Cloud Modeling Languages by Example

Alexander Bergmayr, Manuel Wimmer, Gerti Kappel
Business Informatics Group
Vienna University of Technology, Austria
Email: (bergmayr/wimmer/gerti)@big.tuwien.ac.at

Michael Grossniklaus
Department of Computer and Information Science
University of Konstanz, Germany
Email: michael.grossniklaus@uni-konstanz.de

Abstract—Recently, several proposals towards a cloud modeling language have emerged. As they address the diversity of cloud environments, it is not surprising that these modeling languages support different scenarios. Using a by-example approach based on the scenario of software migration, we demonstrate their representational capabilities and review them according to characteristics common to all modeling languages as well as specific to the cloud computing domain. We report on our findings and present research guidelines for future efforts towards a better alignment of the different cloud modeling languages.

I. INTRODUCTION

Cloud computing has recently emerged as a new possibility how software can be made available to clients as a service. Hereby, software is hosted on a cloud environment [5] and consumable over the network by different clients. For software vendors, this is appealing because of low upfront costs compared to a traditional on-premise solution as well as operational costs that scale with the provisioning and releasing of cloud resources, which are in turn offered as a service by cloud providers. Cloud service offerings range from low-level infrastructure elements, such as raw computing nodes, over higher level platforms, such as a Java execution environment on top of a cloud infrastructure, to ready-to-use software deployed on a cloud platform. Furthermore, software vendors are no longer forced to plan far ahead for resource provisioning [3] because the large-scale datacenters of today’s cloud providers ensure that requested resources are available through their cloud environments. The risk of under- and over-provisioning of such resources is reduced by cloud environments due to their capabilities for elastic scalability on demand. Therefore, resources are not only provisioned as their demand increases but also released once their demand decreases (e.g., number of user requests exceeds or falls below a defined threshold).

Since cloud environments offer novel optimization opportunities (e.g., advanced scalable data persistence solutions), taking full advantage of the cloud requires that the software is prepared for such an environment. However, in order to leverage the possibilities of a cloud environment, the software often needs to comply with certain restrictions that might hinder its functioning (e.g., stateful components in a highly scalable cloud environment). Several previous and ongoing European projects, notably REMICS [41], MODAClouds [2], PaaSage [36], ARTIST [7], and SeaClouds [15], address such optimization opportunities and how they influence the design of new cloud-based software and the re-design of legacy software in the course of a migration towards cloud-based software. Ideally, such design choices can be expressed in terms of models as a basis for the engineering [9], [12], [47] of cloud-based software. This approach calls for an appropriate cloud modeling language.

Recently, diverse proposals towards a cloud modeling language have emerged. However, they have different origins, pursue different goals, and hence provide a complementary and diverse set of language features. Consequently, there is a need to investigate these recent advances towards a cloud modeling language. Existing surveys by Papazoglou and Vaquero [46] and Sun et al. [50] mostly analyze general description languages for service-oriented architectures and low-level formats for resource virtualization with respect to their applicability to cloud computing. While Jamshidi et al. [35] have conducted a systematic literature review of cloud migration research, their survey does not focus on cloud modeling languages.

In this article, we take a different approach by demonstrating the representational capabilities of recent cloud modeling approaches that specifically address the cloud computing domain. Starting from the above-mentioned projects and their related work, we selected ten different approaches: Blueprint [43], CAML [8], CloudMIG [27], CloudML [23], CloudML-UFPE [30], HOT3, MOCCA [39], MULTICLAPP [31], RESERVOIR-ML [20], and TOSCA [10]. For each selected approach, we provide a brief description and then categorize it according to both modeling language characteristics and cloud computing characteristics. We do not claim that this set of cloud modeling languages is complete and leave a systematic literature review for future work. Rather, we focus on the extensional perspective of the modeling languages by demonstrating the approaches in the setting of a cloud migration scenario. We have selected this scenario since it covers the wide spectrum of the software lifecycle. Finally, we critically discuss our findings and present research guidelines towards a more aligned family of cloud modeling languages.

In Section II, we describe characteristics of both modeling languages and cloud computing, and discuss related work. Section III gives an overview of the selected approaches, which are demonstrated based on the Java Petstore scenario in Section IV. In Section V, we draw up research guidelines towards a family of cloud modeling languages, before concluding our work in Section VI.

II. MODELING FOR THE CLOUD COMPUTING

Model-Driven Engineering (MDE) advocates the use of models to raise the level of abstraction and model transformations to increase the degree of automation in the development of software [12]. Modeling languages that are used to create

1Note the difference to CloudML [23], which uses the same acronym.
2http://www.openstack.org
such models and upon which model transformations are typically defined play a central role in general and in this work in particular as we investigate current approaches towards a cloud modeling language. Therefore, we first briefly discuss common characteristics of modeling languages and then give an overview of cloud computing to set the stage for comparing different cloud modeling language proposals.

