Model Driven Cloud Orchestration by Combining TOSCA and OCCI

Fabian Glaser, Johannes Erbel, and Jens Grabowski
University of Goettingen, Institute of Computer Science, Germany
{*fglaser, grabowski*}*@cs.uni-goettingen.de, j.erbel@stud.uni-goettingen.de*

Keywords: Cloud Computing, Open Cloud Computing Interface, Topology and Orchestration Specification for Cloud Applications, Metamodelling

Abstract: To tackle the problem of a cloud-provider lock-in, several standards have emerged in the recent years which aim to provide a unified interface to cloud resources. The Open Cloud Computing Interface (OCCI) thereby focuses on the standardization of a common API for *Infrastructure-as-a-Service* (IaaS) providers and the Topology and Orchestration Specification for Cloud Applications (TOSCA) focuses on the standardization of a template language to enable the proper definition of the topology of cloud applications and their orchestrations on top of an IaaS cloud. TOSCA thereby does not define how the application topologies are created on the cloud. Therefore, it is worthwhile to analyse the conceptual similarities between the two approaches and the possibilities to integrate both. In this paper, we provide an overview of the similarities between the two standardization approaches. Furthermore, we define a concept of a fully model driven cloud orchestrator based on the two standards.

1 INTRODUCTION

With the rise of cloud computing, a multitude of proprietary cloud APIs became available that made it hard for cloud costumers to switch between different cloud providers. To tackle the problem of this *cloud-provider lock-in*, consortia have been formed to develop common standards for interfacing cloud resources. The *Open Cloud Computing Interface* (OCCI) (Nyrén et al., 2016b), developed by the *Open Grid Forum* (OGF)[1,2], thereby aims to provide a standardized managing interface, enabling the customer to manage cloud resources. It has been initially published in 2010 and several open-source implementations have been developed since then supporting all major open-source cloud middleware frameworks, including OpenStack[3], OpenNebula[4] and CloudStack[5].

At a higher level of abstraction, the *Organization for the Advancement of Structured Information Standards* (OASIS)[6] is developing the *Topology and Orchestration Specification for Cloud Applications* (TOSCA)(OASIS, 2013), a template format that aims to standardize the definition of application topologies for cloud orchestration. As such, it enables the customer to define the topology of the cloud application in a reusable manner and to deploy it on TOSCA compliant IaaS clouds. TOSCA has been initially published in 2013 and many major industrial cloud-providers plan supporting it. In contrast to OCCI, TOSCA does not define how the topology are programmatically created on the cloud infrastructure and leaves the implementation to the cloud provider. While the approaches of TOSCA and OCCI are different, both define a model for cloud resources. The goal of this work is to identify the conceptual similarities and differences between the two models and provide a mapping between them where possible. Such a mapping is the first step for building a fully model driven cloud-provider agnostic cloud orchestrator that leverages both TOSCA and OCCI for portable application and infrastructure provisioning and deployment. The remainder of this paper is structured as follows. First we briefly introduce the models of TOSCA and OCCI in Section 2. Then we provide a conceptual comparison and a mapping between the two models in Section 3, followed by the discussion of the model driven cloud orchestrator in Section 4. A feasibility study is discussed in Section 5. We introduce related work in Section 6. Finally, we draw our conclusions and give an outlook on future work in Section 7.

[1] https://www.ogf.org/ogf/doku.php/start
[2] All URLs have been last retrieved on 01/31/2017.
[3] http://www.openstack.org
[4] http://opennebula.org
[5] https://cloudstack.apache.org
[6] https://www.oasis-open.org/

2 BACKGROUND

Both TOSCA and OCCI define languages for modelling cloud resources. Since they hence provide a model for modelling they can be seen as *metamodels* (OMG, 2014). We introduce these metamodels in the following.

