Chapter 4
QoS Assessment and SLA Management

Danilo Ardagna, Michele Ciavotta, Giovanni Paolo Gibilisco, Riccardo Benito Desantis, Giuliano Casale, Juan F Pérez, Francesco D'Andria and Román Sosa González

4.1 Introduction

Verifying that a software system shows certain non-functional properties is a primary concern for Cloud applications.[1] Given the heterogeneous technology offer and the related pricing models currently available in the Cloud market it is extremely complex to find the deployment that fits the application requirements, and provides the best Quality of Service (QoS) and cost trade-offs. This task can be very

[1] In this chapter non-functional properties, QoS and non-functional requirements will be used interchangeably.

D. Ardagna (✉) · M. Ciavotta · G.P. Gibilisco · R.B. Desantis
DEIB, Politecnico di Milano, Piazza L. da Vinci, 32, 20133 Milano, Italy
e-mail: danilo.ardagna@polimi.it

M. Ciavotta
e-mail: michele.ciavotta@polimi.it

G.P. Gibilisco
e-mail: giovannipaolo.gibilisco@polimi.it

R.B. Desantis
e-mail: riccardobenito.desantis@polimi.it

G. Casale · J.F. Pérez
Department of Computing, Imperial College, 180 Queens Gate, London SW7 2AZ, UK
e-mail: g.casale@imperial.ac.uk

J.F. Pérez
e-mail: j.perez-bernal@imperial.ac.uk

F. D'Andria · R. Sosa González
ATOS Spain SA, Subida al Mayorazgo 24B Planta 1, 38110 Santa Cruz de Tenerife, Spain
e-mail: francesco.dandria@atos.net

R. Sosa González
e-mail: roman.sosa@atos.net

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_4

challenging, even infeasible if performed manually, since the number of solutions may become extremely large depending on the number of possible providers and available technology stacks. Furthermore, Cloud systems are inherently multi-tenant and their performance can vary with the time of day, depending on the congestion level, policies implemented by the Cloud provider, and the competition among running applications.

MODAClouds envisions design abstractions that help the QoS Engineer to specify non-functional requirements and tools to evaluate and compare multiple Cloud architectures, evaluating cost and performance considering the distinctive traits of the Cloud.

To better understand the scope of the MODAClouds QoS and SLA tools, referred to as **SPACE 4Clouds for Dev—QoS Modelling and Analysis tool**, Fig. 4.1 provides a high-level overview of the architecture and main actors involved. Each of these tools is the topic of the upcoming sections. In Figure we depict how the Feasibility Study engineer, the Application Developer and the QoS engineer provide inputs to this MODAClouds module. The Feasibility Study engineer provides a set of candidate providers for the application under development. The application developer instead creates a consistent application model and a set of architectural constraints using MODACloudML meta-models (see Chap. 3). Ultimately, the QoS engineer is in charge to define suitable QoS constraints. Simply put, the tool receives in input a set of models describing an application both in terms of functionalities and resource demands. At this point two possible scenarios are possible, in the first one the QoS engineer uses the tool in assessment mode, namely she evaluates the performance and cost based on a specific application deployment (which includes type and number of VMs and PaaS services). In the second scenario the QoS engineer provides

