Cloud Modeling Languages by Example*

Alexander Bergmayr[1], Michael Grossniklaus[2], and Manuel Wimmer[1]

[1]Business Informatics Group, Vienna University of Technology, Austria,
[bergmayr|wimmer]@big.tuwien.ac.at
[2]Department of Computer and Information Science, University of Konstanz, Germany,
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. Based on a by-example approach, 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 unified cloud modeling language.

Keywords: *Cloud Computing • Model-Driven Engineering • 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 since cloud computing has the benefit of low upfront costs compared to a traditional on-premise solution and 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 may 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 ongoing projects, notably REMICS [40], PaaSage [35], MODAClouds [2] and ARTIST [7], currently investigate such optimization opportunities and how they influence the design of new cloud-based software and

*This work is co-funded by the European Commission under the ICT Policy Support Programme, grant no. 317859.

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 [8,11,47] of cloud-based software. This approach calls for an appropriate cloud modeling language.

Contributions of this work. 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. 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. Based on a literature search, we selected ten different approaches: Blueprint [42], CloudMIG [27], the Cloud Modeling Language (CloudML) [30], MOve to Clouds for Composite Applications (MOCCA) [38], REMICS CloudML[1] [13], PIM4Cloud [12], the RESERVOIR Manifest Language [20], Smart Cloud Optimization for Resource Configuration Handling (SCORCH) [22], the Topology and Orchestration Specification for Cloud Applications (TOSCA) [44], and the Unified Service Description Language (USDL) [18]. For each selected approach, we provide a brief description and then categorize it according to both modeling language characteristics and cloud computing characteristics. We put the focus on the extensional perspective of the modeling languages by demonstrating the approaches on the basis of a cloud-based migration scenario. Finally, we critically discuss our findings and present research guidelines towards a unified cloud modeling language.

Structure of this work. 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 unified cloud modeling language, before concluding our work in Section VI.

II. Model-Driven Engineering meets 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 [47]. Modeling languages that are used to create 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.

I. Modeling Language Characteristics

The unification power of models [8] 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. In the following, we briefly summarize common characteristics of a modeling language [10]: *pragmatics*, *syntax* (consisting of *abstract syntax* and *concrete syntax*), and *semantics*.

Pragmatics. The pragmatics of a modeling language refers to its intended purpose and the

[1] Note the difference to CloudML [30], which uses the same acronym.

overall goal that is pursued. There is a strong influence of the pragmatics on the syntax and semantics of a modeling language [37]. 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 [41] 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. Since modern language theory shifts the focus to the abstract syntax, arbitrary concrete syntaxes can be defined. For instance, a language may have one or more textual or graphical syntaxes. In case of UML, the concrete syntax combines graphical with textual elements though the primary focus is on the graphical part.

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 [31]. 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 [31], 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.

II. 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

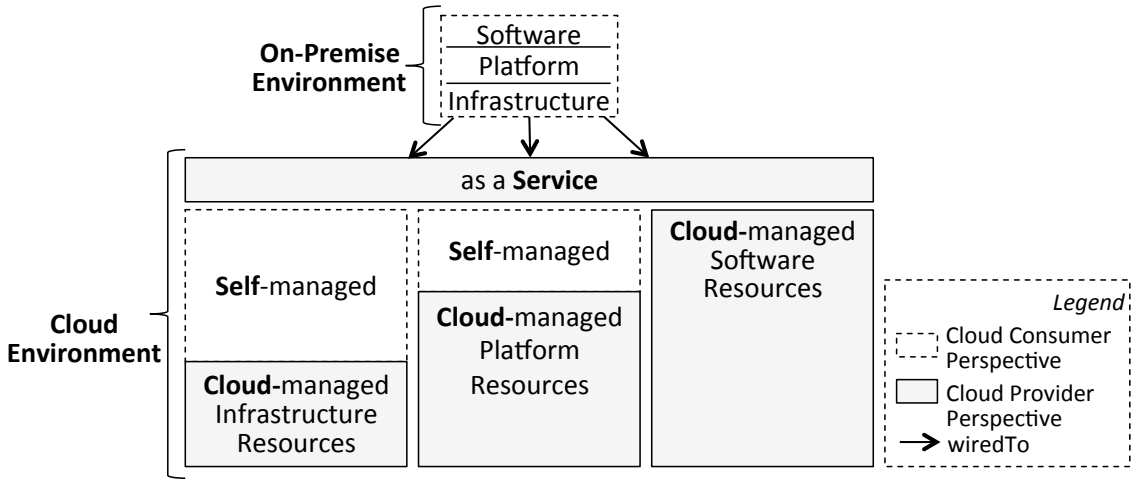

Figure 1: *Virtualized Cloud Resources as a Service*

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* [51] 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 [53]. 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*. In this context, 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 [52] 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 discreet 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. As a consequence, 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].

III. Related Surveys on Modeling Languages for Cloud Computing

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.

III. Overview of Modeling Approaches for Cloud Computing

In this section, we first describe current approaches and then discuss them according to the four modeling language characteristics introduced in Section I.

Blueprint. The Blueprint [43] 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 term of a Virtual Architecture Topology (VAT) for which a graphical notation is suggested. The idea is to publish such blueprints in a public repository [42] to establish a marketplace for cloud services that can be searched and accessed by cloud consumers.

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. Dedicated tool support is offered by CloudMIG Xpress[2], which features automatic computation of optimal cloud-based deployments [26] and conformance checking of legacy software with respect to potential cloud provider [28]. With KDM and CEM, CloudMIG provides reasonable support for 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. A mapping between the two perspectives is automatically generated as far as the application deployment is concerned. However, turning a generated deployment into a running application hosted on a cloud environment is not in the scope of CloudMIG.

[2] http://www.cloudmig.org

CloudML. The description of infrastructure-related cloud services and cloud resources is covered by the Cloud Modeling Language (CloudML) [30]. CloudML 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 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.

MOCCA. The MOve to Clouds for Composite Applications (MOCCA) approach [38] 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.