2.1 TOSCA

According to the specification (OASIS, 2013), TOSCA is "a language to describe service components and their relationships using a service topology, and it provides for describing the management procedures that create or modify services using orchestration processes." Therefore, it is able to describe both the service structure as well as the processes that can be executed on this structure. As the time of this writing, two versions of TOSCA exist. The first is based on XML (OASIS, 2013), and the second is based on YAML (OASIS, 2016). While for TOSCA XML a *XML Schema Definition* (XSD) schema exists, the TOSCA YAML version lacks of a formal metamodel. A simplified metamodel of TOSCA is depicted in Figure 1. A *ServiceTemplate* captures the structure and the life cycle operations of the application. It consists of a *TopologyTemplate* and a *Plan*. Plans define how the cloud application is managed and deployed. TopologyTemplates contain *EntityTemplates*, which are *NodeTemplates* that define e.g., the virtual machines or application components, *RelationshipTemplates* that encode the relationships between the NodeTemplates, e.g., that a certain application component is deployed on a certain virtual machine, or *GroupTemplates*[7] that allow to define groups of NodeTemplates, which e.g. should be scaled together. Additionally, TOSCA defines the EntityTemplates *Capability* and *Requirement*. Capabilities are used to define that a NodeTemplate has a certain ability, e.g., providing a container for running applications, and Requirements are used to define that a certain NodeTemplate requires a certain Capability of another NodeTemplate. All EntityTemplates can have *Properties*, e.g., an IP address for a virtual machine, and a certain *type* that references an *EntityType*. The *EntityType* defines the allowed Properties through *PropertyDefintions*, and have *Interfaces*, which define the *Operations* that can be executed on an instances implementing the type, e.g., the termination of a certain application component, or the restart of a virtual machine. Operations have *Parameters* that define their input and output. In addition to parameters for operations, TOSCA also allows to define input parameters for Plans.
Besides this abstract metamodel, the TOSCA YAML specification defines *normative types* that should be supported by each TOSCA conformant cloud orchestrator. These normative types include e.g., base types for cloud services and virtual machines. More details on the model elements can be found in (OASIS, 2013) and (OASIS, 2016).

2.2 OCCI

According to the OGF, "OCCI is a Protocol and API for all kinds of Management tasks. OCCI was originally initiated to create a remote management API for IaaS model based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring."[8]. The OCCI specification comprises several parts. The OCCI Core Model (Nyrén et al., 2016b) defines a model for cloud resources and their dependencies. OCCI extensions define extensions of the core model to be used for a specific domain. Several extensions are already standardized, e.g., the OCCI Infrastructure Extension (Metsch et al., 2016), which defines compute, network and storage resources for IaaS clouds, and the OCCI Infrastructure Extension for the *Platform-as-a-Service* (PaaS) domain, that defines additional resources for the PaaS Service level. OCCI Renderings define how the OCCI Core Model can be interacted with, e.g., the OCCI HTTP Protocol (Nyrén et al., 2016a) that defines how OCCI resources can be managed over the HTTP protocol.

The OGF does not provide a formal metamodel for OCCI. This gap has been recently addressed by Merle et al. (Merle et al., 2015) and we adopt their metamodel in scope of this work. Figure 2 depicts the OCCI Core Model. The OCCI Core Model is composed of eight elements. The *Category* is the base type for all other classes and provides the necessary identification mechanisms. Categories have *Attributes* that are used to define the properties of a certain class, e.g., the IP address of a virtual machine. Three classes are derived from Category: *Kind*, *Action*, *Mixin*. *Kind* defines the type of a cloud entity, e.g., a compute resource and *Mixins* define how an entity can be extended. Both have *Actions* that define which actions can be executed on an entity. The cloud entities themselves are modelled by the class *Entity*, which provides the base class for cloud *Resources*, e.g., virtual machines, and *Links* that define how the resources are connected.

[7] GroupTemplates and GroupTypes are currently part of the TOSCA YAML rendering, but not part of the TOSCA XML specification.