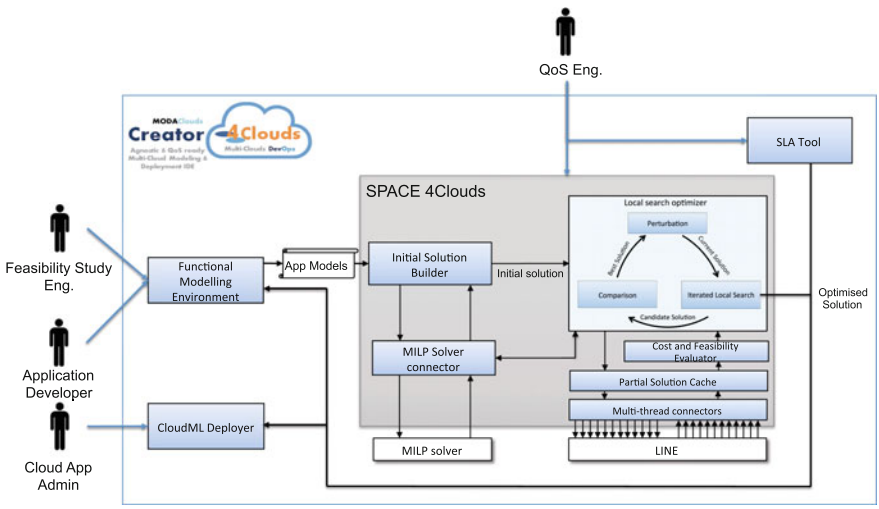

Fig. 4.1 SPACE 4Clouds for Dev—high-level architecture

only a partial configuration and lets the tool face the task of analysing the possible alternatives to return a cost optimised solution that meets the constraints.

In this latter scenario, the module returns a complete deployment description (set of providers, type of VM per tier, number of VMs per hour, type of other services), and also reports useful information about the overall cost and performance. The QoS engineer at that point may choose to accept the solution as it is, to modify the constraints or to change the deployment and evaluate/force different configurations.

This MODAClouds module is composed of three main components:

- **SPACE 4Clouds** has a twofold function. First, it keeps track of candidate solutions and manages their creation, modification, evaluation, comparison and feasibility check. Second, SPACE 4Clouds deals with the design-space exploration and optimisation process by means of a metaheuristic local-search-based approach.
- **LINE** is the component in charge of the evaluation of the performance models (Layered Queuing Networks—LQN) enriched with information about the efficiency and the dynamic behaviour that can affect the Cloud platform.
- **SLA tool** is the component responsible for generating a formal document describing a Service Level Agreement (SLA) among the involved parties in MODAClouds: customers, application providers and cloud providers.

The rest of this chapter is organised as follows: in Sect. 4.2 the MiC case study is presented, SPACE 4Clouds and LINE are described in Sect. 4.3 whereas the SLA tool is detailed in Sect. 4.4.

4.2 Case Study: Meeting in the Cloud (MiC)

In this section, we introduce a web application called Meeting in the Cloud (MiC) that will be used throughout this chapter as a case study. MiC is a web application for social networking that lets the user to profile her topics of interest and to share them with similar users. Moreover, MiC identifies the most similar users in the network according to the registered users' preferences. More specifically, during the registration process, the new user selects her topics of interest from a set of alternatives, providing a preference for each of them in the range 1–5. At the end of the registration, MiC calculates the Pearson coefficient [1] based on the preferences expressed, identifies the users in the system with the most similar interests, and creates a list of contacts for the newcomer. After the registration process, the user can log in into the MiC portal and interact with her *Best Contacts* by writing and reading posts on the selected topics. Users can also change their interests refining their profiles; in this case the system reacts re-evaluating the similarity and updating the list of recommended contacts.

The application, whose main elements are depicted in Fig. 4.2, comprises a Frontend to process the incoming http requests and a Backend developed using JSP and Servlet technologies. A task queue [2, 3] is used to decouple Frontend and Backend

