From Architecture Modeling to Application Provisioning for the Cloud by Combinding UML and TOSCA

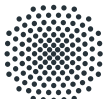
Alexander Bergmayr[1], Uwe Breitenbücher[2], Oliver Kopp[3], Manuel Wimmer[1], Gerti Kappel[1], Frank Leymann[2]
[1]Business Informatics Group, TU Wien, Austria
{bergmayr,wimmer,kappel}@big.tuwien.ac.at
[2]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{breitenbuecher,leymann}@iaas.uni-stuttgart.de
[3]Institute of Parallel and Distributed Systems, University of Stuttgart, Germany
kopp@ipvs.uni-stuttgart.de

BibTeX

```
@InProceedings{Bergmayr_2016,
  author:   = {Alexander Bergmayr and Uwe Breitenb\"{u}cher and
               Oliver Kopp and Manuel Wimmer and Gerti Kappel and
               Frank Leymann},
  title     = {From Architecture Modeling to
               Application Provisioning for the Cloud by
               Combining {UML} and {TOSCA}},
  booktitle = {Proceedings of the 6\textsuperscript{th}
               International Conference on Cloud Computing and
               Services Science},
  year      = {2016},
  publisher = {Scitepress},
  doi       = {10.5220/0005806900970108},
}
```

This publication and contributions were presented at CLOSER 2016
CLOSER 2016 Web site: http://closer.scitevents.org

The original publication is available at
http://dx.doi.org/10.5220/0005806900970108



From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA

Alexander Bergmayr[1], Uwe Breitenbücher[2], Oliver Kopp[2],
Manuel Wimmer[1], Gerti Kappel[1] and Frank Leymann[2]
[1] *Business Informatics Group, TU Wien, Austria*
{*bergmayr,wimmer,kappel*}*@big.tuwien.ac.at*

[2] *IAAS/IPVS, University of Stuttgart, Germany*
{*breitenbuecher,kopp,leymann*}*@informatik.uni-stuttgart.de*

Keywords:
TOSCA, UML, model-driven software engineering, cloud computing, cloud modeling

Abstract:
Recent efforts to standardize a deployment modeling language for cloud applications resulted in TOSCA. At the same time, the software modeling standard UML supports architecture modeling from different viewpoints. Combining these standards from cloud computing and software engineering would allow engineers to refine UML architectural models into TOSCA deployment models that enable automatic provisioning of cloud applications. However, this refinement task is currently carried out manually by recreating TOSCA models from UML models because a conceptual mapping between the two languages as basis for an automated translation is missing. In this paper, we exploit cloud modeling extensions to UML called CAML as the basis for our approach Caml2Tosca, which aims at bridging UML and TOSCA. The validation of our approach shows that UML models can directly be injected into a TOSCA-based provisioning process. As current UML modeling tools lack cloud-based refinement support for deployment models, the added value of Caml2Tosca is emphasized because it provides the glue between architecture modeling and application provisioning.

1 INTRODUCTION

With the emergence of cloud computing, the effort required for application provisioning has considerably been reduced. Cloud services can be acquired on demand [Leymann, 2011] via the Web without the need to negotiate with the cloud provider [Armbrust et al., 2010]. The low upfront costs compared to a traditional on-premise environment and the operational costs that scale with the consumed cloud services are key incentives for companies and engineers to deploy their applications on cloud environments. Several approaches for application modeling and provisioning to the cloud have been proposed [Bergmayr et al., 2014a]. Standardizing the representation of cloud-based deployment models is addressed by TOSCA [OASIS, 2013b]. A deployment model may capture deployment artifacts and targets as a topology and specify a plan how those artifacts must be deployed on the targets by a provisioning engine. At the same time, the software modeling standard UML supports architecture modeling from different viewpoints, including the class, component, and deployment viewpoint. Hence, it appears beneficial to combine standard modeling languages from software engineering and cloud computing [Jamshidi et al., 2013]. This would allow engineers to refine UML architectural models into TOSCA deployment models that enable automatic provisioning of cloud applications by means of TOSCA-compliant containers. From an UML perspective this is beneficial as it allows the application provisioning for UML deployment models. On the other hand, TOSCA deployment models can be considered in the light of UML, thereby gaining insights into the components manifested by deployed artifacts and how they are realized

from a fine-grained structural or even behavioral viewpoint. The deployment viewpoint is provided by both languages, which favors its use in a mapping process.

However, an effective conceptual mapping between UML and TOSCA as the basis for an automated translation between the two languages is still missing. As a result, the translation is currently carried out in a tedious manual step, which is only achievable if engineers are familiar with the peculiarities of both languages and capable to identify the correspondences between them at both levels intensional and extensional [Kühne, 2006]. While at the intensional level common aspects of cloud environments are captured in terms of types, they are instantiated at the extensional level by assigning concrete values to their features. Moreover, due to the generic nature of UML's deployment language, it does not natively support cloud-based deployment models, which hampers the translation between the two languages.

In this paper, we propose a fully automatic transformation approach Caml2Tosca which addresses both the intensional and extensional level of deployment models. From a UML perspective, we exploit cloud-specific extensions as a bridge between the two languages. In previous work, we presented the Cloud Application Modeling Language (CAML) [Bergmayr et al., 2014c] as a lightweight extension to UML's deployment language. It provides intensional abstractions over cloud environments. This enables engineers to create cloud-based deployment models directly in UML and refine them towards a target cloud environment. The refinement task is required as a basis for ultimately carrying out the application provisioning, which is supported by TOSCA containers, such as OpenTOSCA [Binz et al., 2013]. Hence, CAML can serve as the bridge between UML and TOSCA, where Caml2Tosca provides the glue between architecture modeling and application provisioning.

The remainder of this paper is structured as follows. We discuss the background of our work by introducing the metamodels of UML's deployment language including the CAML extensions and TOSCA in Section 2. In Section 3, we present an overview of Caml2Tosca, introduce both modeling levels intensional and extensional, and discuss in detail the conceptual mapping between UML and TOSCA. In Section 4, we validate our approach by presenting a prototypical implementation of the Caml2Tosca transformer and demonstrating it in a practical setting. The evaluation presented in Section 5 shows the practical value of our approach for engineers as current UML modeling tools are capable to adopt CAML. Finally, in Section 6, we discuss related work before we conclude in Section 7.