### A. Modeling Language Characteristics

The unification power of models [9] enables the abstraction from different implementation languages and platforms, thereby turning the focus from low-level implementation details to higher-level domain-specific concepts. By having appropriate modeling languages at hand, the transition from working in the solution space to the problem space can be achieved. Common characteristics of a modeling language [11] can be summarized as follows: pragmatics, syntax, and semantics.

**Pragmatics.** The pragmatics of a modeling language refers to its intended purpose and the overall goal that is pursued. There is a strong influence of the pragmatics on the syntax and semantics of a modeling language [38]. The intended purpose of a modeling language can range from "just" sketching the systems to be developed over providing blueprints that are concrete templates for producing the code manually to cases where the models are the code, i.e., the models are directly executed or the code generation is completely transparent for the user. It has to be further stressed that models are not only applicable in a generative manner, but more and more models are used analytically in software engineering, e.g., for design-space exploration, validation, or even for verification.

**Abstract Syntax.** The abstract syntax of a modeling language defines its concepts and how they relate to each other. It is the common basis of a modeling language since the elements of the abstract syntax are mapped to their concrete notation (cf. concrete syntax paragraph) and to a proper semantic domain (cf. semantics paragraph). In the context of MDE, such elements are typically structured in terms of a metamodel expressed by a class diagram while additional constraints on these elements provide contextual well-formedness rules. For instance, the UML metamodel with several OCL constraints is one well-known example in this respect.

**Concrete Syntax.** The concrete syntax is concerned with the form [42] of a modeling language and defines how abstract elements are realized in a concrete representation. Decorating abstract syntax elements with concrete ones usually increases the readability and intuitive handling of a modeling language. A language may have one or more textual or graphical syntaxes. The concrete syntax of UML is primarily graphical.

**Semantics.** While the concrete syntax of a modeling language aims to leverage correct human interpretations of modeling elements, the machine-interpretable meaning of such elements can only be achieved by explicitly defining their semantics. The semantics thus gives meaning to a modeling language and is defined on top of the abstract syntax elements. Most definitions of semantics are functions that map the abstract syntax elements of one language onto elements of a well-understood formal semantic domain, where the degree of formality may range from plain English to rigorous mathematics [32]. Defining the semantics of a modeling language is far from trivial as it involves a decision about a proper semantic domain, a mapping from valid syntactic elements to a selected semantic domain [32], and the finding of an agreement between stakeholders thereon. Therefore, most modeling languages do unfortunately not have a rigorously defined semantics that goes beyond English prose, even though it is in general a undisputed requirement for the definition of a modeling language. In particular in the light of the growing number of domain-specific languages, this requirement becomes even more important. In practice, however, a useful approach is to implement code generators for modeling languages that produce executable source code from the models. Again considering UML as an example, its semantics is primarily defined in English prose. However, for a key subset of UML, an operational semantics is provided through fUML [48]. Additionally, a plethora of approaches exist in literature that address the generation of source code from UML models, whereas modern UML modeling tools typically come with built-in code generators.

### B. Cloud Computing Characteristics

In cloud computing, resources, such as processing power and storage, platforms, and software, are viewed as commodities that are readily available from large data centers operated by cloud providers. Cloud computing leverages service-oriented architectures to unify elements of distributed, grid, utility, and autonomous computing. One characteristic that sets cloud computing apart from these existing approaches is the dynamic provisioning of resources offered by a cloud provider as a service. Consumers can acquire and release such cloud resources on demand and pay only for what they have actually consumed. This so-called pay-as-you-go principle benefits both the cloud consumer and the cloud provider. From the consumer perspective, the risk of under- or over-provisioning is avoided as the provisioned cloud resources can elastically scale [52] with a consumer’s demand. In contrast, the cloud provider profits from an economy of scale and can offer cloud resources at a price that is lower than the one of an on-premise solution [54]. Cloud providers can utilize their resources to capacity by optimizing the work load scheduling of the different co-located cloud consumers with consideration to their offered quality of service. We refer to the quality of a technical service, which can be expressed, e.g., in terms of latency, availability and security. Ideally, the quality is at least equivalent [53] to the one of an on-premise environment.