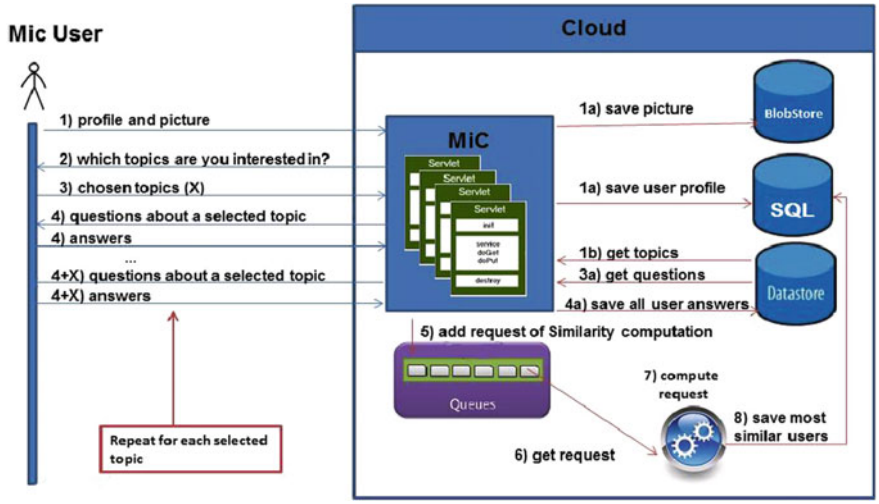

Fig. 4.2 MiC registration steps

in order to make the system capable to evaluate the similarity value in an asynchronous, non-blocking way. The overall application results in this way reactive and responsive all the time. An SQL database stores users' profiles, messages, and best contacts lists. A Blob Service is used to store pictures, while a NoSQL database stores users' interests and preferences. Both are accessed directly by the Frontend. Finally, a Memcache system is used to temporarily store the last retrieved profiles and best contacts messages with the aim of improving the response time of the whole application.

MiC is especially designed to exploit multi Cloud capabilities using a particular Java library, called CPIM, which basically provides an abstraction from the PaaS services provided by the main Cloud Providers, for more details please refer to [4].

4.3 QoS Assessment and Optimisation

SPACE 4Clouds (System PerformAnce and Cost Evaluation on Cloud) is a multi-platform open source tool for the specification, assessment and optimisation of QoS characteristics for Cloud applications. It allows users to describe a software architecture by means of MODACloudML meta-models that express Cloud-specific attributes. Among other things, such models include a user-defined workload in order to assess both performance and cost of the application under different runtime conditions. Users can specify the models defining the Cloud application using Creator 4Clouds graphical interface, while information about the performance of the considered Cloud resources is kept in a SQL database to decouple its evolution from

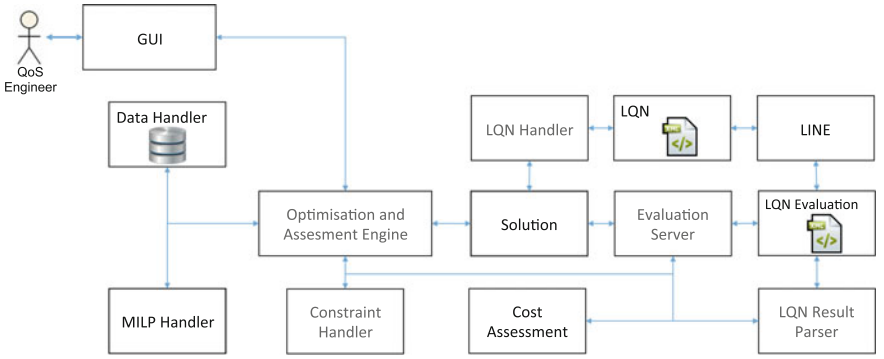

Fig. 4.3 SPACE 4Clouds—architecture

the one of the tool. SPACE 4Clouds can be used either to assess the cost of a complete described solution (i.e. application and Cloud configuration) according to the cost model defined in [5] or (providing only the application model) to find a suitable (even multi-Cloud) configuration that minimises the running cost while meeting QoS requirements.