PIM4Cloud. To support the deployment in cloud environments, PIM4Cloud [12] proposes a component-oriented approach for describing the software down to the infrastructure in terms of a topology. Similarly, the cloud environment can be described in terms of components mainly with respect to the platform level and infrastructure level. Such components are expressed using Scala[3], which can be interpreted by a deployment script provided by PIM4Cloud.

REMICS CloudML. The main purpose of REMICS CloudML [13] is describing and provisioning infrastructure-related cloud resources. Cloud resources are represented in terms of the JavaScript Object Notation (JSON)[4] that serve as input for the provisioning engine that operationalizes REMICS CloudML. The engine implementation builds on the jClouds[5] framework, which abstracts from different cloud environments and, hence, the offerings of supported cloud providers.

RESERVOIR Manifest Language. The RESERVOIR Manifest Language [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 it mainly extends with primitives to describe applications in terms of components and elasticity rules that control the virtual machine configurations in an OpenNebula[6] cloud environment. Hence, explicitly representing potentially offered cloud services is not in the scope of the Manifest Language.

SCORCH. The Smart Cloud Optimization for Resource Configuration Handling (SCORCH) [22] approach operates on a rather low virtualization level for realizing auto-scaling of infrastructures and platforms by optimizing a queue of pre-instantiated virtual machines and their configurations. These configurations are built in reference to a common cloud configuration feature model. SCORCH is implemented based on a constraint solver and relies on several computational models that specify the energy consumption and costs of cloud environments from a cloud provider perspective. Additionally, knowledge about the resources demanded by an application over its lifetime is required by the optimizer. Hence, the primary focus is on cloud resources rather than on cloud applications and even less on cloud services since they cannot be described explicitly. Cloud resources are provisioned on the basis of pre-defined configuration demand models that represent a valid set of features defined in the cloud configuration model.

[3] http://www.scala-lang.org
[4] http://www.json.org
[5] http://www.jclouds.org
[6] http://opennebula.org

Approach	Modeling Language Characteristics			
	Pragmatics	**Abstract Syntax**	**Concrete Syntax**	**Semantics**
Blueprint	Service Description Service Deployment	XML Schema	Graphical	English Prose
CloudMIG	Reverse Engineering Deployment Optimization Conformance Checking	Ecore-based Metamodel	Graphical (CloudMIG Xpress)	English Prose Deployment Optimizer Conformance Checker
CloudML	Resource Description	XML Schema	Textual (XML-based)	English Prose
MOCCA	Reverse Engineering Deployment Optimization	Informal Metamodel	Graphical + Textual (OVF-based)	English Prose Deployment Optimizer
PIM4Cloud	Application Deployment	Informal Metamodel	Textual (Scala-based)	English Prose Deployment Scripts
REMICS CloudML	Resource Provisioning	Informal Metamodel	Textual (JSON-based)	English Prose Resource Configurator
RESERVOIR Manifest Language	VM-Image Scalability Application Deployment	Informal Metamodel	Textual (XML-based)	English Prose Open Nebula Integration
SCORCH	VM-Image Scalability Resource Optimization	Informal Metamodel	Graphical	English Prose Resource Optimizer
TOSCA	Service Description Service Deployment	XML Schema	Textual (XML-based) + Graphical (VINO4Tosca)	English Prose
USDL	Service Description Service Marketplace	Ecore-based Metamodel	Graphical (USDL Tool)	English Prose

Table 1: *Modeling Language Characteristics of Reviewed Approaches*

TOSCA. The Topology and Orchestration Specification for Cloud Applications (TOSCA), standardized by OASIS[7], aims at realizing portable cloud services that are described in terms of so-called service templates, a high-level view on the service deployment topology [9]. TOSCA is based on XML, whereas with Vino4TOSCA[8] [14] a graphical notation for service templates is provided. Such service templates can be operationalized with management plans from which operations exposed by cloud services can be called to initiate, for instance, the application deployment. Management plans rely on existing workflow technology, such as BPMN[9] and BPEL[10], and can be seen as the mapping between the cloud application and cloud environment though at a rather coarse-grained level.

USDL. The Unified Service Description Language (USDL) [18] provides a rich set of modeling elements for describing cloud services from a business and technical perspective. USDL is defined in terms of an Ecore-based metamodel that comes with a dedicated modeling editor[11]. It builds on and extends existing standards for service-oriented architectures, e.g., WSDL and BPEL. Although USDL provides modeling elements commonly useful for describing services in general, the main focus is on so-called marketplace services that are tradeable in the Internet of Services (IoS). Hence, through business aspects, e.g., service levels and pricing schemes and operational aspects, e.g., service interfaces and operations, the cloud provider rather than the cloud comsumer perspective is primarily addressed.

Synopsis. Table 1 summarizes the modeling language characteristics of the reviewed approaches in

[7] https://www.oasis-open.org/committees/tosca
[8] http://vino4tosca.org
[9] http://www.omg.org/spec/BPMN/2.0
[10] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
[11] http://usdleditor.sourceforge.net

terms of pragmatics, abstract syntax, concrete syntax, and semantics. We can observe that while a few modeling languages are defined by formal metamodels based on Ecore or XML Schema, most of the approaches only provide an informal description of their abstract syntax. Similarly, the semantics of the majority of approaches is given in English prose or as a piece of software, rather than as a formal definition. Finally, the approaches can be clustered concerning the pragmatics into the following subsets: (i) service description and deployment (TOSCA, USDL, Blueprint), (ii) resource description and provisioning (SCORCH, REMICS CloudML, CloudML), (iii) application deployment (PIM4Cloud (static support), RESERVOIR Manifest Language (dynamic support)), and (iv) reverse engineering (MOCCA, CloudMIG).

IV. Cloudifying the Good Old Java Petstore

To demonstrate the main 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[12]. Taking existing literature [17,27,38] 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). Even though several approaches support more than one activity, we will demonstrate each approach only once 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.