A key enabling technology of cloud computing is virtualization to abstract from physical resources. While, in theory, the spectrum of virtualization is continuous and all possible trade-offs are imaginable, in practice, cloud environments have converged to three rather discrete points on this spectrum [3]: infrastructure, platform, and software, as shown in Figure 1. Services not only expose the resources offered by a cloud provider but also give information about non-technical aspects, such as pricing and availability. This information is of particular interest to cloud consumers to select the cloud provider that offers services of the required quality at the expected virtualization level. Obviously, the higher the level of virtualization is, the more is managed by a cloud provider. Hence, the on-premise environment is partly or even completely replaced by a cloud environment. In practice, the typical scenario requires to "wire" both environments [1], e.g., by moving some parts of the on-premise environment to the cloud environment.
From an MDE point of view, one challenge is to address both the cloud consumer and the cloud provider perspective mainly because of the required wiring of their environments [25]. MDE can play a major role in this respect not only to externalize necessary domain knowledge and to provide abstractions over diverse cloud environments in terms of models, but also to support the shift from non-cloud environments to cloud environments and between cloud environments through rigorous model transformation techniques [24].

C. Related Surveys on Cloud Modeling Languages

In Papazoglou and Vaquero [46], the authors motivate the need for knowledge-intensive cloud services that comprise metadata of cloud environments, which are currently spread over and confined to the different virtualization levels of such environments. As a consequence, they argue the need for a language that supports the description, the definition of constraints over such descriptions, and the manipulation of cloud services and their metadata. Papazoglou and Vaquero identify and analyze (modeling) languages that fall into these three categories. The set of selected languages spans a broad spectrum, ranging from general languages for service-oriented architectures to low-level formats describing virtual resources.

We share the approach of Papazoglou and Vaquero to use the common virtualization levels as criteria to categorize existing modeling languages, but focus exclusively on modeling languages that respond directly to the requirements of cloud computing. As a result of this focus, we are able to use more fine-grained criteria to analyze existing modeling languages than the work of Papazoglou and Vaquero.

Sun et al. [50] present a survey of service description languages that examines seven different aspects. By analyzing common modeling language characteristics and their capabilities with respect to cloud computing, we cover all of these aspects in this article. In contrast to our work, Sun et al. do not further refine the domain aspect, which is due the fact that their scope goes beyond cloud computing.

Most importantly, our approach is different from the work of Papazoglou and Vaquero [46] and Sun et al. [50] since we give insights into the representational capabilities of recent cloud modeling approaches by demonstrating them according to a concrete cloud-based migration scenario.

Jamshidi et al. [35] conducted a systematic literature review (SLR) of cloud migration research in which they classified 23 publications from 2010 to 2013 according to 12 analysis dimensions. They conclude that cloud migration research is still in its early stages, but their study also provides evidence that the maturity of the field is increasing. Jamshidi et al. [35] do not focus on modeling and modeling languages, which distinguishes their work from ours. Nevertheless, they cite the need for a common research agenda between cloud computing and software engineering researchers, which further motivates our work.

III. CLOUD MODELING APPROACHES

Recently, several cloud modeling approaches have been proposed. In this section, we discuss them according to the four modeling language characteristics introduced in Section II-A.

Blueprint. The Blueprint [44] approach describes Service-based Applications (SBA) in terms of coarse-grained deployment artefacts that provide a uniform representation of SBAs connected with the required cloud service offerings. Blueprints are encoded in XML and typically represented in terms of a Virtual Architecture Topology (VAT) for which a graphical notation is suggested. The idea is to publish such blueprints in a public repository [43] to establish a service marketplace.

CAML. The Cloud Application Modeling Language (CAML) [8] supports representing high-level deployment topologies and their refinement towards concrete cloud offerings. It is realized in terms of a UML internal modeling language based on a model library and profiles. These profiles capture cloud offerings from a technical perspective, e.g., performance, as well as non-technical perspective, e.g., pricing.

CAML fosters reuse of deployment solutions by describing them in terms of templates. As CAML is based on UML, it can directly be applied on UML models, which is especially beneficial for migration scenarios where reverse-engineered UML models are tailored towards a selected cloud environment.

CloudMIG. CloudMIG [27] addresses the migration of legacy applications onto cloud environments. The main focus is on the reverse engineering of applications into representations conforming to the Knowledge Discovery Model (KDM) [45] and their cloud-based deployment. To represent cloud environments, CloudMIG provides the Cloud Environment Model (CEM) that is realized in terms of an Ecore-based metamodel. It supports the cloud application and cloud environment perspective, though the focus with respect to cloud environments is mostly at the infrastructure and partly platform level for which constraints, pricing, and deployments can be specified.

Dedicated tool support is offered by CloudMIG Xpress, which features automatic computation of optimal cloud-based deployments [26] and conformance checking of legacy software with respect to potential cloud providers [28].

CloudML. The main purpose of CloudML [23], formerly PIM4Cloud [13], is to describe application deployments and automate their provisioning mainly based on infrastructure-related cloud offerings. Applications are described from the viewpoint of components that are connected to concrete cloud offerings. CloudML provides an Ecore-based metamodel where deployments are expressed in terms of the JavaScript Object Notation (JSON). They serve as input for the provisioning engine that operationalizes CloudML. The engine implementation builds on the jClouds framework, which abstracts from different cloud environments and, hence, the offerings of supported cloud providers.