2 BACKGROUND

To establish the basis for our work, we introduce UML's deployment modeling concepts and the cloud-specific extensions to them we have developed in previous work [Bergmayr et al., 2014c]. Thereafter, we give an overview on TOSCA's modeling concepts most relevant for our work.

2.1 Cloud extensions to UML

CAML aims at enabling engineers to create deployment models by common cloud modeling concepts and to refine them towards a target cloud environment. Those concepts are captured by CAML's cloud library, whereas UML profiles are employed to support the refinement step. Figure 1 gives an overview of UML's metamodel where the emphasis is placed on the deployment viewpoint. The cloud-specific concepts provided by CAML's cloud library are instances of the generic deployment modeling concepts standardized by UML. They in turn are extended by environment-specific modeling concepts that embody concrete cloud services, e. g., an *M1Medium* compute service of the Amazon AWS environment[1]. CAML provides UML profiles dedicated to cloud environments. They are integrated into a common cloud profile which currently supports stereotypes for Amazon AWS/OpenStack, Google cloud platform[2], and Microsoft Azure[3].

Even though the use of profiles for concrete cloud environments in addition to a cloud library increases the complexity of CAML from a language engineering perspective. However, from an engineer's perspective, it allows them to refine deployment models towards a cloud environment [Ardagna et al., 2012] without direct modifications to the base elements as stereotypes are only associated to them. Moreover, this flexible typing mechanism shows its benefit when changing the refinement, e.g., from an *M1Medium* compute service to an *M1Large* one. It requires only to

[1]Amazon AWS: https://aws.amazon.com
[2]https://cloud.google.com
[3]https://azure.microsoft.com

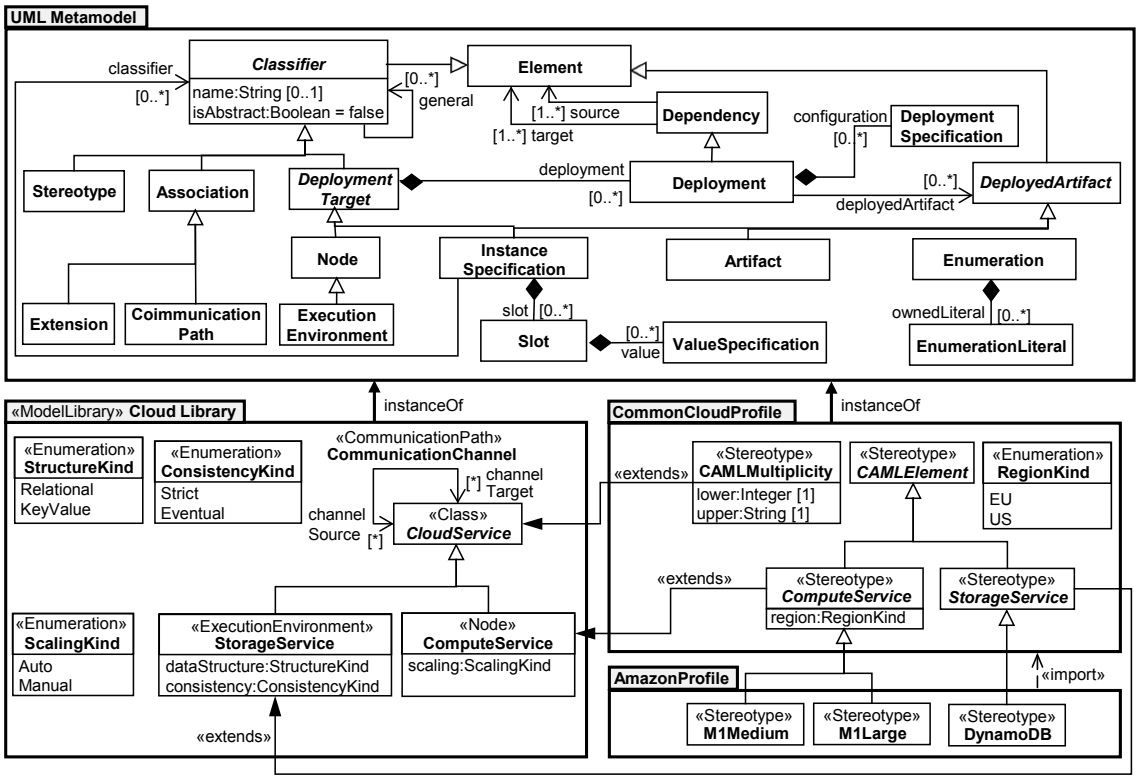


Figure 1: UML deployment concepts and CAML extensions (excerpt).

un-apply and re-apply the stereotypes.

Turning the focus on the cloud library, it is built around the concept of *cloud service*, which is considered as a virtual resource that is expected to be provided and managed by a cloud environment. Specializations of the *cloud service* concept capture common cloud environment capabilities such as computational capacity and data management. The former is embodied by the *compute service* concept. The elastic nature of a cloud environment is managed by dedicated *scalability* strategies. For instance, compute services can be automatically provisioned and released within a certain boundary specified by the *CAMLMultiplicity* (see cloud profile). A *storage service* refers to the data management capabilities of cloud environments. It captures diverse solutions for structuring application data [Fehling et al., 2014] and increasing their availability by relaxing consistency [Vogels, 2009]. To represent relationships between cloud services, *communication channels* are employed. They can also be stereotyped, e. g., to express different forms of communication via specific protocols.

Figure 2 shows how the cloud library and the cloud profile dedicated to Amazon AWS can be applied by means of a simple deployment model. It consists of an automatically scaled compute service that is connected to a key-value storage service for managing data in an eventually consistent way. Both cloud services are refined towards Amazon's cloud environment. The refinement is accomplished by applying stereotypes of the respective UML profile. The modeled compute service refers to Amazon's "M1Medium" service offering, whereas "DynamoDB" is used for the required cloud storage capabilities.

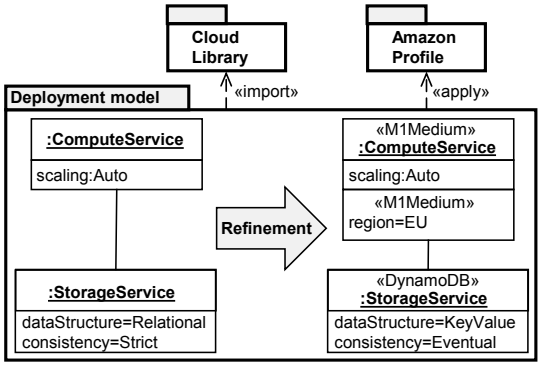


Figure 2: Example deployment model and its refinement towards the Amazon cloud environment.