[8] http://occi-wg.org/

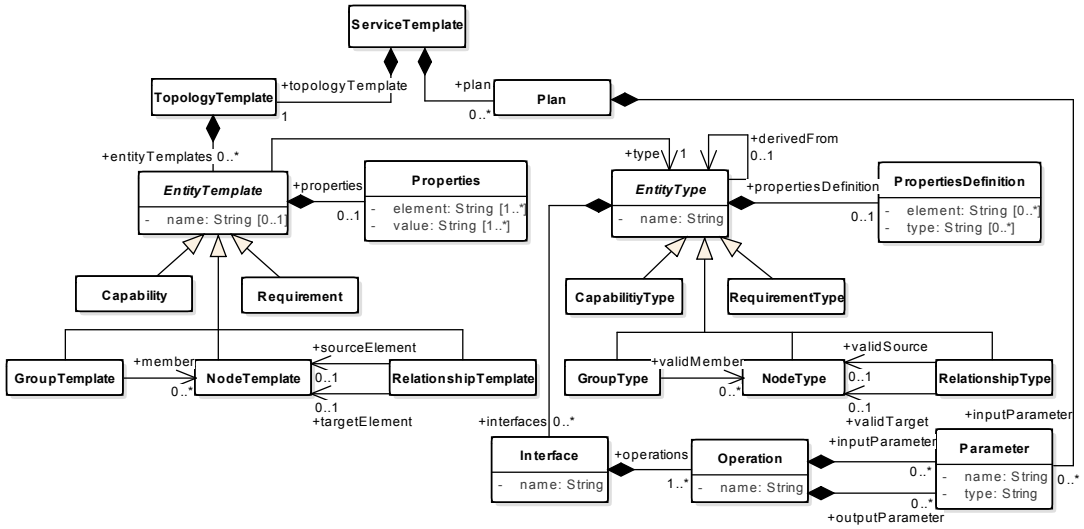

Figure 1: Metamodel of TOSCA (adapted from (Bergmayr et al., 2016)).

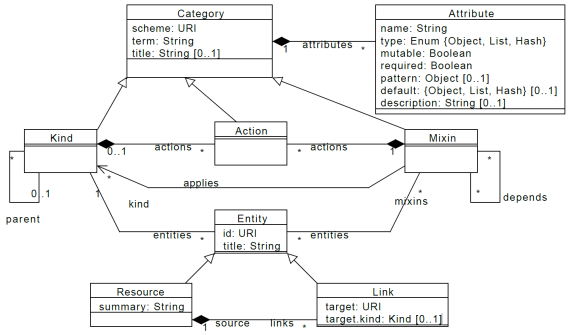

Figure 2: Metamodel of OCCI Core (Nyrén et al., 2016b).

In addition to the Core model, the OGF provides extensions, which define the resources for a certain cloud service level. For example, the OCCI Infrastructure Extension (Metsch et al., 2016) defines the Resources for IaaS. It provides the definitions of the Resources *Compute*, *Network* and *Storage*, and the links that can be established between them, namely *StorageLink*, which connects a compute resource to a storage resource and *NetworkInterface* and *IPNetworkInterface* which connect a compute resource to a network resource.

3 MAPPING TOSCA TO OCCI

While both standards define a metamodel for cloud resources, their focus is different. The focus of OCCI is to provide a standardized API and it does not define concepts to address reusability, composability, and scalability. Instances of the OCCI metamodel are not meant to be stored persistently and to be reused later on as it is the goal of TOSCA. TOSCA on the other side does not define how the defined topology is deployed by means of API calls to the cloud provider as it is done with the OCCI HTTP rendering. Hence, both approaches have their strengths and weaknesses and it is worthwhile to investigate how to integrate them.

Several concepts of TOSCA do not have a one-to-one correspondence in OCCI. These are the concepts for composability, namely the definition of *Capabilities* and *Requirements*, the concepts to achieve scalability, namely *Groups*, *Policies*, and *Parameters* and the concepts to achieve reusability, which are *ServiceTemplates* and *TopologyTemplates*. These are missing due to the different focus of OCCI. Nevertheless, the mapping of most concepts is possible and is summarized in Table 1. EntityTypes are used to define reusable elements in TOSCA. They have *PropertyDefinitions* assigned, that allow to define the properties that a certain EntityType is allowed to have. This matches the purpose of OCCI Categories and their Attributes. Additionally, EntityTypes have Interfaces that define the allowed operations. This matches the Kinds and Actions in OCCI respectively. Hence, we can map all elements that inherit from EntityTypes, namely NodeTypes, RelationshipTypes, and GroupTypes to OCCI Kinds. Their Operations become Actions in OCCI and their Properties become Attributes. For the definition of provisionable elements, TOSCA uses the concept of EntityTemplates, namely NodeTemplates, RelationshipTemplates and GroupTemplates. NodeTemplates get be transformed into Resources and RelationshipTemplates can be transformed in Links. Their is no one-to-one cor-

TOSCA element	OCCI element
EntityType	Kind
Capability/Requirement	Mixin
Operation	Action
Property	Attribute
NodeTemplate	Resource
RelationshipTemplate	Link

Table 1: Mapping of TOSCA elements to OCCI elements.

