# A GENTL Approach for Cloud Application Topologies

Vasilios Andrikopoulos, Anja Reuter,
Santiago Gómez Sáez, and Frank Leymann

IAAS, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
{andrikopoulos,gomez-saez,leymann}@iaas.uni-stuttgart.de,
anja@reutertv.de

**Abstract.** The availability of an increasing number of cloud offerings allows for innovative solutions in designing applications for the cloud and in adapting existing ones for this environment. An important ingredient in identifying the optimal distribution of an application in the cloud, potentially across offerings and providers, is a robust topology model that can be used for the automated deployment and management of the application. In order to support this process, in this work we present an application topology language aimed for cloud applications that is generic enough to allow the mapping from other existing languages and comes with a powerful annotation mechanism already built-in. We discuss its supporting environment that we developed and show how it can be used in practice to assist application designers.

**Keywords:** application topology language, annotation schemes, application distribution, cloud migration.

## 1 Introduction

Cloud computing offers a platform for innovative systems that are partially or fully implemented and/or hosted using cloud offerings. Novel services like Database as a Service (DBaaS) offerings, for example, can be used for designing a new generation of applications, or for adapting existing ones in order to reap the well documented benefits of virtually infinite cloud capacity [5] — of course at a cost. Being able to distribute the application components across cloud offerings, potentially across cloud providers too, opens up the design space for cloud applications significantly [1]. However, the plethora of existing offerings, and the multitude of performance characteristics and pricing models attached to them creates a multi-dimensional problem in identifying the optimal distribution of an application in the cloud.

Toward this goal, in previous work [2], we introduced a design support process and reference architecture that builds on two systems: a *knowledge base* which aggregates information from cloud providers and allows for the identification of appropriate cloud offerings, as well as cost calculation for a given usage profile,

M. Villari et al. (Eds.) : ESOCC 2014, LNCS 8745, pp. 148–159, 2014.

and an *application topology language* and its *supporting environment* which is used for identifying the optimal distribution of the application components across cloud offerings. In [2] we provide a brief introduction to both systems; in this work we focus on the latter, i.e. the topology language and its environment. More specifically, in the following we present the main concepts of the Generalized Topology Language (GENTL), its relationship to existing topology modeling languages, and the implementation of a supporting environment for the language. While the language can be used for different purposes, its main focus for this work is on providing design support capabilities, following the discussion in [2].

The contributions of this work can therefore be summarized by the following:

1. An investigation into existing application topology modeling languages and an identification of their common concepts.
2. The presentation of a generic topology language that allows for the mapping from these languages into a common model, and which supports different types of annotations for additional information to the topology model.
3. An in-depth discussion on the supporting environment for the proposed language.

The rest of this paper is structured as follows: Section 2 discusses some application topology language approaches for background purposes. Based on the identified commonalities between them, Section 3 introduces our proposal for a generalized application topology language. Section 4 enhances the language with a mechanism for annotations that are used for providing additional information to application designers. Section 5 discusses the tooling support for the language. Finally, Section 6 discusses related work, and Section 7 concludes with some future work.

## 2 Background

Cloud management tools like AWS CloudFormation[1], OpenStack[2], OpenNebula[3], and the Flexiant Cloud Orchestrator (FCO)[4] use representations of application topologies aiming at easy deployment and management of cloud resources. The application topology models used are expressed using various means like domain-specific languages (DSLs), visual templates, or graphical models. These models however are specific for each tool and are not portable across providers.

