Chapter 3
The MODAClouds Model-Driven Development

Nicolas Ferry, Marcos Almeida and Arnor Solberg

3.1 Introduction

The Cloud computing market encompasses an ever-growing number of providers offering a multitude of infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS) solutions. In order to exploit the peculiarities of each Cloud solution as well as to optimize performances, availability, and cost, an emergent need is to run and manage multi-Cloud applications [1] (i.e., applications that can execute on multiple Cloud infrastructures and platforms). However, current stacks, libraries and frameworks lack in software engineering methodologies and tools to design, deploy and maintain multi-Cloud systems as stated in the CORDIS reports on Cloud computing [2, 3], "*whilst a distributed data environment (IaaS) cannot be easily moved to any platform provider (PaaS) […], it is also almost impossible to move a service/image/environment between providers on the same level.*"

Model-Driven Development (MDD) [4] techniques are particularly useful to address these challenges. They allow shifting the paradigm from code-centric to model-centric. Models are thus the main artefacts of the development process and enable developers to work at a high level of abstraction, focusing on Cloud concerns rather than implementation details. Model transformations help automating the work of going from abstract concepts to implementation. This approach, which is commonly summarized as "*model once, generate anywhere*", is thus particularly relevant when it comes to design and management of applications across multiple Clouds,

N. Ferry · A. Solberg (✉)
Stiftelsen SINTEF, Postbox 4760 Sluppen, 7465 Trondheim, Norway
e-mail: arnor.solberg@sintef.no

N. Ferry
e-mail: nicolas.ferry@sintef.no

M. Almeida
Softeam Cadextan, 21 Avenue Victor Hugo, 75016 Paris, France
e-mail: marcos.almeida@softeam.fr

E. Di Nitto et al. (eds.), *Model-Driven Development and Operation of Multi-Cloud Applications*, PoliMI SpringerBriefs,
DOI 10.1007/978-3-319-46031-4_3

as well as migrating them from one Cloud to another. Moreover, models can also be used to reason about the application Quality of Service (QoS), and to support design-time exploration methods that identify the Cloud deployment configuration of minimum cost, while satisfying QoS constraints.

In this chapter we present the MODAClouds Model-Driven Development approach to support the design of multi-Cloud applications with guaranteed QoS. The proposed approach relies on a set of tool-supported domain-specific languages (DSLs) collectively called MODACloudML. MODACloudML enables managing multi-Cloud applications in a Cloud provider-independent way while still exploiting the peculiarities of each IaaS and PaaS solution. By supporting both IaaS and PaaS, MODACloudML enables several levels of control of multi-Cloud applications by the Models@runtime engine (see Chap. 9): (i) in case of executing on IaaS or white box PaaS solutions; full control with automatic provisioning and deployment of the entire Cloud stack from the infrastructure to the application, or (ii) in case of executing on black box PaaS solutions; partial control of the application (note that if parts of the multi-Cloud application executes on IaaS or white box PaaS, MODACloudML provides full control of those parts).

The remainder of this chapter is organized as follows. Section 3.2 overviews the typical design process using the MODAClouds design-time tools and MODACloudML. Section 3.3 presents the overall architecture of MODACloudML. Section 3.4 details the list of models that compose MODACloudML before providing examples of some of them. Finally Sect. 3.5 presents some related works and Sect. 3.6 draws some conclusions.

3.2 The Design-Time Development Process

MODACloudML targets different profiles of users, from application developers and providers, who are concerned about the actual deployment artifacts and scripts, to QoS engineers, concerned with application performance and architectural costs. In order to support such diverse profiles, the MODAClouds Integrated Development Environment provides automation tools that facilitate the transition between different models by means of model-to-model transformations. It also provides model-to-text transformations that allow the developer to export/import models from/to specialized tools such as the QoS modelling and analysis tools from MODAClouds.