1http://www.json.org
2http://www.jclouds.org
CloudML-UFPE. The description of infrastructure-related cloud services and cloud resources is covered by CloudML-UFPE [30]. CloudML-UFPE proposes an XML-based approach to represent services offered by cloud providers. The consumer perspective is supported by service requests. As the main focus of CloudML-UFPE is on the infrastructure level, cloud services and resources are represented in terms of nodes and links between them. Nodes have properties for CPU, storage, and memory, whereas links have properties for delay and rate. Service requests contain the required nodes and links according to the specified cloud services, which essentially results in a manual mapping based on identifiers.

HOT. The Heat Orchestration Template (HOT) supports describing deployment templates mainly for cloud environments that operate at the infrastructure level. It is currently developed in the context of OpenStack. HOT is built around the notion of resources that describe the main artifacts, such as compute instances or networks, of a deployment. Resources may get input parameters passed and may produce outputs available to the user. Functions can be used inside of templates to perform specific tasks, such as getting the value of a resource parameter at runtime. OpenStack provides runtime support in terms of a dedicated HOT interpreter.

MOCCA. The MOVE to Clouds for Composite Applications (MOCCA) approach [39] proposes a method for migrating legacy software to a cloud environment. MOCCA comes with a dedicated metamodel that covers modeling elements for representing the architecture and the deployment of the legacy software. Based on these models, the deployment in a cloud environment can be derived and expressed in terms of a clustering of architectural elements and concrete implementation units that are assigned to the virtual resources of a cloud environment. The virtual resources are described in Open Virtualization Format (OVF) [21] to provide support for the actual resource provisioning.

MULTICLAPP. MULTICLAPP [31] aims to support the modeling of cloud applications from a cloud-provider-independent perspective. Thereby, the focus is on the representation of application components modeled in UML and the refinement of these components with a dedicated profile. The idea is to annotate components with stereotypes that are expected to be deployed onto one or multiple environments. Additionally, stereotypes are provided that allow components to be annotated with QoS parameters, such as their response time.

RESERVOIR-ML. RESERVOIR-ML [20] supports the description and configuration of virtualized infrastructures. Configuration is performed at run-time on the basis of monitoring information from deployed applications. The language is based on the Open Virtualization Format (OVF) [21], which mainly extends with primitives to describe applications in terms of components and elasticity rules that control the virtual machine configurations in an OpenNebula cloud environment.

TOSCA. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [10] aims at realizing portable cloud applications that are described in terms of so-called service templates. TOSCA is based on XML, whereas with Vino4TOSCA [14] a graphical notation for service templates is provided. Such service templates can be operationalized with management plans from which operations can be called to initiate, for instance, the provisioning of applications. Management plans rely on existing workflow technologies, such as BPMN[7], and can be seen as the mapping between the cloud applications and cloud environments.

Synopsis. Table I summarizes the modeling language characteristics of the reviewed approaches in terms of pragmatics, abstract syntax, concrete syntax, and semantics. Regarding the pragmatics, the majority of approaches deal with the deployment viewpoint of cloud-based applications. Except CloudML-UFPE and HOT, as these approaches set the focus solely on the description of cloud resources and their provisioning, respectively. Resource provisioning is also supported by CloudML and TOSCA. While CAML supports the refinement of deployments towards concrete cloud offerings, their optimization is supported by MOCCA and CloudMIG. Additionally, CloudMIG allows conformance checks between deployments and cloud offerings. Finally, RESERVOIR-ML supports scaling deployed applications at runtime. Most approaches are realized as external domain-specific languages, i.e., their syntax and semantics is developed from scratch. Only CAML and MULTICLAPP are realized as UML internal languages. Hence, they can directly be applied on UML models. The semantics of the surveyed approaches is given either in English prose or as a piece of software rather than as a formal definition.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Pragmatics</th>
<th>Abstract Syntax</th>
<th>Concrete Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inspector</td>
<td>Service Description</td>
<td>XML Schema</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>CAM</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>CloudMIG</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>CloudML</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>CloudML-UFPE</td>
<td>Resource Provisioning</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>HOT</td>
<td>Resource Provisioning</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>MOCCA</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>MULTICLAPP</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>CLOUDIFYING</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>JAVA</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
<tr>
<td>TOSCA</td>
<td>Application Deployment</td>
<td>XML Profile</td>
<td>Graphical</td>
<td>English prose</td>
</tr>
</tbody>
</table>

TABLE I. LANGUAGE CHARACTERISTICS OF REVIEWED APPROACHES

IV. CLOUDIFYING THE GOOD OLD JAVA PETSTORE

To demonstrate the representational capabilities of the selected cloud modeling approaches, we examine potential activities of a cloud migration scenario that is based on the well-known Petstore reference example, introduced by Sun in 2001. Taking existing literature [18],[27],[39] into consideration, we derived five activities as summarized by the migration process illustrated in Figure 2. The support offered by the selected approaches for each activity is also indicated in this figure (cf. gray boxes). While most approaches support more than one activity, we will demonstrate each approach only once (highlighted in bold) by means of the accompanying tool, if it is publicly accessible. In this way, the concrete syntax of the modeling approaches is put in the spotlight.

