winery

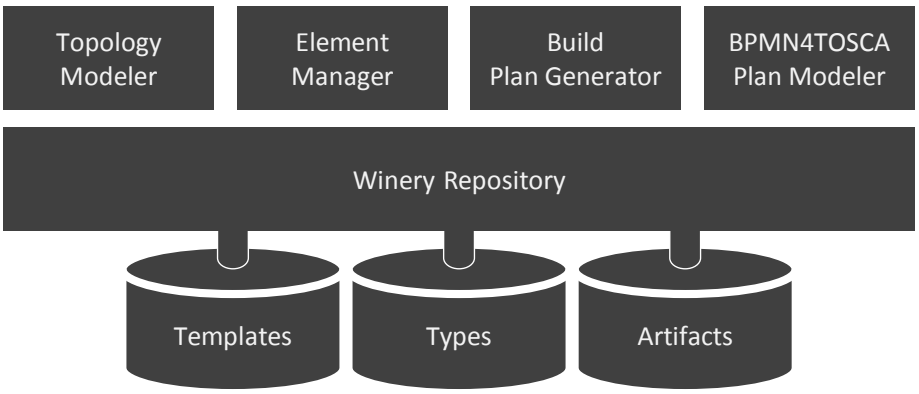

Figure 6: Winery Architecture

declarative provisioning approach (see Section 2.4). Thus, the container provides a *Hybrid TOSCA Runtime Environment* that supports both provisioning flavors. To provide an easy and intuitive user interface for end users, the OpenTOSCA Ecosystem offers the web-based self-service portal Vinothek. This portal offers all applications that are installed in the OpenTOSCA Container and enables end users instantiating new instances of an application by just clicking on a *provisioning button*.

In the following sections, we describe the three tools and the advanced research concepts they support in detail. At first, we introduce Winery (Section 3.1), followed by the OpenTOSCA Container (Section 3.2), and the Vinothek (Section 3.3). In the final section, we give an overview of the developed prototypes (Section 3.4).

3.1 The TOSCA Modeling Tool Winery

Winery [16] is a graphical, web-based TOSCA modeling tool developed at the University of Stuttgart. Figure 6 entails the architecture of Winery consisting of the Topology Modeler, Element Manager, BPMN4TOSCA Modeler, and a BPEL Build Plan Generator, which are connected to the Winery repository component holding all TOSCA related elements such as TOSCA Templates, Types, and Artifacts.

The *Topology Modeler* provides a graphical editor to create TOSCA topology models based on the visual topology modeling language *Vino4TOSCA* [19]. Figure 7 depicts the user interface of this component that allows users to drag and drop Node Templates from a palette into an editor area. Relationship Templates of a certain type can be drawn by clicking on the dropped Node Templates. Moreover, the Topology Modeler supports completing partially modelled topology models [20]. The *Element Manager* provides a system for creation, manipulation, and deletion of all TOSCA entities, for example, Node Types and Implementation Artifacts. The current graphical user interface of the Element Manager is depicted in Figure 8. The Element Manager supports exporting topology models as CSARs: Winery packages all required artifacts of the

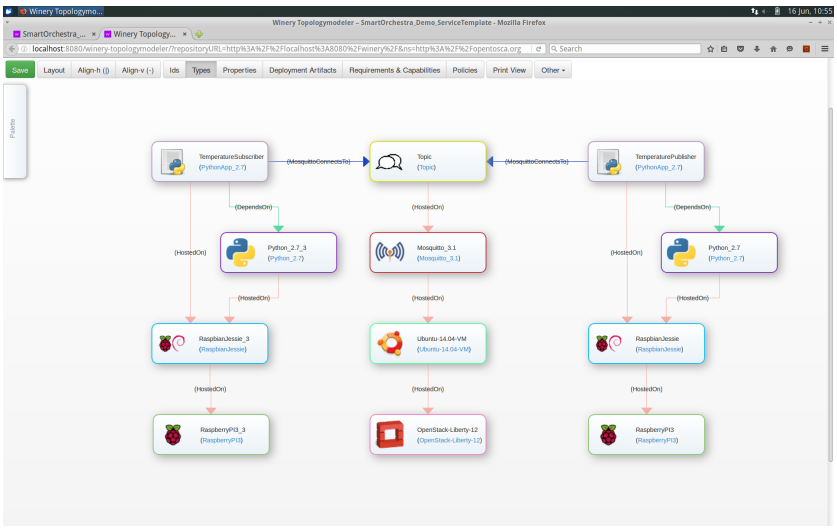

Figure 7: The Winery Topology Modeler showing a topology model of two Raspberry Pis communicating over a message broker hosted on a virtual machine

