Automated Throughput Optimization of Cloud Services via Model-driven Adaptation

Javier Troya, Javier Cubo, José Antonio Martín, Ernesto Pimentel and Antonio Vallecillo
Department of Computer Science, University of Málaga, Málaga, Spain
{*javiertc, cubo, jamartin, pimentel, av*}*@lcc.uma.es*

Keywords: Model-driven Engineering, DSVL, QoS Adaptation, Throughput Optimization, Cloud Service Provisioning.

Abstract: Cloud computing promises easy access, low entry cost and elasticity. However, elastic service provisioning is usually delivered via service replication, which must be supervised manually, hand-picking the services to replicate and ensuring their proper load balance. *Automated service provisioning*, i.e., the function of automatically scaling the services to cope up with their runtime demand, is a research challenge in cloud computing. In this work, we include such scalability analysis early in its development cycle, right at the design stage. We propose a model-driven approach where various QoS parameters can be simulated and analyzed using the *e-Motions* tool. Additionally, the model is automatically transformed to fit the given throughput requirements by replicating the services which cause the bottleneck. In order to evaluate the proposal, we present some initial experimental results run over the *e-Motions* tool.

1 INTRODUCTION

Cloud computing promises low entry cost and elastic scalability. The service provider pays for the amount of resources consumed by its services, and thus this elasticity allows to have fewer or more service instances (properly balanced) depending on the current demand.

However, elastic scalability is usually delivered by easily replicating service instances, what must be supervised manually (Expert Group Report, European Commission, 2010). *Automated service provisioning* (Zhang et al., 2010) is a research challenge in cloud computing.

In this ongoing work, we present an approach to include the scalability analysis of our system early in its development cycle, right at the design stage. Specifically, we propose a model-driven approach where various Quality-of-Service (QoS) parameters can be simulated and analyzed using the *e-Motions* tool (Rivera et al., 2009a; Troya et al., 2013). Additionally, the model is automatically transformed to fit the given throughput requirements. This is done by analyzing the services which pose the bottleneck of our system, replicating them and balancing the load of the workflow while minimizing their cost at the same time.

The main contributions of our approach are i) to include QoS requirements in the model of the cloud system using a Domain-Specific Visual Language (DSVL), ii) to be able to simulate their performance at design time, and iii) to automatically devise replication strategies in order to fulfill the throughput requirements.

We illustrate our approach with a model of a workflow which spans over several cloud services in 2. The initial design of this choreography cannot handle the tight throughput requirements. Instead, this is simulated and analyzed by the QoS observers placed in the model. And, using the simulated values, model-transformation rules are automatically applied to replicate the least performing services, effectively adapting the system to finally fulfill the throughput requirements. In 3, we briefly describe related work which support our approach. We conclude this work with some final remarks in 4.

2 APPROACH

In this section, we present a Model-Driven Engineering (MDE) approach for the definition of a Domain-Specific Visual Language (DSVL) to describe cloud infrastructures. Such DSVL is able to model both functional and non-functional properties. Furthermore, it may self-adapt in terms of elastic scalability by replicating service instances, in order to satisfy the desired non-functional requirements.