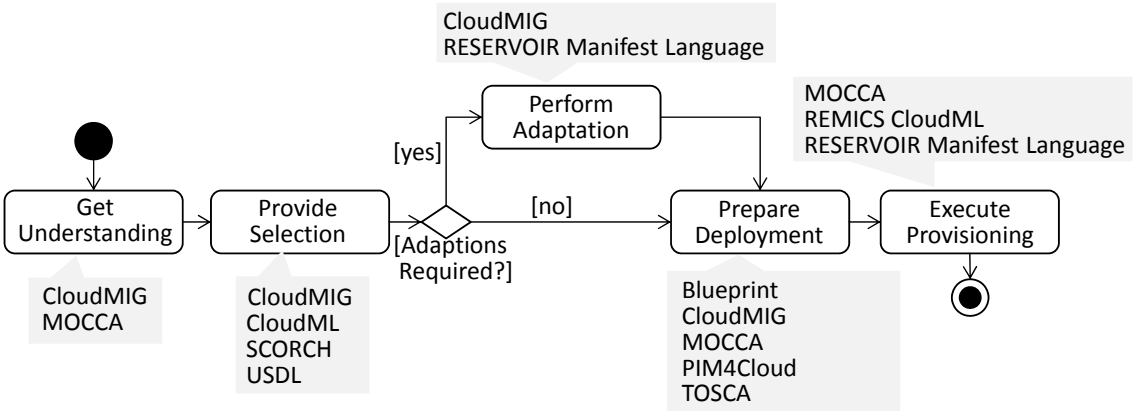

Figure 2: *Migration Process and Support for Activities of Modeling Approaches*

As depicted in Figure 2, first a deep understanding of the legacy application is required [17] 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 migrated application in a cloud environment. In particular, applications to be deployed at the platform and software 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

[12] http://www.oracle.com/technetwork/java/index-136650.html

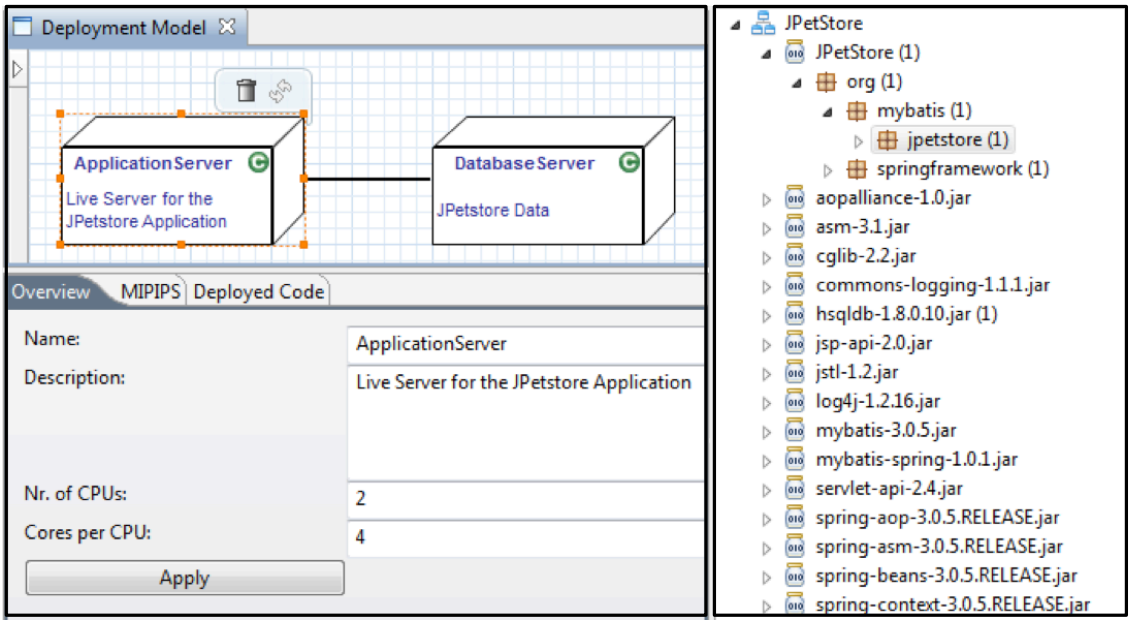

Figure 3: *CloudMIG by Example: (a) Deployment Model, (b) Discovered KDM Model*

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 EC2 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.

I. 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 [15], 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 services, e.g., which libraries have to be supported by PaaS providers.

II. Provide Selection: CloudML, USDL, SCORCH by Example

CloudML. This approach supports the description of infrastructure-related cloud services by describing service types offered by cloud providers mainly in terms of nodes and their properties. Listing 1 shows a concrete Amazon EC2 offering with some node 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 by Example: Service Description*

```
<?xml version="1.0" encoding="ASCII"?>
<serviceDescription:ServicesType ... version="AmazonEC2_M1">
  <nodes ID="M1 Medium Instance 1">
    <ram size="3.75" unit="GB"/>
    <cpu Architecture="64Bit" Cores="2"/>
    <storage size="410.0" unit="GB"/>
    <functionality functionality="Server"/>
    <operatingSystem operatingSystem="Windows7"/>
  </nodes>
  <locations country="US"/>
  <locations country="Europe"/>
</serviceDescription:ServicesType>
```

Listing 2: *CloudML by Example: Resource Description*

```
<?xml version="1.0" encoding="ASCII"?>
<infrastructureDescription:InfrastructureType ... ID="AmazonEC2_M1">
  <nodes ID="M1 Medium Instance 1"/>
  <nodes ID="M1 Medium Instance 2"/>
  <nodes ID="M1 Medium Instance N"/>
  <links ID="Link1"
         owner="M1 Medium Instance 1"/>
  <links ID="Link2"
         owner="M1 Medium Instance 2"/>
  <links ID="LinkN"
         owner="M1 Medium Instance N"/>
</infrastructureDescription:InfrastructureType>
```

USDL. This approach puts focus on the business perspective and provides concepts to describe non-technical aspects. As accounting information is one essential aspect of the "pay-as-you-go" principle in cloud environments, Figure 4 shows in the upper part the charges per hour of the standard Amazon EC2 offering. The price plan captures the cloud service for which a charge is defined, whereas the price metric refers to the unit of measurement by which the cloud service is charged.