Figure 4.3 shows the internal structure of SPACE 4Clouds and the main components are:

- *GUI*: consists of a main configuration window that allows loading the application models to be analysed and configuration parameters for the analysis/optimisation process. The GUI also provides some frames used to visualise the results of the assessment and the progress of the optimisation;
- *Solution*: represents the set of classes that form the internal representation of the application. Since a 24 h horizon is considered, the solution stores 24 records with information about configuration, performance, cost and constraint violations.
- *LQN Handler*: maps the internal representation of a solution on the LQN models used by the solver LINE (see Sect. 4.3.3) for the evaluation; the transformation process supports both IaaS and PaaS services and for multi-Cloud deployments. This component is also responsible for the serialisation of the solution in this format before the evaluation and the parsing of the output of LINE.
- *Evaluation Server*: the role of this component is to decouple the evolution of the different phases of the evaluation between the 24 h model instances for each considered provider contained in each solution. This decoupling allows the solution evaluation to happen in parallel.
- *Data Handler*: is the interface between the SQL database and other components of the tool.
- *Cost Assessment*: is the component responsible for the cost evaluation of the solution.
- *Constraint Handler*: is the component responsible to assess the feasibility of the solution with respect to Architectural and QoS constraints. Constraints are defined via a Domain-Specific Language (DSL) for flexibility and extensibility reasons.

- *Optimisation Engine*: It interacts with other components to evaluate the solutions built with respect to cost, feasibility and performance, and it is responsible for finding the optimal deployment configuration. Its core implements a metaheuristic strategy based on a two-level local search with multiple neighbourhoods.

In the following the describe separately the assessment and optimisation scenarios with the help the MiC use case.

4.3.1 Assessment

In this section we consider the assessment scenario, the one in which the QoS engineer uses SPACE 4Clouds to evaluate the cost and performance of the application under development:

1. Through the GUI the QoS engineer loads the models exported by Creator 4Clouds including also a full deployment configuration (list of providers, type and number of VMs and workload share for each hour), and a description of the incoming workload and QoS constraints.
2. The models are translated into 24 LQN instances. Each instance is tailored to model the application deployment in a particular hour of the day. These instances are then used by the Optimisation and Assessment engine to initialise the structure of a SPACE 4Clouds solution.
3. The set of LQN files is fed into the performance engine, usually LINE, which is in charge of executing the performance analysis.
4. The output of the analysis performed by LINE, stored in an XML file, is read by the LQN Handler and written back in the solution.
5. The solution can then be evaluated in terms of feasibility against user defined constraints by the Constraint Handler component.

We consider the MiC use case presented in Sect. 4.2. For the sake of simplicity only Frontend, Backend and a SQL database are considered, packed together and deployed on a single VM. Let us suppose that all the modelling work has been already done and the QoS engineer has to decide the type and the number of VMs for each hour of the day to be allocated to satisfy a set of constraints. Two candidate Cloud providers have been selected, namely Amazon and Microsoft, based upon the pricing models available on the Internet and on the user's experience. The QoS engineer considers that the daily workload for the application under development will likely follow a bimodal distribution, which he can roughly estimate. She also has to consider the non-functional requirements associated with the ongoing project. In our example the CPU utilisation is imposed to be lower than 80 % and the response time of the *register* functionality to be less than 4 s. Using such information, she devises a preliminary Multi-Cloud configuration (5 medium instances allocated on each provider per hour and 50–50 % workload splitting) and loads it along with the application functional model and the constraint set in SPACE 4Clouds; she chooses

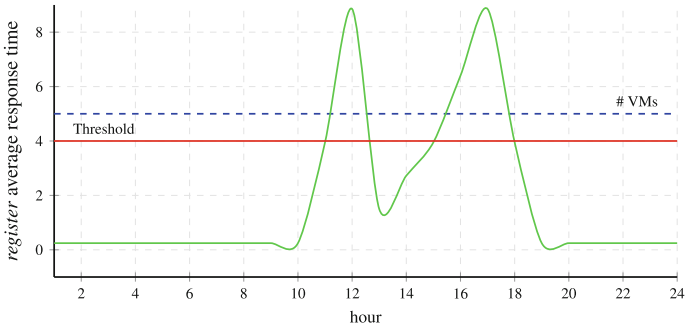

Fig. 4.4 Average response time for MiC *register* functionality

the *assessment* feature and the solution is evaluated and returned. As the reader can see from Fig. 4.4, the response time constraint is violated in the central hours of the day, while the expected daily cost is $34.8.