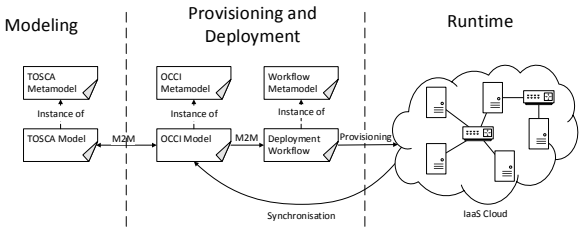

Figure 3: Model driven orchestration with TOSCA and OCCI.

respondents for GroupTypes and GroupTemplates in OCCI. Nevertheless, EntityTemplates that form a group in TOSCA can be transformed into a number of Resources and Links in OCCI. Capabilities and Requirements are used to extend a certain EntityTemplate with certain functionality. This concept can modelled by using OCCI Mixins and the implicated RelationshipTemplates can be modelled by using Links. Based on this mapping we define the concept of a fully model driven cloud orchestrator in the next Section.

4 MODEL DRIVEN CLOUD ORCHESTRATION

Both OCCI and TOSCA are actively developed and have a vibrant research and user community. Hence, they are still subject to change. By adopting a model driven approach for a cloud orchestrator implementation, it is easier to cope with these changes, since most of the code can be generated from formal metamodels and model transformations can be easily adapted accordingly. For the proliferation of the standards, it is important that open-source implementations are provided. Currently, three open-source implementations of TOSCA are available, which provide both the modelling of TOSCA templates and the orchestrated launch of the defined infrastructure: Cloud Application Management Framework (CAMF) (Loulloudes et al., 2015) and OpenTOSCA (Binz et al., 2013) are provided from academia, while Cloudify[9] is developed in the industry. OCCI support is implemented for all major open-source cloud platforms, and with OCCIWare[10] efforts exist to provide an integrated *Integrated Development Environment* (IDE) for the modelling and orchestration of cloud resources with help of the OCCI metamodel. However, the approaches listed above are neither fully model driven nor do they combine both standards. Therefore, we propose a fully model driven cloud orchestrator to provide a playground for future extensions on TOSCA and OCCI. A conceptual overview is depicted in Figure 3. In the modelling step, cloud resources are modelled according to the TOSCA metamodel. A Model-to-Model (M2M) transformation is used to generate a OCCI-compliant resource model from the TOSCA model. From this model, a deployment and provisioning workflow is generated (see e.g., (Breitenbücher et al., 2014) or (Lushpenko et al., 2015) for automatic deployment and provisioning workflow generation). The deployment and provisioning workflow then utilizes the information stored in the OCCI model to initiate the correspondent API calls over the OCCI HTTP rendering to provision the defined resources in the IaaS cloud. We will exemplify this workflow in the next Section.

5 FEASIBILITY STUDY

To evaluate the feasibility of the conceptual mapping and the orchestration workflow introduced above, we use a small TOSCA topology example and will go through the different transformation steps. Since, the current OCCI implementations do not support the PaaS layer yet, we restrict the discussion to the IaaS layer. However, the approach is valid for all service layers, since the same metamodel is used.

5.1 Setup

We implemented the M2M transformation using the *Eclipse Modeling Framework* (EMF)[11] and the Eclipse *Epsilon Transformation Language* (ETL)[12]. As a TOSCA metamodel, we used the XSD provided by the standard and utilized the OCCI metamodel provided by (Merle et al., 2015). We used this model transformator to transform the initial TOSCA topology to an OCCI resource model.
In practice, the Resources, Kinds and Mixins that can be utilized in the OCCI model depend on the specific

[9]http://getcloudify.org
[10]http://www.occiware.org/
[11]https://www.eclipse.org/modeling/emf/
[12]http://www.eclipse.org/epsilon/doc/etl/

IaaS cloud management framework. Our test setup is based on OpenStack[13] and the OCCI implementation *OpenStack OCCI Interface* (OOI)[14]. We used the Java OCCI library *jocci-api*[15] to implement an OCCI model extractor that is able to extract the available Resources, Kinds and Mixins from our OpenStack deployment and serialize it to a model artefact which conforms to the utilized OCCI metamodel. The model transformator links to these available Mixins and Kinds during the transformation of the TOSCA model.