topology into one self-contained CSAR package [21]. In addition, Winery provides advanced tooling such as the automated creation of the TOSCA Lifecycle Interface for new Node Types and the *Implementation Artifact Generator*, which is able to generate Java-based Web Service skeletons that implement the operations defined on a TOSCA Node Type. Thus, users are able to model their topologies in a top-down manner by defining Node Types and their operations including the management interfaces. For implementing these interfaces, the user can request to generate the Java skeletons and implement the intended management logic inside these services. The whole asynchronous Web Service implementation is generated automatically and the service is packaged using Maven[5]. This significantly eases the implementation of management operations since these asynchronous Java Web Services are directly supported by the Management Bus of the OpenTOSCA Container (see Section 3.2). In addition, the Element Manager provides a *Build Plan Generator* that is able to interpret topology models in order to fully automatically generate a corresponding Build Plan using the standardized workflow language BPEL [11]. Generated plans are also packaged into the CSAR during the export. Thus, this enables (i) using plain imperative TOSCA Runtime Environments since all the required logic for provisioning is contained in the CSAR. Moreover, this (ii) forms a hybrid provisioning approach as the generated plans can be customized arbitrarily, if necessary. As a result, this concept combines the benefits of both worlds: simple applications can be provisioned fully automatically by generating Build Plans. However, if a complex application shall be provisioned, a generated Build Plan may serve only as starting point and needs to be modified to handle custom requirements. Details about this plan generation can be found in Breitenbücher et al. [4].

[5] https://maven.apache.org/

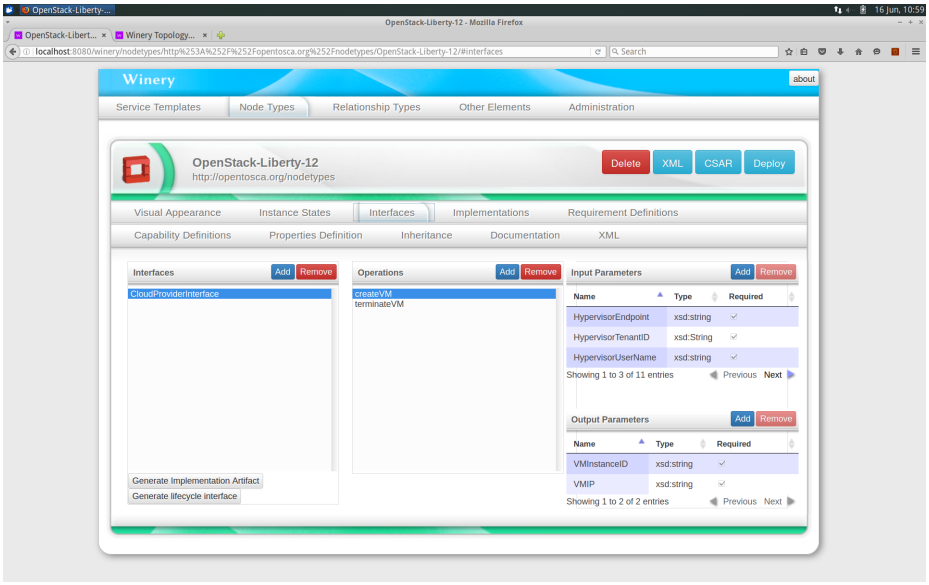

Figure 8: The Winery Element Manager showing the editing options of a Node Type

To support imperatively describing the provisioning of TOSCA topology models, Winery allows modelers the specification of imperative TOSCA management plans. Therefore, the modeling tool provides an editor to model imperative plans using the workflow language *BPMN4TOSCA* [22]. BPMN4TOSCA is an extension of the Business Process Modeling and Notation 2.0 (BPMN) [12] and introduces new domain-specific elements for manipulating TOSCA application instances. Thus, the extension is a BPMN dialect for the TOSCA standard providing special types of activities that enable directly invoking management operations provided by Node Templates and handling properties of Node and Relationship Templates in terms of instance data. Thus, BPMN4TOSCA aims for simplifying the development of management plans and is supported by a graphical editor contained within Winery [23]. Moreover, we are currently finishing a *BPMN4TOSCA to BPEL Transformer* that enables executing created BPMN4TOSCA models in the OpenTOSCA container, which currently supports BPEL as main workflow language. This component transforms BPMN4TOSCA-specific activities (e.g., the *TOSCA Node Management Task* [22]) into BPEL activities that implement exactly the semantics of the corresponding BPMN4TOSCA element.