3http://www.openstack.org
4http://opennebula.org
5http://www.omg.org/spec/BPMN/2.0
6http://www.oracle.com/technetwork/java/index-136630.html
As depicted in Figure 2, first a deep understanding of the legacy application is required [18] before any other step in the process can be conducted. Once a target cloud environment is selected, adaptations on the legacy application are performed, if required. Adaptations may be required to meet the goals of the migration and to exploit the novel cloud-based technologies offered by today's cloud environments. The latter may also involve overcoming constraints imposed by different virtualization levels that hinder the appropriate functioning of the goals of the migration and to exploit the novel cloud-based technologies. This requires the adaptation of the target platform to the target cloud environment. In particular, application virtualization at the platform level typically have to interface with the proprietary framework of the cloud provider, whereas arbitrary dependencies can be deployed and used at the infrastructure level. Nevertheless, platform-level virtualization is appealing as it often provides better support to automate and manage the deployment of applications. For example, the Google App Engine currently does not support EJB at the platform level, but offers a degree of automatic scalability [3] through a strict application structure, which is inherently difficult to achieve at the infrastructure level, where, for instance, Amazon operates. Depending on the virtualization level of the selected cloud service(s) and the pursued migration type [1], the deployment covers different aspects of a software, such as user interface, business logic, or data management, when considering a classical three-layer architecture. After the deployment is prepared, the required cloud resources can be provisioned to finally run the migrated software in the cloud.

### A. Get Understanding: CloudMIG by Example

As conceptual models provide an excellent means for getting an understanding of applications, CloudMIG supports their discovery from legacy applications by building on MoDisco [16], which discovers KDM models from Java applications. CloudMIG provides a tree-based structure to represent KDM models, which can be linked to a deployment model of the legacy application. In Figure 3a, the deployment model of the Petstore is depicted. We assume two nodes for the Petstore scenario, which are further specified in terms of mainly CPU-related properties. In a similar way, a cloud-based deployment is represented, though in this case, a cloud node is instantiated, which refers to a specific cloud environment operated by a cloud provider. Furthermore, we assign the KDM model discovered from the Petstore to the modeled application server, as indicated by "class" icon in upper right corner of the node elements. The reverse-engineered models provide a useful basis for the decision-making of the required cloud offerings.

### B. Provide Selection: CAML, CloudML-UFPTE by Example

CAML enables the refinement of high-level deployment topologies towards concrete cloud offerings. These offerings are captured in terms of UML profiles and meta-profiles. Figure 4 presents some stereotypes specific to the cloud offerings of Amazon. Instance types are offered with different operating systems and they can be located in different regions and availability zones. In turn, cloud offerings are refined by so-called meta-profiles. They facilitate refining cloud provider profiles with technical and non-technical details, e.g., performance of instance types and their costs.

CloudML-UFPTE supports describing infrastructure-related cloud offerings mainly in terms of nodes and their properties. Listing 1 shows a concrete Amazon offering with some node elements.
characteristics and the geographical locations the service is provided for. Listing 2 depicts how the nodes in the service type are linked via identifiers to nodes that represent the infrastructure underlying such a cloud service. Although CloudML explicitly differentiates between virtual and physical resources, the modeling concepts to describe them are almost identical. Hence, the infrastructure description abstracts from specific physical or virtual resources and only presents a generalized view of the nodes and the links that connect them.

```
Listing 1. CloudML-UFPE by Example: Service Description
/serviceDescription:ServiceType version="AmazonML"
  <nodes ID="MI Medium Instance 1"/>
  <nodes ID="MI Medium Instance 2"/>
  <links ID="Link1" owner="MI Medium Instance 1"/>
  <links ID="Link2" owner="MI Medium Instance 2"/>
</serviceDescription:ServiceType>
```