Addressing this deficiency, the Topology and Orchestration Specification for Cloud Applications (TOSCA) [6] is an OASIS standardized language for the portable description of service components, their relationships and management processes. TOSCA documents, or more precisely, Service Templates, contain *node types* defining the properties and interfaces of components, *node templates*

[1] AWS CloudFormation: `https://aws.amazon.com/cloudformation/`
[2] OpenStack: `https://www.openstack.org/`
[3] OpenNebula: `http://opennebula.org/`
[4] FCO: `http://www.flexiant.com/flexiant-cloud-orchestrator/`

representing specific components as a reference to a defined node type, *relationship types* between node types and *relationship templates* instantiating the relationship types, *topology templates* that bring together node and relationship templates, and *management plans* that define how to manage (deploy, provision, update etc.) the application. *Policies* can be attached to node or relationship templates by means of an external language like WS-Policy[5]. TOSCA also allows for the annotation of node types with requirement and capability definitions, as well as the composition of different service templates by e.g. substituting a node template with a service template having the same properties, management interfaces, requirements and capabilities.

A similar approach is Cloud Blueprinting [15] which defines the concepts of *blueprints* as abstract descriptions of cloud service offerings. Blueprints are meant to facilitate cloud service selection, customization and composition into service-based applications. *Blueprint templates* allow application developers to define their requirements in terms of functional capabilities, QoS characteristics, as well as deployment and provisioning resources as target blueprints. A Blueprint document consists of six parts: *general properties* describing the topology, the *offering(s)* described by the document, the *artifacts* necessary to implement these offerings, the *resources* required to deploy these artifacts, the virtual architecture formed by the *relations* between offerings, implementation artifacts and resources, and the *policies* that govern the elements of the document.

CloudML [8] is an approach built on model-driven engineering (MDE) principles with the intention of facilitating the provisioning, deployment, monitoring and adaptation of multi-cloud applications. It provides a DSL for topology modeling, and a runtime environment for the enactment, provisioning, modeling and adaptation of these models. Topology models define the *nodes* of the cloud infrastructure, as well as the *software artifacts* that are deployed on these nodes. Both nodes and artifacts are typed, which allows for reasoning on the topology models. Similarly, the Composite Application Framework (Cafe) [12] provides the means to describe composite service-oriented applications and deploy them automatically across different providers. A Cafe application template consists of an *application model*, a *variability model* containg variability points for the parametrization of the application, and *code artifacts and references*. Application models consist of typed components and implementation elements that allow for nested definition of topologies.

As it can be seen for the discussion above, the topology languages discussed rely on a set of common fundamental concepts with different representations in each language. They all fundamentally use a graph-based view of application topologies which consists of typed components (nodes) and connectors (edges), with the possibility of assembling components into groups, essentially forming subgraphs. Furthermore, components and connectors may have attributes that define them. The degree of granularity in these concepts across the languages however differs. This underlying similarity between these approaches is used for our definition of the GENTL language.

[5] Web Services Policy 1.5 - Framework: `http://www.w3.org/TR/ws-policy/`

## 3 The Generalized Topology Language

The key concepts to be addressed in developing a generic application topology language are *reusability* of existing models, *extensibility* to accommodate future developments, and *composability* of topology models of various granularity levels into larger, more complicated ones. Furthermore, and in order to facilitate the mapping from and to other languages, the topology language should also allow all model elements to capture information that is external to the language itself. In the following section we present our proposal for a language that satisfies these requirements.