In addition to pricing capabilities, USDL provides support for service levels. Amazon EC2 cloud services are committed to an availability agreement which is represented in the lower part of Figure 4. In the provided USDL service level model, Amazon's availability service level for EC2 offerings is depicted, which refers to the annual uptime during the service year.

SCORCH. More details about the internals of cloud environments is provided by SCORCH. Although a deep understanding of such internals is not necessarily required for cloud consumers, cloud providers have an interest in operating their environments energy efficiently and cost

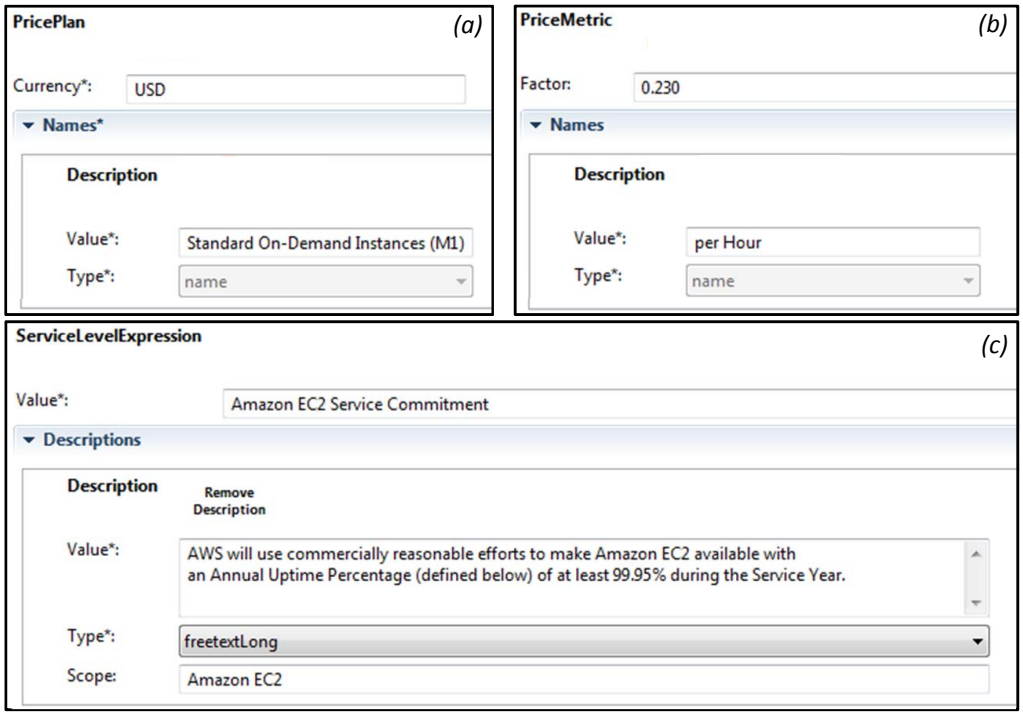

Figure 4: *USDL by Example: (a) Price Plan, (b) Price Metric, and (c) Service Level*

effectively, while ensuring that the cloud resources are virtualized and scheduled as expected by the cloud consumers.

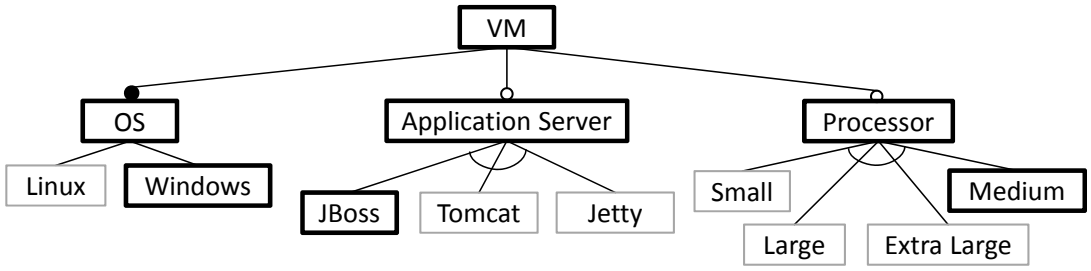

Figure 5: *SCORCH by Example: Feature Model based Configuration*

In the feature model of Figure 5, an excerpt of possible cloud environment configurations is depicted. The features shown with a thick frame represent the configuration assumed to be demanded by our Petstore scenario. While this configuration refers to the cloud consumer perspective, the SCORCH optimizer requires additional configuration models describing the cloud environment, e.g., CPU costs and CPU power consumption, which are part of the cloud provider perspective.

III. Perform Adaptations: RESERVOIR Manifest Language 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 the RESERVOIR Manifest Language.

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 run-time, we attach a *KPI* that refers to a session listener class to the Petstore's *WebUI* component (cf. Listing 3, line 4). 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 Manifest Language by Example: Application Description*

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <rADL:ApplicationDescription ... name="Petstore">
3    <components ovf="PWUI" qualifiedName="PetstoreWebUI">
4      <kpis category="MonitoringAgent" frequency="30.0"
5            qualifiedName="petstore.SessionListener.numberOfSessions"
6            unit="sec"/>
7    </components>
8    <components ovf="PC" qualifiedName="PetstoreController"/>
9    <components ovf="PD" qualifiedName="PetstoreDomain"/>
10   <components ovf="PSB" qualifiedName="PetstoreService"/>
11 </rADL:ApplicationDescription>
```

Listing 4: *RESERVOIR Manifest Language by Example: Elastic Rule*

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <reservoirERM:ElasticityRule ... name="ScaleOutAdjustment">
3    <trigger>
4      <expression>
5        <formula>
6          <formulaElement
7              xsi:type="reservoirERM:ElementSimpleValue"
8              value="petstore.SessionListener.numberOfSessions > N"/>
9        </formula>
10     </expression>
11   </trigger>
12   <actions run="deployNewVM"/>
13 </reservoirERM:ElasticityRule>
```

IV. Prepare Deployment: Blueprint, MOCCA, PIM4Cloud, TOSCA by Example

The majority of approaches provide modeling support to prepare the deployment for our Petstore scenario. In general, our deployment consist 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, REMICS PIM4Cloud and TOSCA in Figure 6.

