A Systematic Review of Cloud Modeling Languages

ALEXANDER BERGMAYR, team
UWE BREITENBÜCHER, University of Stuttgart
NICOLAS FERRY, SINTEF
ALESSANDRO ROSSINI, EVRY
ARNOR SOLBERG, SINTEF
MANUEL WIMMER, TU Wien
GERTI KAPPEL, TU Wien
FRANK LEYMANN, University of Stuttgart

Modern cloud computing environments support a relatively high degree of automation in service provisioning, which allows cloud service customers (CSC) to dynamically acquire services required for deploying cloud applications. Cloud modeling languages (CMLs) have been proposed to address the diversity of features provided by cloud computing environments and support different application scenarios, *e.g.,* migrating existing applications to the cloud, developing new cloud applications, or optimizing them. There is, however, still much debate in the research community on what a CML is and what aspects of a cloud application and its target cloud computing environment should be modeled by a CML. Furthermore, the distinction between CMLs on a fine-grained level exposing their modeling concepts is rarely made. In this article, we investigate the diverse features currently provided by existing CMLs. We classify and compare them according to a common framework with the goal to support CSCs in selecting the CML which fits the needs of their application scenario and setting. As a result, not only features of existing CMLs are pointed out for which extensive support is already provided but also in which existing CMLs are deficient, thereby suggesting a research agenda.

CCS Concepts: **•Software and its engineering → Domain specific languages;**

Additional Key Words and Phrases: cloud computing, domain-specific languages, modeling

ACM Reference Format:
Alexander Bergmayr, Uwe Breitenbücher, Nicolas Ferry, Alessandro Rossini, Arnor Solberg, Manuel Wimmer, Gerti Kappel, and Frank Leymann, 2018. A Systematic Review of Cloud Modeling Languages. *ACM Comput. Surv.* 51, 1, Article 22 (February 2018), 35 pages.
DOI: https://doi.org/10.1145/3150227

1. INTRODUCTION

With the emergence of cloud computing, the effort required to get access to environments with the scale of large distributed data centers has tremendously been reduced. Provisioning resources of a cloud computing environment[1] [Badger et al. 2012; ISO/IEC 2014b] can be carried out on demand [Leymann 2011], without the need to negotiate with the cloud service provider (CSP) [ISO/IEC 2014a]. This is because their offerings are considered as commodities that are readily available as a service and consumable over the network even on a per minute basis. Companies are no longer forced to plan far ahead for resource provisioning [Armbrust et al. 2010]. Instead, the large-scale data centers of today's CSPs ensure that resources are available through the services of

[1]In the following, we use the term "cloud environment" for the sake of brevity

Author's addresses: A. Bergmayr, team Communication Technology Management GmbH, Austria; M. Wimmer and G. Kappel, Business Informatics Group (BIG), TU Wien, Austria; U. Breitenbücher and F. Leymann, Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Germany; N. Ferry and A. Solberg, SINTEF, Norway; A. Rossini, EVRY, Norway.

DOI: https://doi.org/10.1145/3150227

their cloud environments. Hence, cloud computing refers to both the applications hosted on a cloud environment and the systems required to operate its internal resources and expose them as cloud services (*e.g.,* Amazon EC2). As cloud environments offer novel optimization opportunities, *e.g.,* advanced scalable data persistence solutions, applications have to be tailored to exploit them [Andrikopoulos et al. 2013] fully. At the same time, cloud applications need to comply with certain peculiarities of cloud environments that might hinder their functioning, *e.g.,* stateful components in a highly scalable cloud environment. Ideally, architectural decisions can be expressed in terms of models as a basis for the development of cloud applications [France and Rumpe 2010]. This approach calls for appropriate modeling languages and tools that render it truly usable and useful.

The current field of cloud computing encompasses a multitude of different CSPs. The ability to run and manage applications on different cloud systems without going into their technical configuration peculiarities may enlarge the potential set of cloud service customers (CSC)[2] [Sheth and Ranabahu 2010]. However, current cloud solutions are typically heterogeneous, and the provided configuration features are often incompatible. This diversity is a significant obstacle for the realizing the full potential of cloud computing since it harms interoperability and increases vendor lock-in, as well as it requires dedicated CSP specific knowledge for development and administration of cloud systems [Marston et al. 2011]. This challenge needs to be addressed accordingly.

Several languages emerged with partially overlapping but also diverse concepts to represent cloud applications at the model level and to address the diversity of today's cloud environments and their offered services. A standard called TOSCA [OASIS 2013] for representing portable cloud applications and supporting their life-cycle management was adopted by the Organization for the Advancement of Structured Information Standards (OASIS) in late 2013. In this article, we collectively refer to them as *cloud modeling language (CML)*. Generally, a CML can be considered as a domain specific language (DSL) [Mernik et al. 2005] where the domain refers to cloud computing. There is, however, still much debate on what a CML is, what aspects of a cloud application and the target cloud environment should be modeled by a CML, and which of the existing CMLs is appropriate for a particular problem. For example, some CMLs emphasize virtual machine (VM) configuration required for the provisioning of compute services with custom software stacks (*e.g.,* the approach by Nhan *et al.* [Nhan et al. 2012]). Others address also networking aspects such as custom addressing and segmentation of launched VMs (*e.g.,* CloudNaaS [Benson et al. 2011]). While those languages solely target infrastructure services, some CMLs turn the focus on platform services (*e.g.,* StratusML [Hamdaqa et al. 2011]). Independent of the addressed cloud service categories [Badger et al. 2012; ISO/IEC 2014a], some CMLs support the representation of elasticity rules to trigger the provisioning of a compute service if a certain threshold is exceeded (*e.g.,* RESERVOIR-ML [Chapman et al. 2012]). Moreover, they provide dedicated tools to allow CSCs seeking for compute services that satisfy their requirements in terms of performance and costs (*e.g.,* CloudMIG [Frey et al. 2013a]).

Consequently, there is an urgent need to investigate the diverse features of current CMLs. Existing surveys by Papazoglou and Vaquero [Papazoglou and Vaquero 2012] and Sun *et al.* [Sun et al. 2012a] mostly analyze general description languages for service-oriented architectures and low-level formats for resource virtualization with respect to their applicability to cloud computing. Generic service description language can be considered as a source of inspiration for current CMLs as they are often capable of capturing services offered by cloud environments. Some CMLs exploit existing formats for resource virtualization for model serialization. As a result, models created by a CML can directly be interpreted by a cloud environment that supports the selected format. One of the obvious reasons why most of the current existing CMLs were not considered by the surveys of Papazoglou and Vaquero and Sun *et al.* is that the majority of CMLs emerged around or shortly after they carried out their surveys. In the survey of Silva *et al.* [Silva et al. 2013] a systematic literature review (SLR) regarding existing solutions that address the "vendor lock-in" problem in the con-

[2]A cloud service customer (CSC) [ISO/IEC 2014a] consumes services offered by a cloud environment. A cloud service provider (CSP) [ISO/IEC 2014a] operates a cloud environment and manages the exposed services.

text of cloud computing is presented, whereas Jamshidi *et al.* [Jamshidi et al. 2013] conducted an SLR of cloud migration research. However, their surveys do not focus on CMLs. More recently, the representational capabilities of some CMLs were demonstrated in the setting of a cloud migration scenario [Bergmayr et al. 2014b].

This article builds upon the results of existing efforts. It is further influenced by insights gained from investigating individual CMLs, language concepts relevant in the context of architecture modeling and software modeling, features of current cloud environments, and experiences and needs of recently completed and ongoing research projects, ARTIST[3] [Bergmayr et al. 2013], MODAClouds[4] [Ardagna et al. 2012], PaaSage[5] [Jeffery et al. 2013], and REMICS[6] [Mohagheghi et al. 2010]. The systematic review of CMLs presented in this article follows the guidelines of Kitchenham and Charters [Kitchenham and Charters 2007]. We discuss how CMLs differ from architecture description languages (ADLs) because the influence of ADLs on current CMLs is obvious. To classify and compare existing CMLs, we present a concise framework with the main emphasis on their modeling capabilities and the toolset which comes with them. Please note that in this survey we focus on CMLs but omit microservice and container orchestration platforms such as LXC[7] and Kubernetes[8].

The remainder of this article is structured as follows. In Section 2, we discuss related surveys and emphasize the need of this systematic literature review of current CMLs. The classification and comparison framework applied by this review is defined in Section 3. In Section 4, we present the process that was carried out to conduct the review. Our findings obtained from classifying and comparing 19 selected CMLs are presented in Section 5 before we conclude in Section 6.

2. RELATED SURVEYS

In the work of Papazoglou and Vaquero [Papazoglou and Vaquero 2012], the need for knowledge-intensive cloud services[ISO/IEC 2014a] that comprise metadata (*e.g.,* offered services, quality of a service, available service level agreements, technical service specification) of cloud environments is motivated. Currently, the metadata are spread over and confined to the main cloud service categories (*i.e.,* IaaS, PaaS, SaaS) of such environments. As a consequence, they argue for 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 used in the context of service-oriented architecture (*e.g.,* [Mietzner et al. 2008]) to low-level formats describing web resources (*e.g.,* RDF) or virtual resources (*e.g.,* OVF). We share the approach of Papazoglou and Vaquero to use the cloud service categories as introduced by the NIST [Badger et al. 2012] and ISO/IEC [ISO/IEC 2014a; ISO/IEC 2014b] to categorize existing CMLs regarding the target cloud environment they support. However, we focus exclusively on modeling languages tailored to the cloud computing domain, hence claim to be what we call CMLs. As a result, we use more fine-grained criteria to analyze existing CMLs compared to the work of Papazoglou and Vaquero.

Sun *et al.* [Sun et al. 2012a] present a survey that considers seven different aspects of service description languages: domain, coverage, purpose, representation, semantics, intended user, and feature.[9] By analyzing common modeling language characteristics (*i.e.,* coverage, purpose, semantics), their capabilities (*i.e.,* representation) and intended users with respect to the cloud computing domain, we cover all the aspects of this article. In contrast to our work, Sun *et al.* do not further

[3]http://www.artist-project.eu
[4]http://www.modaclouds.eu
[5]http://www.paasage.eu
[6]http://www.remics.eu
[7]https://linuxcontainers.org/lxc
[8]https://kubernetes.io
[9]It is used to capture additional informal comments over a language rather than to provide a feature-based analysis [Kyo et al. 1990].

refine the domain aspect, which is because their scope goes beyond cloud computing. As a result, they include languages used in the context of service-oriented architecture (*e.g.,* SoaML) or semantic web (*e.g.,* OWL-S).

Jamshidi *et al.* [Jamshidi et al. 2013] conducted an 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.* do not focus on modeling techniques and languages in the cloud computing context, 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.

Silva *et al.* [Silva et al. 2013] also conducted an SLR regarding existing solutions that address the "vendor lock-in" problem in the context of cloud computing. They point out that the dependency on a certain cloud environment is a major obstacle to cloud adoption [Dillon et al. 2010]. From an initial set of 721 primary studies, 78 were selected and categorized according to 25 solution types dealing with the portability of cloud applications and how the interoperability between offered cloud services can be improved. Even though some of the introduced solutions types indicate that modeling techniques and languages can counteract portability and interoperability challenges, Silva *et al.* do not further categorize or compare them in terms of more fine-grained criteria. As some of the reviewed CMLs mainly aim for portable cloud applications, the work of Silva *et al.* also further motivates our work.

3. REVIEW FRAMEWORK

The diversity of features provided by today's cloud environments and existing challenges cloud adopters are faced with [Benslimane et al. 2014] has led to the design and development of several CMLs. They have different origins, pursue different goals, and hence provide a partially overlapping but also diverse language features. Still, a closer study of the set of features they propose and their main purpose shows that there is a common theme among them, which we exploited to elaborate our framework for classifying and comparing CMLs.

To establish a thorough framework, we studied the features of individual CMLs and work in the field of cloud computing that discuss core domain concepts. Furthermore, we extracted common characteristics of modeling languages from work in the area of language engineering. While, to a great extent, our classification and comparison framework captures categories supported by all or most existing CMLs, we also argue for features that are only supported by a few of them. They have either been identified in the literature as important to develop cloud applications or have resulted from our own experience gained from participating in research projects such as ARTIST [Bergmayr et al. 2013], MODAClouds [Ardagna et al. 2012], PaaSage [Jeffery et al. 2013] and REMICS [Mohagheghi et al. 2010]. Finally, to validate the practical relevance of our framework, we analyzed features of current cloud environments (*e.g.,* Amazon AWS, Google Cloud Platform, and Microsoft Azure) and concepts of programming libraries (*e.g.,* jclouds[10] and Deltacloud[11], now retired) that provide an abstraction layer on top of cloud environments. In fact, such libraries enable CSCs to connect to cloud environments for carrying out the software provisioning of cloud applications.

Now that we have discussed how our framework has been developed, Figure 1 depicts its main categories and, where appropriate, provides possible manifestations for them. We developed a metamodel for our framework, which enables us to provide a model conforming to this metamodel for the results of each reviewed CML. Providing the framework in terms of a metamodel allows extensions and modifications, which is crucial in a field that is still largely in its infancy.

Considering the *language scope*, we summarize the pragmatics for each reviewed CML and classify them according to widely accepted cloud service categories, *i.e.,* IaaS, PaaS, SaaS, considered as a target. Common *language characteristics* refer to the syntax and semantics of a CML, how

[10]jclouds: https://jclouds.apache.org
[11]Deltacloud: https://deltacloud.apache.org

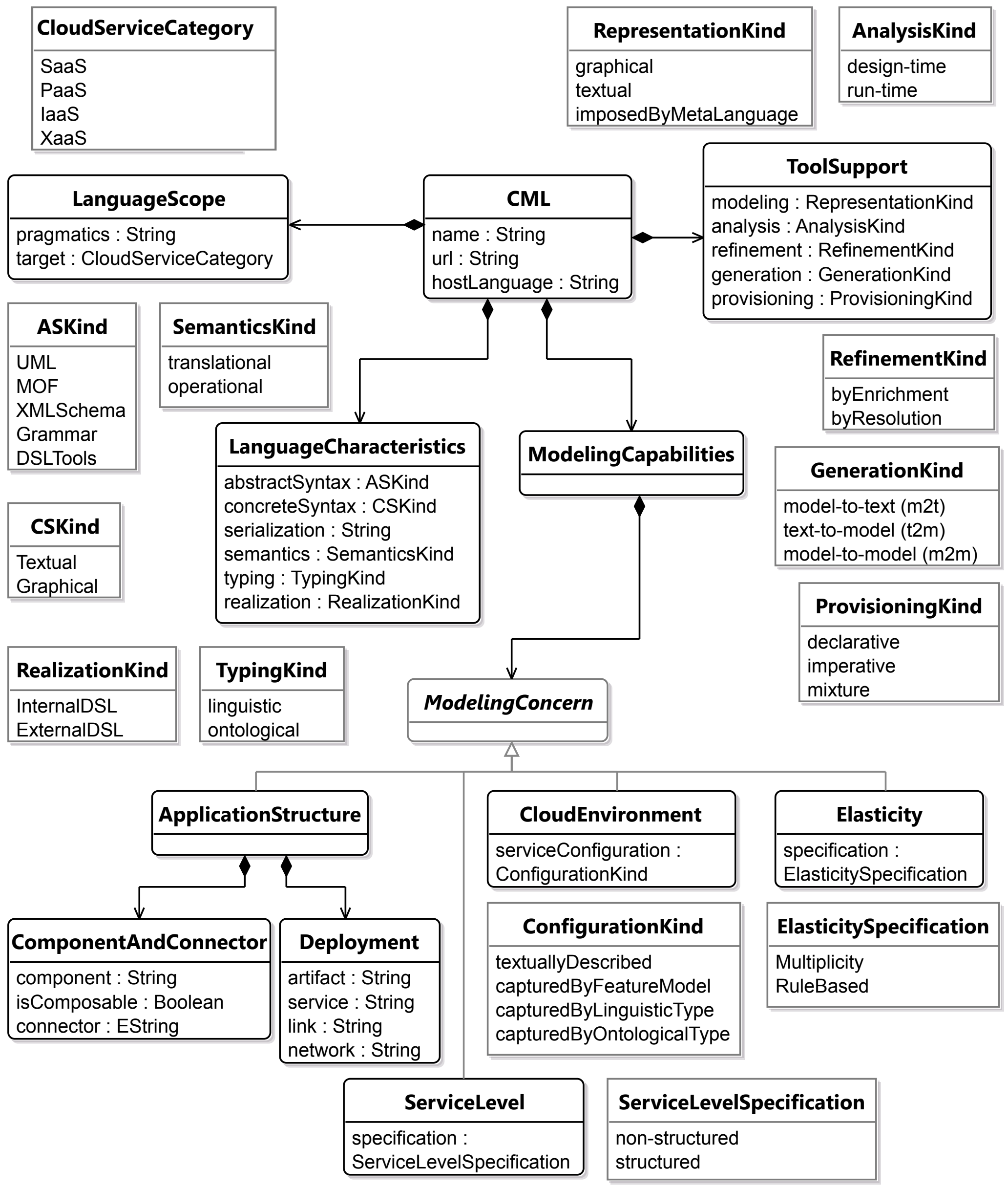

Fig. 1: Classification and comparison framework for CMLs.

it has been realized, and the different kinds of typing mechanisms (*i.e.,* linguistic and ontological) that are supported. The different typing mechanisms seem to be not only relevant from a language engineering perspective but also from an application one, as ontological typing allows extensions to a language without manipulating its definition.