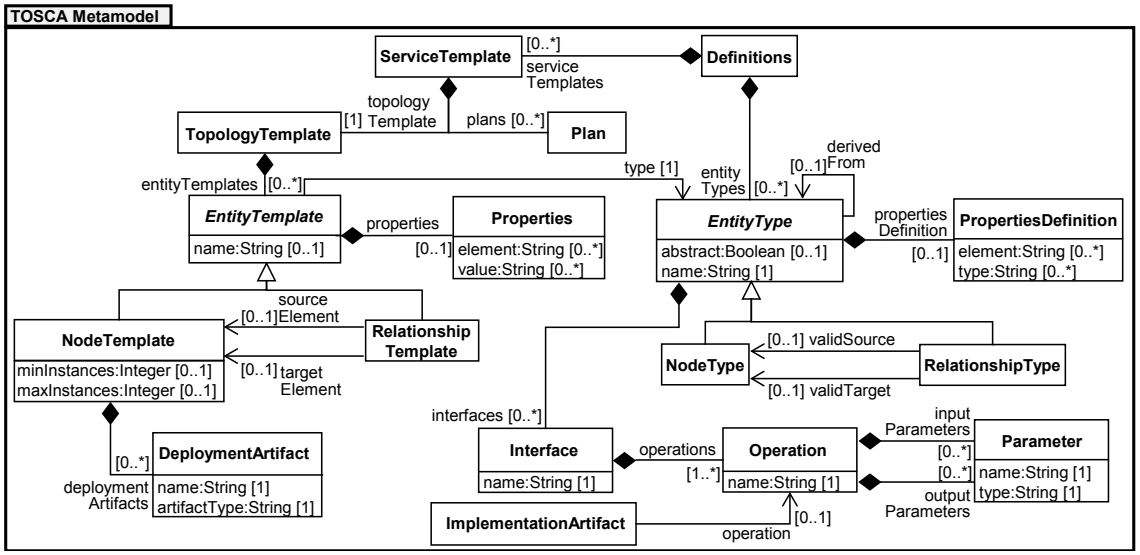


Figure 3: TOSCA metamodel (XML-based language [OASIS, 2013b]).

2.2 TOSCA metamodel

The main concepts of the TOSCA metamodel relevant for our work are depicted in Figure 3. We simplified some constructs, where possible, for the sake of comprehension. For more details, we refer interested readers to the TOSCA specification [OASIS, 2013b] and the primer [OASIS, 2013a]. A compact overview of TOSCA is given by Binz et al. [Binz et al., 2014]. TOSCA consists conceptually of two parts. The *topology template* is used to describe the structure of cloud applications, whereas a deployment *plan* is basically a workflow model that can be invoked to execute certain tasks, e. g., the provisioning of a compute service instance.

Considering the *topology template* in more detail, it is a directed graph that consists of *node templates* and *relationship templates*. The former represents artifacts related to cloud applications and environments, while the latter defines dependencies between those artifacts, e. g., a PHP-based web application is hosted on a web server which in turn is hosted on a compute service. Both kinds of templates are typed. The type of a *node template* is defined by a *node type*, similarly is a *relationship type* used to define the type of a *relationship template*. The respective types can be used to specify a *properties definition* and management *interfaces* including *operations* with *input* and *output parameters*. For example, a compute service type may define a property "public_address" and the operations "start" and "shut down". A template of this type enables the configuration of the public address that is used to access the compute service instance and provides the defined operations to start the compute service instance and shut it down. Generally, types can inherit from each other, which fosters its reuse: compute service types specific to a cloud environment, e. g., Amazon EC2 compute services, may inherit from a generic compute service type that provides common properties and operations. An *implementation artifact* is used to provide a concrete implementation for an operation. In theory, any programming language can be used to implement an operation. From a technical perspective, a TOSCA runtime container must be capable to invoke the implementation of an operation.

Finally, a *DeploymentArtifact* refers to a concrete implementation of a *node template*. For example, a PHP-based web application is considered as a deployment artifact. Also, binaries of web server are *deployment artifacts*. They are required to actually install it.

3 CAML2TOSCA

Bridging the gap between UML and TOSCA leverages not only continuous modeling support for cloud applications but also allows engineers to carry out the application provisioning. In this respect, the target of the application provisioning is a cloud environment. UML provides capabilities to model application architectures from different viewpoints. On the other hand, TOSCA enables the provisioning, management, and termination of

cloud services and applications. In the following, we give first a high-level overview of the underlying approach to combine UML and TOSCA. Thereafter, we clarify how the intensional and extensional modeling levels introduced by UML and TOSCA relate to each other as this is essential for providing a useful mapping between them. Finally, we propose an effective conceptual mapping between UML and TOSCA where CAML takes the role of capturing cloud-specific features from the perspective of UML.

3.1 Overview

Considering the high-level overview presented in Figure 4, the entry point to the CAML2TOSCA approach is a deployment model capturing the desired state of the cloud application provisioning. Creating the deployment model is considered as part of architecture modeling which usually includes to produce a variety of models (or views on models), each of which addressing a concern from a certain viewpoint. In particular, CAML deals with the class, component, and deployment viewpoint. A deployment model refined towards a target cloud environment is considered as input to a tool chain capable to produce a standard-compliant executable cloud service archive (CSAR). We collectively refer to the set of tools comprised by this chain as *TOSCA tools*. They allow engineers to automatically generate a TOSCA-based representation consisting of type definitions and the topology template that corresponds to the injected UML deployment model refined by CAML. Based on those type definitions, the implementations of management operations required for the application provisioning are automatically injected [Kopp et al., 2013]. To enable their execution in an appropriate order, a management plan is orchestrated [Breitenbücher et al., 2014a]. All the produced artifacts by the *TOSCA tools* constitute the CSAR. It can be executed by TOSCA-compliant runtime containers. The presented conceptual

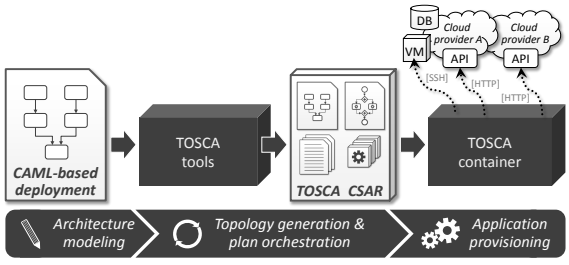


Figure 4: Overview of the CAML2TOSCA approach.

mapping between UML and TOSCA provides the basis for automating the generation of a typed TOSCA topology template from a CAML-based deployment model.