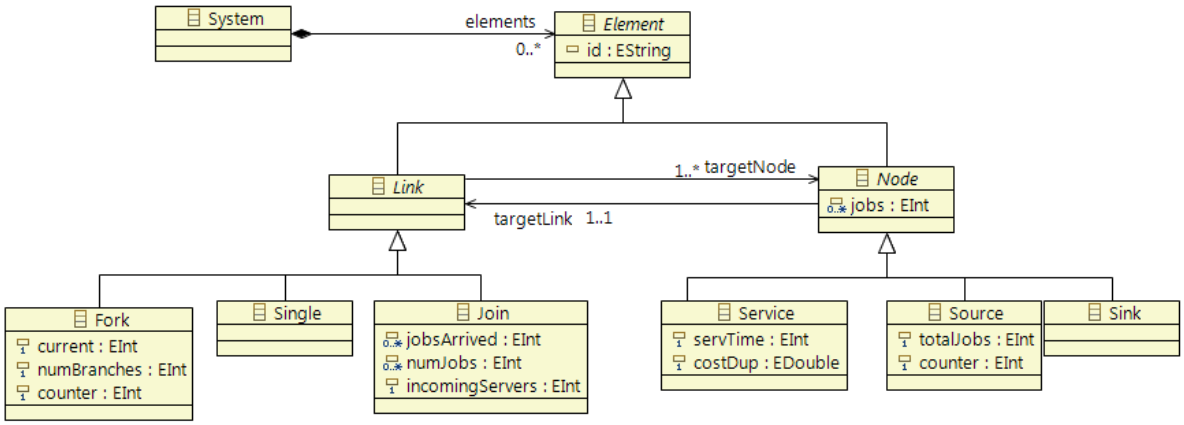


Figure 1: Metamodel of a cloud system.

2.1 Defining Cloud Infrastructures with DSVL

In MDE, any domain-specific language is defined in terms of its abstract and concrete syntaxes, and of its semantics.

2.1.1 Abstract and Concrete Syntaxes

The abstract syntax describes the static structure and it is defined by means of the metamodel shown in 1. A System is composed by a set of Elements, which can be either Links or Nodes. The latter represent the Services of the cloud systems, and also those entities that start the data flow (Source) and those where the data eventually arrives or is consumed (Sink). Services count the number of jobs they serve, and they have information about their service time and the cost of the service (which is also the cost of duplicating it). As for Links, they can be of three types, Fork, Single and Join. Fork links are used to split a job into many, Single links transmit jobs between two Nodes, and Join links are used to combine several jobs into a single one.

A model conforming to this metamodel is shown in 2. It is the initial model used in the case study we use to illustrate our approach. Our case study models a geolocation, photographic service, where the input of the system (the source src) provides a constant stream of geolocalized pictures. These pictures are then sent in parallel to services s1, which provides the map of their location, and s2, which escalates the pictures to an appropriate size and resolution. The resized pictures are then processed by service s3 which applies some image filters. The results coming from s1 and s3 are finally combined and ready to be stored or delivered by the sink (snk) of the system.

The concrete syntax chosen for our model can be seen in 2. It consists of a visual icon assigned to every non-abstract class. The objects with glasses represent our observers, and they are used to record the non-functional properties of the system. In order to be able to introduce them in the models, they need to be defined. As the system, they are defined by means of a metamodel, the one shown in 3. Then, this metamodel is merged together with the system metamodel, so that observer objects can be included in the system specifications.

An observer is an object whose purpose is to monitor the state of the system: the state of the objects, of the actions, or both. Observers, as any other objects, have a state and a well-defined behavior. The attributes of the observers capture their state, and are used to store the variables that we want to monitor. There are two types of observers: individual and general. The former are used to monitor specific objects in the system, while the latter are used to monitor properties of the system as a whole. In our case study, individual observers are aimed at monitoring both the percentage of the time that the monitored service is busy (TimeBusyOb observer) and the evolution in the cost of a service (CostOb observer), considering it can be replicated. The general observers measure the throughput of the system (ThroughputOb observer) and control the logic that deals with the replication of services in each iteration (ControllerOb observer).

2.1.2 Semantics

One way of specifying the dynamic behavior of a DSVL is by describing the evolution of the modeled artifacts along some time model. In MDE, this can be done using model transformations supporting in-place update (Czarnecki and Helsen, 2003). The behavior of the DSVL is then specified in terms of the permitted actions, which are in turn modeled by the transformation rules.