Modeling capabilities of a CML turn the focus on the core domain concepts to model the structure of a cloud application, the services of a cloud environment required to operate the cloud application, and the interconnection between the application and its environment. Cloud applications need to be decomposed into components because the deployment of a cloud application usually enforces to distribute them across a single or even multiple cloud environments [Andrikopoulos et al. 2014c; Petcu 2014]. As a result, a CML must support CSCs to model two essential concerns: (i) cloud environments in terms of their offered services and (ii) cloud application structure in terms of components and their deployment on cloud services. The deployment of application components on cloud services defines their interconnection. Several other technical concerns (*e.g.,* elasticity) and non-technical concerns (*e.g.,* pricing) are desirable, but not sufficient to argue that a given language is not a CML. At the same time, representing an application's structure is not uncommon in the context of architecture modeling [Clements et al. 2003]. In fact, architecture description languages (ADLs) provide concepts to model the high-level structure of an application [Medvidovic and Taylor 2000; Medvidovic et al. 2002]. What differentiates now a CML from an ADL? A CML can be considered as an ADL for a particular domain. However, syntactic elements of a CML capture cloud computing vocabulary, which is usually not the case for a general purpose ADL. As a result, the semantics given to a CML is more specific compared to the semantics of general purpose ADLs intended to be applied to arbitrary domains. For instance, the semantics of a CML can be grounded in translators to executable languages or frameworks in the cloud computing context (*e.g.,* Google App Engine or Microsoft Orleans[12]) or engines that initiate the provisioning of modeled cloud services including the application components on top of them. The latter motivates the importance of explicitly representing the deployment of a cloud application as it specifies the desired state that triggers the provisioning process. Hence, the semantics given to a language and the two essential capabilities of representing the structure of a cloud application and their deployment on cloud services enable us to determine whether or not a certain language is a CML. Finally, even though the suitability of a CML is independent of the provided *tool support*, it appears clear that accompanied tools are beneficial for a CML's usefulness.

3.1. Language scope

3.1.1. Pragmatics. The *pragmatics* of a CML refers to its intended purpose including the overall goal that is pursued. The intended purpose of a modeling language can range from sketching software architectures, over to specifying blueprints for manually realizing application components, over to creating models for generating implementations or directly interpreting or even executing them. Models are not only applied in a generative manner, but more and more they are used analytically in software engineering, *e.g.,* for design-space exploration, optimization, validation, or even for verification. It is worth noting that there is a strong influence of the pragmatics on language characteristics [Karsai et al. 2009], such as syntax and semantics, and how the language is realized.

3.1.2. Target. Cloud environments considered as *target* of a CML can be differentiated according to the commonly accepted cloud service categories [Armbrust et al. 2010; ISO/IEC 2014a]: *infrastructure*, *platform*, and *software*. The higher the degree of virtualization is, the more is usually managed by a cloud environment, and the less is controlled by a CSC. For instance, Google App Engine is a fully-managed platform service in the sense that the application container and the programming language run-time is pre-configured and cannot be manipulated by the CSC. This means that, aside from the application-related artifacts deployed on App Engine, the other artifacts related to the platform down to the infrastructure are immutable and controlled by Google. This is

[12]Orleans: http://research.microsoft.com/en-us/projects/orleans

certainly of interest to a CSC to select services that operate at the expected "virtualization layer". For instance, in the context of software modernization to the cloud, an on-premise environment is partly or even completely replaced by a cloud environment, where in practice the typical scenario requires "wiring" both environments [Andrikopoulos et al. 2013]. A concrete scenario may refer to a cloud application whose frontend is hosted on Amazon's platform service Beanstalk, utilizes the software service Google Maps, and connects to a user-controlled MySQL backend system that runs in a virtual machine hosted on an Amazon EC2 infrastructure service. In this work, we investigate the capabilities of CMLs to represent artifacts related to the main cloud service categories: infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS). We use the abbreviation "XaaS" to refer to all categories.

3.2. Language characteristics

3.2.1. 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 syntax*, *serialization syntax*, and a proper semantic domain. Considering the concrete syntax, it is concerned with the form [Moody 2009] 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 modeling language may have one or more *textual* or *graphical* syntaxes to represent models. In this work, we investigate solely which kind of notation is provided by a CML. To persist or interchange models, they are encoded according to the *serialization syntax* of the modeling language.

3.2.2. Semantics. The *semantics* gives meaning to the syntactic elements of a modeling language. 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 [Harel and Rumpe 2004].

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 [Harel and Rumpe 2004], and the finding of an agreement between stakeholders thereon. Therefore, most modeling languages do not have a rigorously defined semantics that goes beyond natural language specifications, even though it is an 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 transformers that *translate* models of a given language into models of a commonly understood or executable languages such as Java or C#. Another approach is to implement an interpreter that directly *operates* on the models. In the context of CMLs, a model-based provisioning engine is a concrete example of an interpreter. It is important to note that one could implement for a single CML more than one provisioning engine that may behave differently. This raises the question whether or not a provisioning engine should be considered as part of a CMLs semantics definition. However, a provisioning engine gives meaning to modeled cloud services in the sense that it relates them to concrete services of a cloud environment once the provisioning process has been enacted.

3.2.3. Typing. As pointed out by [Atkinson and Kühne 2007], two different kinds of classification mechanisms need to be considered in developing modeling languages. *Linguistic* classification refers to the commonly accepted approach that a user-defined model (token model [Kühne 2006]) is directly expressed by instantiating types contained in a metamodel, which defines the modeling language in use. Linguistic types determine which models are valid instances of a modeling language definition. However, to support engineers to create custom types or even hierarchies of them without modifying the modeling language definition directly, the notion of *ontological* classification has been introduced. Ontological types can be considered as extensions to a modeling language even though in contrast to linguistic types they are not grounded in the language definition. Instead, they are defined using (linguistic) types of the modeling language and often provided in terms of a custom type library to foster their reuse across different application scenarios. As a result, ontological types may capture vital features that are however relevant in a specific context only, and thus lifting them to a linguistic type appears unfavorable.[13] This is certainly of relevance for CMLs. Current cloud environments offer a considerable set of diverse services including processing power usually provided by virtual machines that can be provisioned on-demand. Different virtual machine types may be defined in terms of ontological types while keeping the language definition unchanged. Furthermore, they may capture not only common features but also vital peculiarities imposed by a cloud environment such as the availability zone in which a particular virtual machine instance must be provisioned and the operating system hosted by it. For instance, these features are required to provision a virtual machine in Amazon's cloud environment, whereas in the context of the Google App Engine the operating system of a virtual machine is pre-defined and the distribution of virtual machine replicas is automatically accomplished without granting custom configurations.

3.2.4. Realization. Two different approaches are common for *realizing* a modeling language in general [Fowler 2010] and a CML in particular. Either the language is developed on top of an existing, usually general-purpose language, or it is developed from scratch [Mernik et al. 2005]. Considering the former, they are *internal* in the sense that the selected host language provides the base elements for which extensions and constraints are developed. In contrast, *external* modeling languages have their custom concepts without explicit relationships to any existing language. Generally, there is no simple answer when to create an internal or external modeling language. However, design guidelines [Karsai et al. 2009] and patterns [Mernik et al. 2005] have been proposed to aid engineers in developing DSLs.

3.3. Modeling capabilities

As different stakeholders are usually involved in the development of a cloud application, their concerns need to be covered by the models created with a CML. Essentially, a concern is a stakeholder's interest that "pertains to the development of an application, its operation or any other matters that are critical or otherwise important" [van den Berg et al. 2005].[14] A critical task in the development of a cloud application is its decomposition into deployable components because they must eventually be distributed across a cloud environment or even multiple ones. Hence, the capability of a CML to represent the *structure* of a cloud application certainly matters.

3.3.1. Application structure. A component is a unit of computation in an application, whereas connectors [Medvidovic and Taylor 2000] represent interactions among components. Components and connectors are used to describe the high-level structure of an application in terms of a component configuration. *Composing* components and connectors into another more abstract component is beneficial in particular to hide complex structures of a cloud application.

To deploy a cloud application on the selected target environment, its application components need to be allocated to cloud *services*. More precisely, what needs to be allocated to the cloud services are the implementations of those components. The notion of a deployable *artifact* supports exactly the reference between logical components and connectors to their implementations. Artifacts are supposed to manifest any number of components and connectors. For instance, a cloud application implemented in Java is possibly packaged into several archives, *i.e.,* "JAR files". Those archives can be represented at the model level via artifacts. Allocating them to a cloud service, *e.g.,* a compute service including a Java platform, should have the effect that the "JAR files" are physically allocated to the provisioned service. Furthermore, as cloud services can interact with each other, modeling capabilities are required to connect them explicitly. We use the term *link* to refer to this capability. Artifacts deployed on possibly connected cloud services constitute what is usually called

[13]The linguistic type is often considered as first-class, whereas the ontological as second-class.
[14]A concern is usually supported by a modeling viewpoint, *e.g.,* component and deployment viewpoint.

a deployment configuration.[15] To ensure that connected cloud services can interact with each other, properties related to *networking* concerns often need to be explicitly specified. For instance, a cloud service at the infrastructure level needs to be assigned to a virtual network and possibly connected to a middlebox to ensure that it can be accessed by other cloud services.

It is important to note that we aim to investigate the modeling capabilities of CMLs to represent component, connector, and deployment concerns on a per-concept basis. As a result of this investigation, the different vocabulary introduced by current CMLs is classified according to the common terms of our framework, where the string-valued properties allow us to collect the concrete terms used by them. This effort is a first step towards achieving interoperability between existing CMLs and a core set of common cloud modeling concepts upon which semantic relations among different CMLs can be defined [Malavolta et al. 2010].

3.3.2. Cloud environment. From a cloud application deployment perspective, a modeled cloud service embodies a concrete service offered by the target cloud environment. For instance, several compute services located in different availability zones and a storage service may be required to provide a reliable and scalable cloud applications. The compute services may refer to Amazon EC2 offering and the storage service to its DynamoDB data store solution. As a result, services offered by a *cloud environment* need to be available at the model level. Several possibilities are conceivable to represent a configuration of cloud services required for the deployment of a cloud application. Clearly, they can be described in *textual* form. However, providing them in a structured form would certainly ease their interpretation by tools, *e.g.,* engines that initiate the provisioning of compute and storage services based on a deployment topology. In our framework, we distinguish between three structured-based approaches for capturing cloud services: *feature model*, *linguistic types*, and *ontological types*. Considering the first approach, existing compute and storage services may be captured as features of a certain cloud environment denoting the root concept of the model. In case of the last two approaches, a cloud service is captured in terms of a type as part of a CML. Depending on the typing mechanism a CML supports, a cloud service type is either directly built into the language definition (*i.e.,* linguistic type) or realized as a custom type supplementing the definition of a language without modifying it (*i.e.,* ontological type).

3.3.3. Elasticity. As the main incentive of using cloud services is the capability of cloud environments to scale them with a user's demand [Vaquero et al. 2011], a concern that matters is *elasticity*. Lower and upper bounds of cloud service instances can be specified by a *multiplicity* associated with the modeled service. To specify more sophisticated strategies when a cloud service must be provisioned or released, a rule-based approach [Kritikos et al. 2014] tend to be more powerful compared to specify service multiplicity. The elastic nature of cloud environments is also exploited to utilize them to capacity by optimizing the workload scheduling of the different co-located CSCs with consideration to their required quality of service.

3.3.4. Service level. A concern that matters is the specification of service levels, *e.g.,* referring to latency, availability, and security of a cloud service. Ideally, the quality of a cloud service is at least equivalent [Venters and Whitley 2012] to what can be expected if an on-premise environment is employed to host applications instead of a cloud environment. Currently, only a few of the reviewed CMLs support modeling concepts for capturing service levels at a rather high level. As a result, we distinguish in our framework whether a service level is captured using a *structured* approach or it is described in natural language, *i.e.,* a *non-structured* approach is employed.

3.4. Tool support

3.4.1. Modeling support. The means provided for the use of a CMLs notation and the validation of created models according to its syntax and semantics are subsumed under modeling support.

[15]The term "topology" is often used in this context as well. A deployment configuration or topology is a connected graph that describes deployment artifacts along with targets and relationships between them from a structural perspective.

Depending on how the notation is defined for a CML either *graphical* or *textual* representations of models are provided [Moody 2009]. In rare cases, both kinds of representations can be used. Obviously, this requires that a textual as well as a graphical notation are available for a CML. The notation of a CML may also be *imposed* by the meta-language used to realize it [Neubauer et al. 2015]. For instance, XML-based CMLs for which no further modeling support is available force engineers to express their models directly in XML. If a CML is realized as internal DSL, it should ensure the portability between the modeling tools that support the selected host language of the CML. For instance, UML-based CMLs should be applicable for any standard conformance UML modeling tool. This can be achieved if UML's extension mechanisms are appropriately employed for developing a CML [Selic 2007]. Furthermore, the multiple concerns supported by a CML should ideally be manageable by several dedicated views (*e.g.,* views for the component and deployment concern) while ensuring consistency across them for the same cloud application [Medvidovic and Taylor 2000].

3.4.2. Analysis support. To evaluate or predict certain (non-functional) properties of an application, *e.g.,* operational costs or performance, before it is hosted on a cloud environment is certainly a significant incentive to use a CML. Moreover, selecting an adequate set of services from possibly multiple cloud environments is labor-intensive not only because of inevitable trade-offs between, *e.g.,* operational costs and performance, but also the enormous design space that needs to be explored for an optimal deployment of a cloud application [Harman et al. 2013]. CML toolset developers have thus addressed analysis support for cloud applications and their underlying environments. In addition to *design-time* analysis, support for analyzing cloud applications at *run-time* is of particular interest because they may be migrated among cloud environments if a certain quality of service can no longer be guaranteed.