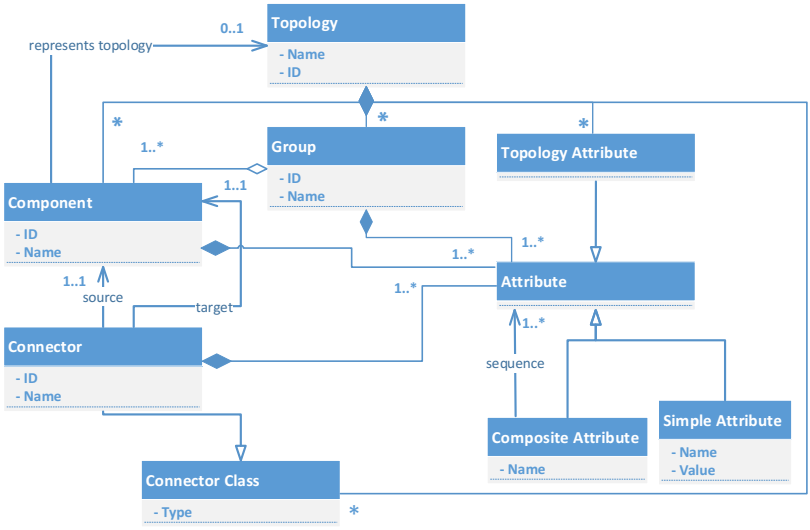



**Fig. 1.** The Metamodel of GENTL

### 3.1 The GENTL Language

The GEneralized Topology Language (GENTL) relies on a generic, but typed system. The metamodel of the language is illustrated in Fig. 1 using UML Class diagram notation. More specifically, GENTL models are built around a *Topology* element with a (unique) ID and a name which acts as a composer of the other elements in the model. Topology elements may have *Topology Attributes* that capture information about the topology model as a whole that cannot be reflected by the other elements. *Topology Attributes* are *Attributes*, either *Simple Attributes* (with a name and a value of string, integer, etc. type) or *Composite Attributes* that organize other Attributes in sequences and allow for nested attribute composition. *Components* have, in addition to a (unique) ID and a name, links to other GENTL Topology elements via the *representsTopology* association

**Table 1.** Mapping between GENTL, and Blueprints and TOSCA

| GENTL | Blueprints | TOSCA |
|---|---|---|
| Topology | Offering | Topology Template |
| Component | Resource Requirement<br>Implementation Artefact | Node Template |
| Connector | Vertical Link<br>Horizontal Link<br>Resource Link | Relationship Template |
| Group | (resourceRequirements)<br>(implementationArtefacts) | Node Type |
| Component Attribute | Resource Requirement Property<br>Implementation Artefact Property | Node Property<br>Node Interface<br>Capability Definition<br>Requirement Definition |
| Group Attribute | | Node Type Property |
| Topology Attribute | Basic Property<br>Offering Property | |
| Connector Class | Link Type | Relationship Type |

allowing for decomposing large topology models and reusing existing ones. Components have one or more Attributes attached to them. A *Connector* captures a relationship between (exactly) two Components, a *source* and a *target*. Connectors also have attributes associated with them, and they belong to one of the available *Connector Classes* that define the type of the Connector, e.g. 'deployed on'. Finally, *Groups* allow for the organization of components into sub-graphs of the topology model with non-exclusive memberships, enabling the creation of views on the topological model. Groups have attributes to provide further information about the components they aggregate.

### 3.2 Mappings from Other Languages

A key feature of GENTL is its generic nature which allows for easy mapping from other topology definition languages. As an example of this capability, Table 1 contains the mapping between GENTL and Blueprints and TOSCA. More specifically, with respect to the former, a Blueprint offering is similar in purpose to the Topology element in GENTL. Blueprint offerings are distinguished between resource requirements and implementation artefacts. Both these element types can be mapped to Components in GENTL, with elements of each type forming a Group in GENTL, and their properties captured as Component or Topology Attributes. Blueprints also support three types of links between elements: Vertical (denoting deployment dependency), Horizontal (denoting functional dependency), and Resource, for connections to external resources like IaaS

offerings. All these links are mapped as Connectors in GENTL, with the three link types modeled as Connector Class elements.

With respect to TOSCA, the mapping between the two languages is rather straightforward. Topology Templates map to Topology elements, Node Templates can be modeled as Components, and Relationship Templates as Connectors. The properties, interfaces, capabilities and requirements definitions of a Node Template are captured as Attributes on the Component, while relationship properties and interfaces are reflected as Connector Attributes. Node and Relationship Types are mapped to Groups and Connector Classes, respectively.

The mapping presented is, of course, unidirectional (from Blueprints/TOSCA to GENTL); bidirectional mapping between topology languages requires ensuring that sufficient information is available on the level of GENTL models. Annotations, as discussed in the following section can be used for this purpose.