In addition, we are currently working on an importer for the TOSCA Simple Profile, which is a new TOSCA serialization format in YAML [7] (see Section 4). Moreover, we are developing new Node Types and new Relationship Types for the domain of IoT and for extending the Cloud portfolio. First IoT-related Node and Relationship Types are shown in Figure 7: The depicted topology model implements a simple use case consisting of two Raspberry Pis communicating over the Mosquitto MQTT Message Broker, which is hosted on a virtual machine running inside an OpenStack.

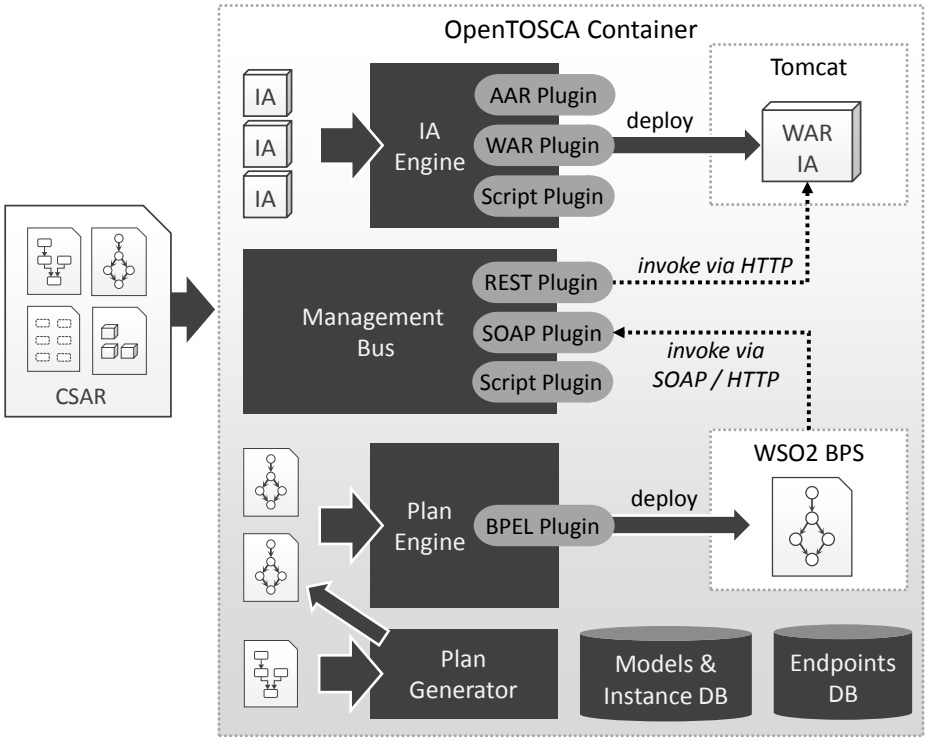

Figure 9: OpenTOSCA Container Architecture

3.2 The OpenTOSCA Container

In this section, we describe the TOSCA Runtime Environment of the OpenTOSCA Ecosystem, which is called *OpenTOSCA Container* [17]. The container is able to process CSARs in a declarative and imperative manner, thus, providing a hybrid provisioning environment that is seamlessly integrated with Winery.

The general processing of the container can be described as follows. The OpenTOSCA Container enables the automated provisioning of Cloud applications that are modeled using TOSCA and packaged as CSARs. To realize the provisioning, the container analyses the contained TOSCA model and invokes the contained Build Plan to instantiate a new application instance of the contained TOSCA model. If there is no Build Plan available, the container generates a Build Plan on its own by using the Plan Generator component and invokes this plan for provisioning the application. During the lifetime of the application, the container enables managing application instances by invoking Management Plans contained in the respective CSAR, for example, to scale application components. Management Plans can be either modeled manually using Winery or generated for provisioning, as mentioned above. Thus, the OpenTOSCA Container supports both imperative and declarative application provisioning approaches.