3.2 Modeling levels

Clarifying how the intensional and extensional level introduced by UML and TOSCA relate to each other is essential for realizing a conceptual mapping between them. Figure 5 depicts the core concepts of UML and TOSCA to create intensional and extensional deployment models. While in UML the various sub-meta-classes of *classifier* are employed to model application architectures from an intensional perspective, the corresponding meta-class in TOSCA is *type*. For instance, the compute service with the *region* property is considered as a concrete UML classifier or TOSCA type, respectively. At the extensional level, the meta-classes *instance specification* of UML and *template* of TOSCA correspond to each other.

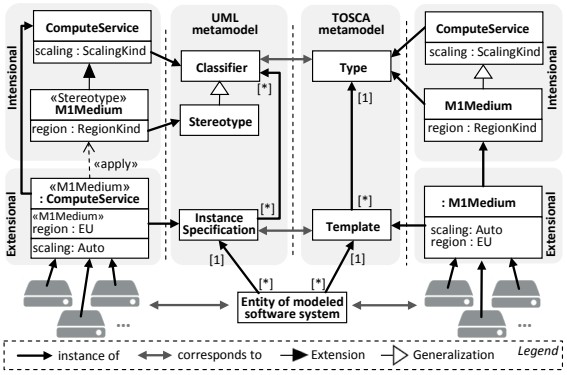


Figure 5: Comparison of TOSCA's and UML's intensional level and extensional level.

Indeed, the stereotype applied to an instance specification needs to be taken into consideration to infer the corresponding type of a produced template because TOSCA does not directly support stereotypes. Still, a stereotype can be considered as a type in TOSCA, which inherits the properties of the base class extended by the stereotype. For instance, the extension between the "M1Medium" stereotype and the compute service is represented as a generalization between the corresponding TOSCA types. This solution implies that a stereotyped artifact of an extensional deployment model needs to be translated into a TOSCA template that is typed by the type of the corresponding stereotype instead of its direct classifier. As a result, the stereotyped compute

service instance at the extensional level is represented as a TOSCA template of type "M1Medium" instead of *compute service*.

Generally, elements modeled at the extensional level are employed to designate elements of a (software) system in the form of a one-to-one mapping concerning their individual features [Kühne, 2006]. Considering the instance of the "M1Medium" compute service, it designates Amazon's M1 medium compute service located in the EU. Obviously, in the context of deployment modeling, several compute services may be comprised by a running cloud application, which requires a multiplicity concept not only for elements of intensional models but also for extensional models. Regarding the former, a one-to-many or many-to-many multiplicity is typically supported by default, i. e., it is not explicitly defined by engineers but integral part of a metamodel. Multiplicities for elements at the extensional level determine the lower-bounds and upper-bounds of real-world entities that are considered as instances of these elements. This is useful for cloud applications where cloud services are provisioned and released on-demand. Clearly, to specify more sophisticated custom rules that trigger the provisioning of new cloud services or force to release them, dedicated language support seems to be required (cf. e. g., [Kritikos et al., 2014]).

3.3 Conceptual Mapping

The proposed conceptual mapping is generic in the sense that any UML deployment model can be translated into a corresponding TOSCA model. As CAML provides standard compliant extensions to UML in terms of custom types, they can be considered as supplementaries to UML's metamodel. In this way, they can be treated in the mapping process similar to UML standard meta-classes.

Generally, concrete classifiers of CAML can be represented as node types in TOSCA and their use at the extensional level can be represented in terms of node templates. A specific case are stereotypes which require special treatment for inferring the type information assigned to produced node templates.

Moreover, standard relationships of UML typically applied for deployment modeling, i. e., *deployment* and *dependency* are also addressed. They are usually applied directly at the extensional level. As a result, they are represented as instances of the corresponding meta-classes at the extensional level instead of an *instance specification*. Both relationship types can certainly be stereotyped if additional features are required or certain vocabularies with specific semantics need to be introduced.

3.3.1 Intensional level

The mapping presented in Table 1 acts as the basis to generate TOSCA type definitions. In fact, *node types* can be generated from concrete *classifers* that constitute CAML's cloud library and profiles. Basically, the signature of a classifier including its name and whether it is abstract or concrete can straightforwardly be mapped to a node type. A *property* of a classifier can be mapped to a *properties definition* of the corresponding node type. Similarly, an *operation* including its input and output *parameters* can be mapped to an *operation* of a node type and added to the exposed management *interface*.

Regarding stereotypes, we need to distinguish whether they extend a base class or inherit from another stereotype. In the former case, the generated node type specializes the corresponding node type of the stereotype's base class, whereas in the latter case, it inherits from the node type of the super stereotype.

Finally, an *association* can be mapped to a *relationship type* where the first *memberEnd* of the *association* corresponds to the *validSource* and the second to the *validTarget*.

Table 1: Mapping for intensional level.