5.2 Example TOSCA Topology

The logical structure of the TOSCA model is depicted on the right hand side in Figure 4. We use the graphical notation used in the standard (OASIS, 2016) to depict the TOSCA topology. The Properties of the NodeTemplates are omitted here for brevity. The topology consists of two virtual machines *vm1* and *vm2*, which are connected to the network *network* via *port1* and *port2* respectively. In addition, *vm1* is attached to an external block-storage device *volume*. The virtual machines, the ports, the volume and the network are modelled as TOSCA NodeTemplates. The connections between them are modelled with RelationshipTemplates. The virtual machines have the *Capabilities OperatingSystem*, which models the operating system to be used, and *Container*, which models the technical specification of the virtual machine, e.g., the number of compute cores, and RAM. To be able to attach the block-storage volume to vm1, the NodeTemplate volume has the Capability *Attachment*. To be able to bind the ports to the virtual machines and link them to the network, the ports have the Capabilities *Linkable* and *Bindable*. Finally, the corresponding *Requirements* are modelled for the NodeTemplates to be able to establish the relationships. All NodeTemplates, RelationshipTemplates, and Capabilities use TOSCA normative EntityTypes as they are defined in the TOSCA YAML specification. An overview of the utilized types is given in the second column of Table 2.

5.3 Corresponding OCCI model

The results after transforming the example into an OCCI model are shown on the left hand side of Figure 4 in an UML object diagram. The NodeTemplates vm1 and vm2 are transformed into Resources with the OCCI Kind *Compute*. The NodeTemplate volume is transformed into a Resource with the Kind *Storage*, and the NodeTemplate network is transformed into a Resource with the Kind *Network*. In the OCCI Infrastructure model, there is no corresponding element to the NodeType *Port*, and the corresponding information is stored in the Links with the OCCI Kind *NetworkInterface* which connects the virtual machines to the network. These Links are additionally associated with the Mixin *ipnetworkinterface* to allow to setup specific IP addresses. The attachment of the block-storage to vm1 is transformed into a Link of the OCCI Kind *StorageLink*. The Capabilities OperatingSystem and Container are transformed into references to the cloud-provide specific Mixins to model images with a certain operating system, and certain virtual machine types respectively. The Resource network is associated to the Mixin ipnetwork to allow the IP specific configuration of the network. An overview of the mapped OCCI Kinds and Mixins is given in the third column of Table 2.

Element	TOSCA Type	OCCI Element
vm1, vm2	tosca.nodes.Compute	Compute
network	tosca.nodes.network.Network	Network + IPNetwork
volume	tosca.nodes.BlockStorage	Storage
port1, port2	tosca.nodes.network.Port	-
OperatingSystem	tosca.capabilities.OperatingSystem	Mixin (OpenStack specific)
Container	tosca.capabilities.Container	Mixin (OpenStack specific)
Linkable	tosca.capabilities.network.Linkable	-
Bindable	tosca.capabilities.network.Bindable	-
Attachment	tosca.capabilities.Attachment	-
LinksTo	tosca.relationships.network.LinksTo	NetworkInterface + IPNetworkInterface
BindsTo	tosca.relationships.network.BindsTo	NetworkInterface + IPNetworkInterface
AttachesTo	tosca.relationships.AttachesTo	StorageLink

Table 2: Mapping of TOSCA Types to OCCI Elements.

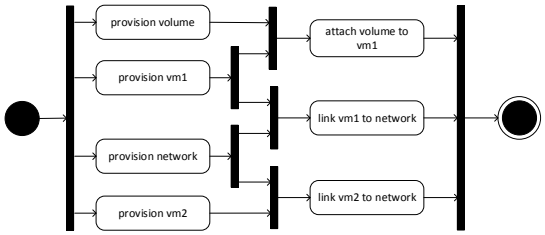

Figure 5: Example Provisioning Plan.

5.4 Provisioning order with the OCCI HTTP Protocol

Figure 5 shows the provisioning plan for the example. Each action in the diagram corresponds to a single call to the OCCI server. The Resources *vm1*, *vm2*, *network* and *volume* can be provisioned in parallel. Links can be created, when the two corresponding Resources are ready.