## 4 GENTL Annotations

Annotation schemes are used to provide information that is attached, but not directly related to the application topology itself, and which can be used for several purposes like metering, billing, matching, and management. Different languages provide different mechanisms for this purpose, and existing topology annotation schemes can be classified to one or more of the following categories depending on their intended use:

1. *Discovery:* These annotations describe the capabilities or requirements of topologies and/or their elements, ranging from functional interface descriptions to QoS characteristics and semantics annotations, and used for matching purposes.
2. *Provision and Management:* These are used to automate the deployment of applications, and support tasks ranging from simple installation of components to complex system adaptations at runtime.
3. *Design Support:* This type is used to provide decision making support during the design or migration of an application to the cloud. For example, applicable design patterns can be identified and captured through these annotations.

Blueprints, for example, allow for the definition of QoS information in both offerings and resource requirements elements, which are used for discovery of matching offerings. This information is expressed as policies that are attached to the blueprint elements. TOSCA supports both discovery, and provision and management type plans by means of policy types and templates for non-functional characteristics, and management plans expressed in languages like BPEL or BPMN. In [2] we discuss how design support annotations can be used to identify the most cost efficient deployment of the application in the cloud through interaction with the application designer. In addition, and depending on the level of automation in processing the information captured in them, annotations can be *automatic* (intended for processing entirely by machines), *human-oriented* (e.g. in natural

language), or *hybrid* as a combination of them. Furthermore, annotations may be *static* or *dynamic*, requiring e.g. input from the user. Discovery annotations, for example, are usually static and automatic annotations, while provision and management annotations combine static and dynamic with automatic or hybrid characteristics.

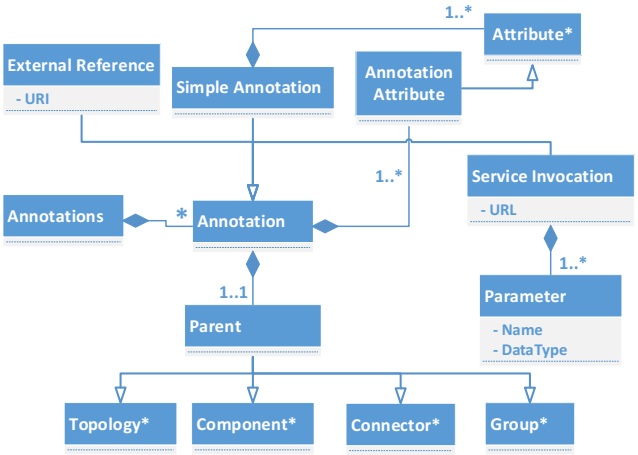



**Fig. 2.** The Annotations Metamodel of GENTL (*Class** refers to Fig. 1)

As discussed in the introduction, the main focus of GENTL is on providing design support, which constitutes the main requirement on the language with respect to its annotation scheme. However, and in order to preserve the generic nature of the language, GENTL Annotations as summarized by Fig. 2 are designed to support all types of annotations discussed above. Following the example of the WS-Policy Framework, GENTL Annotations are defined on a separate document that contains references to the topology definition(s) elements. This allows multiple annotation documents to be created for the same topology, as well as reuse of existing annotations by reorganizing the references to elements. More specifically, an *Annotation* element has one or more *Annotation Attributes* that are of type Attribute (as defined in Fig. 1), and is attached to one *Parent* that can be a Topology element, or any Component, Connector or Group inside a topology model. In addition, an Annotation is one of the following types: a *Simple Annotation* (collection of Attributes), an *External Reference* (a URI referring to a resource actually containing the annotation, e.g. a TOSCA management plan), or a *Service Invocation* (containing a request endpoint, and request *Parameters* for invoking the endpoint). The first two types are static, while the last one is dynamic. The *Annotations* element groups together multiple Annotation elements into one document.

## 5 Tooling Support

Providing the right tooling support is essential for the usability of any topology language. There are some fundamental requirements towards providing an environment for GENTL users, namely: *platform independence*, capability to *import* existing topology models and their annotations from other languages into GENTL, and an easy to use *graphical environment* incorporating automatic graph layout and dynamic interaction functionalities. In the following we discuss how the GENTL Environment that we developed satisfies these requirements.