OpenTOSCA Container Architecture In Figure 9, an overview of the OpenTOSCA Container architecture is depicted. The container mainly consists of the following components, which are explained in detail in the following subsections: The *Implementation Artifact Engine* (IA Engine) is responsible for processing the Implementation Artifacts contained within the CSAR. Similarly, the *Plan Engine* is responsible for the processing of Management Plans. The *Management Bus* [24, 25] is a communication middleware inside the container that enables plans to invoke different kinds of management operations through a unified interface. The *Plan Generator* [4] allows generating imperative Build Plans based on declarative TOSCA models. Also, the OpenTOSCA Container needs data storages to manage information about, for example, known CSARs or available service endpoints. These information are stored in two independent databases: the *TOSCA Models & Instance Database* and the *Endpoints Database*.

Implementation Artifact Engine Management Plans implement management logic that has to be automated for a certain application, for example, a *Migration Plan* specifies all activities, their execution order, and the data flow between the activities to migrate a component from one infrastructure to another. To implement these activities, Management Plans may orchestrate various kinds of software and services, for example, scripts or APIs of Cloud providers. These low-level operations are often not contained in the Management Plans themselves and, usually, are located in the application's target environment, for example, a creation mechanism for virtual machines that is exposed by a hypervisor as API or an installation script for the Apache Tomcat application server that has to be executed on the target virtual machine. To realize the execution of such management operations, TOSCA provides the notion of Implementation Artifacts that implement the management operations provided by Node Templates (see Section 2.2). Implementation Artifacts are contained in the CSAR that is passed to the OpenTOSCA Container and processed by the Implementation Artifact Engine. They can be divided in either local or remote Implementation Artifacts [24]. In OpenTOSCA, *Local Implementation Artifacts* are SOAP-based Web Application Archives (WARs)[6] that are executed by the OpenTOSCA Container by deploying them on a local Apache Tomcat Servlet Container[7]. For being invokable by plans, endpoints of the deployed Implementation Artifacts are stored in the *Endpoints Database*. Contrary, *Remote Implementation Artifacts* are, for example, shell scripts that are executed in the application's target environment. Therefore, such Implementation Artifacts are not deployed locally by the Implementation Artifact Engine but are just stored in the *Models and Instance Database*, which can be accessed by plans via an API. To enable appropriate processing according to the Implementation Artifact types, the Implementation Artifact Engine implements an extensible plugin-system with a plugin for each Implementation Artifact type. At the moment, the OpenTOSCA Container and its Implementation Artifact Engine contains plugins to support the processing of Java Web Application Archives (WARs) and Axis Archives (AARs). To support other types of Implementation Artifacts, the provided plugin interface of the engine enables an easy development of new plugins.

[6] Skeletons for this kinds of Implementation Artifacts can be automatically generated using Winery, see Section 3.1.

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

Plan Engine Similar to the Implementation Artifact Engine, the Plan Engine is responsible for the processing of the Management Plans contained in a CSAR. Since plans implement imperative provisioning and management logic for the application inside the CSAR, plans need to be invokable and executable. Therefore, the OpenTOSCA Container employs a local workflow engine, namely the WSO2 Business Process Server (BPS)[8], which is used by the Plan Engine to deploy BPEL-based Management Plans and to make them executable. TOSCA proposes modeling plans using workflow languages since they provide a robust process execution environment, cf. Section 2.3. However, as there are different kinds of languages available, the Plan Engine provides a plugin-system for adding support for other languages, for example, BPMN, too.