UML/CAML	TOSCA
uml:Model m	**add** Definitions d
uml:Classifier c	**add** NodeType nt **if** c.oclIsTypeOf(uml:Class) nt.name = c.name nt.abstract = c.isAbstract nt.derivedFrom = -- obtain CAML base element **if**(c.oclIsTypeOf (Stereotype) **and** c.extension.oclIsDefined()) **else** c.general **add** PropertiesDefinition pd **if** c.attribute.notEmpty() **add** Interface i **if** c.ownedOperation.notEmpty()
uml:Property p	**for each** uml:Property p **in** c.attribute pd.element = -- create element definition from p.name pd.type = -- create element type from p.type
uml:Operation uo	**add** Operation o **for each** uml:Operation uo **in** c.ownedOperation o.name = uo.name **add** Parameter p **for each** uml:Parameter up **in** uo.ownedParamer p.name = up.name p.type = up.type o.inputParameters = uo.ownedParameter **where** p.direction = in o.outputParameters = uo.ownedParameter **where** p.direction = out nt.interfaces.operations = o
uml:Association a	**add** NodeRelationship nr nr.name = a.name nr.validSource = a.memberEnd.at(1) nr.validTarget = a.memberEnd.at(2)

3.3.2 Extensional level

The mapping presented in Table 2 provides the basis to generate a TOSCA *topology template* from a UML deployment model refined towards a cloud environment. Thus, the emphasis is now placed on *instance specifications* and the generation of corresponding *node templates* and *relationship templates* from them. If a stereotype is applied to an instance specification the inferred type refers to the node type of the stereotype instead of the type assigned to the instance specification. Indeed, we need to consider the properties of both the type assigned and the stereotype applied to the instance specification. If the assigned type refers to an *artifact*, a TOSCA *deployment artifact* must be created additionally. It describes the type of an artifact that is actually deployed, e.g., a web application implemented in PHP.

The *deployment* relationship in UML is directly mapped to the predefined TOSCA relationship template *ttl*[4]*:hosted on*. Even though there is also a predefined TOSCA relationship template *ttl:depends on* that fits well to the *dependency* relationship in UML, we need to consider the fact that a dependency may be stereotyped to specialize its semantics. As a result, a possibly applied stereotype determines on the type of the produced relationship template.

Table 2: Mapping for extensional level.

UML/CAML	TOSCA
uml:Model m	**add** Definitions d **add** ServiceTemplate st st.name = m.name **add** TopologyTemplate tt
uml:InstanceSpecification is **switch**(is.classifier.oclType())	
case : uml:Classifier c **using** rst = cs.getAppliedSubstereotype (cpp:CAMLElement) mst = cs.getAppliedStereotype (cpp:CAMLMultiplicity)	**add** NodeTemplate nt nt.name = is.name nt.type = is.classifier **if** rst.oclIsUndefined() **else** rst.name nt.minInstances = is.getValue(mst, 'lower') nt.maxInstances = is.getValue(mst, 'upper') **add** Properties **add** Element e, Value v **for each** uml:Slot sl **in** is.slots e = sl.definingFeature.name v = sl.getValues() **add** Element e, Value v **for each** uml:Property p **in** rst.attribute e = p.name v = is.getValue(rst, p.name) **add** DeploymentArtifact da **if**(c = uml:Artifact) da.name = c.name da.artifactType = -- obtain it from c.filename
case uml:Association a **using** rst = cc.getAppliedSubstereotype (ccp:CAMLElement)	**add** RelationshipTemplate rt rt.name = is.name rt.type = is.classifier **if** rst.oclIsUndefined() **else** rst.name **add** Properties **add** Element e, Value v **for each** uml:Property p **in** rst.attribute e = p.name v = is.getValue(rst, p.name)
uml:Deployment d	**add** RelationshipTemplate rt name = "HostedOn" type = ttl:HostedOn sourceElement = d.source targetElement = d.target
uml:Dependency d **using** rst = cc.getAppliedSubstereotype (ccp:CAMLElement)	**add** RelationshipTemplate rt name = "DependsOn" **if** rst.oclIsUndefined() **else** rst.name type = ttl:DependsOn **if** rst.oclIsUndefined() **else** rst.name **add** Properties **add** Element e, Value v **for each** uml:Property p **in** rst.attribute e = p.name v = is.getValue(rst, p.name) sourceElement = d.source targetElement = s.target

4 VALIDATION

To validate the practical feasibility of our approach, we implemented the CAML2TOSCA transformer on top of Eclipse[5] and integrated it into the open-source ecosystem *OpenTOSCA*, which supports modeling, provisioning, and managing of TOSCA-based cloud applications. In the following, we present the tool chain leveraging architecture modeling and application provisioning based on UML and TOSCA. Thereafter, we introduce a detailed application scenario to validate our approach in a real-life scenario.

4.1 Prototypical Implementation

We implemented a Java-based open-source prototype of the CAML2TOSCA transformer[6]. It is grounded in the presented conceptual mapping between UML and TOSCA (see Section 3.3). From a technical perspective, the CAML2TOSCA model transformer comes as an Eclipse plug-in where the conceptual mapping between UML and TOSCA is implemented by means of the proven model transformation language ATL[7] [Jouault et al., 2008]. To employ ATL, the source and target metamodels must be available in a compatible format. Metamodels represented by EMF's[8] Ecore are directly supported by ATL. However, language definitions expressed in terms of an XML Schema are not compatible with it. For that purpose, we automatically

[4]ttl: Tosca type library

[5]Eclipse: http://www.eclipse.org

[6]The open-source prototype is vailable at: https://github.com/alexander-bergmayr/caml2tosca

[7]ATL: https://eclipse.org/atl

[8]Ecore: https://eclipse.org/modeling/emf

reverse-engineered an Ecore-based representation from the XML-based metamodel of the TOSCA standard [Neubauer et al., 2015]. To produce an XML-based representation of the TOSCA models generated by the CAML2TOSCA transformer, the standard serialization mechanisms provided by EMF are exploited. EMF provides dedicated model converters to translate between the serialization formats used by Ecore and XML Schema. We integrated the CAML2TOSCA transformer into OpenTOSCA[9] as shown in Figure 6. The resulting system involves two kinds of users: (i) *Application engineers* model their applications in UML, which are then automatically provisioned to be used by (ii) *business users*. To create the high-level architecture of a cloud application including its desired deployment on a cloud environment, engineers can employ the Papyrus Eclipse UML[10] modeling tool for which CAML plug-ins are available. In a first step, the deployment model is refined towards the selected target cloud environment by applying the CAML cloud profile, see ①. This ensures that a properly typed TOSCA-based representation can be generated from a deployment model created in UML and refined by CAML. To support the application provisioning, the CAML deployment model is translated into a corresponding TOSCA topology topology, see ②. As the generated TOSCA representation describes only the desired deployment topology, executable artifacts for the actual application provisioning are required, e. g., a script to install a web server on a virtual machine or to configure a web application. Those artifacts are automatically injected into the TOSCA topology by Winery [Kopp et al., 2013], which is part of OpenTOSCA to model TOSCA-based cloud applications. Winery injects implementations of all management operations required for the provisioning by looking up the corresponding node types and relationship types in a local repository and embedding the required artifacts and type definitions into the TOSCA topology, see ③. To execute the required operations, a BPEL-based deployment plan is automatically generated [Breitenbücher et al., 2014a]. It orchestrates the operation implementations in an appropriate order, see ④. The generated deployment plan, topology template, and all required artifacts and type definitions are packaged as portable CSAR. The OpenTOSCA container consumes the CSAR for