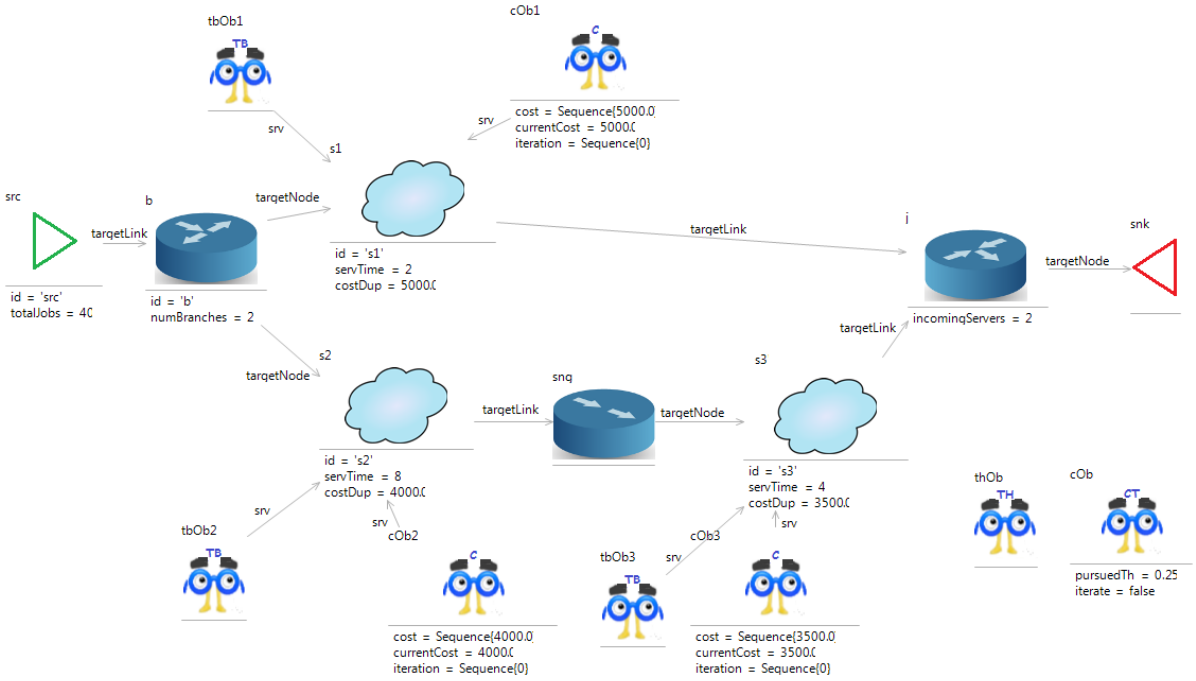


Figure 2: Initial model of our case study.

There are several approaches that propose in-place model transformations to deal with the behavior of a DSVL, from textual to graphical (see (Rivera et al., 2008) for a brief survey). Our approach provides a very intuitive way to specify behavioral semantics, close to the language of the domain expert and the right level of abstraction (de Lara and Vangheluwe, 2008).

In-place transformations are composed of a set of rules, each of which represents a possible *action* of the system. These rules are of the form $l : [\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$, where l is the rule's label (its name); and LHS (Left-Hand Side), RHS (Right-Hand Side), and NAC (negative application conditions) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the precondition for the rule to be applied, whereas the RHS one represents its postcondition, i.e., the effect of the corresponding action. Thus, a rule can be applied, i.e., triggered, if an occurrence (or match) of the LHS is found in the model and none of its NAC patterns occurs.

Generally, if several matches are found, one of them is non-deterministically selected and applied, producing a new model where the match is substituted by the appropriate instantiation of its RHS pattern (the rule's *realization*). The model transformation proceeds by applying the rules in a non-deterministic order, until none is applicable — although this behavior can be usually modified by some execution control mechanism (Rivera et al., 2009c).

In (Rivera et al., 2009b), the authors showed how time-related attributes can be added to rules to represent features like duration, periodicity, etc. Moreover, they also included the explicit representation of *action executions*, which describe actions currently executing.

There are two types of rules to specify time-dependent behavior, namely, *atomic* and *ongoing* rules. Atomic rules represent atomic actions, with a specific duration. They can be cancelled, but cannot be interrupted. Ongoing rules represent interruptible continuous actions. Atomic rules can be periodic, and atomic and ongoing rules can be scheduled, or be given an execution interval, by the lower and upper bounds of the rules.