Designing a Cloud application through the design-time environment is typically a multi-stage process as depicted in Fig. 3.1. First, users specify, through the IDE, the application architecture and all its functional aspects as well as QoS requirements. In the next stage, designers may decide to refine these models, for instance, by selecting a certain class of database services and certain kinds of computational resources. In MODAClouds, this process is achieved by QoS engineers supported by the Line and SPACE 4Clouds tools (see Chap. 4). Line can be used to estimate the performance of the identified solution (e.g., response time and throughput), whilst SPACE 4Clouds can be used to find the minimum-cost multi-Cloud deployment configuration. At this

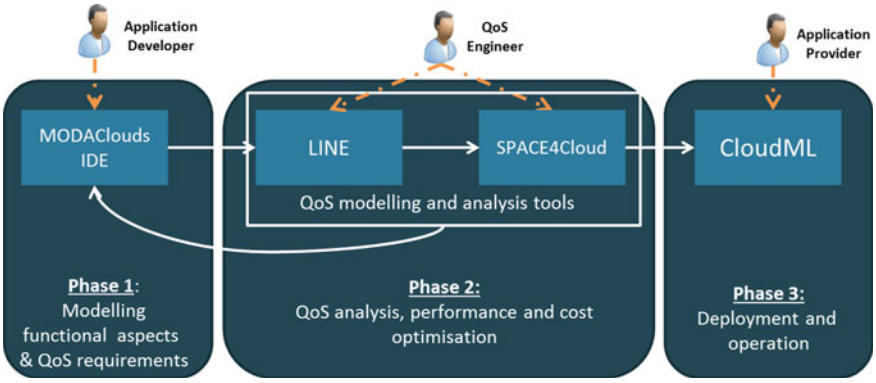

Fig. 3.1 MODAClouds design-time approach workflow

stage, an iterative process may be started to tune the models of the application until a suitable solution is identified. The output of this process is a CLOUDML deployment model that can then be used by the application provider to automatically deploy the multi-Cloud application.

All these tools rely and can be used to produce the models that compose MODACloudML. In the next sections we present the overall architecture of MODAClouds as well as the list of models it is made of.

3.3 Overall Language Architecture

The MODACloudML architecture is inspired by the OMG Model-Driven Architecture (MDA) [5], which is a model-based approach for the development of software systems. The MDA relies on three types of models for three layers of abstractions. The closer to the system a layer is, the more technical the description. These three MDA layers, from the more abstract to the more detailed, are:

- The Computational Independent Model (CIM), which describes what the system is expected to do but hides all the technical details related to the implementation of the system.
- The Platform Independent Model (PIM), which describes views of the systems in a platform independent manner so that it can be mapped to several platforms at the PSM levels.
- The Platform Specific Model (PSM), which refines the PIM with technical details required for specifying how the system can use a specific platform.

Some of the main benefits of the MDA are to facilitate the portability, interoperability and reusability of parts of the system which can be easily moved from one platform to another, as well as the maintenance of the system through human readable and reusable specifications at various levels of abstraction.

From the Cloud perspective, the introduction of new layers of abstraction improves the portability and reusability of Cloud related concerns amongst several Clouds. Indeed, even if the system is designed for a specific platform including framework, middleware, or Cloud services, these entities often rely on similar concepts, which can be abstracted from the specificities of each Cloud provider. Typically, the topology of the system in the Cloud as well as the minimum hardware resources required to run it (e.g., CPU, RAM) can be defined in a Cloud-agnostic way. Thanks to this new abstraction layer, one can map a platform specific model to one or more Cloud providers.

The MODACloudML architecture refines the PSM abstraction layer by dividing it into two sub-levels: the Cloud Provider-Independent Models (CPIM) level and the Cloud Provider-Specific Models (CPSM) level, whilst the CIM and PIM can be grouped into a so called Cloud-enabled Computational Independent Model (CCIM) level. MODACloudML thus relies on the following three layers of abstraction: (i) the Cloud-enabled Computation Independent Model to describe an application and its data, (ii) the Cloud-Provider Independent Model to describe Cloud concerns related to the application in a Cloud-agnostic way, and (iii) the Cloud-Provider Specific Model to describe the Cloud concerns needed to deploy and provision the application on a specific Cloud.

3.4 MODACloudML Sub Models

The models that compose MODACLoudML are presented and organised according to the modelling level they belong in Fig. 3.2.