Management Bus Various kinds of Implementation Artifacts may be used to implement a certain management operation: On the one side, Web Services typically are used to offer infrastructure functionalities in the form of APIs, for example, to offer operations for creating virtual machines. On the other side, scripts and configuration management tools such as Chef [26] are often used to implement the TOSCA Lifecycle Interface [6], for example, to implement the install operation of a Web Server as shell script that is executed on the operating system that shall host the Web Server. Thus, if Management Plans need to orchestrate diverse operation implementations, they have to deal with the low-level, technical details of their invocation [3]. As a result, Management Plans become complex models that are hard to create and even harder to maintain. Due to this issues, we developed the *OpenTOSCA Management Bus*, which provides a uniform interface for invoking different kinds of Implementation Artifacts. In OpenTOSCA, we offer a SOAP/HTTP-based API that can be easily invoked using BPEL workflow models. Thus, plans themselves only invoke the operation via this API while all technical details about the actual execution of the associated Implementation Artifact is handled and hidden by the bus. The bus chooses also the right Implementation Artifact according to the operation that shall be invoked. To enable extensibility, the Management Bus implements a plugin-system, which currently provides plugins for invoking SOAP/HTTP and HTTP-based Web Services, shell scripts, Chef [26], and Ansible [27]. The Management Bus also supports the asynchronous communication used by the Java-based Web Services generated using the modeling tool Winery (cf. Section 3.1). Details about the Management Bus can be found in Wettinger et al. [24, 25].

Plan Generator We described the OpenTOSCA Container as a TOSCA Runtime Environment that is capable of imperative and declarative provisioning (cf. Section 2.3 and Section 2.4). Whilst the imperative provisioning is realized by utilizing Build Plans that are shipped within the CSAR, the declarative provisioning is realized using the Plan Generator. The Plan Generator is an integrated processor of declarative TOSCA topology models and generates an imperative BPEL-based Build Plan. Details about this plan generation are described in Breitenbücher et al. [4]. As depicted in Figure 9, generated Build Plans are passed to the Plan Engine, which makes them executable by deploying them onto the local workflow engine as described in Section 3.2.

[8] http://wso2.com/products/business-process-server/

(a) The Vinothek Overview page shows the available applications.

(b) The Vinothek Application Details page shows the details of the Moodle application.

Figure 10: Screenshots of the OpenTOSCA Vinothek

3.3 The Self-Service Portal Vinothek

Beside a TOSCA Runtime Environment, the OpenTOSCA Ecosystem also provides a self-service portal that can be easily used by end users to provision new instances of CSARs installed in the OpenTOSCA Container. This self-service portal is called *Vinothek* [18] and provides a simple, web-based graphical user interface shown in Figure 10a. The Vinothek lists all installed CSARs and provides detailed information when selecting one of them as shown in Figure 10b. Further, within this view, the provisioning of the displayed application can be triggered by clicking on a simple button. Also, required input parameters, for example, credentials, can be provided, too.

The Vinothek is implemented according to the web-based Client-Server architecture. In Figure 11, the architecture is depicted. The *graphical user interface (GUI)* is implemented using Java Server Pages and HTML5. The *RESTful API* delegates the HTTP-requests received from the GUI to the *TOSCA Application Lifecycle Manager*, which handles the provisioning of applications. To enable integrating different TOSCA Runtime Environments, the requests are delegated to the *TOSCA Runtime Integration Layer*, which integrates various runtimes via plugins. Additionally to the processing of provisioning tasks, the TOSCA Runtime Integration Layer is responsible for gathering information from the connected TOSCA Runtimes, for example, the available CSARs and when the provisioning of an application instance is finished. Currently, there is only one plugin available for integrating the OpenTOSCA Container. Thus, the self-service portal Vinothek serves as an abstraction layer for end users, since a simple user interface can be used to start a provisioning instead of invoking Build Plans manually.

In addition to the TOSCA specification, we extended the CSAR format to contain additional information in the form of meta-information encompassing, e.g., icons, screenshots, and descriptions about the modeled application as shown in Figure 10b.

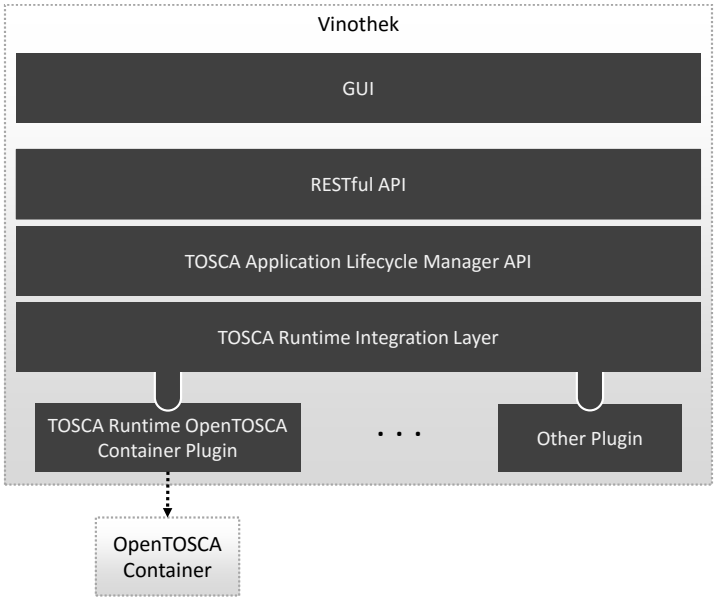

Figure 11: The Vinothek Architecture Overview (adapted from [18]).