C. Perform Adaptations: RESERVOIR-ML by Example

Clearly, the required adaptations for a successful migration to the cloud can be diverse and related to different virtualization levels of cloud environments. To address the elastic scalability at infrastructure level, we consider RESERVOIR's capabilities to trigger elastic rules from an application. In Listing 3, the components of the Petstore are modeled according to RESERVOIR-ML. The elasticity rule depicted in Listing 4 triggers the instantiation of a new virtual machine if the number of active sessions exceeds a predefined number N of sessions. To monitor the number of active sessions at runtime, the KPI refers to a session listener class of the Petstore’s WebUI component. We implemented Java’s SessionListener interface for the Petstore to provide one potential source to gather application level measurements. They are collected by monitoring agents provided by the RESERVOIR monitoring framework. In a similar way, a “scale-in” rule could be defined in order to suspend a virtual machine, if the session count falls below a given threshold.

```
Listing 3. RESERVOIR-ML by Example: Application Description
  <components ovf="FWI" qualifiedName="PetstoreWebUI"/>
  <components ovf="EC" qualifiedName="PetstoreController"/>
  <components ovf="PD" qualifiedName="PetstoreDomain"/>
  <components ovf="PBS" qualifiedName="PetstoreService"/>
</RESERVOIR-ML:ApplicationDescription>
```

```
Listing 4. RESERVOIR-ML by Example: Elastic Rule
  <reservoirERM: ElasticityRule name="ScaleOutAdjustment">
    <trigger>
      <actions run="deployNewVM"/>
    </trigger>
  </reservoirERM: ElasticityRule>
```

D. Prepare Deployment: Blueprint, MOCCA, MULTICLAPP, TOSCA by Example

The majority of approaches provide modeling support to prepare the deployment for our Petstore scenario. In general, our deployment consists of the components that constitute the Petstore, the execution environment required for the components, and the nodes based on which the environment is hosted. We consider potential solutions according to Blueprint, MOCCA, MULTICLAPP, and TOSCA in Figure 5.

```
Fig. 5. (a) Blueprint by Example: Virtual Architecture Topology, (b) MOCCA by Example: (1) Deployment Model, (2) Cafe Archive Component Model, (c) TOSCA by Example: Service Template, and (d) MULTICLAPP by Example: Annotated Component Model
```

The properties of the nodes and the links that connect them are specified in the Annotated Component Model. In Figure 5, the nodes represent the components, whereas the links represent the communication between them. The relationships between the components are defined in the Topology Diagram. The deployment model specifies the placement of the components on the nodes, the communication between them, and the resources required for their operation. The execution environment required for the components is specified in the service template, which includes the operating system, the application server, and the database management system.
addressed by connecting platform-related components with (virtual) resources described in terms of OVF, which may serve as an interchange format, provided that the selected cloud environment offers support in this respect.

TOSCA provides nodes and relationships to represent deployments. Nodes can be attached with multiplicities to express the number of potential running instances. In addition to templates, TOSCA provides types, which are considered reusable entities instantiated by templates. For instance, in our concrete deployment, the modeled application server is of type JBoss. In a similar way, relationships are considered, which means that the name attached to a link is basically the type of the relationship.

MULTICLAPP proposes a component-oriented approach similar to MOCCA to represent a deployment. However, the deployment is described independently of a cloud provider. Hence, only the types of offerings needed in the cloud environment as well as their required quality are described. MULTICLAPP does not necessarily distinguish between different virtualization levels, but solely between artifacts of the application being deployed and the offerings that are expected to be provided by the cloud environment.

E. Execute Provisioning: CloudML, HOT by Example

The provisioning of cloud resources is supported by CloudML. As shown in Listing 5, the requirements from a cloud consumer perspective for the Amazon solution are specified by the required middleware, i.e., a JBoss application container to run the Petstore application and the minimal VM requirements: number of Cores, amount of Storage space and RAM. These cloud resource requirements serve as input for the CloudMLEngine. It is capable to perform the provisioning by a semi-automatic matching between the defined requirements and the cloud offerings.

Listing 5. CloudML by Example: Resource Requirements and Provisioning

```xml
<net.cloudml.core:CloudMLModel name="Petstore-RS">  
  <providers name="aws-ec2"/>
  <resources downloadCommand="wget -P " URL" installCommand="..."/>
  <requiredExecutionPlatform name="Required-OS">
    <os name="Ubuntu"/>
  </requiredExecutionPlatform>
  <vms name="MI.Medium" provider="aws-ec2">
    <vms minCores="2" minStorage="50" minRam="1000">
      <os name="Ubuntu"/>
    </vms>
    <vms name="Provider-OS" owner="MI.Medium"/>
    <vms family="Provider-OS" values="Ubuntu"/>
  </vms>
</net.cloudml.core:CloudMLModel>
```

HOT allows templates to be expressed for cloud-based deployments. In contrast to CloudML, HOT directly specifies the offerings of a cloud provider instead of taking the perspective of the requirements for cloud offerings. Listing 6 depicts a template that specifies a compute offering in terms of a resource. It allows selecting from three different compute types that are specified in terms of a constrained parameter. The middleware required to run the Petstore application is installed as part of the resource provisioning (see line 16).