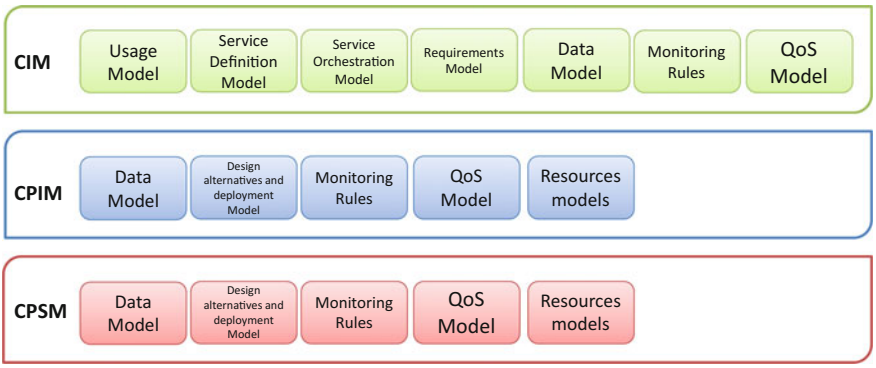

Fig. 3.2 The MODACloudML models

3.4.1 CCIM Models

The CCIM models, which define what the system is expected to do but hide the Cloud-related concerns, are the following:

Service Definition Model: describes the software to be developed as a set of components or services. It includes the typical constructs needed for describing the structure of a software system.

Usage Model: specifies the way users are expected to exploit the functionality of the software to be. It consider a 24 h time-horizon. Each single point in time of the usage model can be exploited by QoS tools regarding the search for optimal solutions.

Service Orchestration: describes the behaviour of the glue between components and services. It can be annotated with stochastic information used to express the probability for some behavioural path to be followed which can in turn be exploited by QoS analysis and optimisation tools.

Requirements Model: completes and formalizes the service functional description. Business and QoS requirements can be associated to a Service or to a specific service operation.

Data Model: describes the main data structures associated with the software to be. It can be expressed in terms of typical Entity Relational (ER) diagrams and enriched by a metamodel that specifies functional and non-functional data properties.

QoS Model: includes information concerning expected QoS characteristics (e.g., response time) at the application level. QoS contraints can be attached to specific application component/services.

In the following we exemplify the usage of the Service Orchestration models to specify the overall architecture of the SensApp case study.

3.4.2 Example

At the CCIM level, an application is described as a set of high level services following a Service Oriented Architecture (SOA) [6]. The application is specified as a set of business-aligned reusable services that can be combined into high-level business processes and solutions within the context of an enterprise.

Figure 3.3 depicts a simple functional architecture of the SensApp case study specified with the MODAClouds IDE as a Service Orchestration model. SensApp [7] is a typical Cloud-based application that acts as a buffer between sensor networks and Cloud-based systems. On the one hand, it facilitates sensors to continuously push data while, on the other hand, it provides higher level services with notification and query facilities.

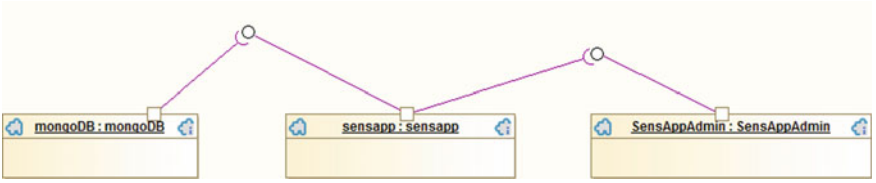

Fig. 3.3 SensApp CCIM architecture

The overall architecture of SensApp consists of a core service called *SensApp* to manage the sensors and their data coupled with a MongoDB[1] database to store sensor descriptions and meta-data as well as the measurements. The *SensApp admin* uses the public REST API of *SensApp* and provides capabilities to manage sensors and visualise data using a graphical user interface. For the sake of simplicity, other concerns such as the detailed description of interfaces, or the behaviour of services and users are not presented in this figure.

The models at the CCIM level are used to semi-automatically generate part of the CPIM models. In particular, the Service Definition Models and the Service Orchestrations Model, which can partially be generated through reverse engineering techniques, are used to initiate the Design Alternatives and deployment models whilst the CCIM data models are used to initiate the CPIM data models.