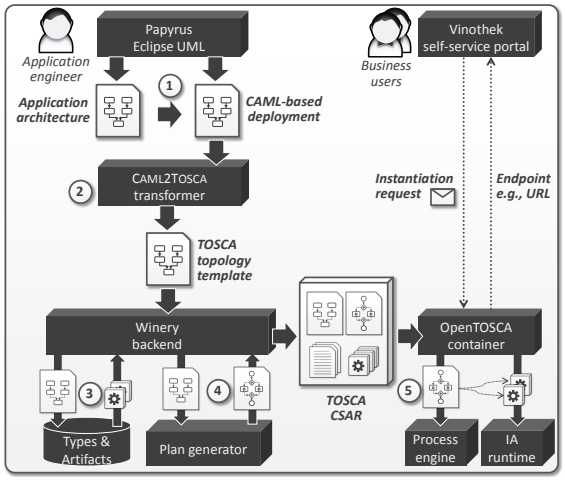


Figure 6: Tool chain for application modeling and provisioning to the cloud.

installing it. The CSAR enables the container to provision the modeled application. In fact, the generated deployment plan is executed on a local workflow engine, see ⑤. As TOSCA allows implementations of management operations using arbitrary technologies, operation implementations that are not executed in the application's target cloud environment, e. g., local services that wrap cloud environment APIs, are deployed on a local IA runtime (Implementation artifact runtime) and bound to the deployment plan [Wettinger et al., 2014b, Wettinger et al., 2014a]. Finally, business users can trigger the provisioning of cloud applications via the Vinothek [Breitenbücher et al., 2014b], which is a self service portal that offers those applications.

4.2 CAML2TOSCA by Example

To demonstrate our approach, we refer to the Moodle learning platform[11], which is a LAMP-based application. The CAML deployment model of Moodle is shown in Figure 7. It represents the application structure together with an excerpt of domain classes, the manifestation of these components by deployable artifacts, and a possible deployment of these artifacts on OpenStack. The depicted extensional deployment model contains two components, *MoodleWebApp* and *MoodleDB*. They are connected to a LAMP-based application stack on top of two compute services: the first compute service hosts the business tier while the data tier is hosted on the second one. Moreover, the two compute services are refined towards Open-

[9] OpenTOSCA: https://github.com/OpenTOSCA
[10] Papyrus: https://eclipse.org/papyrus
[11] https://moodle.org

Stack. The stereotypes are covered by CAML's cloud profile, where the base elements to which they can be applied are captured by the cloud library of CAML. It also covers a set of base types to specify application stacks for cloud deployment models.

The CAML deployment model is translated into a functionally equivalent TOSCA topology template by the CAML2TOSCA transformer. This topology is enriched by Winery and packaged into a CSAR, which can be consumed by the OpenTOSCA container. An excerpt of the CSAR is depicted in Figure 8. It shows the definition of the "Apache Web Server" type assigned to a template. Properties of the "Apache Web Server"

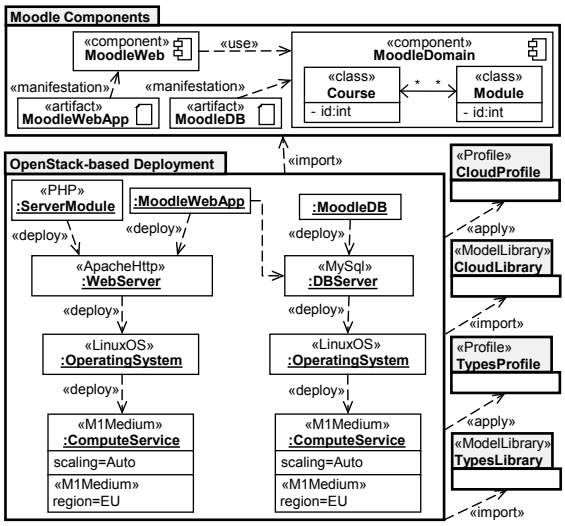


Figure 7: CAML deployment model for Moodle on OpenStack.

CSAR

Definitions

```
<NodeTemplate name="Apache Web Server" type="ns2:ApacheWebServer">
 <Properties>
  <ns2:ApacheWSProperties xmlns:ns2="..." xmlns="...">
   <httpdport>80</httpdport>
  </ns2:ApacheWSProperties>
 </Properties>
</NodeTemplate>
<NodeType name="ApacheWebServer">
 <DerivedFrom typeRef="ns1:WebServer"/>
 <PropertiesDefinition element="tns:ApacheWSProperties"/>
  <Interfaces>
   <Interface name="http://docs.oasis-open.org/tosca/ns/interfaces/lifecycle">
    <Operation name="install"/>
    <Operation name="deploy"/>
   </Interface>
  </Interfaces>
</NodeType>
```

Types

```
<xs:complexType name="tApacheWSProperties">
 <xs:element default="80" name="httpdport" type="xs:int"/>
</xs:complexType>
<xs:element name="ApacheWSProperties" type="tApacheWSProperties"/>
```

Figure 8: CSAR for Moodle on OpenStack.

type are captured by an XML schema to support their validation when they are used. To automatically provision the Moodle application, the used node types and relationship types must be available in the model repository of Winery for injecting the required provisioning logic. We achieved this by semantically aligning CAML's cloud profile with the respective TOSCA types. The CAML deployment model can thus be transformed into a corresponding TOSCA representation without changing its overall semantics. As a result, CAML deployment models can directly be injected into a TOSCA-based provisioning process. In addition, behavioral aspects, i. e., the implementations of operations required for the provisioning can be embedded seamlessly without additional manual effort when creating the CAML deployment model. However, very specific stereotypes or TOSCA types, respectively, may require further manual adaptation of the produced TOSCA topology or deployment plan. For example, a special script may need to be added to the TOSCA topology in order to automatically establish a connection between two custom business components.