### 5.1 The GENTL Environment

The GENTL Environment was developed as a Web application, providing platform-independent access by means of most popular Web browsers. For this purpose, a project in the Django framework[6] was created. Django is based on Python and offers an object-relational mapper that enables the definition of data models in Python. The data models are persisted in a built-in database and are accessed either through API calls or SQL statements directly to the database. The resulting Django project consists of the following set of Python applications:

**Topology App:** handles the topology data and is responsible for the graph visualization (using Graphviz[7] and the pydot[8] interface between Python and Graphviz for this purpose). Topology elements are implemented as Python objects and stored in Django's database in tables containing the data model instances.

**Annotation Apps:** implement the annotation model through two different applications — a Static Annotation App for Simple Annotations and External References, and a Dynamic Annotation App for Service Invocation Annotations.

**Transformation App:** bundles the importing functionalities — currently for Blueprint and TOSCA topology models based on the mappings discussed in Section 3.2.

Beyond importing existing topologies, the GENTL Environment allows also for the modeling of application topologies from scratch. Furthermore, it supports exporting GENTL application topology models into a serialized XML format, with exporting to other languages being ongoing work. The source and binary files for the GENTL Environment are available online[9] under Apache License 2.0.

### 5.2 User Interface

The main Graphical User Interface (GUI) of the GENTL Environment is shown in Fig. 3. The top menu allows for the addition of new elements in the topology

[6] Django: `https://www.djangoproject.com/`
[7] Graphviz: `http://www.graphviz.org/`
[8] pydot: `http://code.google.com/p/pydot/`
[9] The GENTL Environment: `http://www.iaas.uni-stuttgart.de/GENTL`