3.4.3 CPIM and CPSM Models

CPIM and CPSM levels are composed of the same set of models. CPIM models are derived from CCIM models and are in turn refined into CPSM models. The set of models that compose these two levels are the following:

Design Alternative and Deployment Model: at the CPIM level, it describes the assignment of application components to underlying resources. This includes services, platforms and infrastructural resources. At the CPSM level, it characterizes Cloud resources of a specific Cloud provider.

Data Model: at the CPIM level, this model refines the CCIM data model to describe data model in terms of logical models as flat model, hierarchical model and relational model. Finally, at the CPSM level, it describes the data model based on the specific data structures implemented by the Cloud providers.

Monitoring Rules: this model describes the monitoring rules aiming at controlling the execution of specific application components/data/connectors assigned to specific resources. They are used to indicate to the run-time platform what components/services to monitor.

[1] https://www.mongodb.org.

QoS Model: this model includes information concerning QoS characteristics of Cloud resources in both a provider-independent (CPIM level) and provider-specific (CPSM level) way. It includes cost information, thus, offering the possibility to estimate an upper-bound for application costs.

Resources Model: this model represents different Cloud environment and offerings and can be used as a catalogue of available resources. This catalogue is particularly useful as a basis for the specification of CPIM and CPSM models. It is also used in order to evaluate performance and cost of applications, as proposed by the decision making and analysis tools, as well as during the selection of the resource to be used by a multi-Cloud application.

In the following we exemplify the usage of the deployment model to specify the component deployment and orchestration in the Cloud. Deployment models are specified using CLOUDML.

CLOUDML [8, 9] consists of: (i) a domain-specific language (DSL) for specifying the provisioning and deployment of multi-Cloud applications; and (ii) a models@runtime environment for enacting the provisioning, deployment, and adaptation of these applications. While the CLOUDML language is part of MODACloudML, the models@runtime environment is integrated as part of the MODAClouds IDE. This way, developers can take advantage of the CCIM models and of the optimization tools in order to specify deployment models. CLOUDML allows developers to model the provisioning and deployment of a multi-Cloud application at both the CPIM and CPSM levels of abstractions. This two-level approach is agnostic to any development paradigm and technology, meaning that the application developers can design and implement their applications based on their preferred paradigms and technologies.

CLOUDML is inspired by component-based approaches [10] that facilitate separation of concerns and reusability. In this respect, deployment models can be regarded as assemblies of components exposing ports (or interfaces), and bindings between these ports. In a nutshell, CLOUDML enables to express the following concepts (we refer the reader to [9] for details):

- **Cloud**: Represents a collection of VMs offered by a particular Cloud provider.
- **External component**: Represents a reusable type of VM or PaaS solution.
- **Internal component**: Represents a reusable type of application component to be deployed on an external component.
- **Port**: Represents a required or provided interface to a feature of a component.
- **Relationship**: Represents a communication between ports of two application components, they express dependencies between components.
- **Hosting**: Represents the fact that a component uses another as execution platform.

In addition, CLOUDML implements the type-instance pattern [11], which also facilitates reusability. This pattern exploits two flavors of typing, namely ontological and linguistic [12]. Figure 3.4 illustrates these two flavors of typing. SL (for Small Linux) represents a reusable type of VM. It is linguistically typed by the class VM (for Virtual Machine). SL1 represents an instance of the VM SL. It is ontologically typed by SL and linguistically typed by VMInstance.

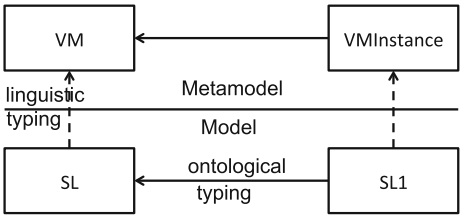

Fig. 3.4 Linguistic and ontological typing

The transformation from CPIM to CPSM consists in: (i) adding the actual data resulting from the resolution of the constraints defined in the external component types (e.g., actual number of cores, RAM size, storage size), and (ii) adding data required for the deployment and management of the application that are Cloud provider-specific. Thanks to this enrichment, it is possible to retrieve data about the actual resources provisioned including how they can be accessed and how they can be configured. Such data is particularly useful during the process of configuration of the components and their bindings.