A special kind of object, named *Clock*, represents the current global time elapse. This allows designers to use it in their timed rules.

The semantics of our system are specified by means of a set of behavioral rules that model the permitted actions in the system. Examples of such rules are shown in Figures 4(a) and 4(b), which model how a job is sent from a service to many services through a fork link. Two rules are used to model this.

In the first rule, Service-Fork, the job is forwarded from the service to the link connected to it. The condition in the LHS, written in the Object Constraint Language (OCL), checks that the service contains at least a job and that the counter of the link is set to 0. This counter attribute is used by the link to subse-

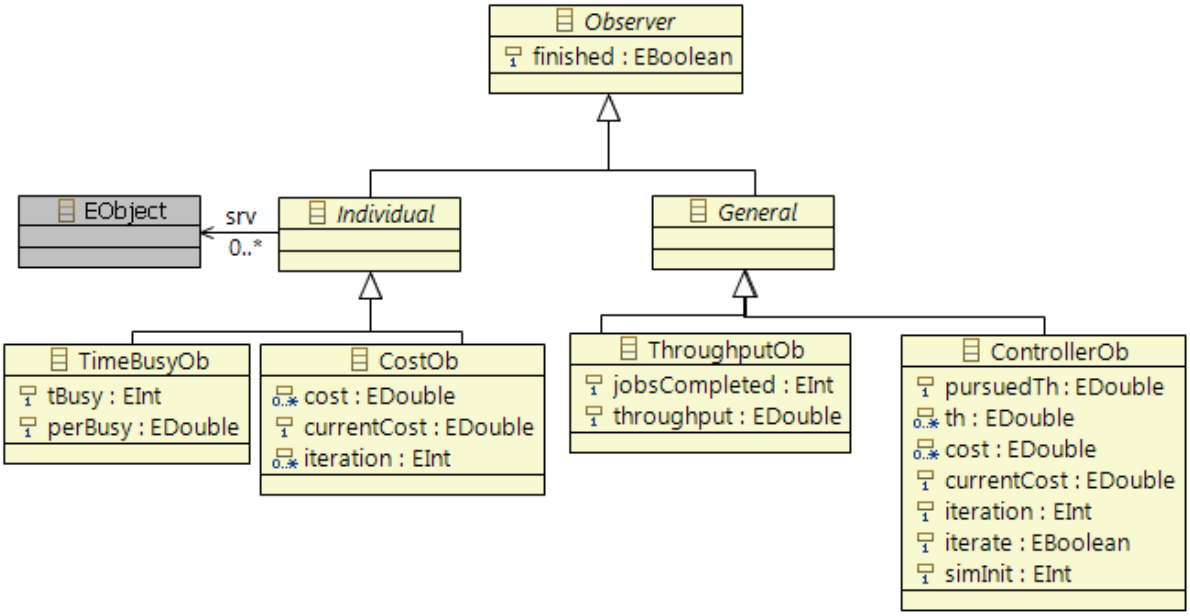


Figure 3: Observers metamodel.

quently send the job to all its outgoing services. In the rule's RHS, the job is removed from the service and placed in the link, and the number of outgoing services is given to the counter attribute of the link. The TimeBusyOb observer associated to the service is updated with the addition of the time consumed by this rule, which is the service time of the service.

The other rule, Fork-Services, models the second part of this process. The OCL condition in the LHS checks that the job is not already within the service doing the matching and that the link's counter is bigger than 0 (modeling that there are still services to which the job must be sent). In the RHS, the counter value is decreased and the job is added to the service's jobs. This rule is instantaneous (its duration is 0), since it has been added to model the logic of the fork links. Similar rule have been defined for single and join links. The complete set of behavioral rules is shown in (Atenea, 2012).