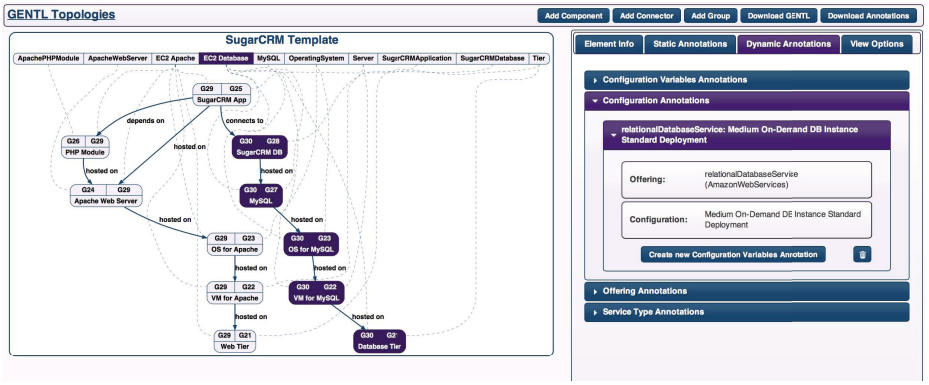


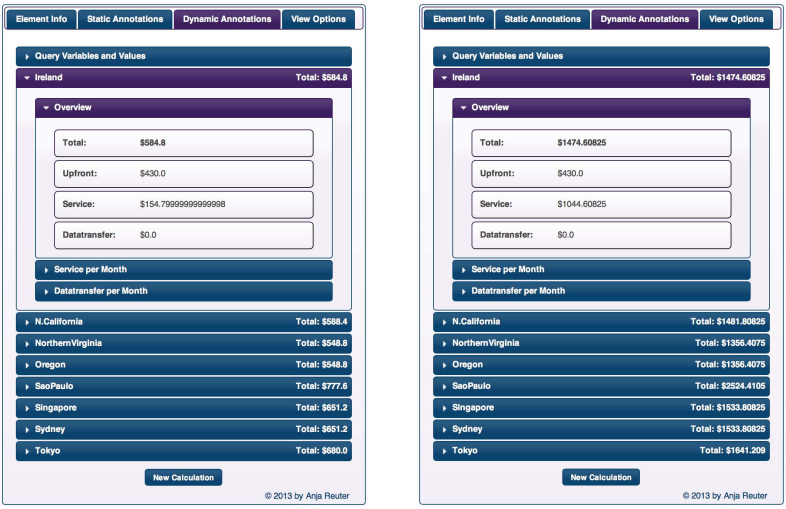

**Fig. 3.** The SugarCRM Application in the GUI of the GENTL Environment

model, exporting the model in a serialized GENTL document, and exporting the annotations (in a separate document). Following the 'GENTL Topologies' link leads to an initial screen that lists all models currently in the database, as well as importing an existing GENTL, TOSCA or Blueprint model (not shown in the figure). The topology model itself is visualized in the left pane of the screen as a graph with Group elements arranged at the top of the pane and connected to individual Components and Connectors with dashed lines to show membership in this group. Selecting a group ('EC2 Database' in Fig. 3) highlights the members of the group. The right pane of the GUI is used to show the element attributes, as well as provide access to the annotations (both static and dynamic) of each element. In the case shown in Fig. 3, the 'EC2 Database' group is annotated with the information that it is (also) deployable on a Medium On-Demand Instance DB (Standard Deployment) of the Amazon RDS[10] service instead of an EC2[11] instance. This dynamic annotation actually builds on the integration with the Nefolog system [2] for identifying and retrieving the details of this offering and can dynamically change to another cloud service offering if the requirements for this group change. Using the cost calculation capabilities of Nefolog, this information can be used to provide a projection of the operational expenses of using alternative deployment groups, as shown in Fig. 4.

## 6 Related Work

Related works in the literature build on application topology models to optimize the distribution of an application across cloud offerings. The optimization involves different dimensions, usually however in combination with operational expenses. For example, the work in [14] presents DADL, a language to describe the architecture, behavior and needs of a distributed application to be deployed

[10] Amazon Relational Database Service (Amazon RDS): `http://aws.amazon.com/rds/`
[11] Amazon Web Services Elastic Compute Cloud: `http://aws.amazon.com/ec2/`




(a) EC2 Reserved Instance (b) RDS Standard Deployment

**Fig. 4.** Cost Calculation in the GENTL Environment for two AWS Offerings

on the cloud, as well as describing available cloud offerings for matching purposes. Similarly, in [3], the authors propose an approach that matches and dynamically adapts the allocation of infrastructure resources to an application topology in order to ensure SLAs. CloudMig [9] builds on an initial topology of the application that is adapted through model transformation in order to optimize the distribution of the application across cloud offerings. A similar approach is proposed by the MODAClouds work [4] which uses CloudML (see Section 2) for the definition of the application topology model.

The approach in [13] uses a Palladio-based application topology model in order to distribute an application across different cloud providers aiming at optimizing for availability and operational expenses. The MOCCA framework [11] deals with the same problem by introducing variability points in the application topology in order to cope with possible alternative deployment topologies. CMotion [7] uses an approach based on topology modeling, generation of alternative topologies, and consequent evaluation and selection of one of those alternatives based on multiple criteria. The work in [1] uses the notion of typed graphs for similar purposes and proposes a formal framework to support this effort. In a similar approach, MADCAT [10] incorporates to the topology model scalability elements, and refines the topology model from a high-level application topology to a ready for deployment one. While GENTL allows for mapping from different application topology language however, the above works rely on a single topology language. In this respect our proposal offers the means for a more generic approach that decouples from the specifics of each language used.

## 7 Conclusion

The existence of initiatives like TOSCA allow for the automated deployment and management of applications in a distributed manner across cloud offerings and providers. This empowers application designers to pursue "smarter", more efficient application topologies that span multiple offerings. An important component in this effort is a robust application topology modeling language that acts as the foundation for any optimization of the distributed deployment of the application. Toward this goal, in this work we present the Generalized Topology Language (GENTL) which builds on the common characteristics of existing approaches. More specifically, in the previous we presented the main concepts of the language, as well as the mappings that allow transforming topology models from other languages to GENTL. Following on, we discussed the annotation mechanism developed for the language, enabling the addition of both static and dynamic additional information to the topology models in the language. Finally, we also presented the environment that we developed for the language as a Web application.