3.4.4 Example

Figure 3.5 depicts the deployment model of SensApp at the CPIM level specified with the MODAClouds IDE. The overall system will be deployed using two different virtual machines (VMs), the first VM will host SensApp and the second the SensApp Admin. Both VMs (*CloudNodeInstance* and *ML*) have differents characteristics and are thus specified as instances of different types (*SL* and *ML*). Both SensApp and its admin, in order to be executed properly, have to be hosted in a Servlet container. In this case they are both hosted on the same type of Jetty container called *JettySC*. This type of relationship is depicted in the figure by arrows between blue ports. In addition, SensApp has to communicate with the database in order to store and retrieve sensors data. This type of relationship is depicted by arrows between purple ports.

3.5 Related Work

In the literature several efforts aimed to offer support for designing, optimizing and managing multi-Cloud applications. In particular, several EU projects provide methodologies and tools to support the design and management of Cloud-based applications. However, to the best of our knowledge, none of them propose an integrated approach offering models that can be used for performance and cost analysis and optimisation, as well as deployment and runtime management of multi-Cloud applications.

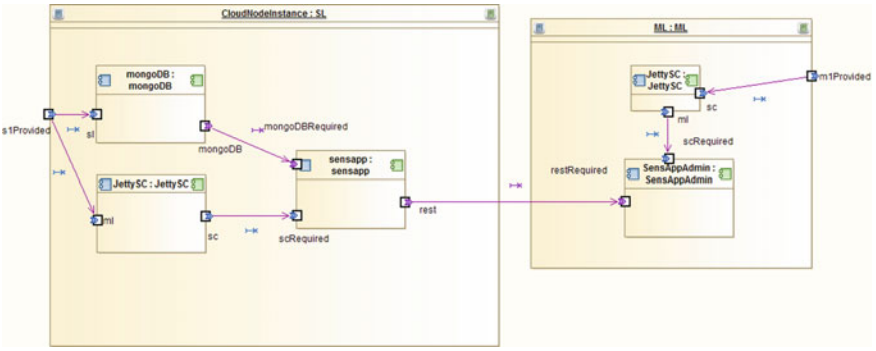

Fig. 3.5 Deployment model of SensApp at the CPIM level

The Cloud Application Modeling Language (CAML) [13] is being developed within the ARTIST EU FP7 project[2] and supports the provider-independent specification of deployment topologies and their refinement into provider-specific deployment. The main focus of the ARTIST project being the migration of legacy application to the Cloud as well as the feasibility study of such migration, the language has been defined as an UML internal modeling language based on a model library and profiles. This way, it can be directly applied on UML models, which is especially beneficial for migration scenarios where reverse-engineered UML models are tailored towards a selected Cloud environment. These CAML profiles also capture Cloud offerings from a functional and non-functional perspectives including cost aspects.

In order to cover the necessary aspects of the specification and execution of multi-Cloud applications, the PaaSage project[3] adopts the Cloud Application Modelling and Execution Language (CAMEL). CAMEL integrates and extends existing DSLs, including Cloud Modelling Language (CLOUDML) [8, 9], Saloon [14, 15], and the Organisation part of CERIF [16], for specifying multiple aspects of multi-Cloud applications, such as provisioning, deployment, providers, organisations, users, and roles. Moreover, CAMEL adds DSLs for specifying aspects such as metrics, requirements, goals, scalability rules [17, 18], security controls, execution contexts, execution histories, etc. CAMEL is designed and implemented with the Eclipse Modelling Framework (EMF)[4] on top of the Connected Data Objects (CDO)[5] persistence solution. MODAClouds and PaaSage are collaborating on the research and development of CLOUDML. However, PaaSage does not offer a specific approach for the design-time optimization of multi-Cloud applications.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [19, 20] is a specification developed by the OASIS consortium, which provides a

[2] http://www.artist-project.eu/.

[3] https://www.paasage.eu.