Blueprint. This approach explicitly differentiates between implementation artifacts that typically refer to the software level and the resources at the platform and infrastructure level required for

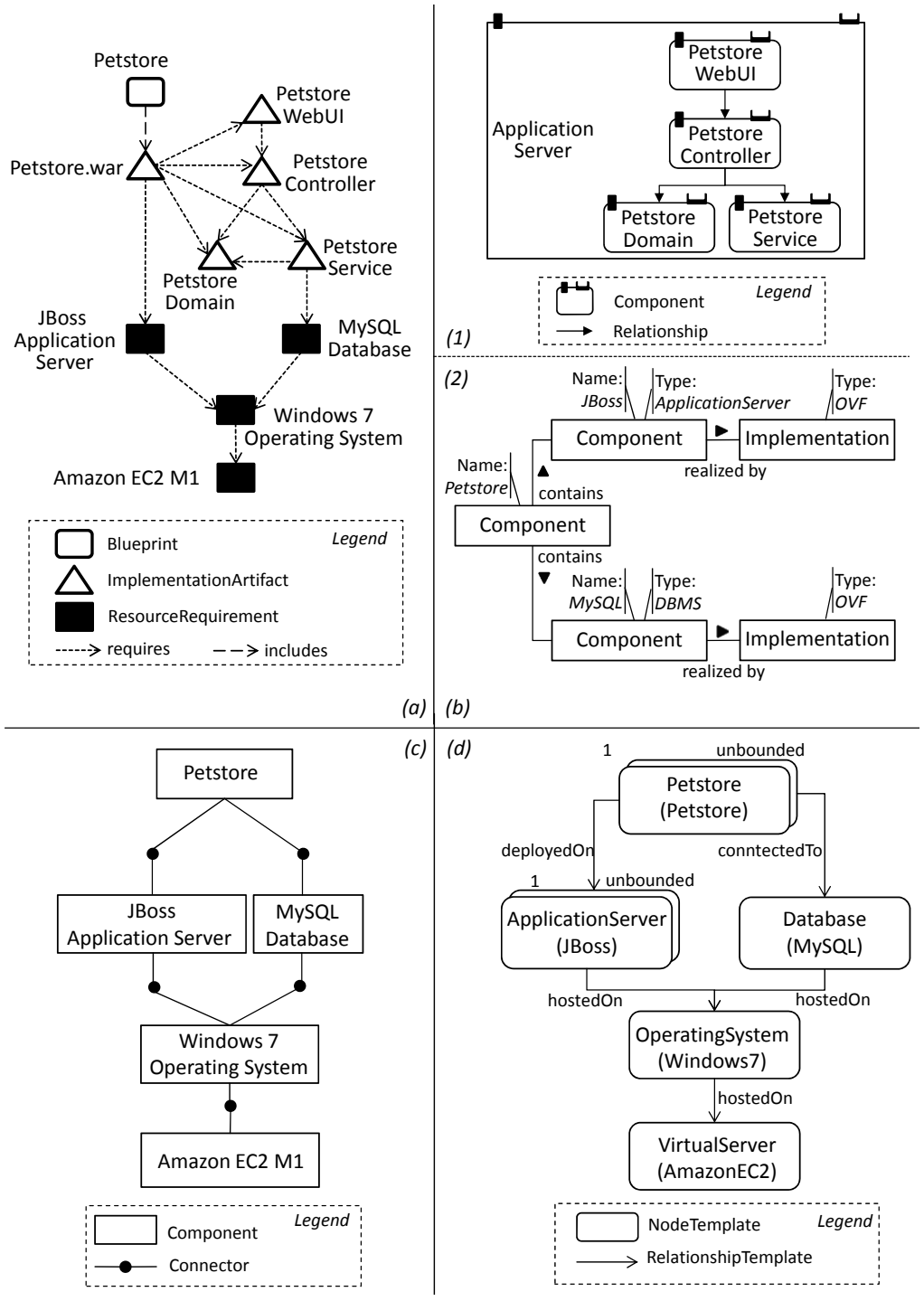

Figure 6: (*a*) *Blueprint by Example: Virtual Architecture Topology,* (*b*) *MOCCA by Example:* (1) *Deploymnet Model,* (2) *Cafe Archive Component Model,* (*c*) *REMICS PIM4Cloud by Example: Component Model, and* (*d*) *TOSCA by Example: Service Template*

their operation. We model the Petstore components in terms of implementation artifacts that require resources such as the represented application server, the database management system and the operating system, all of which are hosted on an Amazon EC2 environment.

MOCCA. In contrast to Blueprint, MOCCA advocates components to represent the Petstore scenario from the software level down the platform level, whereas the infrastructure level is 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.

PIM4Cloud. As depicted in Figure 6, PIM4Cloud proposes a component-oriented approach similar to MOCCA to represent a deployment. However, in PIM4Cloud a component subsumes both implementation artifacts and resource requirements of Blueprint, which is only partly the case in MOCCA.

TOSCA. In order to represent deployments, TOSCA provides nodes and relationships. Nodes in TOSCA are comparable to components in PIM4Cloud, although in TOSCA, nodes can be attached with multiplicities to express the number of potential running instances of a particular node. 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.

V. Execute Provisioning: REMICS CloudML by Example

The provisioning of cloud resources is supported by REMICS's second effort towards a cloud modeling language: CloudML. As shown in Listing 5, the requirements from a cloud consumer perspective for the Amazon EC2 solution are specified by the minimal required number of *Cores*, amount of *Disk* space and *RAM*, and the location of a node is represented.

These cloud resource requirements then serve as input for the REMICS *CloudMLEngine*, which is capable of actually provisioning the required cloud resources. Currently, connectors for Amazon EC2 and RackSpace are supported by the provided interpreter of REMICS CloudML.