3.4.3. Refinement support. Explicit refinement support can ensure that modifications to models expressed by a CML are carried out in a stepwise systematic manner. Model refinement can be considered as a process of transforming a given high-level model into a more concrete model. For instance, deployment topologies of cloud applications are often modeled independently of the target cloud environment in a first step. The refinement of the deployment topology towards the target cloud environment is conducted in a second step [Ardagna et al. 2012]. This approach is particularly beneficial if a cloud application needs to be migrated between environments. The high-level models representing the cloud application are retained and *enriched* by environment-specific information to accomplish the refinement. For instance, environment-specific information can be captured in terms of custom (ontological) types [Atkinson et al. 2009] or profiles [Langer et al. 2012]. Refinement may also include the process of discovering appropriate concrete solutions that are already available, *e.g.,* an application service that is hosted on a cloud environment. To enable this kind of refinement, both the requirements of high-level models and the capabilities of existing more concrete models must be appropriately described, such that the former can be *resolved* according to the latter.

3.4.4. Generation support. Applying generative techniques is promising because executable artifacts can be produced for possible multiple target cloud environments from a single set of (architectural) models. Even though the environment for which artifacts were generated may change over time, the investment in creating models is retained [Greenfield and Short 2003] provided that generators are capable of producing those artifacts for the new environment. This includes not only the generation of implementations for the application itself but also deployment scripts and vice versa, *i.e.,* the generation of models from lower level code artifacts. Furthermore, a deployment plan expressed in terms of a workflow model may be generated from a deployment configuration to enact the application provisioning. In this work, we distinguish between three kinds of generative techniques [Czarnecki and Helsen 2006] possibly supported by a CML: *model-to-code*, *code-to-model*, and *model-to-model*.

3.4.5. Provisioning support. One key characteristic of cloud environments is the support for dynamic service provisioning. CSCs can provision and release cloud services on demand and pay only for what they have consumed. A provisioning engine aims to automate such processes and the (re-)deployment of application-related artifacts including the required middleware on top of those services. Considering the support for application and service provisioning in the light of Talwar's classification [Talwar et al. 2005], current CML's inherently apply a model-based approach as they represent a deployment configuration in terms of a model. In case it is directly interpreted by a provisioning engine, the approach can be characterized as *declarative* because the created model describes only what has to be provisioned, but without providing any details about how the provisioning shall be executed. In contrast to a declarative approach, an *imperative* approach explicitly prescribes how the provisioning must be executed. For instance, a deployment script[16] or a workflow model is often used to capture the respective provisioning actions.[17]

Considering the two approaches from the perspective of a CML user, the declarative approach is less invasive compared to the imperative approach because it requires describing solely the desired state of the provisioning in terms of a deployment configuration [El Maghraoui et al. 2006; FuWeili et al. 2017; Breitenbücher et al. 2014]. Since this loss of control is not always desirable, a *mix* of the two approaches is supported by some CMLs. For instance, if a CML supports in addition to a deployment configuration the specification of deployment scripts that shall be executed at a certain point during the provisioning of an application component, then both approaches can be combined to a certain degree. Such approaches typically employ life-cycle definitions that subdivide the provisioning of an application component into multiple phases. Those definitions provide a hook for custom scripts or other implementations that must be executed in a certain phase.

4. REVIEW PROCESS

To conduct the systematic review of CMLs, we followed the guidelines recommended by Kitchenham and Charters [Kitchenham and Charters 2007]. The review commenced in mid-2014 in the context of the ARTIST project, where some CMLs—with no claim for completeness—were demonstrated in the setting of a cloud migration scenario [Bergmayr et al. 2014b]. However, this demonstration revealed already an initial set of CMLs, which was useful in several phases of the review process. For instance, we used the initial set of CMLs of [Bergmayr et al. 2014b] to assess the quality of the search queries we formulated in an early phase of the review process, as those CMLs had to be covered by the obtained records. By this, we followed the guidelines by Kitchenham and Charters [Kitchenham and Charters 2007] concerning the development, evaluation, and tuning of the search queries by running a pilot study of the review process before executing the full study. Furthermore, we exploited them to develop a list of keywords required to conduct a keyword-based search as part of the study selection phase. The main phases we carried out in the course of the review process are described in the following Sections 4.1 to 4.4.

4.1. Research questions

The aim of this systematic literature review is to provide an overview of current CMLs, classify their main characteristics and core capabilities including the toolset they support, and identify the gaps and future research directions for CML development. The overall objective is defined by four research questions (**RQ**) as follows:

RQ1	What are the main purposes of current CMLs?
RQ2	What are their characteristics from a language engineering perspective?
RQ3	What core cloud modeling capabilities do they provide?
RQ4	What toolset is accompanied with today's CMLs?

[16]A deployment script is sometimes executed at the remote environment that is considered as the target of the application provisioning.
[17]A workflow model usually captures the data flow and the control flow among actions in an explicit way.

4.2. Data sources and search strategy

In this systematic review, the electronic databases recommended by [Petersen et al. 2008; Kitchenham et al. 2009; Kitchenham et al. 2010] have been used to search for primary studies.[18] We decided to search for publications in the period from 2006 to 2015 as Amazon Web Services (AWS) was officially launched in 2006 and the term "cloud computing" appeared around 2007 [Venters and Whitley 2012]. Since cloud computing is a highly diverse research topic, determining on the publication sources to search for relevant research works is a difficult task.

Hence, instead of taking this decision solely based on our expertise and knowledge in the area of cloud computing, model-driven engineering (MDE), and related research areas, we additionally formulated a search query with the main aim to figure out the publication sources where a CML may have been published. Based on the topic of this systematic review and the research questions proposed in Section 4.1, we defined the terms of the search queries according to the recommendations of [Kitchenham and Charters 2007]. We considered the terms "model", "modeling", "modelling", "language", "ontology", "profile", and "domain" as the main constituents of the search query. In addition, we limited the search to studies that are indexed by the keyword "cloud computing". After several tests, we selected the search query that returned the largest result set. Depending on the electronic database, the syntax of the search query obviously differs.

The exact search queries we executed against the selected electronic databases, including the number of records we received as a result, are summarized in Table I. Based on the obtained initial set of records, we determined the set of publication sources by a manual selection process in order to limit the records of publications considered in study selection process. We selected those publication sources that seemed to be relevant for this review. For instance, we discarded sources dedicated to topics such as "e-health applications and services" or "green computing". The selected publication sources are available online in the electronic appendix.

Electronic database	Search query	Records
ACM Digital Library http://portal.acm.org	"query": {(model, modeling, modelling, language, ontology, profile, domain) AND keywords.author.keyword:(+"cloud computing")} "filter": {"publicationYear":{"gte":2006 }},{owners.owner=GUIDE}	3,208
IEEE Xplore http://ieeexplore.ieee.org	((model OR modeling OR modelling OR language OR ontology OR profile) AND "Author Keywords":"cloud computing") ? and refined by Year: 2005–2016	4,020
ScienceDirect http://www.sciencedirect.com	(model OR modeling OR modelling OR language OR ontology OR profile) and KEYWORDS("cloud computing")[All Sources(Computer Science)]	899
Scopus http://www.scopus.com	ALL (model OR modeling OR modelling OR "language" OR ontology OR profile OR domain) AND KEY(cloud computing) AND PUBYEAR > 2005 AND SUBJAREA (comp)	13,284
SpringerLink http://www.springerlink.com	"cloud computing" AND (model OR modeling OR modelling OR language OR ontology OR profile) within "Computer Science" AND 2005–2016	12,341

Table I: Search queries executed against electronic databases

4.3. Study selection

To select the most relevant and important studies, inclusion and exclusion criteria were developed in a first step, see Section 4.3.1. They were applied in several stages of the study selection process as described in Sections 4.3.2 to 4.3.5.

[18]We only discarded Google Scholar (http://scholar.google.com) from the list of recommended electronic databases as it hardly allows publications to be downloaded in a batch process and a suitable format.

4.3.1. Inclusion and exclusion criteria. Studies relevant to this review must propose language concepts for modeling cloud applications. Those concepts must be defined in terms of a grammar[19] or a metamodel. For instance, even though the work of Sun *et al.* [Sun et al. 2012b] proposes a toolkit for managing cloud services, the language concepts used to represent them have not formally been defined but rather sketched using a single example. Similarly, CAMEL [Rossini 2015; Rossini et al. 2017] is currently developed and adopted by several projects (PaaSage, CloudSocket[20] [Woitsch and Utz 2015], CACTOS[21] [Östberg et al. 2014], and Melodic[22]) to support the modeling and execution of applications distributed over multiple cloud environments. However, until now, only an overview of how CAMEL is used and which existing DSLs it integrates and extends has been published. This criterion also excludes research works that introduce modeling methodologies independent of a CML (*e.g.,* MADCAT [Inzinger et al. 2014]).

Furthermore, proposed language concepts must enable engineers to model a cloud application independent of the concrete target cloud environment. Shielding models from possible changes of target cloud environments is one main requirement of a modeling language in general [Atkinson and Kühne 2003] and so also desirable for a CML. However, this does not mean that a CML should not provide capabilities for creating environment-specific models at all. Ideally, it should allow engineers to refine environment-independent models into models specific to the target cloud environment [Ardagna et al. 2012], which is commonly known as the transition from a platform-independent model (PIM) to a platform-specific model (PSM) in MDE. Languages that solely support PSMs, *i.e.,* they are directly bound to a cloud environment such as Amazon's CloudFormation[23] and OpenStack's HOT[24], are thus excluded from this review. However, such languages are potential transformation targets for CMLs to automate the provisioning of modeled application deployments. In this respect, approaches such as jclouds and Deltacloud (now retired) may also be considered as they provide an abstraction layer on top of the programming libraries provided for cloud environments. These approaches support a variety of cloud environments via dedicated connectors.

To limit the scope of this review, we consider studies that propose language concepts targeting mainly the user rather than the provider of a cloud environment. One of the reasons for this decision is that approaches addressing the CSP perspective tend to connect proposed language concepts with internal resources of a cloud environment. This is hardly possible for a CML targeting CSCs as providers of a cloud environment usually offer cloud services to their users without giving many details of the internal resources underlying those services, if at all. For instance, SCORCH [Dougherty et al. 2011] assumes a scenario in which auto-scaling is realized by provisioning and releasing pre-instantiated virtual machines from a queue, where its optimizer aims at determining the length of this queue and the configuration of the pre-instantiated virtual machines in the queue. Moreover, SCORCH is based on several computational models that specify, for example, the energy consumption of resources and the costs for consuming such resources, which is of interest for cloud providers. This is because they mainly benefit from utilizing their resources to capacity by co-locating different CSCs on the same infrastructure resources. Providing explicit specifications of such infrastructure resources is supported by Cloud# [Liu and Zic 2011] for improving the understanding of how resources in a cloud environment are virtualized, scheduled, and isolated from each other.

The inclusion and exclusion criteria that were applied in this systematic review are described as follows.

Inclusion criteria

[19]Please note that a grammar is not only defining the concepts by the abstract syntax but also their concrete appearance on the basis of the concrete syntax.
[20]https://www.cloudsocket.eu
[21]http://www.cactosfp7.eu
[22]http://melodic.cloud
[23]CloudFormation: https://aws.amazon.com/de/cloudformation
[24]Heat Orchestration Template (HOT): http://docs.openstack.org/developer/heat

(1) Studies that report on language concepts for cloud application modeling
(2) Proposed language concepts that are suitable to create models independent of target cloud environments
(3) Studies that address the CSC perspective

Exclusion criteria

(1) Studies or approaches that are bound to a cloud environment
(2) Studies that address the CSP perspective
(3) Studies that are not written in English

The selection of the primary studies was carried based on the initial set of records we obtained from executing the search queries against the data sources given in Table I. To extract the studies relevant for this review from the total number of studies, we passed through several pruning stages, where in each stage the number of studies was significantly reduced compared to the result of the previous stage. Overall, four researchers conducted the four pruning stages as illustrated in Figure 2.

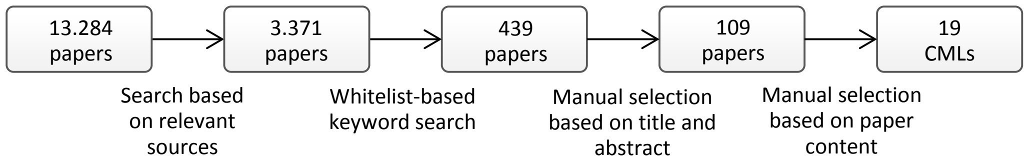
13.284 papers
3.371 papers
439 papers
109 papers
19 CMLs
Search based on relevant sources
Whitelist-based keyword search
Manual selection based on title and abstract
Manual selection based on paper content

Fig. 2: Study selection process

4.3.2. Pruning stage 1: Search limited to relevant publication sources. Based on the initial set of records obtained from the electronic databases, we extracted those studies that were published in the selected publication sources. Additionally, the removal of duplicates as a result of using different electronic databases was carried out at this stage. Duplicates were identified by considering the title, authors, and publication year of a study. Overall, in this stage, we selected around 25 percent of the total number of records for the next pruning stage.

4.3.3. Pruning stage 2: Whitelist-based keyword search. To accomplish this stage, we defined in a first step a list of keywords from which at least one must be present in a certain study to consider it for the next pruning stage. We elaborated the whitelist on the basis of CMLs we were already aware of [Bergmayr et al. 2014b] and our own experience in the area of MDE and cloud computing. The keywords we defined for the whitelist are summarized in Table II. After conducting the second pruning stage, we ruled out around 85 percent of studies. Thus, the number of studies were considerably pruned. The reduction to a manageable number of studies at this stage was necessary as the two remaining stages are hardly achievable automatically and so were conducted manually.

4.3.4. Pruning stage 3: Manual selection based on title and abstract. As it is too often somewhat difficult to determine the relevance of a study to a systematic review from solely considering its title, we decided to evaluate in this stage both the title and abstract of each study against the inclusion and exclusion criteria. Generally, we acted in a conservative manner in this stage, as in some cases even more information in addition to the title and abstract of a study is required to determine whether a study is relevant for this review. Finally, in this pruning stage, we ruled out around three-quarter of the studies, thus leaving 108 studies for the fourth pruning stage.

Keyword		
[application|service] developer	[application|service] definition	deployment topology
[application|service] mode(l)ler	[application|service] architecture	infrastructure(-)as(-)a(-)service
[application|service] engineer	[application|service] topology	platform(-)as(-)a(-)service
[application|service] requirement	resource provisioning	software(-)as(-)a(-)service
[application|service] component	resource description	language engineering
[application|service] deployment	cloud service [engineering]	problem(-)oriented(-)language
[application|service] mode(l)l(ing)	cloud service	domain(-)specfic(-)language
[application|service] life(-)cycle	cloud application	domain modelling
[application|service] provisioning	cloud mode(l)l(ing) [language]	meta(-)mode(l)ling
[application|service] distribution	deployment mode(l)l(ing) [language]	meta(-)model

Table II: Keywords of the elaborated whitelist.

4.3.5. Pruning stage 4: Manual selection based on study content. In the final pruning stage, we carefully read the remaining studies under the consideration of the goal of this systematic review and the defined inclusion and exclusion criteria. After all, we selected 19 relevant studies that are presented in Table III. The table provides in addition to a representative name of each selected approach the main publications from which the data has been extracted relevant for this review. Finally, we also performed a quality check of the final selected set of papers. In particular, we used backward snowballing [Wohlin 2014] with the set of relevant studies by using their reference lists to identify new papers to include in the study. However, based on this method, we could not identify more relevant papers. Thus, we conclude that our search process already worked well to identify the relevant literature.