3.4 Prototype

All components of the OpenTOSCA Ecosystem are released under the open-source license Apache 2.0 [28], except Winery, which is released under both the Eclipse Public License (EPL) [29] and the Apache 2.0 License. The OpenTOSCA Ecosystem can be obtained from GitHub[9] and Eclipse[10].

Winery is implemented in Java 1.7 and is available as Web Application Archive (WAR). The whole Winery project is split into the Winery Repository itself and the Topology Modeler, both are implemented as separate Web applications. The repository is accessible through a RESTful interface, for which the project also entails a client library. The GUI components access data trough the RESTful interface and expose the graphical Web interface implemented with JavaScript and Java Server Pages (JSP). The BPMN4TOSCA Modeler[11] and BPMN4TOSCA to BPEL Transformer[12] are not part of the main release of Winery, as they are not part of the Eclipse project. The modeler is a prototypical implementation that allows modeling a subset of BPMN4TOSCA, mainly the BPMN4TOSCA tasks themselves, start/end events, and basic gateways. The transformer, which is currently completed, can take models specified using the modeler and transform them into BPEL models that are executable using the WSO2 BPS.

[9] https://www.github.com/OpenTOSCA
[10] https://projects.eclipse.org/projects/soa.winery
[11] https://github.com/winery/BPMN4TOSCAModeler
[12] https://github.com/winery/BPMN4TOSCA2BPEL

The OpenTOSCA Container is built on top of the OSGi framework[13], a component-based model for Java applications. The functions of the container can be accessed through a RESTful HTTP API, which provides operations for the upload of TOSCA CSARs, their processing, provisioning, and management. For the container to operate as expected, the Implementation Artifact and Plan Engines employ an Apache Tomcat and a WSO2 BPS, respectively. The self-service portal Vinothek is a single Web application implemented in Java using Java Server Pages. It is connected to the OpenTOSCA Container via the RESTful API.

4 Future Work

In this section, we describe the planned future work and ongoing research activities regarding TOSCA and the OpenTOSCA Ecosystem.

Support YAML Profile The TOSCA committee released a TOSCA Simple Profile for the data serialization format YAML [7]. The goal of the Simple Profile is to reduce the learning curve and ease the adoption of TOSCA to model Cloud applications. Therefore, the YAML rendering of TOSCA maps to a subset of the original TOSCA Specification. Future work entails the support for TOSCA models defined with the YAML Simple Profile, which calls for the development of a transformation component.

Integration of a IoT-Message Broker In order to support rapidly emerging IoT services, we plan to integrate an IoT-Message Broker into the OpenTOSCA Container. Besides the invocation of application operations as outlined in Section 3.2, this will enable OpenTOSCA to support (i) the messaging pattern publish and subscribe as well as (ii) diverse IoT communication protocols, such as MQTT [30] or CoAP [31]. Moreover, we plan to apply our work on security policies to this domain [32].

Integration of a BPMN Engine To implement the imperative provisioning approach, the OpenTOSCA Container supports BPEL. We are working on extending the OpenTOSCA Ecosystem to support BPMN 2.0. This enhances the usability of the ecosystem because developers can choose which workflow language they prefer to use.

Application Bus Beside the OpenTOSCA Management Bus, which provides a central, unified component for invoking arbitrary kinds of Implementation Artifacts implementing management operations, we are working on a service bus that enables an easy communication between components contained in TOSCA models. Moreover, this bus will allow the invocation of modeled operations through a unified interface. However, these operations provide business functionality instead of management functionality. Since TOSCA does not support defining interfaces for invoking business functionality, we will extend the standard accordingly for this purpose.