Listing 6. HOT by Example: Resource Provisioning

```xml
<heat_template_version: 2013-05-23
parameters:
  compute_type:
  type: string
  allowed_values: [m1.small, m1.medium, m1.large]
resources:
  petstore-vm:
    type: OS:Novaserver
    properties:
      flavor: (get_param: compute_type)
      user_data:
        str_replace:
          #!/bin/bash -v
          #!/usr/bin/env python
          yum -y install java mysql
          yum -y install java jboss mysql
          yum -y install java jboss mysql

F. Synopsis

To analyze the selected approaches in terms of their capabilities with respect to cloud computing, we use the dimensions as afore discussed in Section II-B and summarize in Table II the support provided by each approach. We can observe four areas of interest. First, approaches that exploit UML or KDM, i.e., CAML, CloudMIG, and MULTICLAPP provide the most complete support to represent the on-premise environment. Second, we note that while all approaches support the mapping of cloud resources, only very few of them, i.e., CloudMIG, CloudML, and MULTICLAPP provide a semi-automatic matching of cloud offerings to application requirements. Third, it is not surprising that approaches, which stem from the domain of service-oriented architectures, i.e., Blueprint, support the representation of cloud offerings as they are typically considered as a service. However, CAML also provides UML profiles to represent such cloud offerings. Finally, the support for cloud-specific characteristics in all approaches is rather scattered at the time of this study.

V. OBSERVATIONS AND RESEARCH DIRECTIONS

The investigation of existing modeling approaches in the domain of cloud computing has revealed several interesting observations and corresponding research directions. In the following, we summarize these observations by discussing (i) the proposed modeling languages according to common language engineering characteristics, (ii) the diffusion of these languages among them and existing standard modeling languages, and (iii) their applicability with respect to the peculiarities of cloud computing.

<table>
<thead>
<tr>
<th>Approach</th>
<th>On-Premise Environment</th>
<th>Wiring</th>
<th>Cloud Environment</th>
<th>Specifics</th>
<th>High-Quality Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blueprint</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CAML</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CloudMIG</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CloudML</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CloudMLUFF</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>MULTICLAPP</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>HOT</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

TABLE II. DOMAIN CHARACTERISTICS OF REVIEWED APPROACHES
A. Language Engineering Characteristics

Observation: Heterogeneous language engineering background. It seems that there are two dominant meta-languages applied in the field of cloud modeling. While Blueprint, CloudML-UFPE, RESERVOIR-MI, and TOSCA define their modeling elements in terms of XML Schemas, CloudMIG, CloudML, and MCCA provide an Ecore-based metamodel. Related to the second approach, UML is applied as a meta-language. However, instead of defining a modeling language from scratch as done in the previous mentioned approaches, a UML-internal modeling language is specified using UML profiles and UML libraries. Both CAML and MULTICLAPP apply this approach. HOT follows a different approach by providing a text-based syntax conforming to the YAML metaformat.

Research directions. Ideally, proposals for new modeling languages come together with a machine-interpretable and human Usable language definition preferably in a commonly accepted format. Guidelines for defining high quality language definitions are presented in the literature (e.g., [38], [40], [49]). Once they are defined, sharing them in terms of an open repository (e.g., AlanMod's Metamodel Zoo9 or ReMoDD10) allows them to be accessed and may further stimulate their development in the new modeling languages. However, the definition of these languages is still not formally presented in the literature. Clearly, a single representation that usefully denotes the semantics of a modeling language would be the preferred way [17].

B. Modeling Language Diffusion

Observation: High diversity in current cloud modeling approaches. Current cloud modeling approaches pursue different goals, propose, hence, provide diverse modeling elements, and show various levels of maturity. While these approaches provide a considerable set of complementary cloud modeling elements, they also show similarities not only in the modeling elements they propose but also in their pragmatics. Although in this article the focus is on the extensional perspective rather than on the intensional one, the demonstration of the approaches revealed possibilities for their integration. For instance, the majority of approaches deal with the deployment towards cloud environments. Interestingly, however, we noticed that most approaches are hardly aware of each other. One exception is CAML which describes an initial mapping to TOSCA to exploit the OpenTOSCA runtime environment for resource provisioning. Still, a well-connected mix of existing cloud modeling elements is currently not available.

Research directions. The observed diversity of the current approaches is beneficial in the sense that a broad spectrum of modeling elements for the cloud computing domain is available. At the same time, the exchange of models between approaches and provided tools, respectively, is hardly supported. Thus, the finding of a common ground between the current approaches is required. Clearly, the semantics of the modeling languages play a major role [37] since useful mappings, which are the basis for an integration, can otherwise hardly be identified. Still, a common metamodel [4] may serve as a useful means in such an integration endeavor.