CML	References
Blueprint	[Nguyen et al. 2011; Nguyen et al. 2012]
Caglar *et al.*	[Caglar et al. 2013]
CAML	[Bergmayr et al. 2014a; Bergmayr et al. 2014; Bergmayr et al. 2016]
clADL	[Pérez and Rumpe 2013; Hermerschmidt et al. 2014]
CloudDSL	[Silva et al. 2014]
CloudMIG	[Frey and Hasselbring 2010; Frey and Hasselbring 2011; Frey et al. 2013b; Frey et al. 2013a]
CloudML-SINTEF*	[Ferry et al. 2013; Ferry et al. 2014]
CloudML-UFPE*	[Gonçalves et al. 2011]
CloudNaaS	[Benson et al. 2011]
GENTL	[Andrikopoulos et al. 2014b; Andrikopoulos et al. 2014a]
Holmes	[Holmes 2014a; Holmes 2014b; Holmes 2015b; Holmes 2015a]
MOCCA	[Leymann et al. 2011]
MULTICLAPP	[Guillén et al. 2013b; Guillén et al. 2013a]
Nhan *et al.*	[Nhan et al. 2012]
PDS	[Lu et al. 2013]
RESERVOIR-ML	[Chapman et al. 2010; Chapman et al. 2012]
StratusML	[Hamdaqa et al. 2011; Hamdaqa and Tahvildari 2014; Hamdaqa and Tahvildari 2015; Hamdaqa and Tahvildari 2016]
TOSCA	[OASIS 2013; OASIS 2013; Binz et al. 2014]
VAMP	[Etchevers et al. 2011a; Etchevers et al. 2011b]

* As both languages use the same acronym, we added suffixes to better highlight their difference.

Table III: Selected approaches

4.4. Data extraction

In a first step, we converted our proposed classification and comparison framework for CMLs into several spreadsheets. They were used to collect the relevant data. The single columns of the created spreadsheets were derived from the properties of the classes constituting the framework. Following this approach, the properties determine the concrete data items that we had to extract from the selected studies. For instance, the scope of a certain CML is described by a single row, which captures its pragmatics and the kind of target cloud environments that are supported. The scope of a CML has been derived from the natural language descriptions of the respective studies. Similar for the toolset provided by a CML, we extracted the relevant data from the available studies. Concerning the characteristics of a CML and its modeling capabilities, we extracted them from the language definition.

Considering the process of collecting and interpreting the data itself, a series of consensus meetings were held with the goal to carefully analyze each reviewed CML according to the properties of the prepared tables. As a result of this process, we established the basis necessary to answer the research questions as defined in Section 4.1.

5. RESULTS

Until now several CMLs have been proposed. In the following, we classify and compare them along the dimensions of the presented review framework (see Section 3) with the main aim to answer the defined research questions (see Section 4).

CML	Pragmatics	Target
Blueprint	Cloud service composition and description of deployment configurations	XaaS
Caglar *et al.*	Cloud service simulation and description of deployment plan configurations	IaaS
CAML	Cloud application architecture description and refinement of deployment configurations towards target cloud environment	XaaS
clADL	Architecture description of interactive cloud services and generation of implementations for the cyber-physical systems domain	XaaS
CloudDSL	Description of deployment configurations	XaaS
CloudMIG	Application migration to the cloud with emphasis on optimal deployment configurations and their conformance with target cloud environments	PaaS IaaS
CloudML-SINTEF	Automated provisioning of multi-cloud applications and re-configuration of provisioned cloud services at run-time	XaaS
CloudML-UFPE	Description of cloud services	IaaS
CloudNaaS	Description of deployment configurations with emphasis on network aspects	IaaS
GENTL	Description of deployment configurations with emphasis on cost-efficient application provisioning	XaaS
Holmes	Description of deployment configurations and their automated provisioning	XaaS
MOCCA	Optimal (re)arrangement of (existing) deployment configurations for application provisioning to multiple target cloud environments	XaaS
MULTI-CLAPP	Application code generation for target cloud environments from component configurations	XaaS
Nhan *et al.*	Feature model based software stack (re-)configuration and their automated provisioning	IaaS
PDS	Deployment plan generation from described deployment configurations	IaaS
RESERVOIR-ML	Description of deployment configurations with emphasis on application-triggered elasticity rules for infrastructure-related cloud services	IaaS
StratusML	Generation of executable deployment descriptor and run-time adaptation rule from described deployment configurations	XaaS
TOSCA	Description of portable composite cloud applications for their automated provisioning and life-cycle management	XaaS
VAMP	Automated provisioning of distributed cloud applications with emphasis on support for establishing communication between components	IaaS

Deployment configuration: connected graph of deployment artifacts, targets, and their relationships (see Section 3.3)
Deployment plan: imperative description of the provisioning process
Component configuration: connected graph of components and connectors (see Section 3.3)

Table IV: Language scope

5.1. RQ1: What are the main purposes of current CMLs?

5.1.1. Pragmatics. The majority of CMLs deals with the description of cloud deployment configurations. CAML considers them as part of the overall cloud application architecture. To refine cloud deployment configurations towards the target environment, CAML provides dedicated UML profiles. CloudMIG aims at migrating on-premise deployment configurations into optimal cloud deployment configurations and assuring that those configurations conform to the target cloud environment. GENTL and MOCCA also address optimization of cloud deployment configurations. While GENTL places emphasis on cost-efficient application provisioning, MOCCA primarily deals with the distribution of cloud application components to multiple target cloud environments. CloudML-SINTEF exploits cloud deployment configurations not only at design-time but also at run-time for model-based reconfigurations of provisioned cloud services. CloudNaaS places emphasis on capturing networking aspects (*e.g.,* addressing and segmentation of compute services at the infrastructure) by cloud deployment configurations, whereas PDS exploits them for generating deployment plans. Deployment plans[25] are suggested by TOSCA to describe the process used to create and terminate cloud services and to manage them throughout their whole lifetime. In the approach of Caglar *et al.*, deployment plan configurations are created based on previous simulation results obtained from CloudSim [Calheiros et al. 2011].

CMLs that are capable of describing cloud deployment configurations also support the representation of cloud services. In contrast to these CMLs, CloudML-UFPE places emphasis solely on describing cloud services without providing dedicated concepts to model a cloud deployment configuration. Still, cloud services described by CloudML-UFPE can be considered as a potential source for describing cloud deployment configurations. Blueprint addresses the composition of cloud services.

Several CMLs aim at automating the provisioning of cloud services and possible application components deployed on top of them. To exploit TOSCA-based provisioning support, CAML provides a mapping to TOSCA. CloudML-SINTEF comes with its provisioning engine. Such an engine is also available for TOSCA. Other approaches (Holmes, Nhan *et al.* and PDS) rely on configuration management systems, such as Cloud-Init or Chef, whereas StratusML generates deployment descriptors for platform-related cloud services (*e.g.,* Azure App Service). In contrast to those approaches, VAMP exploits OVF to describe VM configurations including application components. It provides a dedicated protocol for exchanging configuration parameters (*e.g.,* remote addresses and ports) between remote VMs to establish the communication among application components. Generally, generative techniques play an important role in automated provisioning to the cloud because a variety of artifacts (*e.g.,* deployment plans or scripts, run-time models, and VM images) are automatically produced. Aside from generating deployment or provisioning-related artifacts, the goal of MULTICLAPP is to generate application code for the Java environment. Generation of cloud application code is also supported by clADL. It proposes an architecture style for modeling interactive cloud services in the context of cyber-physical systems (*e.g.,* services that process sensor data from industrial production machines). Cloud environments are considered as the deployment target for those services.

A few CMLs emphasize the representation of elasticity rules (RESERVOIR-ML, StratusML) capable of triggering the provisioning of cloud service at application run-time.

5.1.2. Target. Almost half of the CMLs are capable of representing cloud services at any of the three considered cloud service categories. CMLs that target IaaS mainly deal with the description of compute services or the configuration of VMs. CloudMIG provides cloud service descriptions for both categories IaaS and PaaS, whereas StratusML considers PaaS up to SaaS.

5.1.3. Summary of CML scope. Current CMLs pursue different goals and show various levels of maturity. Still, they also show similarities with respect to their pragmatics. For instance, the

[25] In the TOSCA specification the term "management plan" is used.

majority of CMLs deal with the description of deployment configuration and some of them even support automated application provisioning. On the other hand, the observed diversity of the current CMLs is beneficial in the sense that a broad spectrum of application scenarios is supported. At the same time, the exchange of models between approaches and provided tools, respectively, is hardly supported. As a result, a well-connected mix of existing CMLs is currently not available. The finding of common ground between the current approaches is thus highly desirable. GENTL did already a first step in this direction. Mappings from Blueprint and TOSCA to GENTL are presented in the work of Andrikopoulos *et al.* [Andrikopoulos et al. 2014a]. In this respect, the semantics of the CMLs play a major role [Kappel et al. 2006] since useful mappings, which are the basis for language interoperability (*cf. e.g.,* [Malavolta et al. 2008]), can otherwise hardly be identified. A common metamodel [Atzeni et al. 2005] may serve as a useful means in such an endeavor.

Another interesting aspect is that most CMLs are used solely at design-time, whereby the representation of the cloud application at run-time is outside the scope of most CMLs. For instance, run-time information may provide the current status and workload of a certain provisioned compute service. CloudML-SINTEF is capable of annotating the design-time model with run-time information, which allows run-time adaptations to be performed not only by human operators but also reasoning engines in order to manipulate models at run-time automatically. This capability is facilitating "models@run-time" [Blair et al. 2009], which is an architectural pattern for dynamically adaptive systems that leverages upon models at both design-time and run-time.

5.2. RQ2: What are the characteristics of CMLs from a language engineering perspective?

Main language characteristics of current CMLs are summarized in Table V. While they appear to be primarily relevant for language engineers, some of them may also be of interest for the users of a CML. For instance, the notation of a CML clearly affects its users.

5.2.1. Syntax. Considering how the abstract syntax of CMLs is represented, two meta-languages seem to be dominant in the field of cloud computing: MOF and XML Schema. The majority of CMLs provide a MOF-based metamodel.[26] Two of them uses UML as a host language (CAML and MULTICLAPP). Around a quarter define their modeling elements in terms of XML Schemas (BLUEPRINT, CloudML-UFPE, PDS, TOSCA, VAMP). The remaining languages follow either a grammar-based approach (clADL, CloudNaaS, Holmes) or rely on Microsoft's DSL tools[27] (StratusML).

Regarding the concrete syntax of CMLs, the majority provides either a graphical notation or a textual one. A few of them provide both. In case of CloudML-UFPE and VAMP models need to be expressed directly in XML. Even though the TOSCA standard does not define a graphical notation, with Vino4TOSCA [Breitenbücher et al. 2012] service templates can be visually represented. The representational capabilities of some of the reviewed CMLs are demonstrated in [Bergmayr et al. 2014b].

While the serialization format of a language is usually imposed by the meta-language used to define it (*e.g.,* XMI is the standard interchange format for MOF-based metamodels), some CMLs support alternative formats mainly for compatibility between tools. For instance, the provisioning engine of CloudML-SINTEF can also consume models serialized in the JSON format.

5.2.2. Semantics. Turning the focus from the syntactical aspects of the CMLs to their semantics, the majority of approaches comes with a toolset (see Table VIII) that directly interprets or executes the models of a CML, *e.g.,* a provisioning engine. In this case, the semantics of the CMLs is defined based on an operational approach. A translational approach is applied for defining the semantics of CAML, clADL and MULTICLAPP. Both CAML and MULTICLAPP provide a mapping to Java for generating application code from models. In addition, CAML also provides a mapping to

[26]Since Ecore is a reference implementation of the essential part of MOF, CMLs which provide an Ecore-based metamodel follow a MOF-based approach.
[27]https://msdn.microsoft.com/en-us/library/aa905334.aspx

CML	Syntax			Semantics	Realization	Typing
	Abstract	Concrete	Serialization			
Blueprint	XML schema	graphical	XML	operational	external	linguistic
Caglar *et al.*	MOF	graphical	custom XML	operational	external	linguistic
CAML	UML	graphical	XMI	translational	internal UML	linguistic ontological
clADL	Grammar	textual	custom	translational	internal MontiArc	linguistic
CloudDSL	MOF	graphical	XMI	English prose only	external	linguistic
CloudMIG	MOF	graphical	XMI	operational	external	linguistic ontological
CloudML-SINTEF	MOF	textual graphical	XMI JSON	operational	external	linguistic ontological
CloudML-UFPE	XML schema	textual in XML	XML	operational	external	linguistic
CloudNaaS	Grammar	textual	custom	operational	external	linguistic
GENTL	MOF	graphical	XML	operational	external	linguistic ontological
Holmes	Xtext* grammar	textual	custom XMI	operational	external	linguistic ontological
MOCCA	MOF	graphical	XMI	operational	external	linguistic
MULTI-CLAPP	UML	graphical	XMI	translational	internal UML	linguistic ontological
Nhan *et al.*	MOF Feature model	textual graphical	XMI JSON SXFM	operational	external	linguistic
PDS	XML schema	textual	XML Ruby	operational	external	lingusitic
RESERVOIR-ML	MOF	textual graphical	XMI XML	operational	external	linguistic
StratusML	Microsoft DSL toolkit	graphical	XML	operational	external	linguistic
TOSCA	XML schema	textual graphical	XML YAML	operational	external	linguistic ontological
VAMP	XML schema	textual in XML	XML	operational	internal OVF	linguistic

* Xtext is a language engineering framework: https://eclipse.org/Xtext compatible with MOF

Table V: Language characteristics

TOSCA for reasons of exploiting existing TOSCA containers, *e.g.,* OpenTOSCA [Binz et al. 2013], which enable automatic provisioning of cloud applications. clADL is grounded in FOCUS [Broy and Stølen 2001], which enables the formal specification of distributed systems in terms of components communicating via channels. The communication between the components is formally represented by the concept of streams [Ringert and Rumpe 2011]. MULTICLAPP provides a mapping to Java for generating application code from models. Finally, in case of CloudDSL neither an operational nor a translational semantics is defined. However, it seems that a mapping from CloudDSL to TOSCA is currently being developed.

5.2.3. Realization. Four out of the 19 CMLs are realized as internal domain-specific languages. Both CAML and MULTICLAPP are embedded in UML. While MULTICLAPP solely relies on UML's profile mechanism to annotate components that are expected to be deployed onto a cloud environment, CAML exploits both the library concept of UML as well as profiles. The modeling concepts of CAML's cloud library are generic in the sense that they are independent of a cloud environment, whereas profiles are used to capture services offered by cloud environments. This is in contrast to MULTICLAPP, where generic cloud modeling concepts are captured by means of a profile. Additionally, MULTICLAPP comes with a feature model for capturing concrete cloud services. clADL is realized on top of MontiArc [Haber et al. 2012], which is a textual domain-specific language for modeling distributed interactive systems, whereas VAMP is integrated into OVF.

5.2.4. Typing. More than a third of the reviewed CMLs support in addition to linguistic typing also ontological typing. From a language engineering perspective, all CMLs primarily rely on a spanning hierarchy [Atkinson and Kühne 2005] where ontological types are considered orthogonal to linguistic types. Thus, ontological types are referenced by instances of linguistic types. The latter would require a stacking hierarchy [Atkinson and Kühne 2005]. In UML, an additional typing dimension can be accomplished by combining libraries and profiles. CAML applies this approach.

5.2.5. Summary of CML characteristics. One aspect that requires consideration refers to the interoperability between CMLs. The heterogeneities imposed by the different meta-languages used to implement them certainly impede such an endeavor. Realizing a mapping between two CMLs defined with different meta-languages would require the implementation of a technical bridge in addition to the definition of language correspondences. Hence, the use of different meta-languages for realizing CMLs poses a challenge for exchanging models between them. With CMLs that solely provide an XML Schema but not a human-usable notation, the users are bound to the verbose angle-bracket syntax, which is complex in terms of human-comprehension and therefore impedes maintainability [Badros 2000]. For instance, XMLText [Neubauer et al. 2015] provides a semi-automatic approach for generating Xtext-based grammars with a human-readable textual concrete syntax while ensuring backward compatibility to the original XML-based representation.