2.2 Simulation and Model Transformation

Once we have the cloud application modeled using our DSVL, we are ready to perform simulations and check how the system auto-scales in order to satisfy a required level of throughput. Since the observers are objects included in the system, we simply have to check the value of their attributes after the simulation in order to see the performance measures and how the system has evolved and self-adapted. The whole approach is implemented in the *e-Motions* tool (Rivera et al., 2009a), which runs as an Eclipse plugin and executes the model transformation rules by encoding them as rewriting rules over Maude.

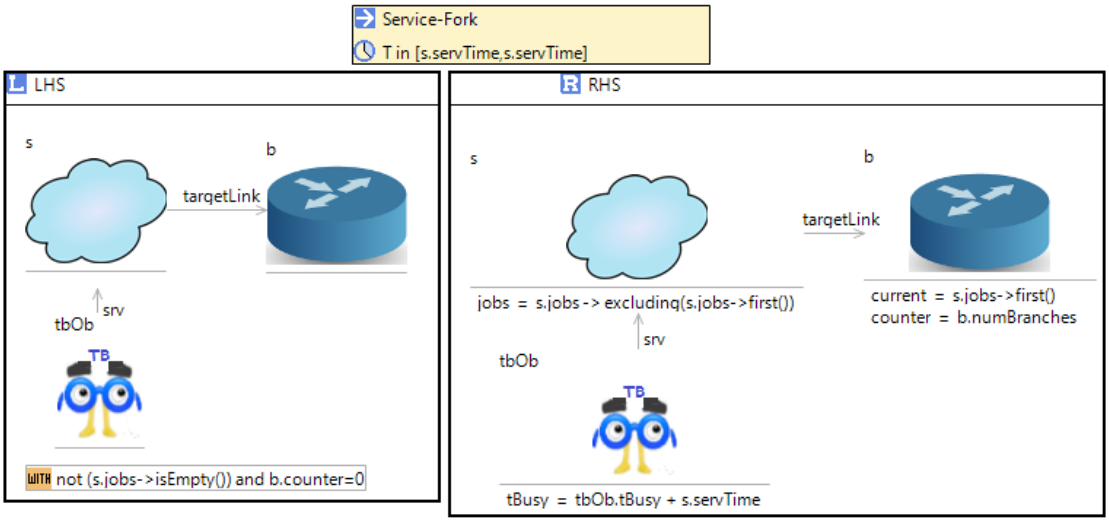
Table 1: Evolution of the throughput and the system.

Iteration	Throughput	Service	Cost
0	0.12	–	12500
1	0.24	s2	16500
2	0.24	s3	20000
3	0.48	s2	24000

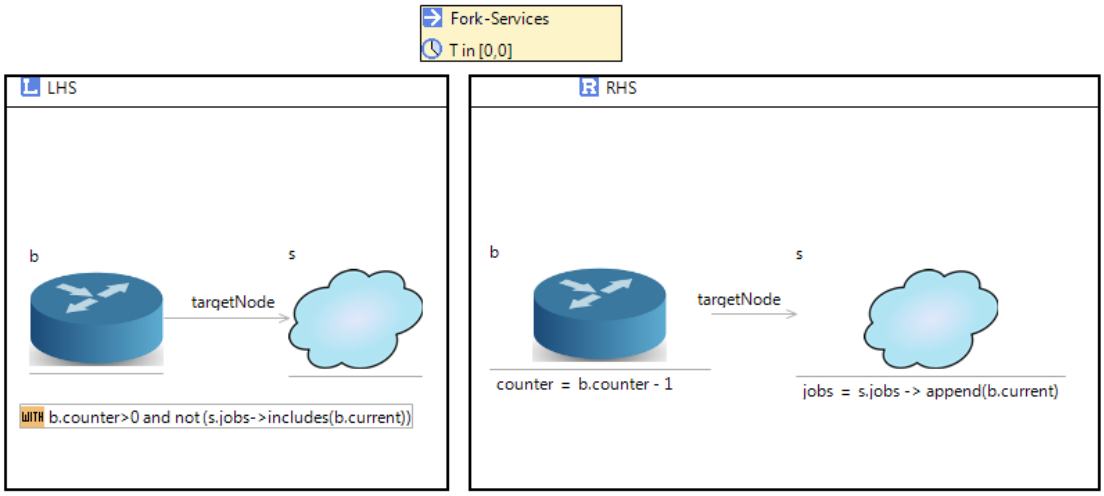
The auto-scaling criteria in our approach is intuitively explained as follows. If the observed throughput is lower than the expected value, then the service with the slower service time is duplicated. If there are more than one service with the same service time, then it duplicates the cheapest.