5 EVALUATION

Today, several modeling tools support UML. The aim of this study is to investigate on their methods for deployment modeling in general and support for cloud-based deployment targets in particular. We created the deployment model of Figure 7 in each of the selected tools with the aim to answer the research question as follows.

RQ: What are the methods of current UML modeling tools to represent cloud-based deployment models and what are the practical implications?

Our comparison criteria mainly address (*i*) the levels at which deployment models are represented, (*ii*) the support for multiplicities not only at type level but also at instance level, and (*iii*) the offered possibilities to refine environment-independent deployment models towards a selected cloud environment. Regarding the second criterion, the support for multiplicities at the instance level is not directly supported by the UML standard. However, defining them for modeled application artifacts and cloud services appears to be of particular importance. The multiplicities determine the lower bounds of running application artifacts and cloud services as well as their upper bounds since in a highly scalable cloud environment [Vaquero et al.,

2011] they are provisioned as their demand increases but also released once their demand decreases.

Comparison criteria. As deployment models are specified by exploiting the intensional and extensional level, the first criterion refers exactly to the capability of UML modeling tools to support both levels. The second criterion is dedicated to the support of multiplicities at the extensional level because this seems of particular interest for modeling cloud applications. Finally, to investigate the need of UML libraries and UML profiles covering cloud-specific domain concepts, the third criterion addresses the support of current UML modeling tools for refining deployment models towards a target cloud environment.
$-CC1$: Is deployment modeling supported at both levels intensional and extensional?
$-CC2$: Is the definition of multiplicities supported for elements at the extensional level?
$-CC3$: Is the refinement of deployment models towards a cloud environment supported?

Selected tools. The selected set of commercial and open-source UML modeling tools that claim to support deployment modeling are summarized in Table 3.

Evaluation procedure. First, we first imported the deployment model of Figure 7 including the required cloud library and profile into the modeling tools. Thereafter, we evaluated the capabilities of the modeling tools offered in the standard settings and explored the different wizard configurations if supported.

Results. The results of our study are summarized in Table 3. All evaluated UML modeling tools support both the intensional and extensional level to create deployment models. Support for the extensional level slightly differs between the tools because some of them offer to directly create instances of deployment artifacts, where the respective instance specification assigned with a classifier is generated automatically, i.e., without additional user interaction. Regarding multiplicities at the extensional level, only Enterprise Architect supports them by default. Most of the tools lack cloud-based refinement support for UML deployment models. Only Rational Software Architect introduces a cloud node concept which is resembled by CAML's compute service. This emphasizes the value of CAML not only to leverage the refinement of deployment models towards a cloud environment but also as a bridge from UML to TOSCA. Finally, even though this evaluation focuses on UML, it is worth noting that

Table 3: Comparison results

		UML				
		Deployment		Multiplicities		Cloud Support
Modeling Tool		Intensional level	Extensional level	Intensional level	Extensional level	
Name	Version					
Altova UML	2015	supported	supported via Object Diagram	supported	not supported	no support
ArgoUML	0.34	supported	directly supported	supported	not supported	no support
Enterprise Architect	9.3	supported	supported via Object Diagram	supported	supported	SOMF-based
Magic Draw	18.0	supported	directly supported	supported	not supported	no support
Rational Software Architect	8.5.1	supported	directly supported	supported	not supported	partial support
Papyrus	1.0.0	supported	supported via Object Diagram	supported	not supported	no support
Visual Paradigm	12.1	supported	supported via Object Diagram	supported	not supported	no support

Enterprise Architect supports describing and analyzing cloud environment topologies as part of the service-oriented modeling framework (SOMF)[12].

6 RELATED WORK

Several approaches introduce a considerable set of cloud modeling concepts and propose tool support to automate the application provisioning. The Blueprint [Nguyen et al., 2011] approach describes service-based applications by coarse-grained deployment artifacts that are connected with concrete cloud services. Describing such cloud services in an XML-based language is supported by CloudML-UFPE [Gonçalves et al., 2011]. Similarly, Zephyrus [Cosmo et al., 2014] allows specifying service types associated with constraints, e.g., the maximum number of replicas of an application component, that are attempted to be satisfied when an extensional deployment model is derived from an initial set of types. Dependencies between them are expressed via ports through which the derivation of possible deployment configurations can be automated. The notion of ports is also exploited by CloudML [Ferry et al., 2013] to specify cloud-oriented deployments. Both Zephyrus and CloudML come with dedicated tools to automate the software provisioning based on specified deployment models, which is also supported by the approach of Holmes [Holmes, 2014] and StratusML [Hamdaqa and Tahvildari, 2015]. Creating deployment models is addressed by CloudMIG [Frey and Hasselbring, 2011] where the main focus is on migration scenarios to the cloud. Migrating existing applications to the cloud is also addressed by MOCCA [Leymann

[12]SOMF: http://www.sparxsystems.com/somf

et al., 2011]. It allows the representation of the architecture and deployment of existing applications. Moreover, it is capable to derive an optimal clustering of architectural elements and concrete implementation units that are assigned to virtual resources of a cloud environment. They are described in the Open Virtualization Format [DMTF, 2013] (OVF) to provide support for the actual resource provisioning. OVF is extended by RESERVOIR-ML [Chapman et al., 2012] with primitives to describe applications in terms of components and elasticity rules that control the virtual machine configurations in an OpenNebula[13] cloud environment.

Modeling concepts of all these approaches are reflected by our approach on a level of abstraction that supports engineers in the design and deployment of cloud applications. As a result, modeling concepts required to, e. g., achieve the optimization of an application deployment (cf. [Frey and Hasselbring, 2011]), express elasticity rules (cf. [Chapman et al., 2012]), or exploit workload models (cf. [Ferry et al., 2013, Hamdaqa and Tahvildari, 2015]) are not completely captured by our approach. However, it enables deployment models to be specified in such a way that they are seamlessly applicable on UML models usually created throughout application modeling activities, e.g., class models to specify the realization of components, because our approach is based on UML. Consequently, well-connected model-based views on cloud applications from both perspectives environment-independent as well as environment-specific are supported. The refinement of the former is supported by UML profiles in general [Bergmayr et al., 2014b] and CAML's cloud profile in particular. Cloud-specific stereotypes allow extensional deployment models to be refined towards a cloud environment without re-modeling of the deployed artifacts as it is the case for existing cloud modeling approaches. This additional flexible typing dimension and the benefits of a multi-viewpoint language exploited by our approach differentiates it from existing approaches.