The solution is clearly infeasible and the QoS engineer has to pull her sleeves up and fine-tune the configuration, perhaps acting on the number of VMs and the workload splitting between the selected Clouds per hour. This is a non-trivial task since, for instance, varying the workload share directed to a certain provider affects the response time and implies an adjustment of the number of VMs running at that particular hour. A similar reasoning applies to the VM types involved. After long fine tuning, the user identifies a feasible solution with the following cost: $39.4. The solution in point has the same types of VMs of the original one and the same workload percentage for each of two providers but uses a larger number of VMs in the hours between 10 a.m. and 19 p.m.

At this point the user can be satisfied with her work but we will see in the next section that there is still room for improvement without sacrificing feasibility, exploiting the optimisation feature of SPACE 4Clouds.

4.3.2 Optimisation

The aim of this section is to provide a brief description of the optimisation strategy implemented within SPACE 4Cloud. A two-step approach has been developed; in the first step an initial valid configuration of the system is derived automatically starting from a partially specified application description given by the QoS engineer. In order to do so, a Mixed Integer Linear Problem (MILP) is built and efficiently solved [6]. This solution is based on approximated performance models, in fact, the QoS associated to a deployment solution is calculated by means of an M/G/1 queuing model with processor sharing policy. Such performance model allows calculating the average response time of a request in closed form. Our goal is to determine quickly

an approximated initial solution (list of Cloud providers, types of VMs, number of VMs and hourly load balancing) that is then further improved.

In the second step a local-search-based optimisation algorithm iteratively improves the starting Cloud deployment exploring several configurations. A more expressive performance model (LQN) is employed to derive more accurate estimates of the QoS by means of the LINE solver. More specifically, the algorithm implemented exploits the assessment feature to evaluate several, hopefully distinct Cloud configurations. It has been designed to explore the solution space using a bi-level approach that divides the problem into two levels delegating the assignment of the VM type to the first (upper) level, and the load balancing and the definition of the number of replicas to the second (lower) level. The first level implements a stochastic local search with tabu memory; at each iteration the VM type used for a particular tier is changed randomly from all the available VM types, according to the architectural constraints. The tabu memory is used to store recent moves and avoid cycling of the candidate solutions around the same configurations. Once the VM size is fixed the solution is refined by gradually reducing the number of VMs until the optimal allocation is found. Finally the workload is balanced among the Cloud providers by solving a specific MILP model. This whole process is repeated for a pre-defined number of iterations, updating the final solution each time a feasible and cheaper one is found.

Returning to the example begun in the previous section, let us imagine that the QoS engineer has at her disposal only the functional and non-functional description of the application and an indication on the possible shape and average value of the workload. The user in point can leave to SPACE 4Clouds the task of choosing the most suitable set of providers (limited to two providers for a fair comparison with the scenario in the previous section), the type and number of VMs for each provider and hour, and the hourly workload share for each provider. In this second case a feasible and optimised solution is returned in around 20 min and the related cost is $19.33 that is 50 % lower than the solution devised by trial and error in the previous section. At this point one may wonder, how is the solution from SPACE 4Clouds different from the one obtained by the QoS engineer? Fig. 4.5 depicts the number of VMs per hour for the selected Cloud providers. We can see that Microsoft has been replaced