In our case study (see the model in 2), it was required a minimum throughput of 0.25 jobs per time slot. Table 1 presents, for each iteration in our model, the throughput and cost of the system, and the service that was duplicated from the previous iteration. In each iteration, a fixed number (40 in our case) of packets are sent from the source to the sink, after which the QoS properties are calculated. If the throughput constraint is not satisfied, the system auto-scales and a new iteration is launched automatically.

We can see that the initial cost of the services was 12500. The simulation of the original version of the model (2) obtained a throughput of 0.12. Then the model is transformed so as to duplicate the number of instances of the least performing service (service



(a) Service-Fork rule



(b) Fork-Services rule

Figure 4: Rules for sending a job through a fork link.

s2) in iteration 1. This increased the cost of the whole system to a total of 16500 and the resulting throughput increased to 0.24. After iteration 1, we have two instances of service s2 and one of s3. Consequently, the service time of both is now the same: 4 (recall that the service time in service s1 is 2, see 2, so it is not considered for duplication). However, duplicating service s3 is cheaper. For this reason, s3 is duplicated after iteration 1, so the cost of the system increases to 20000, but the throughput remains as 0.24. The reason why this value remains the same is that services s2 and s3 are connected in series. Thus, if one service is faster than the other, a bottleneck is caused in the latter, and viceversa. So, in order to avoid this local bottleneck, both services should have the same service time. After iteration 3, the system automatically duplicates service s2, achieving a final throughput of 0.49 and with a total cost of 24000.

3 RELATED WORK

In this section, we comment on related work in two dimensions: (i) adaptation of QoS, and (ii) model-driven simulation.

There are several efforts towards QoS adaptation of cloud-based systems (Stantchev and Schröpfer, 2009; Cao et al., 2009; Calheiros et al., 2011). Specifically, in (Stantchev and Schröpfer, 2009) the authors propose an approach to negotiate, monitor and enforce Service-Level-Agreements (SLAs) when the infrastructure is

not owned by the controller of the system. Their work is applicable to cloud and grid computing scenarios. They promote the replication of services as an effective way to boost performance and dependability, but they achieve this via predefined replication strategies. Our approach, however, is able to generate such strategies automatically based on a simulation of the system and its performance.

OPTIMIS (Ana Juan Ferrer et al., 2012) is a holistic approach which enables flexible and dynamic provisioning of cloud services, based on adaptive self-preservation. This mechanism is a key to meet predicted and unforeseen changes in resource requirement. This approach deserves further consideration once the tools are delivered. Our work is less ambitious than theirs but, nonetheless, our contribution towards the simulation and analysis of various QoS parameters, including automatic model transformation to fit the given throughput requirements, will complement other solutions for elastic service provisioning.

As regards model-driven simulation efforts, ARENA (Rockwell Automation, 2011) offers similar analysis tools where the model can be visually simulated in a MDE fashion. However, they use their proprietary notations and therefore it cannot be extended with its own DSVLs. Our approach, based in *e-Motions*, supports the creation of new metamodels and their domain-specific visual notation. Additionally, we provide transformation rules which refine the model so as to fulfill the throughput requirements.

The observers used in our approach are reminiscent of those used in the MARTE (OMG, 2008) specification. The advantage of including these observers into a DSVL (as it is done in our approach) is that we were able to i) make explicit the throughput requirements in the specification, ii) simulate and reason over the model, and iii) automatically transform the model according to the information extracted via these observers. This is not possible with MARTE observers.