5.5 Findings

The generated model is fully conformant with the OCCI metamodel defined by (Merle et al., 2015).

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

[14] https://github.com/openstack/ooi

[15] https://github.com/Misenko/jOCCI-api

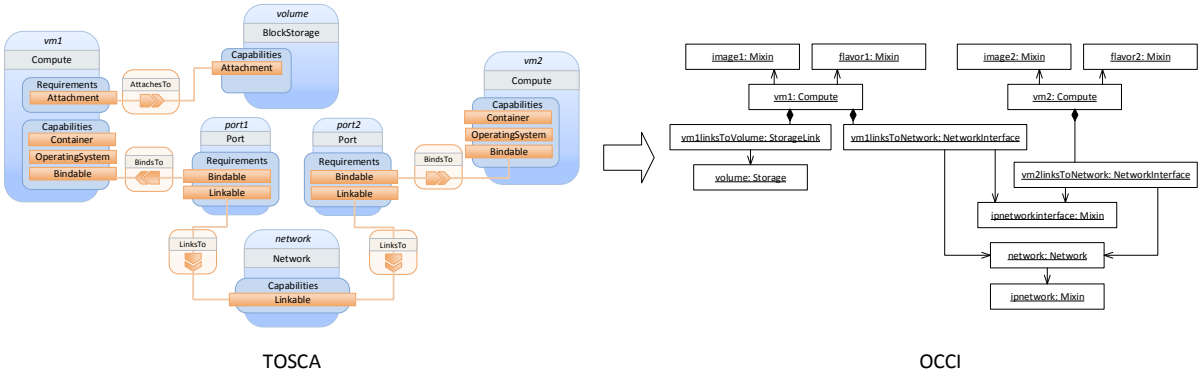

Figure 4: Transformation of an example TOSCA topology to OCCI.

We saw that most of the utilized TOSCA normative Types have a one-to-one correspondence with OCCI elements. In case of the many-to-one relationship of the ports in the example, we implemented a post-processing in the transformation step. In this step, we replaced the created port Resource and the corresponding Links with a single Link connecting the virtual machines to the Network. Such a post-processing step might be necessary also for other TOSCA normative types. There are also some cases where there is a one-to-many relationship, and one TOSCA element might correspond to several elements in OCCI, e.g. when a GroupTemplate is used to control the scaling of several other NodeTemplate. Current IaaS middlewares impose certain restrictions on what can be provisioned, such that the Capabilities defined in the TOSCA model need to be mapped to the available Mixins. Such a map would need to be provided for each IaaS infrastructure. Overall, the results of this initial feasibility study are promising and it showed that TOSCA and OCCI can be used complementary for model driven cloud orchestration.

6 RELATED WORK

Besides TOSCA, several other orchestration template formats exist, which have been developed by different cloud providers or communities, e.g., OpenStacks Heat Orchestration Template Language[16] and the Amazons CloudFormation template format[17]. They are not considered in this paper, since our focus is on interoperability of standards. Merle et al. (Merle et al., 2015) defined a metamodel for OCCI with help of EMF to provide a common basis for the generation and conformance testing of OCCI tools. This metamodel is used by (Paraiso et al., 2016) to model the deployment of applications with help of containers. Both works have been published in scope of the OCCIWare[18] project, that aims to provide a fully integrated IDE to support the whole cloud application management life cycle on multiple clouds based on OCCI. Interoperability with TOSCA is not considered. With the Eclipse Incubation Project CAMF (Loulloudes et al., 2015), Loulloudes et al. attempt to build a whole IDE to manage cloud applications with the help of TOSCA. In the scope of the project different adapters have been developed to deploy the defined TOSCA topology on multiple clouds. However, no model driven mapping and interaction with OCCI is provided. Regarding the modelling of cloud applications, several extensions to UML have been developed to capture cloud application specifics, e.g., (Bergmayr et al., 2014), (Kamali et al., 2014), (Guillén et al., 2013). In addition, Bergmayr et al. (Bergmayr et al., 2016) show how to convert refined UML models to TOSCA templates. Their approach is also based on an Ecore metamodel generated from the TOSCA XSD. These works consider the modelling of cloud applications, but do not take the mapping to certain API calls into account. Ferry et al. (Ferry et al., 2014) define a models at runtime approach for the management of cloud applications. Their approach is based on a modelling language called *CloudML* and is not based on standards.