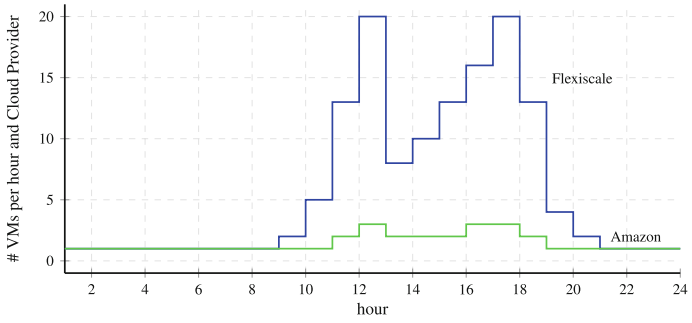

Fig. 4.5 VMs allocated per hour on Amazon and Flexiscale cloud providers

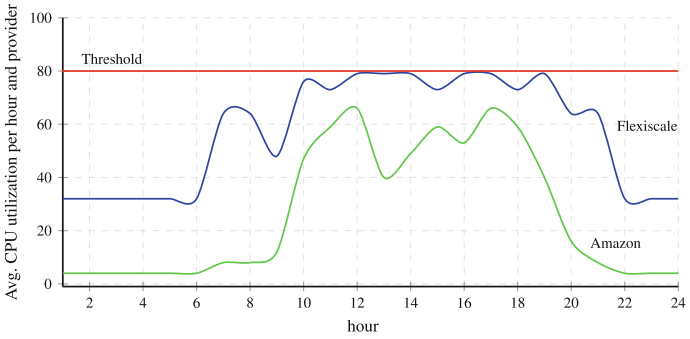

Fig. 4.6 CPU utilization per hour on Amazon and Flexiscale cloud providers

by Flexiscale and that the number of VMs allocated varies hourly from 1 through 20 differently for each provider. Moreover, distinct (more powerful) VM types have been selected and the workload has been split in 80–20 %, where the larger part has been assigned to Flexiscale. Finally, Fig. 4.6 reports the average CPU utilization per Cloud provider, that is clearly below the threshold of 80 % imposed by the user.

4.3.3 LINE

LINE [7] is a tool for the performance analysis of cloud applications. LINE has been designed to automatically build and solve performance models from high-level descriptions of the application. This description can be in the form of a Layered Queueing Network (LQN) model. From this description, LINE is able to provide accurate estimates of relevant performance measures such as application response time or server utilisation. LINE can also provide response times for specific components of the application, enabling the pinpointing of components causing a degradation in the QoS. LINE can therefore be used at design time to diagnose whether the deployment characteristics are adequate to attain the desired QoS levels.

Although other tools are available for performance modelling (such as SimuCom [8] and LQNS [9]), LINE stands apart for a number of reasons.

- In addition to provide average performance measures, LINE can compute *response time distributions*, which can be directly used to assess *percentile Service Level Agreements* (SLAs), e.g., that 95 % of the requests for the *register* functionality are processed in less than 6 s.
- LINE features a reliability model, namely *random environments* [10], to capture a number of conditions that may affect the application, including servers breakdowns and repairs, slow start-up times, resource heterogeneity and contention in multi-tenancy, a key property of cloud deployments.

- LINE is able to model *general request processing times*, which can be used to represent the resource demands posed by the very broad range of cloud applications.
- LINE offers a parallel execution mode for the efficient solution of a large number of performance models.

4.4 SLA Management

As far as SLA management is concerned, in the MODAClouds context we consider three possible actors, Cloud Service Providers (CSPs), which are responsible for the efficient utilization of the physical resources and guarantees their availability for the customers; Application Providers (APs) that are responsible for the efficient utilization of their allocated resources in order to satisfy the SLA established with their customers (end users) and achieve their business goals and customers, which represent the legitimate users for the services offered by the application providers. Usually, CSPs charge APs for renting Cloud resources to host their applications. APs, in turn, may charge their Customers for the use of their services and need to guarantee their customers' SLA. SLA violations, indeed, have an impact on APs reputation and revenue loss incurred in the case of Cloud-hosted business applications. In both circumstances penalty-based policies have to be enforced.

MODAClouds therefore devises a two-level SLA system; the first level (Customer-AP) describes the service offered by the Application Provider to its users. The guarantee terms in this SLA should only watch observable metrics by the end user. At the other level, AP-CP SLA describes the QoS expected from the Cloud provider. In this SLA level, there is one agreement per Virtual Machine or PaaS service.