Listing 5: *REMICS CloudML by Example: Resource Requirements*

```
<?xml version="1.0" encoding="ASCII"?>
<remicsCloudMl:Template xmi:version="2.0" ...
  <nodes id="Amazon EC2 M1 Medium Instance">
    <properties xsi:type="remicsCloudMl:Core" min="2"/>
    <properties xsi:type="remicsCloudMl:Disk" min="400"/>
    <properties xsi:type="remicsCloudMl:RAM" min="3"/>
    <properties xsi:type="remicsCloudMl:Location" value="Europe"/>
  </nodes>
</remicsCloudMl:Template>
```

VI. Synopsis

To analyze the selected approaches in terms of their capabilities with respect to cloud computing, we use the dimensions of cloud computing as afore discussed in Section II and summarize in Table 2 the support provided by each approach. We can observe three areas of interest. First, we note that while all approaches support the mapping of cloud resources, only very few of them provide an automatic matching of resources to application requirements. Second, it is not surprising that approaches, which stem from the domain of service-oriented architectures, i.e., Blueprint, TOSCA, and USDL, provide the most complete support for this characteristic of cloud

Approach	On-Premise Environment			Wiring		Cloud Environment									
						Service			Resource			Specifics			
	Infrastructure	Platform	Software	Mapping	Matching	Infrastructure	Platform	Software	Infrastructure	Platform	Software	Dynamic Provisioning	Pay-as-you-Go Principle	Elastic Scalability	Quality of Service
Blueprint	~	+	~	+	-	+	+	+	~	~	~	-	-	-	-
CloudMIG	+	+	+	+	+	+	~	-	+	~	-	-	+	-	~
CloudML	+	-	-	+	-	+	-	-	+	-	-	-	-	-	-
MOCCA	~	+	~	+	-	-	-	-	~	~	-	~	-	-	-
PIM4Cloud	~	~	~	+	-	-	-	-	~	~	-	-	-	-	-
REMICS CloudML	+	-	-	+	+	+	-	-	-	-	-	+	-	-	-
RESERVOIR Manifest Language	+	~	~	+	-	-	-	-	+	~	-	+	-	+	-
SCORCH	~	~	-	+	-	-	-	-	+	~	-	~	~	+	-
TOSCA	~	~	~	+	-	+	+	+	~	~	~	~	-	-	-
USDL	~	~	~	+	-	+	+	+	~	~	+	-	+	-	+

Table 2: *Domain Characteristics of Reviewed Approaches*

environments. Finally, the support for cloud-specific characteristics in all approaches is still very initial at the time of this study.

V. Observations and Research Directions towards a Unified Cloud Modeling Language

The review of existing modeling approaches in the domain of cloud computing has revealed several interesting observations. In the following, we summarize these observations by discussing (i) the *formality* of the proposed modeling languages according to common language characteristics, (ii) the *diffusion* of these languages among them and existing standard model languages and (iii) their *applicability* with respect to the specifics of cloud computing. For each observation, we present research directions that will lead towards a unified cloud modeling language.

I. Modeling Language Formality

Observation: Formal metamodels not always publicly available. Although a formal metamodel leverages the applicability and interpretation of a modeling language, only half of the cloud modeling approaches we demonstrated in this article provide such a metamodel for the public. While Blueprint, CloudML and TOSCA define their modeling elements in terms of XML Schemas, CloudMIG and USDL provide an Ecore-based metamodel. The other half of the approaches sketches their proposed modeling elements using informal graphical notation elements. We re-modeled the metamodels for these approaches with Ecore to demonstrate their representational capabilities and to ensure that at least all the models presented in this article are valid instances of their

metamodel. The ability to validate metamodel extensions is itself an important benefit, but a formal metamodel also enables the definition of transformation rules on metamodels or the reuse of metamodels.

Research directions. Ideally, proposals for new modeling languages come together with a machine-interpretable metamodel preferably defined on the basis of a commonly accepted format, such as MOF or Ecore. Guidelines for defining good quality metamodels are presented in the literature (e.g., [37, 39, 49]). Once they are defined, sharing them in terms of an open repository (e.g., AtlanMod's Metamodel Zoo[13] or ReMoDD[14]) allows them to be accessed and may further stimulate their reuse in the development of new modeling languages.

Observation: Semantics matters. Of course, semantics always matters for modeling languages because it is the foundation for correct interpretation and automation. While CloudML provides modeling capabilities to describe infrastructure-related aspects, and TOSCA as well as Blueprint allow expressing deployment plans and dependencies between deployable software artifacts at a useful level of granularity, it remains unclear how they map to formats interpretable by cloud environments. MOCCA and the RESERVOIR Manifest Language build on top of OVF, which is interpretable by several cloud providers. Hence, OVF is used as a reference for providing meaning to their deployment-related modeling capabilities. Furthermore, the RESERVOIR Manifest Language is integrated in the OpenNebula environment which gives the necessary semantics for the execution of elasticity rules. In a similar way, CloudMIG's CEM is operationalized by the provided tool-support and mapped to some extent to KDM, which can again be considered as a reference semantics, though it is not directly interpretable. Models created in PIM4Cloud are executable as the provided language is embedded in Scala. REMICS CloudML provides an interpreter for executing the provisioning of cloud resources that builds on top of the jClouds framework. Finally, SCORCH translates all models into a CSP and uses a standard CSP solver for doing the calculations. In summary, most of the approaches give meaning to their modeling elements by referring/translating to other (modeling) languages (e.g., CloudMIG, MOCCA, RESERVOIR Manifest Language, SCORCH) or providing tool-support in the sense of interpreters (e.g., CloudMIG, PIM4Cloud, REMICS CloudML, RESERVOIR Manifest Language), while only a few approaches do not go beyond English prose in this respect.

Research directions. Standardized techniques for defining the semantics of a modeling language are still not available [16]. In practice, a mixture of existing techniques are applied, which is also reflected by our observation. Defining references to another (modeling) language is a lightweight technique to provide meaning for a modeling language though it is important to select a reference language which is well-known by the target audience of the modeling language. Standardized, well-established languages may serve as preferred candidates for such references. Although individually developed code generators for modeling languages may serve well for operationalizing a metamodel, the semantics is still not explicitly represented. Clearly, a single representation that usefully denotes the semantics of a modeling language would be the preferred way [16].