7 CONCLUSIONS AND OUTLOOK

In the paper, we presented an approach to combine TOSCA and OCCI for model and standard driven cloud orchestration. We defined an initial mapping between the metamodel elements of TOSCA and

[16]https://wiki.openstack.org/wiki/Heat

[17]https://aws.amazon.com/cloudformation/

[18]https://www.occiware.org/

OCCI and explained how we will adopt this mapping for a model driven cloud orchestrator based on the two standards. By adopting a model driven approach, it becomes easier to incorporate changes to both evolving standards and to provide a playground for new concepts. We will continue with the evaluation of the mapping between TOSCA normative types and OCCI elements. Furthermore, we will use this approach to support the adaptation of models at runtime to keep the model of the infrastructure and the application deployment consistent with its actual state in the Cloud. This will also allow us to react to changing workloads.

ACKNOWLEDGEMENTS

We thank the Simulationswissenschaftliches Zentrum Clausthal-Göttingen (SWZ) for financial support.

REFERENCES

Bergmayr, A., Breitenbücher, U., Kopp, O., Wimmer, M., Kappel, G., and Leymann, F. (2016). From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA. In *6th International Conference on Cloud Computing and Services Science (CLOSER)*.

Bergmayr, A., Troya, J., Neubauer, P., Wimmer, M., and Kappel, G. (2014). UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In *3rd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*, pages 56–65.

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 *Service-Oriented Computing*, pages 692–695. Springer.

Breitenbücher, U., Binz, T., Kepes, K., Kopp, O., Leymann, F., and Wettinger, J. (2014). Combining Declarative and Imperative Cloud Application Provisioning Based on TOSCA. In *IC2E*, pages 87–96. IEEE Computer Society.

Ferry, N., Brataas, G., Rossini, A., Chauvel, F., and Solberg, A. (2014). Towards Bridging the Gap Between Scalability and Elasticity. In *4th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 746–751.

Guillén, J., Miranda, J., Murillo, J. M., and Canal, C. (2013). A UML Profile for Modeling Multicloud Applications. In *Service-Oriented and Cloud Computing*, pages 180–187. Springer.

Kamali, A., Mohammadi, S., and Barforoush, A. A. (2014). UCC: UML profile to cloud computing modeling: Using stereotypes and tag values. In *7th International Symposium on Telecommunications (IST)*, pages 689–694. IEEE.

Loulloudes, N., Sofokleous, C., Trihinas, D., Dikaiakos, M. D., and Pallis, G. (2015). Enabling Interoperable Cloud Application Management through an Open Source Ecosystem. *IEEE Internet Computing*, 19(3):54–59.

Lushpenko, M., Ferry, N., Song, H., Chauvel, F., and Solberg, A. (2015). Using adaptation plans to control the behavior at runtime. In Bencomo, N., Götz, S., and Song, H., editors, *CEUR Workshop Proceedings*, volume 1474. CEUR.

Merle, P., Barais, O., Parpaillon, J., Plouzeau, N., and Tata, S. (2015). A Precise Metamodel for Open Cloud Computing Interface. In *8th IEEE International Conference on Cloud Computing (CLOUD)*, pages 852–859. IEEE.

Metsch, T., Edmonds, A., and Parák, B. (2016). Open Cloud Computing Interface - Infrastructure. [Available online: `http://ogf.org/documents/GFD.224.pdf`].

Nyrén, R., Edmonds, A., Metsch, T., and Parák, B. (2016a). Open Cloud Computing Interface - HTTP Protocol. [Available online: `http://ogf.org/documents/GFD.223.pdf`].

Nyrén, R., Edmonds, A., Papaspyrou, A., Metsch, T., and Parák, B. (2016b). Open Cloud Computing Interface - Core. [Available online: `http://ogf.org/documents/GFD.221.pdf`].

OASIS (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA) 1.0. [Available online: `http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html`].

OASIS (2016). TOSCA Simple Profile in YAML Version 1.0. [Available online: `http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html`].

OMG (2014). MDA Guide rev. 2.0. OMG Document ormsc/2014-06-01 [Available Online: `http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf`].

Paraiso, F., Challita, S., Al-Dhuraibi, Y., and Merle, P. (2016). Model-Driven Management of Docker Containers. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, San Francisco, United States.