[13] https://www.osgi.org

Integration with Docker Docker is an actively emerging open-source technology for packaging applications using containers. We are currently working on Node Types that support Docker Deployment Artifacts as well as on a *Docker Engine* Node Type. To enable the automated provisioning and integration with conventional non-containerized components, we extend the OpenTOSCA Plan Generator to handle such containers. Consequently, the huge and continuously growing ecosystem of publicly shared Docker containers[14] can be utilized as building blocks of TOSCA-based topology models. In addition, OpenTOSCA's Implementation Artifact Engine is being extended to support Docker Implementation Artifacts. This allows for creating Implementation Artifacts using arbitrary technologies, exposing technology-agnostic interfaces such as RESTful or SOAP Web service APIs. Furthermore, Docker Compose, Docker Machine, and Docker Swarm are utilized to further simplify and streamline the bootstrapping of instances of the OpenTOSCA Ecosystem on diverse infrastructures such as developer machines, on-premise servers, and Cloud environments.

Acknowledgements

This work is partially funded by the BMWi projects CloudCycle (01MD11023), Migrate! (01ME11055), SePiA.Pro (01MD16013F), and SmartOrchestra (01MD16001F).

References

1. Leymann, F.: Cloud Computing: The Next Revolution in IT. In: Proc. 52th Photogrammetric Week, Wichmann Verlag (2009) 3–12
2. Fehling, C., Retter, R.: Composite as a Service: Cloud Application Structures, Provisioning, and Management. it - Information Technology Special Issue: Cloud Computing (2011) 188–194
3. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In: Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS 2013), Springer (2013) 130–148
4. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), IEEE (2014) 87–96
5. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0 (2013) http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.
6. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0 (2013) http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.html.
7. OASIS: TOSCA Simple Profile in YAML Version 1.0 – Committee Specification 1.0 (2016) http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/cs01/TOSCA-Simple-Profile-YAML-v1.0-cs01.html.
8. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: Advanced Web Services. Springer (2014) 527–549

[14] https://hub.docker.com

9. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing (2012) 80 – 85
10. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)
11. OASIS: Web Services Business Process Execution Language (BPEL) Version 2.0. (2007) http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
12. Object Management Group, Inc.: Business Process Model and Notation (BPMN) Version 2.0. (2011) http://www.omg.org/spec/BPMN/2.0/.
13. 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), SciTePress (2013) 475–482
14. 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), Springer (2006) 404–423
15. Eilam, T., Elder, M., Konstantinou, A., Snible, E.: Pattern-based Composite Application Deployment. In: Proceedings of the 12th International Symposium on Integrated Network Management (IM 2011), IEEE (2011) 217–224
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), Springer (2013) 700–704
17. 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), Springer (2013) 692–695
18. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: 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 (2014) 69–72
19. 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), Springer (2012) 416–424
20. Hirmer, P., Breitenbücher, U., Binz, T., Leymann, F.: Automatic Topology Completion of TOSCA-based Cloud Applications. In: Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI), GI (2014) 247–258
21. Wettinger, J., Breitenbücher, U., Kopp, O., Leymann, F.: Streamlining DevOps Automation for Cloud Applications using TOSCA as Standardized Metamodel. Future Generation Computer Systems (2016) 317–332
22. 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), Springer (2012) 38–52
23. 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), CEUR Workshop Proceedings (2015) 51–54
24. 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), SciTePress (2014) 559–568

25. Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: Streamlining cloud management automation by unifying the invocation of scripts and services based on TOSCA. International Journal of Organizational and Collective Intelligence (IJOCI) (2014) 45–63
26. Taylor, M., Vargo, S.: Learning Chef: A Guide to Configuration Management and Automation. O'Reilly (2014)
27. Hochstein, L.: Ansible: Up and Running. O'Reilly Media, Inc. (2014)
28. The Apache Software Foundation: Apache License, Version 2.0 (2016) http://www.apache.org/licenses/LICENSE-2.0.
29. The Eclipse Foundation: Eclipse Public License - Version 1.0 (2016) https://www.eclipse.org/legal/epl-v10.html.
30. OASIS: OASIS Message Queuing Telemetry Transport (MQTT) TC – OASIS (2014) https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard.
31. Bormann, C.: CoAP – Constrained Application Protocol – Overview (2016) http://coap.technology/.
32. Waizenegger, T., Wieland, M., Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Mitschang, B., Nowak, A., Wagner, S.: Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences, Springer (2013) 360–376

All links were last followed on July 4th, 2016.