II. Modeling Language Diffusion

Observation: High diversity in current cloud modeling approaches. Current cloud modeling approaches pursue different goals, propose, hence, 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

[13] http://www.emn.fr/z-info/atlanmod/index.php/Zoos
[14] http://www.cs.colostate.edu/remodd/v1/

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, Blueprint, CloudMIG, MOCCA, PIM4Cloud and TOSCA deal with the deployment to cloud environments and the description of cloud services is addressed by Blueprint, CloudML, TOSCA and USDL. Interestingly, however, we notice that these approaches are hardly aware of each other and we only found one work [19] that deals with identifying potential integration points between two approaches, i.e., TOSCA and USDL. Thus, 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 [36] since useful mappings, which are the basis for an integration, can otherwise hardly be identified. A common metamodel [4] may serve as a useful means in such an integration endeavor.

Observation: Little attention paid to standard modeling languages. The cloud modeling approaches studied in this article indisputably introduce novel modeling elements for cloud computing. At the same time, standard modeling languages, such as UML, already provide a rich set of elements to model software-, platform-, and infrastructure-related artifacts. Nevertheless, most approaches re-introduce slightly modified versions of component-like (e.g., MOCCA, TOSCA, PIM4Cloud, and the RESERVOIR Manifest Language) and deployment-like (e.g., CloudMIG and REMICS CloudML) modeling elements well known from UML. While the need for such modeling capabilities is clear, it is surprising that the reuse of existing modeling standards is neglected. As a result, these approaches are hardly compatible to well-established standard software modeling languages, which also limits their practical applicability in a broad spectrum of scenarios.

Research directions. Instead of introducing slightly adapted UML-like modeling elements without linking to UML, an alternative is to provide at least compatibility with the UML metamodel through, for instance, model transformations describing the correspondences between the languages' elements in order to enable partial model exchange, or even better to extend the UML metamodel in a light-weight manner through, for instance, UML profiles. Such profiles can then cover the specifics of the cloud computing domain. As a result, not only the full expressive power of UML can be exploited but also the reuse of existing UML models [33,34] can be ensured.

III. Modeling Language Applicability

Observation: Non-functional requirements as first-class entities. 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 USDL, service descriptions of cloud-based offerings address pricing mainly for the purpose of informing cloud consumers in the sense of a service marketplace. CloudMIG covers pricing information of cloud environments for the optimization of software deployments [26]. SCORCH exploits cost information for optimizing the internals of cloud environments concerning pre-instantiation of virtual machines. Apart from pricing, USDL provides modeling capabilities for service levels which can be used to represent cloud providers commitments to certain non-functional properties, such as service availability. The RESERVOIR Manifest Language 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 [32] 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 [17] and even dedicated views [23] 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 the RESERVOIR Manifest Language.

Research directions. Different cloud-based migration types and their potential impact on adaptations for each of these types is discussed in Andrikopoulos *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 towards a unified cloud modeling language, 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 article, 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 towards a unified cloud modeling language. Even though unification 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 an approach based on multi-views, 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 point of view of the cloud service consumer. In this context, the most important challenge is how to align cloud modeling languages with existing modeling standards such as UML or SysML.

References

[1] Andrikopoulos, V., Binz, T., Leymann, F., Strauch, S.: How to Adapt Applications for the Cloud Environment. Computing **95**(6), 493–535 (2013)

[2] Ardagna, D., Nitto, E.D., Mohagheghi, P., Mosser, S., Ballagny, C., D'Andria, F., Casale, G., Matthews, P., Nechifor, C.S., Petcu, D., Gericke, A., Sheridan, C.: MODAClouds: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. In: Proc. Intl. Workshop on Modeling in Software Engineering (MISE), pp. 50–56 (2012)

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)

[4] Atzeni, P., Cappellari, P., Bernstein, P.A.: ModelGen: Model Independent Schema Translation. In: Proc. Intl. Conf. on Data Engineering (ICDE), pp. 1111–1112 (2005)

[5] Badger, M.L., Grance, T., Patt-Corner, R., Voas, J.M.: Cloud Computing Synopsis and Recommendations. Tech. rep., NIST Computer Security Division (2012)

[6] Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009)

[7] Bergmayr, A., Bruneliere, H., Cánovas Izquierdo, J.L., Gorroñogoitia, J., Kousiouris, G., Kyriazis, D., Langer, P., Menychtas, A., Orue-Echevarria Arrieta, L., Pezuela, C., Wimmer, M.: Migrating Legacy Software to the Cloud with ARTIST. In: Proc. European Conf. on Software Maintenance and Reengineering (CSMR), pp. 465–468 (2013)

[8] Bézivin, J.: On the Unification Power of Models. Software and System Modeling **4**(2), 171–188 (2005)

[9] Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Computing **16**(3), 80–85 (2012)

[10] Bjørner, D.: Software Engineering 1: Abstraction and Modeling. Software Engineering. Springer (2010)

[11] Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool (2012)

[12] Brandzæg, E., Mohagheghi, P., Mosser, S.: Towards a Domain-Specific Language to Deploy Applications in the Cloud. In: Proc. Intl. Conf. on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING), pp. 213–218 (2012)

[13] Brandzæg, E., Mosser, S., Mohagheghi, P.: Towards CloudML, a Model-Based Approach to Provision Resources in the Clouds. In: Proc. Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE) (2012)

[14] Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: Proc. Intl. Conf. on Cooperative Information Systems, pp. 416–424 (2012)

[15] Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering. In: Proc. Intl. Conf. on Automated Software Engineering (ASE), pp. 173–174 (2010)

[16] Bryant, B.R., Gray, J., Mernik, M., Clarke, P.J., France, R.B., Karsai, G.: Challenges and Directions in Formalizing the Semantics of Modeling Languages. Comput. Sci. Inf. Syst. **8**(2), 225–253 (2011)

[17] Canfora, G., Penta, M.D., Cerulo, L.: Achievements and Challenges in Software Reverse Engineering. Commun. ACM **54**(4), 142–151 (2011)