Proposals for new CMLs or extension to them come ideally together with a machine-interpretable and human-usable language definition. The latter is preferably available in a commonly accepted format. Once they are defined, sharing them via an open repository such as AtlanMod's Metamodel Zoo[28] or ReMoDD[29] allows them to be easily accessed. It may also further stimulate their reuse in the development of new languages or additional features for them.

Another interesting aspect is that currently little attention is paid to general-purpose software modeling languages, such as UML, even though they provide modeling concepts to represent software, platform, and infrastructure artifacts from different viewpoints (only two approaches provide cloud-specific extensions to UML). With the relatively recent emergence of TOSCA and its standardization by OASIS, it appears obvious that aligning cloud modeling approaches with existing software modeling approaches to provide continuous modeling support is highly required [Jamshidi et al. 2013].

5.3. RQ3: What core cloud modeling capabilities do CMLs provide?

Probably most important to the users of CMLs are their modeling capabilities. A black-box view on the modeling concerns addressed by current CMLs is summarized in Table VII. The results of a closer investigation of the component and deployment viewpoint are given in Table VI.

[28]Metamodel Zoo: http://www.emn.fr/z-info/atlanmod/index.php/Zoos
[29]ReMoDD: http://www.cs.colostate.edu/remodd/v1

CML	Application structure	Cloud environment services	Elasticity	Service level
Blueprint	Component Deployment	textually described	Multiplicity	structured by policies
Caglar *et al.*	Deployment	linguistic types	✗	✗
CAML	Class Component Deployment	ontological types	Multiplicity	✗
clADL	Component Deployment	ontological types	✗	✗
CloudDSL	Deployment	textually described	✗	✗
CloudMIG	Class Deployment	ontological types	Rule-based	✗
CloudML-SINTEF	Component Deployment	ontological types	Multiplicity	✗
CloudML-UFPE	Deployment	linguistic types	✗	✗
CloudNaaS	Deployment	linguistic types	✗	✗
GENTL	Component Deployment	ontological types	✗	structured by annotations
Holmes	Deployment	ontological types	Multiplicity	✗
MOCCA	Component Deployment	ontological types	✗	✗
MULTI-CLAPP	Component Deployment	feature model	✗	structured by properties
Nhan *et al.*	Deployment	linguistic types	✗	✗
PDS	Deployment	linguistic types	✗	✗
RESERVOIR-ML	Component Deployment	linguistic types	Rule-based	✗
StratusML	Task Deployment	linguistic types	Rule-based	✗
TOSCA	Component Deployment	ontological types	Multiplicity	structured by policies
VAMP	Component Deployment	linguistic types	✗	✗

[*] Blueprint does not directly support to represent service levels but instead exploits existing languages such as WS-Policy or SLAng.

Table VI: Cloud modeling concerns

5.3.1. Application structure. As expected, all the reviewed CMLs provide capabilities to model the structure of a cloud application from a deployment viewpoint. Around half of the CMLs support in addition to the deployment viewpoint also the component viewpoint. Reasons for including the latter viewpoint are manifold. For instance, representing the interaction between a cloud application and a cloud service requires the consideration of application components. Connectors are used to represent the interconnection among them and with cloud services. Having components explicitly represented is also of particular relevance for application modernization to the cloud if components are replaced by cloud services already offered by the selected target environment. Furthermore, components may need to be re-allocated to deployable artifacts if the cloud application is migrated only partially.

In addition to the component and deployment viewpoints of a cloud application, CloudMIG also addresses the class viewpoint. In case of CAML, the class viewpoint is exploited to generate application code from structural models annotated with platform-specific UML profiles [Bergmayr et al. 2016]. The profiles are automatically generated from programming libraries, *e.g.,* DynamoDB and Objectify, provided or recommended by cloud environments. As CloudMIG aims at supporting application migration to the cloud, it captures technical constraints of cloud environments (*e.g.,* directly instantiating Java's `Thread` class is not permitted on the Google App Engine) against which conformance checks are carried out automatically. Dedicated validators operate at the class level of KDM models [OMG 2011] to investigate whether the application satisfies all the captured environment constraints. KDM models are directly associated with cloud nodes, which explains the absence of components and connectors in CloudMIG.

Considering the component and deployment viewpoints in more detail (see Table VI), the high diversity of the vocabulary used by the reviewed CMLs is revealed. At the same time, it shows that even though various syntactical terms have been proposed, they can be classified at least according to a high-level categorization. Almost all CMLs that deal with both viewpoints component and deployment support artifacts as a means of allocating concrete implementations to cloud services. In case of PDS, the artifact is specific to Java-based applications. RESERVOIR-ML and StratusML do not distinguish between logical components and their implementations in terms of deployable artifacts. Components are directly allocated to cloud services where the interconnection between them is solely modeled from the deployment viewpoint by connecting cloud services to each other. As a result, connectors between components can hardly be modeled in particular if a variety of components are allocated to cloud services.

Regarding cloud services, it is important to note that, as a result of the varying pragmatics of the CMLs, they refer to different categories (see the superscript annotation determining the category). For instance, CMLs targeting IaaS mainly focus on the representation of compute services and their configuration in terms of virtual machine characteristics such as CPU, memory, and local disc space. As a result, they can hardly deal with PaaS or SaaS-related cloud services. Moreover, deployment models represented by CMLs that also support PaaS or even SaaS-related cloud services may only be partially translated to CMLs with a focus on IaaS. Clearly, the border between IaaS and PaaS is becoming blurred with the emergence of new "cross-layered" cloud services (*e.g.,* Azure App Service on different virtual machines). Still, the cloud service category appears to be relevant to consider. For instance, defining useful correspondences between the language concepts with the goal to achieve better interoperability between CMLs requires considering the supported target cloud environment.

Finally, TOSCA distinguishes between deployment artifact and implementation artifact. The former refers to the implementations of application components, whereas the latter is used to capture the implementation of a management operation. A management operation is attached to a deployment model and executed in the context of the application provisioning, *e.g.,* "start" of a provisioned cloud service or "install" of an application-related artifact.

5.3.2. Cloud environment. Considering how CMLs represent cloud environment services, the majority captures them in terms of linguistic or ontological types that are instantiated to model the cloud application structure from a deployment or component viewpoint or even both. Generally, CMLs that advocate ontological types over linguistic types for capturing cloud services provide a richer collection of pre-defined custom service types mainly because their definition is less intrusive compared to defining linguistic types. Capturing cloud services in terms of a feature model can be considered as an orthogonal approach compared to providing service types. From a language engineering perspective, a feature model is not expressed using a CML but rather one of the existing notations (*cf. e.g.,* [Czarnecki and Eisenecker 2000]). Both MULTICLAPP and the approach of Nhan *et al.* exploit feature models to let engineers configure the target cloud environment by select-

CML	Component & connector		Deployment			
	Component	Connector	Service	Link	Networking	Artifact
Blueprint	Blueprint	Resource-Requirement	Blueprint^X	Require	✗	Implementation-Artifact
Caglar *et al.*	✗	✗	VM^I	✗	✗	Cloudlet
CAML	Component^C	Prov./Req. Interfaces	CloudService^X	Communication-Channel	Addressing	Artifact
clADL	Component^C	Prov./Req. Interfaces	Runtime^X	Endpoint	✗	Artifact
CloudDSL	✗	✗	Service^X	Endpoint	✗	Resource
CloudMIG	✗	✗	CloudNode^I	Cloud Link	Addressing	KDM model
CloudML-SINTEF	Component	Relationship	External-Component^X	Hosting	Addressing	Resource
CloudML-UFPE	✗	✗	Node^I	Link	Addressing	✗
CloudNaaS	✗	✗	Group^I	Virtual Net	Addressing Segmentation NetworkService	✗
GENTL	Component	Connector	Component^X	Connector	✗	✗
Holmes	✗	✗	Hosting Unit^X	Ports	Security-Groups	Service
MOCCA	Component	Component-Relation	Virtual-System	Network Port Profiles	Addressing	Artifact
MULTI-CLAPP	Cloud Artefact Element^C	Prov./Req. Interfaces	Cloud Artefact Interface^X	✗	✗	Cloud Artefact
Nhan *et al.*	✗	✗	VMNode^I	Connection	Addressing	Software-Component
PDS	Service	✗	Node^I	Database connection	Segmentation	War file
RESERVOIR-ML	Component	✗	Virtual-Machine-Descriptor^I	Network Port Profiles	Addressing Segmentation	✗
StratusML	Service Task Activity	Connector Call	Task^{S,P}	Connector Endpoint	Addressing	Service Adaptation
TOSCA	Node-Template	Relationship-Template	Node-Template^X	Relationship-Template	Addressing	Deployment-Artifact Implementation-Artifact
VAMP	Component	Binding	Dynamic-Virtual-System^I	Network Port Profiles	Addressing	✗

^{C} Component is composable
^{I} IaaS
^{P} PaaS
^{S} SaaS
^{X} XaaS (Services of all three categories can be modeled)

Table VII: Component and deployment viewpoint

ing the required cloud services. They are considered as a feature of a cloud environment denoting the root concept of the feature model. As feature models are often used to represent the commonalities and variabilities of a certain domain explicitly, they appear to be a useful source to define cloud service types assuming that the analyzed domain refers to cloud computing. The variabilities indicate the information required to instantiate a cloud service type in a CML [Mernik et al. 2005].

5.3.3. Elasticity. Turning the focus on how the elastic nature of cloud environments can be treated at the model level, less than half of the CMLs allow engineers to define upper and lower-bounds for service instances or elasticity rules based on which new service instances are provisioned or released. Considering the CMLs supporting a rule-based approach, they all rely on the notion of event condition action (ECA) rules that are triggered by monitors observing the cloud environment. Events may refer to increasing virtual machine workload when considering the infrastructure level or growing response time of application servers operating at the platform level. CloudMIG seems to focus solely on infrastructure-level events whereas both RESERVOIR-ML and StratusML do also consider events created at the platform-level.

5.3.4. Service level. Regarding the representation of service levels, only few CMLs have reported on how they can be captured. Blueprint exploits existing languages such as WS-Policy[30] or SLAng[31] to express QoS constraints, *e.g.,* “response time < 3 sec” or “Data storage is only within the Netherlands”. GENTL is capable of representing those QoS constraints in terms of typed annotations. MULTICLAPP provides some stereotypes capable of capturing QoS constraints using key-value pairs along with an operator that can be selected from a predefined enumeration. It covers standard relational operators such as “equal to”, “greater than”, or “less than”. Finally, TOSCA supports the definition of policies for expressing non-functional behavior or a kind of QoS that a node type can declare to expose. A declared policy type can be instantiated via a policy template used within a node template and processed by a TOSCA container. For instance, policies are processed during service provisioning to guarantee that the provisioned services of a cloud environment satisfy the requirements captured by policies [Waizenegger et al. 2013].

5.3.5. Summary of CML modeling capabilities. The reviewed CMLs provide a considerable set of modeling concepts to support a variety of viewpoints relevant in the context of cloud application and service modeling. Still, modeling concepts for capturing non-functional aspects are underrepresented. The spectrum of non-functional requirements is certainly broad, however directly attaching information, such as service levels and pricing [Cardoso et al. 2010], to the deployment artifacts and targets may be a further improvement [Glinz 2007]. As a result, technical and non-technical aspects are brought together in a possibly single view, which can support the selection of an appropriate CSP and the optimization of the application provisioning. Regarding the latter aspect, CAML covers pricing information of cloud environments for the selection of cloud services, whereas CloudMIG and GENTL provide this information for the optimization of deployment configuration.

Considering the modeling concepts of Table VII in the light of interoperability, it gives a first impression of how they relate to each other. Still, it is a first step in this direction as achieving interoperability between the CMLs also requires the consideration of concept specializations and possible available custom types. For instance, the TOSCA primer provides a set of pre-defined custom types that may not only be of value for TOSCA users, but cloud application engineers in general. As a result, conceptual mappings between the CMLs as a basis for accomplishing interoperability among them need to consider both intensional and extensional levels [Kühne 2006].[32] This requires a good understanding of the modeling levels addressed by a CML, which may include not only the design-time models but also run-time models [Rossini et al. 2015].

[30]http://www.w3.org/TR/ws-policy
[31]http://uclslang.sourceforge.net
[32]Custom types are at the intensional level. They are instantiated at the extensional level by assigning concrete values to their features.

CML	Modeling	Analysis	Refinement	Generation	Provisioning
Blueprint	Graphical model editor	✗	resolution	✗	✗
Caglar *et al.*	Grapical model editor	✗	✗	m2t: Deployment script Simulation code	imperative
CAML	Arbitrary UML model editor	✗	enrichment by UML profiles	m2t: UML-Java t2m: Java-UML m2m: CAML-TOSCA	declarative[T]
clADL	Textual model editor	✗	✗	m2t: Monticore facility	✗
CloudDSL	Graphical model editor	✗	✗	✗	✗
CloudMIG	Graphical model editor	design-time: Conformance checking and Deployment optimization	✗	t2m: Java-KDM	✗
CloudML-SINTEF	Textual & graphical model editor	✗	enrichment by ontological types	m2m: Runtime model m2t: Adaption script	declarative
CloudML-UFPE	Directly repre-sented in XML	✗	✗	✗	✗
CloudNaaS	✗	✗	✗	m2t: network rules	declarative
GENTL	Graphical model editor	design-time: Deployment optimization	enrichment by ontological types	✗	✗
Holmes	Textual model editor	✗	enrichment by ontological types	m2t: Deployment script	declarative
MOCCA	Graphical model editor	design-time: Deployment optimization	✗	m2m: Deployment plan	declarative
MULTI-CLAPP	Arbitrary UML model editor	✗	refinement by UML profile	m2t: Service adapter m2m: Deployment plan	✗
Nhan *et al.*	Graphical feature configuration editor	✗	resolution	m2t: Deployment script	declarative
PDS	Directly repre-sented in XML	✗	✗	✗	declarative
RESERVOIR-ML	Textual & graphical model editor	✗	✗	m2t: Deployment script	declarative
StratusML	Graphical multi-view model editor	design-time: Performance Availability Price analysis	✗	m2t: Deployment descriptor[*] Adaptation rules m2m: Performance models	✗
TOSCA	Graphical model editor	✗	enrichment by ontological types and Implementation artifacts	m2m: Deployment plan	mixture
VAMP	Directly repre-sented in XML	✗	✗	m2t: VM image	declarative

[*] A deployment descriptor is directly interpreted by the cloud environment.
[T] Based on OpenTOSCA as CAML provides a mapping to TOSCA.

Table VIII: Tool support

5.4. RQ4: What toolset is accompanied with existing CMLs?

Given the fact that the first CML was published in 2010, a considerable set of diverse tools beyond model editors for CMLs is already available as summarized in Table VIII.

5.4.1. Modeling support. The majority of CMLs provide a custom textual or graphical model editor. Both CloudML-SINTEF and RESERVOIR-ML provide a textual as well as a graphical notation to represent models. The latter uses a textual approach for representing elasticity rules. UML-based CMLs (CAML and MULTICLAPP) can be used by arbitrary modeling tools supporting UML and its profile mechanism. Most current UML modeling tools provide multi-view model editors. Also, StratusML comes with a multi-view model editor implemented on top of Microsoft's DSL tools. Even though the TOSCA specification does not define a standard graphical notation, Winery [Kopp et al. 2013] is a web-based model editor capable of visually representing TOSCA models. For some of the reviewed CMLs, a dedicated model editor is missing. CloudML-UFPE, PDS, and VAMP are XML-based languages where models are intended to be represented directly in XML. In case of CloudNaaS, only a parser is available for its language. Hence, engineers are forced to use a plain text editor to represent network policies.