Observation: Little attention paid to general-purpose software modeling languages. The cloud modeling approaches studied in this article indisputably introduce novel modeling elements for cloud computing. At the same time, general-purpose software modeling languages, such as UML, provide modeling concepts to represent software, platform, and infrastructure artifacts from different viewpoints. Nevertheless, only two approaches, i.e., CAML and MULTICLAPP, provide extensions to UML to deal with cloud modeling requirements. Most other approaches define slightly modified versions of component-like (e.g., CloudML, MCCA, RESERVOIR-MI) and deployment-like (e.g., CloudMIG and TOSCA) modeling concepts from scratch. While their need is clear, it is surprising that the reuse of existing modeling concepts is not taken into account even though compatibility with well-established software modeling languages would facilitate their practical applicability in a broad spectrum of scenarios [33], [34].

Research directions. Compatibility between languages can be achieved by model transformations that encode the respective correspondences between their metamodels to enable at least partial model exchange. In addition, existing metamodels...
can directly be reused or extended in a lightweight manner (e.g., by profiles as in CAML and MULTICLAPP). Such extensions can then cover the characteristics of the cloud computing domain. As a result, not only the full expressiveness of software modeling languages is exploited but also reuse of existing models is ensured.

C. Modeling Language Applicability

Observation: Non-functional requirements matter. In the area of cloud computing, non-functional requirements are of significant relevance for cloud consumers. While the spectrum of non-functional requirements is broad, pricing is one aspect that is addressed by several approaches from different perspectives. In CAML, cloud offerings are annotated with costs mainly for the purpose of informing cloud consumers [19] and selecting cloud environments. CloudMIG covers pricing information of cloud environments for the optimization of software deployments [26]. RESERVOIR-ML deals with application performance indicators to acquire or release virtual machine instances depending on predefined elasticity rules.

Research directions: Directly attaching information, such as pricing and service levels to the deployment artifacts may be a further improvement [29], thereby bringing technical and non-technical aspects together in a single view, which can help to ease the selection of an appropriate cloud provider. With respect to the general field of performance engineering, approaches are available that go beyond the performance indicators proposed by the cloud modeling languages. For instance, Kieker [51] provides support for software run-time performance analysis, whereas Palladio [6] allows attaching UML component diagrams with parameters for predicting performance already at design-time.

Observation: Extensible pragmatics of cloud modeling languages. Throughout the demonstration of current cloud modeling approaches, we observed that they already provide considerable support for the cloud computing domain. However, we also observed that some activities in the “moving-to-the-cloud” scenario are less well supported than others. To gain an understanding of legacy software, CloudMIG advocates the use of KDM, which offers reasonable support by providing an overview of and dependencies between legacy artifacts in terms of an inventory model. Still, higher abstractions are desirable in this respect [18] and even dedicated views [222] on such abstractions are inevitable, particularly when adaptations are required to achieve a successful migration [1]. Although MOCCA proposes the use of architectural models on high abstraction levels, they need to be manually created. Furthermore, adaptations on such high abstraction levels may also require to transfer them to lower abstraction levels until they take effect on the executable source code. CloudMIG takes one important step in this direction by checking the conformance between legacy software artifacts and cloud environments. However, support for automatic correction of conformance violations is currently initial at best. Finally, models used at design-time may also be utilized during run-time, e.g., for monitoring or re-configuration, as is currently only foreseen in the context of RESERVOIR-ML.

Research directions: Different cloud-based migration types and their potential impact on adaptations for each of these types is discussed by Andriopoulos et al. [1], which can be considered not only as a basis for current and future migration scenarios, but also as a starting point for elaborating on cloud-based optimization opportunities. They are preferably captured by future efforts, for instance, in terms of patterns that are operationalized through dedicated model transformations. Through these efforts, the forward engineering phase in a cloud-based migration scenario and the engineering and operation of new cloud-based software may considerably be improved.

VI. CONCLUSIONS

In this paper, we reported on the state-of-the-art of cloud modeling languages by examining existing proposals using a by-example approach based on the scenario of migrating an existing application to the cloud. By applying the different languages, we made several observations that identified future research directions. Even though some alignment is clearly needed, it is however unclear how to actually realize it. For example, a one-size-fits-all approach resulting in the one cloud modeling language is challenging because current languages are highly diverse in their pragmatics, e.g., the focus can either be on the service provider or on the service consumer side. An alternative approach is to have clear correspondences between the different languages and allow for a multi-viewpoint approach or aim for a family of languages, which resolves current language heterogeneities in an non-intrusive manner. Furthermore, it is not clear if cloud modeling is an activity that is performed in isolation from software modeling or if these two concerns should be addressed at the same time. This question is particularly relevant, if considered from the cloud service consumer point of view. In this context, the most important challenge is how to align cloud modeling languages with existing software modeling languages in order to provide continuous modeling support.

REFERENCES