[18] Cardoso, J., Barros, A., May, N., Kylau, U.: Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments. In: Proc. Intl. Conf. on Services Computing (SCC), pp. 602–609 (2010)

[19] Cardoso, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: Cloud Computing Automation: Integrating USDL and TOSCA. In: Proc. Intl. Conf. on Advanced Information Systems Engineering (CAiSE), pp. 1–16 (2013)

[20] Chapman, C., Emmerich, W., Márquez, F.G., Clayman, S., Gallis, A.: Software Architecture Definition for On-Demand Cloud Provisioning. Cluster Comput. **15**(2), 79–100 (2012)

[21] DMTF, Open Virtualization Format (OVF) (2013). URL http://dmtf.org/standards/ovf. Version 2.0.0

[22] Dougherty, B., White, J., Schmidt, D.C.: Model-Driven Auto-Scaling of Green Cloud Computing Infrastructure. Future Generation Computer Systems **28**(2), 371–378 (2011)

[23] Ducasse, S., Pollet, D.: Software architecture reconstruction: A process-oriented taxonomy. IEEE Trans. Software Eng. **35**(4), 573–591 (2009)

[24] France, R., Rumpe, B.: Modeling for the cloud. Software and System Modeling **9**(2), 139–140 (2010)

[25] France, R., Rumpe, B.: The Evolution of Modeling Research Challenges. Software and Systems Modeling **12**(2), 223–225 (2013)

[26] Frey, S., Fittkau, F., Hasselbring, W.: Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud. In: Proc. Intl. Conf. on Software Engineering (ICSE), pp. 512–521 (2013)

[27] Frey, S., Hasselbring, W.: The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. Intl. J. Advances in Software **4**(3&4), 342–353 (2011)

[28] Frey, S., Hasselbring, W., Schnoor, B.: Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms. J. Softw. Maint. Evol.: Res. Pract. (2012)

[29] Glinz, M.: On Non-Functional Requirements. In: Proc. Intl. Conf. on Requirements Engineering, pp. 21–26 (2007)

[30] Gonçalves, G., Endo, P., Santos, M., Sadok, D., Kelner, J., Merlander, B., Mångs, J.E.: CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds. In: Proc. Intl. Conf. on Cloud Computing Technologies and Science (CloudCom), pp. 399–406 (2011)

[31] Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer **37**(10), 64–72 (2004)

[32] van Hoorn, A., Waller, J., Hasselbring, W.: Kieker: a framework for application performance monitoring and dynamic software analysis. In: Proc. Int. Conf. on Performance Engineering, pp. 247–248 (2012)

[33] Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven Engineering Practices in Industry. In: Proc. Intl. Conf. on Software Engineering (ICSE), pp. 633–642 (2011)

[34] Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical Assessment of MDE in Industry. In: Proc. Intl. Conf. on Software Engineering (ICSE), pp. 471–480 (2011)

[35] Jeffery, K., Horn, G., Schubert, L.: A Vision for Better Cloud Applications. In: Proc. Intl. Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud), pp. 7–12 (2013)

[36] Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In: Proc. Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS), pp. 528–542 (2006)

[37] Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schindler, M., Völkel, S.: Design Guidelines for Domain Specific Languages. In: Proc. Intl. Workshop on Domain-Specific Modeling (DSM), pp. 7–13 (2009)

[38] Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving Applications to the Cloud: An Approach Based on Application Model Enrichment. Int. J. Cooperative Inf. Syst. **20**(3), 307–356 (2011)

[39] Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM Comput. Surv. **37**(4), 316–344 (2005)

[40] Mohagheghi, P., Berre, A.J., Henry, A., Barbier, F., Sadovykh, A.: REMICS - Reuse and Migration of Legacy Applications to Interoperable Cloud Services. In: Towards a Service-Based Internet, *LNCS*, vol. 6481, pp. 195–196. Springer (2010)

[41] Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Trans. Software Eng. **35**(6), 756–779 (2009)

[42] Nguyen, D.K., Lelli, F., Papazoglou, M.P., van den Heuvel, W.J.: Blueprinting Approach in Support of Cloud Computing. Future Internet **4**(1), 322–346 (2012)

[43] Nguyen, D.K., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M.P., van den Heuvel, W.J.: Blueprint Template Support for Engineering Cloud-Based Services. In: Proc. European ServiceWave Conference, pp. 26–37 (2011)

[44] OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA) (2013). URL https://www.oasis-open.org/committees/tosca. Version 1.0

[45] OMG, Knowledge Discovery Model (KDM) (2011). URL http://www.omg.org/technology/kdm. Version 1.3

[46] Papazoglou, M.P., Vaquero, L.M.: Knowledge-Intensive Cloud Services: Transforming the Cloud Delivery Stack. In: J. Kantola, W. Karwowski (eds.) Knowledge Service Engineering Handbook, pp. 447–492. CRC Press (2012)

[47] Selic, B.: MDA Manifestations. UPGRADE: The European Journal for the Informatics Professional **9**(2), 12–16 (2008)

[48] Selic, B.: The Less Well Known UML. In: Formal Methods for Model-Driven Engineering, *LNCS*, vol. 7320, pp. 1–20. Springer (2012)

[49] Spinellis, D.: Notable Design Patterns for Domain-Specific Languages. J. Syst. Softw. **56**(1), 91–99 (2001)

[50] Sun, L., Dong, H., Ashraf, J.: Survey of Service Description Languages and Their Issues in Cloud Computing. In: Proc. Intl. Conf. on Semantics, Knowledge and Grids (SKG), pp. 128–135 (2012)

[51] Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Dynamically Scaling Applications in the Cloud. SIGCOMM Comput. Commun. Rev. **41**(1), 45–52 (2011)

[52] Venters, W., Whitley, E.A.: A Critical Review of Cloud Computing: Researching Desires and Realities. JIT **27**(3), 179–197 (2012)

[53] Walker, E.: The Real Cost of a CPU Hour. Computer **42**(4), 35–41 (2009)