5.4.2. Analysis support. Only two of the reviewed CMLs bring analysis support to deployment topologies. CloudMIG's toolset is capable of validating a cloud application against constraints defined over cloud environments (see Section 5.3). Furthermore, it supports engineers seeking for the Pareto optimal set of deployment configurations with the objective to minimize response time, the number of violations of an upper-bound response time, and costs. In fact, the solutions in the Pareto optimum are a trade-off between performance and costs. MOCCA addresses the problem of distributing cloud application components over a single or multiple cloud environments. For that reason, a deployment topology is translated into a labeled connected graph where the labels capture the information (*e.g.,* data throughput per time unit or the workload of an application component) according to which the set of optimal graph partitions are calculated using a cohesion metric.

5.4.3. Refinement support. Considering the toolset of CMLs for refinement tasks, the Blueprint approach enables the resolution of defined service requirements by performing a string-based matching against service offerings. As a result of this resolution process, explicit interconnections between concrete service blueprints are derived. A resolution process is also carried out by the approach of Nhan *et al.* to derive a valid configuration of a feature model from an initial one. In fact, the resolution is achieved by additionally selecting required features that are not covered by the initial configuration. As some CMLs propose to model a cloud application independent of a cloud environment in a first step (CAML, CloudML-SINTEF, MULTICLAPP, the approach of Holmes, StratusML, and TOSCA), they employ dedicated approaches to achieve a refinement towards the target cloud environment. In essence, all of them exploit predefined environment-specific information that is associated with an environment-independent model. How the environment-specific information is in fact captured and associated with a model from a technical perspective varies between the CMLs mainly because they are realized on top of different platforms. Furthermore, three of the CMLs (CloudML-SINTEF, the approach of Holmes, and StratusML) employ matching techniques to achieve the refinement step semi-automatically.

5.4.4. Generation support. The majority of CMLs exploit generative techniques. Caglar *et al.* provide a code generation facility capable of producing simulation code for CloudSim. Similarly, CAML, clADL and MULTICLAPP[33] support automated forward engineering capabilities. CAML and CloudMIG support model-based reverse engineering. Whereas the former uses UML for representing generated models, the latter exploits KDM. Several CMLs support the generation of deployment scripts, descriptors, and plans from deployment configuration (see the approaches of Caglar *et al.*, Holmes, and Nhan *et al.*, MOCCA, MULTICLAPP, RESERVOIR-ML, StratusML, TOSCA).

[33]Whereas clADL exploits Monticore's code generation facility [Krahn et al. 2010], CAML and MULTICLAPP focus on code generation for the Java platform.

Virtual machine images can be generated from deployment configuration by the VAMP. They are represented in OVF. A few approaches deal with environment-related artifacts such as a run-time model (CloudML-SINTEF), adaption scripts or rules that allow the manipulation of provisioned cloud services at run-time (CloudML-SINTEF and StratusML), and network rules capable of re-provisioning virtual networks to support different fail-over scenarios (CloudNaaS).

5.4.5. Provisioning support. Finally, more than half of the reviewed CMLs come with tool support capable of automatic application and service provisioning. Most of them suggest a declarative approach, thereby reducing the effort for engineers to describe the provisioning actions explicitly. Interestingly, the CML of Caglar *et al.* supports the opposite approach. Their CML provides modeling concepts for expressing the provisioning actions in a strict imperative style. The TOSCA-compliant run-time container OpenTOSCA [Binz et al. 2013] combines the declarative and imperative approach [Breitenbücher et al. 2014]. It enables automated plan-based provisioning and management of cloud applications, which allows imperative deployment scripts to be integrated into a deployment plan. The latter is generated from a deployment configuration represented in terms of TOSCA.

5.4.6. Summary of CML tool support. Current CMLs span a broad spectrum of tools supporting engineers in the design, development, and provisioning of cloud applications. Moreover, the existing set of tools enable them to deal with different application scenarios, *e.g.,* migration of a non-cloud application to a cloud environment, distribution of application components in a single or multi-cloud environment, optimization of cloud applications. Having categorized the core tools provided by existing CMLs can be considered as a first step towards a canonical CML toolkit [Medvidovic and Taylor 2000]. Overall, existing CMLs have placed the greatest emphasis on modeling, generation, and provisioning of cloud applications and the least on analysis and refinement.

5.5. Threats to validity

Two main threats may jeopardize the internal validity of this systematic literature review. First, the search of potentially relevant research works was carried out on a set of publication sources that we determined. Since cloud computing is a highly diverse research topic, we may have excluded publication sources in which research works relevant to this review are published. To reduce the possibility of missing publication sources, we formulated in a first step a relatively generic search query for obtaining a set of potentially relevant publication sources. From this obtained initial set of publication sources, we excluded those sources that seemed to be out of the scope of this review in a second manual step. Second, the definition of keywords used to prune the set of studies is a critical task, in the sense that many studies are already excluded before the manual search process is carried out. To reduce the risk of defining a keyword list that is too restrictive, we carefully tested it with a set of studies we were aware of from previous work. Concerning external threats to validity, we cannot claim any results regarding language features outside of our classification and comparison framework even though some CMLs are realized as extensions to existing general-purpose or architecture description languages.

6. CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Several CMLs accompanied by a considerable set of tools have been proposed so far. As they address the diversity of modern cloud environments and their services, existing CMLs pursue different goals, differ in scope, and provide thus partially overlapping but also diverse modeling concepts. Consequently, the investigation of the diverse features currently provided by CMLs is of high interest. In this article, we presented a common classification and comparison framework with the goals: (i) to support CSCs in selecting the CML that fits the needs of their application scenario and setting; and (ii) to investigate language characteristics and concepts of existing CMLs for which extensive support is already provided and also in which current CMLs are deficient. In this respect, the current framework is an important advance for highlighting the essence of current CMLs and providing an effective means to compare them. We expect that our framework for CMLs is going to be refined, *e.g.,* with consideration of application behavior in addition to the structure, and extended, *e.g.,* with consideration of other cloud service categories or support for extensibility. The main findings of our evaluation of 19 CMLs are summarized in 6.1 whereas possible future research directions are drawn up in 6.2.

6.1. Main Findings

Finding 1: High diversity in current cloud modeling languages. Current CMLs pursue different, sometimes even specific goals. Hence, they propose diverse modeling concepts and show various levels of maturity. At the same time, there is a common theme among them, as the majority of CMLs are capable of representing the structure of cloud applications in terms of components and their deployment on cloud services. However, even though there are structural correspondences among most of the languages, there is still the question of whether there are semantic mismatches that are only evident during run-time. This is particularly true for CMLs that use different run-time environments. As MDE follows the convention over configuration principle, the conventions may be different in different run-time environments [Bergmayr et al. 2015].

Finding 2: Little attention paid to existing standardized software modeling languages. Even though general-purpose languages such as UML and ADLs provide modeling concepts to represent applications from a variety of viewpoints, only two CMLs currently provide extensions to UML. With the relatively recent adoption of the TOSCA standard, it appears even more desirable to align modeling CMLs that emerged in the area of cloud computing and software engineering for providing continuous modeling support.

Finding 3: Primary focus of CMLs on design-time aspects. The majority of currently existing CMLs is primarily used for representing design-time artifacts such as application components manifested by deployable artifacts or deployment targets. However, considering run-time aspects at the model level appears also promising for capturing the current status and workload of a certain provisioned compute service. This run-time information can be exploited for various tasks such as adaptation or optimization as the models@run-time paradigm [Blair et al. 2009] propagates it. A few CMLs have already taken up this idea to further use the models in the life-cycle of an application.

Finding 4: Considerable set of tools for current CMLs. Current CMLs are equipped with a broad spectrum of tools supporting engineers in the design, development, and provisioning of cloud applications. Overall, existing CMLs have put much emphasis on modeling, generation, and provisioning of cloud applications, but only limited efforts on analysis and refinement of cloud applications.

Finding 5: Lack of interoperability between CMLs. The exchange of models between CMLs requires currently manual effort for replicating those models in the different languages. Model transformations for an automated model exchange are not available even though scenarios such as application modernization would benefit from a combined set of tools provided by different CMLs. In previous work, we have outlined a modernization process that consists of several activities requiring automation support. While there is no single CML to cover all of these activities, a combination of CMLs and their accompanying tools would be suitable to perform them [Bergmayr et al. 2014b]. However, there is currently a lack of interoperability among CMLs. As a result, a tedious manual model recreation task is required for performing certain activities in the modernization process.

Finding 6: Heterogeneous language engineering background. From a language engineering perspective, two dominant meta-languages are used to realize CMLs: MOF and XML Schema. Some CMLs are realized using a grammar-based approach while others employ meta-languages of proprietary language workbenches such as Microsoft's DSL tools. The heterogeneities imposed by the different meta-languages used to implement them certainly impedes to achieve interoperability between CMLs. However, the good news is that the CMLs are realized with explicit language definitions, which allows to reason about the language concepts in a precise manner. Furthermore, for several meta-language combinations, there are already bridges available to translate the language definitions as well as the model representations between different metamodeling stacks [Kurtev et al. 2002].

Finding 7: Creation, evolution, and eventual convergence of CMLs. Based on the survey information we derived, we have also analyzed the evolution of the different CMLs. Seven CMLs have been developed before the first official version of the TOSCA standard has been published in 2013. Even more interestingly, eleven CMLs emerged mostly in parallel with the TOSCA standard. Some CMLs are still further developed in newer publications. However, it has to be noted that there has not been a single new CML published after 2014. This may have several reasons, one being the existence and emergence of the TOSCA standard, which also contributes to the convergence of different CMLs as some extensions of these CMLs aim for integration with TOSCA. Furthermore, while all other languages have their metamodel—this also applies to the two UML extensions where only the host language is standardized, but not the extensions themselves—having interoperability with a standardized language may be of practical importance.

Finding 8: Sparsely connected CML research communities. Another finding based on the gathered survey information is that different research communities have contributed on CMLs, or even if it is the same research community, then the languages have been proposed for very specific fields. We documented in a citation matrix how the different CMLs cite each other in their corresponding (denoted with an "X"). The matrix can be found in the electronic appendix (see Figure 3). For example, it can be seen how Blueprint refers to StratusML. The citation matrix also contains information about the possibility to cite papers. This means, when a paper cannot be referenced because it was not yet published, we use a less than sign, meaning that the CML has been published before the other CML. Empty cells indicate that it was most likely possible to cite the paper based on the publication date, but it has not been done. An interesting outcome of the citation matrix is that some CMLs are not even citing one of the other CMLs and that some CMLs are not cited by any of the other ones. One explanation may be that the CMLs considered a very particular case such as cyber-physical systems combined with cloud computing or migration of existing systems to cloud platforms. Another difference between the CMLs that are unaware of each other is that they are coming from different paradigms such as component-orientation vs. service-orientation. The most cited CMLs in the context of the citation matrix are CloudML-SINTEF and TOSCA. Please note that we could not trace potential influences for TOSCA as the standard document does not provide references to other CMLs.

6.2. Future Research Directions

Direction 1: Continuous cloud modeling support. Since most CMLs introduce their own set of modeling concepts, a well-connected mix of CMLs is currently not available. Surprising is the fact that the supported modeling concepts for the component and deployment viewpoint are largely inconsistent. Having the TOSCA standard, it is desirable to align existing and potential new CMLs for providing continuous modeling support, *e.g.,* by achieving interoperability among the languages.

Direction 2: Run-time models for cloud applications. Run-time aspects at the model level appear promising for capturing the status and workload of a provisioned compute service. This run-time information can be exploited for various tasks, *e.g.,* optimization and adaptation. For instance, if a modeled compute service is replaced by another one for some purpose, *e.g.,* to save costs or increase performance, the running service instances need to be adapted according to the changes in the run-time model. These changes could then be propagated to the running cloud services.

Direction 3: Simulation of deployment configurations. Analyzing and predicting non-functional properties such as costs and performance before the actual application provisioning is carried out seems of high interest. The simulation of deployment configurations refined towards a cloud environment could contribute to predicting operational costs of certain cloud services. It could support engineers to evaluate the effectiveness of defined elasticity rules before they are applied to the running application. This requires an operational semantics that captures the behavior of cloud services to simulate them rather than execute their provisioning. As more than half of the CMLs are realized based on MOF, a first promising step could be to explore fUML [OMG 2016] because it can be used to define the operational semantics of MOF-based languages [Mayerhofer et al. 2013].

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

This work was supported by the European Commission under the ICT Policy Support Programme, grant no. 317859, grant no. 318484, and grant no. 317715, by the BMWi project SmartOrchestra grant no. 01MD16001F and by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development. We would like to thank the authors of the surveyed approaches for providing us with their feedback on an earlier version of this work. Moreover, we would like to thank the reviewers for their valuable comments and recommendations to improve our work.

REFERENCES

Vasilios Andrikopoulos, Tobias Binz, Frank Leymann, and Steve Strauch. 2013. How to Adapt Applications for the Cloud Environment. *Computing* 95, 6 (2013), 493–535.

Vasilios Andrikopoulos, Anja Reuter, Santiago Gómez Sáez, and Frank Leymann. 2014a. A GENTL Approach for Cloud Application Topologies. In *Proc. of European Conf. on Service-Oriented and Cloud Computing (ESOCC)*. 148–159.

Vasilios Andrikopoulos, Anja Reuter, Mingzhu Xiu, and Frank Leymann. 2014b. Design Support for Cost-Efficient Application Distribution in the Cloud. In *Proc. of Intl. Conf. on Cloud Computing (CLOUD)*. 697–704.

Vasilios Andrikopoulos, Santiago Gómez Sáez, Frank Leymann, and Johannes Wettinger. 2014c. Optimal Distribution of Applications in the Cloud. In *Proc. of Intl. Conf. on Advanced Information Systems Engineering (CAiSE)*. 75–90.

Danilo Ardagna, Elisabetta Di Nitto, Parastoo Mohagheghi, Sébastien Mosser, Cyril Ballagny, Francesco D'Andria, Giuliano Casale, Peter Matthews, Cosmin-Septimiu Nechifor, Dana Petcu, Anke Gericke, and Craig Sheridan. 2012. MODAClouds: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. In *Proc. of Intl. Workshop on Modeling in Software Engineering (MISE)*. 50–56.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (2010), 50–58.

Colin Atkinson, Matthias Gutheil, and Bastian Kennel. 2009. A Flexible Infrastructure for Multilevel Language Engineering. *IEEE Trans. Software Eng.* 35, 6 (2009), 742–755.

Colin Atkinson and Thomas Kühne. 2003. Model-Driven Development: A Metamodeling Foundation. *IEEE Software* 20, 5 (2003), 36–41.

Colin Atkinson and Thomas Kühne. 2005. Concepts for Comparing Modeling Tool Architectures. In *Proc. of Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. 398–413.

Colin Atkinson and Thomas Kühne. 2007. A Tour of Language Customization Concepts. *Advances in Computers* 70 (2007), 105–161.

Paolo Atzeni, Paolo Cappellari, and Philip A. Bernstein. 2005. ModelGen: Model Independent Schema Translation. In *Proc. of Intl. Conf. on Data Engineering (ICDE)*. 1111–1112.

Mark Lee Badger, Timothy Grance, Robert Patt-Corner, and Jeffery M. Voas. 2012. *Cloud Computing Synopsis and Recommendations*. Technical Report USP 800-146. National Institute of Standards and Technology (NIST). http://csrc.nist.gov/publications/nistpubs/800-146/sp800-146.pdf.

Greg J. Badros. 2000. JavaML: A Markup Language for Java Source Code. *Computer Networks* 33, 1-6 (2000), 159–177.

Younes Benslimane, Michel Plaisent, Prosper Bernard, and Bouchaib Bahli. 2014. Key Challenges and Opportunities in Cloud Computing and Implications on Service Requirements: Evidence from a Systematic Literature Review. In *Proc. of Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*. 114–121.

Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. 2011. CloudNaaS: A Cloud Networking Platform for Enterprise Applications. In *Proc. of Intl. Symposium on Cloud Computing (SOCC)*. 1–13.