Cloud modeling support based on UML is proposed by MULTICLAPP [Guillén et al., 2013]. It presents a UML profile for the purpose of representing applications components that are expected to be deployed on a cloud environment. As MULTICLAPP does not support refining application components towards cloud services provided by a certain cloud environment, our approach is different in the sense that CAML's cloud profile is applied to achieve exactly this environment-specific refinement. Moreover, translating deployment models refined towards a cloud environment into TOSCA brings management support to engineers as TOSCA-compliant containers enable automatic application provisioning based on extensional deployment models.

[13] OpenNebula: http://opennebula.org

7 CONCLUSION

Our proposed CAML2TOSCA approach bridges the gap between UML and TOSCA. As a result, engineers are capable to combine the capabilities of both languages where the deployment viewpoint is exploited for realizing the conceptual mapping between them. Cloud-specific extensions to UML called CAML are exploited to accomplish the bridge towards TOSCA. To automate the translation from UML to TOSCA, we implemented the CAML2TOSCA transformer. It is grounded in the presented conceptual mapping between the two languages and provides the necessary glue to leverage a full-fledged tool chain for architecture modeling and application provisioning based on UML and TOSCA. While the evaluation of our approach shows its practical value, several lines of future work need to be investigated. We aim to realize a repository of common deployment types applicable to both UML and TOSCA. Bi-directional transformations at the intensional level can play a key role to synchronize those types between UML and TOSCA and possibly other languages, such as CloudML. Finally, providing simulation support for deployment models to support prediction about non-functional properties such as costs and performance before the actual application provisioning is carried out seems highly desirable. We plan to explore how fUML [OMG, 2013] can be employed to provide behavioral semantics for CAML in a similar way as it can be used to define behavioral semantics of MOF-based metamodels [Mayerhofer et al., 2013].

ACKNOWLEDGEMENTS

This work is co-funded by the EC, grant no. 317859 (ARTIST project), and the German government, grant no. 03ET4018B (NEMAR project).

REFERENCES

Ardagna, D., Nitto, E. D., Casale, G., Petcu, D., Mohagheghi, P., Mosser, S., Matthews, P., Gericke, A., Ballagny, C., D'Andria, F., Nechifor, C., and Sheridan, C. (2012). MODAClouds: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. In *MISE@ICSE*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A View of Cloud Computing. *Commun. ACM*, 53(4).

Bergmayr, A. et al. (2014a). Cloud Modeling Languages by Example. In *SOCA*.

Bergmayr, A. et al. (2014b). JUMP - From Java Annotations to UML Profiles. In *MODELS*.

Bergmayr, A. et al. (2014c). UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In *CloudMDE@MoDELS*.

Binz, T. et al. (2013). OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *ICSOC*.

Binz, T. et al. (2014). TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services*. Springer.

Breitenbücher, U. et al. (2014a). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *IC2E*.

Breitenbücher, U. et al. (2014b). Vinothek – A Self-Service Portal for TOSCA. In *ZEUS*.

Chapman, C., Emmerich, W., Márquez, F. G., Clayman, S., and Galis, A. (2012). Software Architecture Definition for On-Demand Cloud Provisioning. *Cluster Comput.*, 15(2).

Cosmo, R. D., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., and Agahi, A. (2014). Automated synthesis and deployment of cloud applications. In *ASE*.

DMTF (2013). DMTF, Open Virtualization Format (OVF). Version 2.0.0.

Fehling, C. et al. (2014). *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. (2013). Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In *CLOUD*.

Frey, S. and Hasselbring, W. (2011). The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. *Intl. J. Advances in Software*, 4(3&4).

Gonçalves, G. E., Endo, P. T., Santos, M. A., Sadok, D., Kelner, J., Melander, B., and Mångs, J. (2011). CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds. In *CloudCom*.

Guillén, J., Miranda, J., Murillo, J. M., and Canal, C. (2013). A UML Profile for Modeling Multicloud Applications. In *ESOCC*.

Hamdaqa, M. and Tahvildari, L. (2015). Stratus ML: A Layered Cloud Modeling Framework. In *IC2E*.

Holmes, T. (2014). Automated Provisioning of Customized Cloud Service Stacks using Domain-Specific Languages. In *CloudMDE@MoDELS*.

Jamshidi, P., Ahmad, A., and Pahl, C. (2013). Cloud Migration Research: A Systematic Review. *IEEE T. Cloud Computing*, 1(2).

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39.

Kopp, O. et al. (2013). Winery – modeling tool for TOSCA-based cloud applications. In *ICSOC*.

Kritikos, K., Domaschka, J., and Rossini, A. (2014). SRL: A scalability rule language for multi-cloud environments. In *Proc. of Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*, pages 1–9.

Kühne, T. (2006). Matters of (meta-)modeling. *Software and System Modeling*, 5(4).

Leymann, F. (2011). Cloud Computing. *it - Information Technology*, 53(4).

Leymann, F. et al. (2011). Moving Applications to the Cloud: An Approach Based on Application Model Enrichment. *Int. J. Cooperative Inf. Syst.*, 20(3).

Mayerhofer, T., Langer, P., Wimmer, M., and Kappel, G. (2013). xMOF: Executable DSMLs based on fUML. In *SLE*.

Neubauer, P. et al. (2015). XMLText: From XML Schema to Xtext. In *SLE*.

Nguyen, D. K., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M. P., and van den Heuvel, W. (2011). Blueprint Template Support for Engineering Cloud-Based Services. In *ServiceWave*.

OASIS (2013a). TOSCA Primer v1.0. http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html.

OASIS (2013b). TOSCA v1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

OMG (2013). Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.1. http://www.omg.org/spec/FUML/1.1.

Vaquero, L. M., Rodero-Merino, L., and Buyya, R. (2011). Dynamically Scaling Applications in the Cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1).

Vogels, W. (2009). Eventually Consistent. *Commun. ACM*, 52(1).

Wettinger, J. et al. (2014a). Streamlining Cloud Management Automation by Unifying the Invocation of Scripts and Services Based on TOSCA. *IJOCI*, 4(2).

Wettinger, J. et al. (2014b). Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA. In *CLOSER*.