4 CONCLUSIONS AND PERSPECTIVES

In this ongoing work, we have presented an approach to tackle automatic service provisioning via service replication and supported by a DSVL. We proposed to include the scalability analysis at the design stage. Our proposal is based on model-driven adaptation mechanism, where various QoS parameters can be simulated and analyzed using the *e-Motions* tool. Furthermore, with the purpose of fulfilling the required throughput, we have made the system to automatically transform by replicating and load-balancing the services which cause the bottleneck.

Our approach has been evaluated with a concrete case study of cloud services. We obtained promising results since the tool was able to automatically refine the initial models so as to fit the throughput requirements.

As regards future work, we are moving on to explicitly include the concept of load-balancer in our model, to analyze in detail its impact on the performance of the system, and to finally implement the resulting model and the transformation rules in the clouds.

In addition, although our approach is to replicate services as a mean to address service provisioning, we perform the scalability analysis at the design stage, and thus we do not address dynamic adaptation. However, we plan to study and compare other strategies such as adaptive self-preservation or dynamic reconfiguration of Virtual Machines.

ACKNOWLEDGEMENTS

This work has been partially supported by the projects TIN2008-05932 (ReSCUE), TIN2008-031087, TIN2011-23795 and TIN2012-35669 (SOFIA), all funded by the Spanish Ministry of Science and Innovation and FEDER, and the project P11-TIC-7659 funded by the Andalusian Government.

REFERENCES

Ana Juan Ferrer et al. (2012). Optimis: A holistic approach to cloud service provisioning. *Future Generation Comp. Syst.*, 28(1):66–77.

Atenea (2012). Modeling of Cloud System Infrastructures. http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions/Cloud.

Calheiros, R., Ranjan, R., and Buyya, R. (2011). Virtual machine provisioning based on analytical performance and QoS in cloud computing environments. In *Proc. of ICPP*, pages 295–304.

Cao, B.-Q., Li, B., and Xia, Q.-M. (2009). A service-oriented qos-assured and multi-agent cloud computing architecture. In Jaatun, M., Zhao, G., and Rong, C., editors, *Cloud Computing*, volume 5931 of *LNCS*, pages 644–649. Springer.

Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of MDA*.

de Lara, J. and Vangheluwe, H. (2008). Translating model simulators to analysis models. In *Proc. of FASE 2008*, number 4961, pages 77–92. Springer.

Expert Group Report, European Commission (2010). The future of cloud computing - opportunities for european

cloud computing beyond 2010. http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf.

OMG (2008). A UML profile for MARTE: Modeling and analyzing real-time and embedded systems. Technical report, OMG.

Rivera, J., Durán, F., and Vallecillo, A. (2009a). A graphical approach for modeling timedependent behavior of DSLs. In *In Proc. of VLHCC'09*.

Rivera, J. E., Durán, F., and Vallecillo, A. (2009b). A graphical approach for modeling time-dependent behavior of dsls. In *Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'09)*, Corvallis, Oregon (US). IEEE Computer Society.

Rivera, J. E., Guerra, E., de Lara, J., and Vallecillo, A. (2008). Analyzing rule-based behavioral semantics of visual modeling languages with Maude. In *Proc. of SLE'08*, number 5452, pages 54–73, Tolouse, France. Springer.

Rivera, J. E., Vallecillo, A., and Durán, F. (To appear in 2009c). Formal specification and analysis of domain specific languages using Maude. *Simulation: Transactions of the Society for Modeling and Simulation International*.

Rockwell Automation (2011). Arena simulation software. Available at: http://www.arenasimulation.com/ Last checked: Dic. 2012.

Stantchev, V. and Schröpfer, C. (2009). Negotiating and enforcing qos and slas in grid and cloud computing. In *Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing*, GPC '09, pages 25–35, Berlin, Heidelberg. Springer-Verlag.

Troya, J., Vallecillo, A., Durán, F., and Zschaler, S. (2013). Model-driven performance analysis of rule-based domain specific visual models. *Information and Software Technology*, 55(1):88 – 110.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18.