Alexander Bergmayr, Uwe Breitenbücher, Oliver Kopp, Manuel Wimmer, Gerti Kappel, and Frank Leymann. 2016. From Architecture Modeling to Application Provisioning for the Cloud by Combining UML and TOSCA. In *Proc. Intl. Conf. on Cloud Computing and Services Science (CLOSER)*. 97–108.

Alexander Bergmayr, Hugo Bruneliere, Javier Cánovas, Jesús Gorroñogoitia, George Kousiouris, Dimosthenis Kyriazis, Philip Langer, Andreas Menychtas, Leire Orue-Echevarria, Clara Pezuela, and Manuel Wimmer. 2013. Migrating Legacy Software to the Cloud with ARTIST. In *Proc. of European Conf. on Software Maintenance and Reengineering (CSMR)*. 465–468.

Alexander Bergmayr, Michael Grossniklaus, Manuel Wimmer, and Gerti Kappel. 2014. JUMP – From Java Annotations to UML Profiles. In *Proc. of Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. 552–568.

Alexander Bergmayr, Michael Grossniklaus, Manuel Wimmer, and Gerti Kappel. 2016. Leveraging Annotation-based Modeling with JUMP. *Software and Systems Modeling* (2016). to appear.

Alexander Bergmayr, Alessandro Rossini, Nicolas Ferry, Geir Horn, Leire Orue-Echevarria, Arnor Solberg, and Manuel Wimmer. 2015. The Evolution of CloudML and its Applications. In *Proceedings of the 3rd International Workshop on Model-Driven Engineering on and for the Cloud 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 29, 2015*. 13–18.

Alexander Bergmayr, Javier Troya, Patrick Neubauer, Manuel Wimmer, and Gerti Kappel. 2014a. UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In *Proc. of Intl. Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*. 56–65.

Alexander Bergmayr, Manuel Wimmer, Gerti Kappel, and Michael Grossniklaus. 2014b. Cloud Modeling Languages by Example. In *Proc. of Intl. Conf. on Service-Oriented Computing and Applications (SOCA)*. 137–146.

Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak, and Sebastian Wagner. 2013. OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications. In *Proc. of Intl. Conf on Service-Oriented Computing (ICSOC)*. 692–695.

Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. 2014. TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services*, Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel (Eds.). Springer.

Gordon S. Blair, Nelly Bencomo, and Robert B. France. 2009. Models@run.time. *IEEE Computer* 42, 10 (2009), 22–27.

Uwe Breitenbücher, Tobias Binz, Kálmán Képes, Oliver Kopp, Frank Leymann, and Johannes Wettinger. 2014. Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *Proc. of Intl. Conf. on Cloud Engineering (IC2E)*. 87–96.

Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann, and David Schumm. 2012. Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In *On the Move to Meaningful Internet Systems: OTM – Confederated Intl. Conferences: CoopIS, DOA-SVI, and ODBASE*. 416–424.

Manfred Broy and Ketil Stølen. 2001. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer.

Faruk Caglar, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. 2013. Model-driven Performance Estimation, Deployment, and Resource Management for Cloud-hosted Services. In *Proc. of Intl. Workshop on Domain-Specific Modeling (DSM)*. 21–26.

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Softw. Pract. Exper.* 41, 1 (2011), 23–50.

Jorge Cardoso, Alistair Barros, Norman May, and Uwe Kylau. 2010. Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments. In *Proc. of Intl. Conf. on Services Computing (SCC)*. 602–609.

Clovis Chapman, Wolfgang Emmerich, Fermín Galán Márquez, Stuart Clayman, and Alex Galis. 2010. Software architecture definition for on-demand cloud provisioning. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, Chicago, Illinois, USA, June 21-25, 2010*. 61–72. DOI:http://dx.doi.org/10.1145/1851476.1851485

Clovis Chapman, Wolfgang Emmerich, Fermín Galán Márquez, Stuart Clayman, and Alex Gallis. 2012. Software Architecture Definition for On-Demand Cloud Provisioning. *Cluster Comput.* 15, 2 (2012), 79–100.

Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. 2003. Documenting Software Architectures: Views and Beyond. In *Proc. of Intl. Conf. on Software Engineering (ICSE)*. 740–741.

Krzysztof Czarnecki and Ulrich W. Eisenecker. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

Krzysztof Czarnecki and Simon Helsen. 2006. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal* 45, 3 (2006), 621–646.

Tharam S. Dillon, Chen Wu, and Elizabeth Chang. 2010. Cloud Computing: Issues and Challenges. In *Proc. of Intl. Conf. on Advanced Information Networking and Applications (AINA)*. 27–33.

Brian Dougherty, Jules White, and Douglas C. Schmidt. 2011. Model-Driven Auto-Scaling of Green Cloud Computing Infrastructure. *Future Generation Computer Systems* 28, 2 (2011), 371–378.

Kaoutar El Maghraoui, Alok Meghranjani, Tamar Eilam, Michael Kalantar, and AlexanderV. Konstantinou. 2006. Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools. In *Proceedings of the 7th International Middleware Conference (Middleware 2006)*. Springer, 404–423.

Xavier Etchevers, Thierry Coupaye, Fabienne Boyer, Noel de Palma, and Gwen Salaun. 2011b. Automated Configuration of Legacy Applications in the Cloud. In *Proc. of Intl. Conf. on Utility and Cloud Computing (UCC)*. 170–177.

Xavier Etchevers, Thierry Coupaye, Fabienne Boyer, and Noel De Palma. 2011a. Self-Configuration of Distributed Applications in the Cloud. In *Proc. of Intl. Conf. on Cloud Computing (CLOUD)*. 668–675.

Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. 2013. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In *Proc. of Intl. Conf. on Cloud Computing (CLOUD)*. 887–894.

Nicolas Ferry, Hui Song, Alessandro Rossini, Franck Chauvel, and Arnor Solberg. 2014. Cloud MF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications. In *Proc. of Intl. Conf. on Utility and Cloud Computing (UCC)*. 269–277.

Martin Fowler. 2010. *Domain-Specific Languages*. Addison-Wesley.

Robert France and Bernhard Rumpe. 2010. Modeling for the Cloud. *Software and Systems Modeling* 9, 2 (2010), 139–140.

Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. 2013a. Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud. In *Proc. of Intl. Conf. on Software Engineering (ICSE)*. 512–521.

Sören Frey and Wilhelm Hasselbring. 2010. Model-Based Migration of Legacy Software Systems into the Cloud: The CloudMIG Approach. *Softwaretechnik-Trends* 30, 2 (2010), 1–2.

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

Sören Frey, Wilhelm Hasselbring, and Benjamin Schnoor. 2013b. Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms. *Journal of Software: Evolution and Process* 25, 10 (2013), 1089–1115.

FuWeili, Roly Perera, Paul Anderson, and James Cheney. 2017. μPuppet: A Declarative Subset of the Puppet Configuration Language. In *31th European Conference on Object-Oriented Programming (ECOOP 2017)*. Dagstuhl LIPIcs.

Martin Glinz. 2007. On Non-Functional Requirements. In *Proc. of Intl. Conf. on Requirements Engineering (RE)*. 21–26.

Glauco Gonçalves, Patricia Endo, Marcelos Santos, Djamel Sadok, Judith Kelner, Bob Merlander, and Jan-Erik Mångs. 2011. CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds. In *Proc. of Intl. Conf. on Cloud Computing Technologies and Science (CloudCom)*. 399–406.

Jack Greenfield and Keith Short. 2003. Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. In *Proc. of Intl. Conf on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 16–27.

Joaquín Guillén, Javier Miranda, Juan Manuel Murillo, and Carlos Canal. 2013a. A Service-oriented Framework for Developing Cross Cloud Migratable Software. *J. Syst. Softw.* 86, 9 (2013), 2294–2308.

Joaquín Guillén, Javier Miranda, Juan Manuel Murillo, and Carlos Canal. 2013b. A UML Profile for Modeling Multicloud Applications. In *Proc. of European Conf. on Service-Oriented and Cloud Computing (ESOCC)*. 180–187.

Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. 2012. *MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems*. Technical Report. Software Engineering Lab, RWTH Aachen University, Aachen. http://webdoc.sub.gwdg.de/ebook/serien/ah/AIB/2012-03.pdf.

Mohammad Hamdaqa, Tassos Livogiannis, and Ladan Tahvildari. 2011. A Reference Model for Developing Cloud Applications. In *Proc. of Intl. Conf. on Cloud Computing and Services Science (CLOSER)*. 98–103.

Mohammad Hamdaqa and Ladan Tahvildari. 2014. The (5+1) Architectural View Model for Cloud Applications. In *Proc. of Intl. Conf. on Computer Science and Software Engineering (CASCON)*. 46–60.

Mohammad Hamdaqa and Ladan Tahvildari. 2015. StratusML: A Layered Cloud Modeling Framework. In *Proc. of Intl. Conf. on Cloud Engineering (IC2E)*. 96–105.

Mohammad Hamdaqa and Ladan Tahvildari. 2016. StratusPM: An Analytical Performance Model for Cloud Applications. In *10th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, MESOCA 2016, Raleigh, NC, USA, October 3, 2016*. 24–31. DOI:http://dx.doi.org/10.1109/MESOCA.2016.13

David Harel and Bernhard Rumpe. 2004. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer* 37, 10 (2004), 64–72.

Mark Harman, Kiran Lakhotia, Jeremy Singer, David Robert White, and Shin Yoo. 2013. Cloud Engineering is Search Based Software Engineering too. *Journal of Systems and Software* 86, 9 (2013), 2225–2241.

Lars Hermerschmidt, Antonio Navarro Perez, and Bernhard Rumpe. 2014. A Model-based Software Development Kit for the SensorCloud Platform. In *Trusted Cloud Computing*, Helmut Krcmar, Ralf Reussner, and Bernhard Rumpe (Eds.). Springer, 125–140.

Ta'id Holmes. 2014a. Automated Provisioning of Customized Cloud Service Stacks using Domain-Specific Languages. In *Proc. of Intl. Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*. 46–55.

Ta'id Holmes. 2014b. Facilitating Development and Provisioning of Service Topologies through Domain-Specific Languages. In *EDOCW*. 422–425.

Ta'id Holmes. 2015a. Facilitating Agile Prototyping of Cloud Applications - A Model-based Approach. In *Proceedings of the MoDELS 2015 Demo and Poster Session co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015)*. 52–55.

Ta'id Holmes. 2015b. Facilitating Migration of Cloud Infrastructure Services: A Model-Based Approach. In *Proc. of Intl. Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*. 7–12.

Christian Inzinger, Stefan Nastic, Sanjin Sehic, Michael Vögler, Fei Li, and Schahram Dustdar. 2014. MADCAT: A Methodology for Architecture and Deployment of Cloud Application Topologies. In *Proc. of Intl. Symposium on Service Oriented System Engineering (SOSE)*. 13–22.

ISO/IEC. 2014a. Information technology - Cloud computing - Overview and vocabulary 17788. (2014). http://standards.iso.org/ittf/PubliclyAvailableStandards/c060544_ISO_IEC_17788_2014.zip.

ISO/IEC. 2014b. Information technology - Cloud computing - Reference architecture 17789. (2014). http://standards.iso.org/ittf/PubliclyAvailableStandards/c060545_ISO_IEC_17789_2014.zip.

Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. 2013. Cloud Migration Research: A Systematic Review. *IEEE Trans. Cloud Computing* 1, 2 (2013), 142–157.

Keith Jeffery, Geir Horn, and Lutz Schubert. 2013. A Vision for Better Cloud Applications. In *Proc. of Intl. Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud)*. 7–12.

Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. 2006. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *Proc. of Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS)*. 528–542.

Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2009. Design Guidelines for Domain Specific Languages. In *Proc. of Intl. Workshop on Domain-Specific Modeling (DSM)*. 7–13.

Barbara Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen G. Linkman. 2009. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Information & Software Technology* 51, 1 (2009), 7–15.

Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report. Keele University and Durham University Joint Report, EBSE-2007-01. http://community.dur.ac.uk/ebse/biblio.php?id=51

Barbara Kitchenham, Rialette Pretorius, David Budgen, Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen G. Linkman. 2010. Systematic Literature Reviews in Software Engineering - A Tertiary Study. *Information & Software Technology* 52, 8 (2010), 792–805.

Oliver Kopp, Tobias Binz, Uwe Breitenbücher, and Frank Leymann. 2013. Winery – A Modeling Tool for TOSCA-Based Cloud Applications. In *Proc. of Intl. Conf. on Service-Oriented Computing (ICSOC)*. 700–704.

Holger Krahn, Bernhard Rumpe, and Steven Völkel. 2010. MontiCore: A Framework for Compositional Development of Domain Specific Languages. *Intl. Journal on Software Tools for Technology Transfer* 12, 5 (2010), 353–372.

Kyriakos Kritikos, Jörg Domaschka, and Alessandro Rossini. 2014. SRL: A Scalability Rule Language for Multi-cloud Environments. In *Proc. of Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*. 1–9.

Thomas Kühne. 2006. Matters of (Meta-)Modeling. *Software and Systems Modeling* 5, 4 (2006), 369–385.

Ivan Kurtev, Jean Bézivin, and Mehmet Akşit. 2002. Technological spaces: An initial appraisal. In *Proc. of Federated Conferences CoopIS and DOA*.

Kang Kyo, Sholom Cohen, James Hess, William Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh. http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm

Philip Langer, Konrad Wieland, Manuel Wimmer, and Jordi Cabot. 2012. EMF Profiles: A Lightweight Extension Approach for EMF Models. *Journal of Object Technology* 11, 1 (2012), 1–29.

Frank Leymann. 2011. Cloud Computing. *it - Information Technology* 53, 4 (2011), 163–164.

Frank Leymann, Christoph Fehling, Ralph Mietzner, Alexander Nowak, and Schahram Dustdar. 2011. Moving Applications to the Cloud: An Approach Based on Application Model Enrichment. *Int. J. Cooperative Inf. Syst.* 20, 3 (2011), 307–356.

Dongxi Liu and John Zic. 2011. Cloud#: A Specification Language for Modeling Cloud. In *Proc. of Intl. Conf. on Cloud Computing (CLOUD)*. 533–540.

Hongbin Lu, Mark Shtern, Bradley Simmons, Michael Smit, and Marin Litoiu. 2013. Pattern-Based Deployment Service for Next Generation Clouds. In *Proc. of World Congress on Services (SERVICES)*. 464–471.

Ivano Malavolta, Henry Muccini, and Patrizio Pelliccione. 2008. DUALLY: A Framework for Architectural Languages and Tools Interoperability. In *Proc. of Intl. Conf. on Automated Software Engineering (ASE)*. 483–484.

Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Damien A. Tamburri. 2010. Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies. *IEEE Trans. Software Eng.* 36, 1 (2010), 119–140.

Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. 2011. Cloud Computing - The Business Perspective. *Decis. Support Syst.* 51, 1 (2011), 176–189.

Tanja Mayerhofer, Philip Langer, Manuel Wimmer, and Gerti Kappel. 2013. xMOF: Executable DSMLs based on fUML. In *Proc. of Intl. Conf on Software Language Engineering (SLE)*. 56–75.

Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. 2002. Modeling Software Architectures in the Unified Modeling Language. *ACM Trans. Softw. Eng. Methodol.* 11, 1 (2002), 2–57.

Nenad Medvidovic and Richard N. Taylor. 2000. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Software Eng.* 26, 1 (2000), 70–93.

Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-Specific Languages. *ACM Comput. Surv.* 37, 4 (2005), 316–344.

Ralph Mietzner, Frank Leymann, and Mike P. Papazoglou. 2008. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. In *Proc. of Intl. Conf. on Internet and Web Applications and Services (ICIW)*. 156–161.