[4] https://www.eclipse.org/modeling/emf/.

[5] https://www.eclipse.org/cdo/.

language for specifying the components comprising the topology of Cloud-based applications along with the processes for their orchestration. TOSCA is comparable to CLOUDML, however the language has been conceived for design-time modelling only.

3.6 Conclusion

The MODAClouds Model-Driven Development approach relies on the so called MODACloudML which integrates a set of domain-specific languages. These languages cover the specifications of both functional and non functional aspects of multi-Cloud applications. Thanks to the three levels architecture, multi-Cloud applications can be designed in a Cloud provider-independent way thus reducing vendor lock-in before being refined with provider-specific information thus allowing to exploit the peculiarities of each provider.

References

1. Petcu D (2014) Consuming resources and services from multiple clouds. J Grid Comput 1–25
2. SSAI Expert Group (2010) The future of cloud computing. Technical report
3. SSAI Expert Group (2012) A roadmap for advanced cloud technologies under H2020. Technical report
4. Schmidt DC (2006) Guest editor's introduction: model-driven engineering. IEEE Comput 39(2):25–31
5. OMG: OMG model-driven architecture. http://www.omg.org/mda/
6. MacKenzie M, Laskey K, McCabe F, Brown P, Metz R (2006) Reference model for service oriented architecture 1.0. Technical report, OASIS
7. Mosser S, Fleurey F, Morin B, Chauvel F, Solberg A, Goutier I (2012) SENSAPP as a reference platform to support cloud experiments: from the internet of things to the internet of services. In: SYNASC 2012: 14th international symposium on symbolic and numeric algorithms for scientific computing. IEEE Computer Society, pp 400–406
8. Ferry N, Rossini A, Chauvel F, Morin B, Solberg A (2013) Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In: O'Conner L (ed) Proceedings of CLOUD 2013: 6th IEEE international conference on cloud computing. IEEE Computer Society, pp 887–894
9. Ferry N, Song H, Rossini A, Chauvel F, Solberg A (2014) CloudMF: applying MDE to tame the complexity of managing multi-cloud applications. In: Proceedings of UCC 2014: 7th IEEE/ACM international conference on utility and cloud computing
10. Szyperski C (2011) Component software: beyond object-oriented programming, 2nd edn. Addison-Wesley Professional
11. Atkinson C, Kühne T (2002) Rearchitecting the UML infrastructure. ACM Trans Model Comput Simul 12(4):290–321
12. Kühne T (2006) Matters of (meta-)modeling. Softw Syst Model 5(4):369–385
13. Bergmayr A, Troya J, Neubauer P, Wimmer M, Kappel G (2014) UML-based cloud application modeling with libraries, profiles and templates. In: Proceedings of workshop on CloudMDE, pp 56–65

14. Quinton C, Rouvoy R, Duchien L (2012) Leveraging feature models to configure virtual appliances. In: CloudCP 2012: 2nd international workshop on cloud computing platforms. ACM, pp 2:1–2:6
15. Quinton C, Haderer N, Rouvoy R, Duchien L (2013) Towards multi-cloud configurations using feature models and ontologies. In: MultiCloud 2013: international workshop on multi-cloud applications and federated clouds. ACM, pp 21–26
16. Jeffery K, Houssos N, Jörg B, Asserson A (2014) Research information management: the CERIF approach. IJMSO 9(1):5–14
17. Kritikos K, Domaschka J, Rossini A ((2014 (To Appear))) SRL: a scalability rule language for multi-cloud environments. In: Proceedings of CloudCom 2014: 6th IEEE international conference on cloud computing technology and science
18. Domaschka J, Kritikos K, Rossini A ((2014 (To Appear))) Towards a generic language for scalability rules. In: Proceedings of CSB 2014: 2nd international workshop on cloud service brokerage
19. Palma D, Spatzier T (2013) Topology and orchestration specification for cloud applications (TOSCA). Technical report, Organization for the Advancement of Structured Information Standards (OASIS) (June)
20. Kopp O, Binz T, Breitenbücher U, Leymann F (2013) Winery–a modeling tool for tosca-based cloud applications. In: Service-oriented computing. Springer, pp 700–704