With respect to the latter, and beyond adding additional mappings from, e.g. CloudML to GENTL, the main task for the immediate future is enabling the automated deployment and management of GENTL application models in a cross-solution manner. For this purpose, we plan to define a language-specific annotation type that will allow application designers to provide the necessary information for the mapping to a deployable language. In addition, in future work we intend to use GENTL as the underlying language for the definition of $\alpha$- and $\gamma$-topologies (application-independent and -specific topology models, respectively) as discussed in [1]. By these means we will be able to provide a comprehensive design support solution to application designers in identifying the most efficient distributed deployment of their application.

**Acknowledgment.** This work is partially funded by the FP7 EU-FET project 600792 ALLOW Ensembles.

## References

1. Andrikopoulos, V., Gómez Sáez, S., Leymann, F., Wettinger, J.: Optimal Distribution of Applications in the Cloud. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 75–90. Springer, Heidelberg (2014)
2. Andrikopoulos, V., Reuter, A., Mingzhu, X., Leymann, F.: Design Support for Cost-efficient Application Distribution in the Cloud. In: Proceedings of CLOUD 2014. IEEE Computer Society (to appear, 2014)
3. Antonescu, A.F., Robinson, P., Braun, T.: Dynamic topology orchestration for distributed cloud-based applications. In: Second Symposium on Network Cloud Computing and Applications (NCCA), pp. 116–123 (2012)

4. Ardagna, D., Di Nitto, E., Mohagheghi, P., et al.: MODAclouds: A model-driven approach for the design and execution of applications on multiple clouds. In: 2012 ICSE Workshop on Modeling in Software Engineering (MISE), pp. 50–56. IEEE (2012)
5. Armbrust, M., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009)
6. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable cloud services using tosca. IEEE Internet Computing 16(3) (2012)
7. Binz, T., Leymann, F., Schumm, D.: CMotion: A Framework for Migration of Applications into and between Clouds. In: Proceedings of SOCA 2011, pp. 1–4. IEEE Computer Society (2011)
8. Ferry, N., Rossini, A., Chauvel, F., Morin, B., Solberg, A.: Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In: Proceedings of CLOUD 2013, pp. 887–894. IEEE Computer Society (2013)
9. Frey, S., Hasselbring, W.: The cloudmig approach: Model-based migration of software systems to cloud-optimized applications. International Journal on Advances in Software 4(3&4), 342–353 (2011)
10. Inzinger, C., Nastic, S., Sehic, S., Vögler, M., Li, F., Dustdar, S.: Madcat a methodology for architecture and deployment of cloud application topologies. In: Proceedings of SOSE 2014. IEEE (to appear, 2014)
11. Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S.: Moving applications to the cloud: An approach based on application model enrichment. International Journal of Cooperative Information Systems 20(03), 307–356 (2011)
12. Mietzner, R., Unger, T., Leymann, F.: Cafe: A generic configurable customizable composite cloud application framework. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009, Part I. LNCS, vol. 5870, pp. 357–364. Springer, Heidelberg (2009)
13. Miglierina, M., Gibilisco, G., Ardagna, D., Di Nitto, E.: Model based control for multi-cloud applications. In: 5th International Workshop on Modeling in Software Engineering (MiSE), pp. 37–43 (2013)
14. Mirkovic, J., Faber, T., Hsieh, P., Malaiyandisamy, G., Malaviya, R.: DADL: Distributed Application Description Language. Tech. Rep. ISI-TR-664, USC/ISI (2010), `ftp://www.isi.edu/isi-pubs/tr-664.pdf`
15. Papazoglou, M.P., van den Heuvel, W.J.: Blueprinting the cloud. Internet Computing 15(6), 74–79 (2011)