Parastoo Mohagheghi, Arne-Jørgen Berre, Alexis Henry, Franck Barbier, and Andrey Sadovykh. 2010. REMICS – REuse and Migration of Legacy Applications to Interoperable Cloud Services – REMICS Consortium. In *Proc. of European Conf. on Towards a Service-Based Internet (ServiceWave)*. 195–196.

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

Patrick Neubauer, Alexander Bergmayr, Tanja Mayerhofer, Javier Troya, and Manuel Wimmer. 2015. XMLText: From XML Schema to Xtext. In *Proc. of Intl. Conf. on Software Language Engineering (SLE)*. 71–76.

Dinh Khoa Nguyen, Francesco Lelli, Mike P. Papazoglou, and Willem-Jan van den Heuvel. 2012. Blueprinting Approach in Support of Cloud Computing. *Future Internet* 4, 1 (2012), 322–346.

Dinh Khoa Nguyen, Francesco Lelli, Yehia Taher, Michael Parkin, Mike P. Papazoglou, and Willem-Jan van den Heuvel. 2011. Blueprint Template Support for Engineering Cloud-Based Services. In *Proc. of European Conf. on Towards a Service-Based Internet (ServiceWave)*. 26–37.

Tam Le Nhan, Gerson Sunyé, and Jean-Marc Jézéquel. 2012. A Model-driven Approach for Virtual Machine Image Provisioning in Cloud Computing. In *Proc. of European Conf. on Service-Oriented and Cloud Computing (ESOCC)*. 107–121.

OASIS. 2013. Topology and Orchestration Specification for Cloud Applications (TOSCA). (2013). https://www.oasis-open.org/committees/tosca.

OASIS. 2013. Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer. (2013). http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html.

OMG. 2011. Knowledge Discovery Metamodel (KDM). (2011). http://www.omg.org/spec/KDM.

OMG. 2016. Semantics of a Foundational Subset for Executable UML Models (FUML). (2016). http://www.omg.org/spec/FUML/.

Per-Olov Östberg, Henning Groenda, Stefan Wesner, James Byrne, Dimitrios S. Nikolopoulos, Craig Sheridan, Jakub Krzywda, Ahmed Ali-Eldin, Johan Tordsson, Erik Elmroth, Christian Stier, Klaus Krogmann, Jörg Domaschka, Christopher B. Hauser, Peter J. Byrne, Sergej Svorobej, Barry Mccollum, Zafeirios Papazachos, Darren Whigham, Stephan Rüth, and Dragana Paurevic. 2014. The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation. In *Proc. of Intl. Conf. on Cloud Computing Technology and Science (CloudCom)*. 26–31.

Michael P. Papazoglou and Luis M. Vaquero. 2012. Knowledge-Intensive Cloud Services: Transforming the Cloud Delivery Stack. In *Knowledge Service Engineering Handbook*, Jussi Kantola and Waldemar Karwowski (Eds.). CRC Press, 447–492.

Antonio Navarro Pérez and Bernhard Rumpe. 2013. Modeling Cloud Architectures as Interactive Systems. In *Proc. of Intl. Workshop on Model-Driven Engineering for High Performance and CLoud computing (MDHPCL)*. 15–24.

Dana Petcu. 2014. Consuming Resources and Services from Multiple Clouds. *J. Grid Comput.* 12, 2 (2014), 321–345.

Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. In *Proc. of Intl. Conf. on Evaluation and Assessment in Software Engineering (EASE)*. 68–77.

Jan Oliver Ringert and Bernhard Rumpe. 2011. A Little Synopsis on Streams, Stream Processing Functions, and State-Based Stream Processing. *International Journal of Software and Informatics (IJSI)* 5, 1-2 (2011), 29–53.

Alessandro Rossini. 2015. Cloud Application Modelling and Execution Language (CAMEL) and the PaaSage Workflow. In *Proc. of Workshops of European Conf. on Service-Oriented and Cloud Computing (ESOCC)*. 437–439.

Alessandro Rossini, Juan de Lara, Esther Guerra, and Nikolay Nikolov. 2015. A Comparison of Two-Level and Multi-level Modelling for Cloud-Based Applications. In *Proc. of European Conf. on Modeling Foundations and Applications (ECMFA)*. 18–32.

Alessandro Rossini, Kiriakos Kritikos, Nikolay Nikolov, Jörg Domaschka, Daniel Seybold, Frank Griesinger, Daniel Romero, Michal Orzechowski, Georgia Kapitsaki, and Achilleas Achilleos. 2017. The Cloud Application Modelling and Execution Language (CAMEL). *OPen Access Repository of Ulm University (OPARU)* (2017). DOI:http://dx.doi.org/10.18725/OPARU-4339

Bran Selic. 2007. A Systematic Approach to Domain-Specific Language Design Using UML. In *Proc. of Intl. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. 2–9.

Amit Sheth and Ajith Ranabahu. 2010. Semantic Modeling for Cloud Computing, Part 1. *IEEE Internet Computing* 14, 3 (2010), 81–83.

Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. 2013. A Systematic Review of Cloud Lock-In Solutions. In *Proc. of Intl. Conf. on Cloud Computing Technology and Science (CLOUDCOM)*. 363–368.

Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. 2014. Cloud DSL: A Language for Supporting Cloud Portability by Describing Cloud Entities. In *Proc. of Intl. Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*. 36–45.

Le Sun, Hai Dong, and Jamshaid Ashraf. 2012a. Survey of Service Description Languages and Their Issues in Cloud Computing. In *Proc. of Intl. Conf. on Semantics, Knowledge and Grids (SKG)*. 128–135.

Yih Leong Sun, Terence Harmer, Alan Stewart, and Peter Wright. 2012b. Mapping Application Requirements to Cloud Resources. In *Proc. of European Parallel Processing Workshops (Eur-Par)*. 104–112.

Vanish Talwar, Dejan S. Milojicic, Qinyi Wu, Calton Pu, Wenchang Yan, and Gueyoung Jung. 2005. Approaches for Service Deployment. *IEEE Internet Computing* 9, 2 (2005), 70–80.

Klass van den Berg, Jose Maria Conejero, and Ruzanna Chitchyan. 2005. *AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation*. Technical Report. D9 AOSD-Europe-UT-01, AOSD-Europe. http://doc.utwente.nl/64104

Luis Miguel Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. 2011. Dynamically Scaling Applications in the Cloud. *Computer Communication Review* 41, 1 (2011), 45–52.

Will Venters and Edgar A. Whitley. 2012. A Critical Review of Cloud Computing: Researching Desires and Realities. *Journal of Information Technology* 27, 3 (2012), 179–197.

Tim Waizenegger, Matthias Wieland, Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Bernhard Mitschang, Alexander Nowak, and Sebastian Wagner. 2013. Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing. In *On the Move to Meaningful Internet Systems: OTM – Confederated Intl. Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE*. 360–376.

Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE'14)*. 38:1–38:10.

Robert Woitsch and Wilfrid Utz. 2015. Business Process as a Service: Model Based Business and IT Cloud Alignment as a Cloud Offering. In *Proc. Intl. Conf. on Enterprise Systems (ES)*. 121–130.

Online Appendix to:
A Systematic Review of Cloud Modeling Languages

ALEXANDER BERGMAYR, team
UWE BREITENBÜCHER, University of Stuttgart
NICOLAS FERRY, SINTEF
ALESSANDRO ROSSINI, EVRY
ARNOR SOLBERG, SINTEF
MANUEL WIMMER, TU Wien
GERTI KAPPEL, TU Wien
FRANK LEYMANN, University of Stuttgart

A. SELECTED PUBLICATION SOURCES

Workshop	Acronym
Intl. Workshop on Model-Driven Engineering for High Performance and CLoud Computing	MDHPCL
Intl. Conference on Cluster Computing Workshops and Posters	Cluster Workshops
Intl. Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems	MESOCA
Intl. Workshop on European Software Services and Systems Research - Results and Challenges	S-Cube
Intl. Workshop on Modeling in Software Engineering	MiSE
Intl. Workshop on Analysis and Programming Languages for Web Applications and Cloud Applications	APLWACA
Intl. Workshop on Security in Cloud Computing	Cloud Computing
Intl. Workshop on Distributed Cloud Computing	DCC
Intl. Workshop on Cloud Services, Federation, and Open Cirrus Summit	FederatedClouds
Intl. Workshop on Hot Topics in Cloud Services	HotTopiCS
Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems	MASCOTS
Intl. Workshop on Multi-Cloud Applications and Federated Clouds	MultiCloud
Intl. Enterprise Distributed Object Computing Workshop	EDOCW
Intl. Workshop on Grid Computing	-
Intl. Workshop on Cloud Computing Platforms	CloudCP
Intl. Workshop on Cloud Data and Platforms	CloudDP
Intl. Workshop on Software Engineering Challenges of Cloud Computing	CLOUD@ICSE
Intl. Workshop on Model-Driven Engineering on and for the Cloud	CloudMDE
Intl. Workshop on on Model-Driven Engineering for High Performance and CLoud computing	MDHPCL

© 2018 ACM. 0360-0300/2018/02-ART22 $15.00
DOI: https://doi.org/10.1145/3150227

Conference	Acronym
Symp. on Cluster Computing and the Grid	CCGRID
Asia-Pacific Services Computing Conf.	APSCC
Conf. on Computer Science and Service System	CSSS
Conf. on Cloud Computing in Emerging Markets	CCEM
Conf. on Cluster Computing	CLUSTER
Latin American Conf. on Cloud Computing and Communications	LatinCloud
Conf. on Cloud Computing and Intelligence Systems	CCIS
Conf. on Cloud Computing and Services Science	CLOSER
Conf. on Cloud Computing	CLOUD
Conf. on Cloud Computing Technology and Science	CloudCom
Conf. on Web Services	ICWS
Conf. on Model-Driven Engineering and Software Development	MODELSWARD
Symp. on Modeling, Analysis, Simulation of Computer and Telecommunication Systems	MASCOTS
Conf. on Software Engineering	ICSE
Europ. Software Engineering Conf./Symp. on the Foundations of Software Engineering	ESEC/FSE
Conf. on Software Engineering and Service Science	ICSESS
Conf. on Software and Data Technologies	ICSOFT
Conf. on Software Engineering and Applications	ICSOFT-EA
Computer Science and Engineering Conf.	ICSEC
Conf. on Communications and Information Technology	ICCIT
Conf. on Services Computing	SCC
Asia-Pacific Services Computing Conf.	APSCC
Conf. on Service-Oriented Computing and Applications	SOCA
Conf. on Utility and Cloud Computing	UCC
Conf. on Web Services	ICWS
Conf. on Cloud and Service Computing	CSC
Conf. on P2P, Parallel, Grid, Cloud and Internet Computing	3PGCIC
Asia Pacific Cloud Computing Congress	APCloudCC
Symp. on Cloud and Services Computing	ISCOS
Conf. on Internet Computing for Engineering and Science	ICICSE
Conf. on Cloud and Green Computing	CGC
Conf. on Cloud and Ubiquitous Computing and Emerging Technologies	CUBE
Conf. on Cloud Computing and Big Data	CLOUDCOM
Conf. on Information Science and Cloud Computing Companion	ISCC-C
Conf. on Information Science and Cloud Computing	ISCC
Conf. on Cloud Engineering	IC2E
Conf. on Future Internet of Things and Cloud	FiCloud
Symp. on Service-Oriented System Engineering	SOSE
Europ. Conf. on Web Services	ECOWS
Conf. on Grid and Cloud Computing	GCC
Conf. on Cluster Computing	ICCC
Conf. on System of Systems Engineering	SoSE
Conf. on Cloud Computing Technologies, Applications and Management	ICCCTAM
Conf. on Cloud Computing and Internet of Things	CCIOT
Conf. on Service Science	ICSS
Symp. on Cloud Computing	SoCC
Conf. on Internet and Web Applications and Services	ICIW
Enterprise Distributed Object Computing Conf.	EDOC
Conf. on Systems, Programming, Languages and Applications: Software for Humanity	SPLASH
World Congress on Services	SERVICES
Conf. on World Wide Web	WWW
Conf. on Distributed Applications and Interoperable Systems	DAIS
Europ. Conf. on Service-Oriented and Cloud Computing	ESOCC
Conf. on Model-Driven Engineering Languages and Systems	MoDELS

Journal	Acronym	Publisher
Communications of the ACM	CACM	ACM
Computing Surveys	CSUR	ACM
Transactions on Internet Technology	TOIT	ACM
Transactions on Software Engineering and Methodology	TOSEM	ACM
Communications of the Association for Information Systems	CAIS	AIS
Advances in Information Sciences and Service Sciences	AISS	CIS
Computer Systems Science and Engineering	-	CRL
Computer Communications	-	Elsevier
Computer Standards and Interfaces	-	Elsevier
Future Generation Computer Systems	FGCS	Elsevier
Information and Software Technology	IST	Elsevier
Journal of Systems and Software	JSS	Elsevier
Journal of Visual Languages and Computing	JVLC	Elsevier
Intl. Journal of Web Information Systems	IJSWIS	Emerald
IBM Journal of Research and Development	IBM RD	IBM
IBM Systems Journal	Systems	IBM
Computer		IEEE
Cloud Computing	-	IEEE
Communications Letters	-	IEEE
Internet Computing	-	IEEE
Software	-	IEEE
Transactions on Computers	TC	IEEE
Transactions on Cloud Computing	TCC	IEEE
Transactions on Services Computing	TSC	IEEE
Transactions on Software Engineering	TSE	IEEE
Intl. Journal of Grid and Utility Computing	IJGUC	Inderscience
Intl. Journal of Web and Grid Services	IJWGS	Inderscience
Intl. Journal of Network Management	-	John Wiley & Sons
Computer Journal	-	Oxford Journals
Communications in Computer and Information Science	CCIS	Springer
Computing	-	Springer
Cluster Computing	-	Springer
Journal of Cloud Computing	JoCCASA	Springer
Journal of Internet Services and Applications	-	Springer
Service Oriented Computing and Applications	-	Springer
Software and Systems Modeling	SoSym	Springer
Intl. Journal of Cooperative Information Systems	IJCIS	World Scientific
Intl. Journal of Software Engineering and Knowledge Engineering	IJSEKE	World Scientific
Computer Science and Information Systems	ComSIS	-
Computing and Informatics	-	-
Journal of Internet Technology	JIT	-
Journal of Object Technology	JOT	-

B. CITATION MATRIX OF SELECTED APPROACHES

		Cited Approach																		
Approach	**[earliest,latest]**	Blueprint	Caglar et al.	CAML	clADL	CloudDSL	CloudMIG	CloudML-SINTEF	CloudML-UFPE	CloudNaaS	GENTL	Holmes	MOCCA	MULTICLAPP	Nhan et al.	PDS	RESERVOIR-ML	StratusML	VAMP	TOSCA [2013]
Blueprint	2011,2012		<	<	<	<		<		<	<	<		<	<	<		x		<
Caglar et al.	2013			<		<				<	<	<								
CAML	2014,2016	x					x	x	x			x	x	x			x	x		x
clADL	2013,2014																			
CloudDSL	2014							x	x									x		x
CloudMIG	2010,2013			<								<								
CloudML-SINTEF	2013,2014																			x
CloudML-UFPE	2011		<	<	<	<		<		<	<	<		<	<	<	x			
CloudNaaS	2014																			
GENTL	2014	x					x	x					x							x
Holmes	2014,2015			x				x												x
MOCCA	2011		<	<	<	<		<		<	<	<		<	<	<				<
MULTICLAPP	2013			<				x				<								
Nhan et al.	2012		<	<		<		<		<	<	<		<		<				<
PDS	2013			<								<								
RESERVOIR-ML	2010,2012		<	<	<	<		<		<	<	<		<		<				<
StratusML	2011,2016		x				x	x	x											
VAMP	2011		<	<	<	<		<		<	<	<		<	<	<	x			<

Fig. 3: Citation matrix