The lifecycle of an SLA can be split up in several different phases:

1. preparation of the service offer as a **template**,
2. location and mediation of the **agreement**,
3. **assessment** of the agreement during execution and
4. termination and decommission of the agreement.

Within MODAClouds we designed and implemented a policy-driven SLA framework that focus on the phase 1–3 of the described lifecycle. It comprises a REST server (the SLA core) and a set of additional helper tools: the SLA Mediator and the SLA Dashboard. The Mediator tool acts as a layer atop the core, to implement some MODAClouds specific behaviour. The SLA Dashboard shows the violations and penalties of agreements in a more user-friendly way.

Figure 4.7 shows how the SLA Components are organised and how they are related to other MODAClouds components, in particular:

- SLA Repository: manages the persistence of SLA Templates, SLA Contracts and the relation between Services/Contracts/Templates.

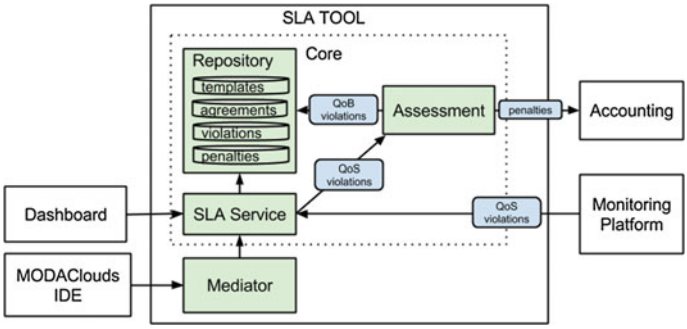

Fig. 4.7 SLA tool: architecture

- SLA Mediator: maps the QoS constraints defined by the QoS Engineer in SLA Agreements of both SLA levels.
- Assessment: computes the possible business violations, notifying any observer (like an external Accounting component) of raised penalties.

Finally, we want to remark that the tool has been implemented following to be fully compliant (concepts, agreements and templates) with the WS-Agreement[2] specification. This choice made it a tool more flexible and potentially applicable to contexts other than MODAClouds.

References

1. Pearson K (1895) Note on regression and inheritance in the case of two parents. Proc R Soc Lond 58:240–242
2. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co. Inc
3. Schmidt D, Stal M, Rohnert H, Buschmann F (2001) Pattern-oriented software architecture patterns for concurrent and networked objects. Wiley
4. Giove F, Longoni D, Yancheshmeh SM, Ardagna D, Di Nitto E (2013) An approach for the development of portable applications on PaaS clouds. Closer 2013 Proc Aachen Ger 30:591–601
5. Franceschelli D. and Ardagna D. and Ciavotta M. and Di Nitto E.: SPACE4CLOUD: a tool for system performance and costevaluation of cloud systems. Proceedings of the 2013 international workshop on multi-cloud applications and federated clouds, 2013, pp 27–34
6. Ardagna D, Gibilisco GP, Ciavotta M, Lavrentev A (2014) A multi-model optimization framework for the model driven design of cloud applications. Search-Based Softw Eng 8636:61–76. Springer
7. Pérez JF, Casale G,(2013) Assessing SLA compliance from Palladio component models. In: Proceedings of the 2nd workshop on management of resources and services in cloud and sky computing (MICAS). IEEE Press

[2]Web Services Agreement Specification (WS-Agreement) http://www.ogf.org/documents/GFD.192.pdf.

8. Becker S, Koziolek H, Reussner R (2009) The Palladio component model for model-driven performance prediction. J Syst Softw 82(1):3–22
9. Franks G, Maly P, Woodside M, Petriu DC, Hubbard A (2009) Layered queueing network solver and simulator user manual. Real-Time and Distributed Systems Lab Carleton Univ Canada
10. Casale G, Tribastone M (2011) Fluid analysis of queueing in two-stage random environments. In: Eighth international conference on quantitative evaluation of systems (QEST). IEEE, pp 21–30