Towards the model-based predictive performance analysis of Cloud adaptive systems with e-Motions

Patrícia Araújo de Oliveira, Francisco Durán & Ernesto Pimentel
University of Málaga, Spain
{patricia,duran,ernesto}@lcc.uma.es

We use graph transformation to define an adaptive component model, what allows us to carry on predictive analyses on dynamic architectures through simulations. Specifically, we build on the e-Motions definition of the Palladio component model, and then specify adaptation mechanisms as generic adaptation rules. We illustrate our approach with rules modelling the increase in the number of CPU replicas used by a component, and the distribution of works between processors, reacting, respectively, to saturated queues or response time constraints violations. We evaluate alternative scenarios by analysing their performance, and discuss on its consequences in practice.

1 Introduction

Predictive analysis methods in the design phase offer the possibility of estimating the impact of design decisions, which may help in the accomplishment of operational optimal results, before the deployment of the system, and therefore minimizing the required effort and cost. In fact, having the capability of predicting problems related with performance constraints, scalability issues or reliability risks when the system is being modeled has for long been proven very useful. This kind of predictive information about Quality of Service (QoS) metrics allows the adoption of decisions during the design phase, even before the system has been implemented or deployed, so mitigating the impact of bad designs or decisions. In order to get precise analysis of a system, we need an appropriate model both of the application itself and of the context where it is going to be deployed. The more accurate the model is, the more precise the prediction. The same techniques may also help in the calibration of quality parameters, providing support to estimate optimal values for them.

A number of different approaches have been used to make this kind of analysis, including techniques based on stochastic networks, Petri nets and statistical methods. Unfortunately, current predictive analysis tools do not deal in a satisfactory way with dynamic architectures, which characterize current technologies, such as cloud platforms. The reason of this limitation is that these tools operate on static structures, which cannot be modified along their analysis process. This is the case, for instance, of the Palladio tool [12], one of the currently more successful predictive analysis tools. The Palladio Simulator is widely used in both industry and academia, and can be used to predict QoS properties (performance and reliability) from software architecture models. However, Palladio only supports the modeling of static architectures, with no support for systems that dynamically adapt to context changes, which is the case, for example, in cloud scenarios. Recently, there have been attempts such as those in [13] or [1] to build on Palladio support for dynamic systems. Others tools such as D-Klaper [11], MEDEA [9] or [15] have proposed alternative solutions for performance prediction of dynamic systems. As we explain in Section 6, none of these proposals provides a satisfactory solution to the problem.

We claim that with our approach we can model the dynamic aspects of systems and their environments, and provide accurate predictive information on their execution. Although in this work we present

Submitted to:
PROLE 2017

initial ideas on the approach by focusing on response time, we will present enough details so the reader can gather an idea of the possibilities of our proposal. We believe that the approach may be used for the predictive analysis of typical performance metrics, such as throughput and resource usage, but also others such as reliability, cost or security.

We use graph transformation systems to model systems and their environments, adaptation mechanisms and their analysis. Although users may use graph transformation to express all of this from scratch, using systems such as Groove [10] or e-Motions [18], we believe it would not be a good idea. Instead, we build on top of well-known description languages, such as the language of Palladio for the specification of systems' architectures. We then model adaptation mechanisms using graph transformation, and perform the analysis of the entire systems thus described. In the future we will provide appropriate bridges to be able to specify adaptation mechanisms in already existing languages. For instance, we are considering the use of the SYBL language [7] for the specification of adaptation strategies for cloud systems. In this way, one can combine Palladio descriptions of systems with SYBL descriptions of adaptation mechanisms, and perform its analysis in our analysis tool.

We build on the e-Motions specification of the Palladio Simulator in [16], where given the metamodel of Palladio (the PCM, Palladio Component Metamodel), and its operational semantics given with a set of graph-transformation rules, we can perform predictive analysis of (static) systems of systems specified using the Palladio editors. In other words, we can specify a system in Palladio and feed it into e-Motions, using it as an external tool on which to perform its analysis. Indeed, this e-Motions specification of Palladio is executable, and introduces the possibility of adding new features and capabilities to Palladio through the definition of new e-Motions rules. In this work, we propose using such possibility for representing the dynamic behaviour of self-adaptation mechanisms, such as cloud features related with elasticity. In this way, our approach will allow us a predictive analysis based on simulations for dynamic systems, and in particular for applications deployed on cloud platforms. Among the different methods for analysis, statistical methods offer the possibility of making performance estimations at (relatively) low cost for complex systems, thanks to the expressiveness of the specification languages, and, although they may exhibit certain imprecision, the margins are usually acceptable.

In this work we define transformation rules modelling two scalability strategies usual in cloud environments, and thus showing some of the possibilities for the definition of adaptation and its analysis. Specifically, we have modelled the increase in the number of CPU replicas used by a component, and the distribution of works between processors, reacting, respectively, to saturated queues or response time constraints violations. Indeed, by analysing these, and possibly other alternative adaptation mechanisms, one may infer which will be the best adaptation mechanisms in specific situations, and which the best calibration of their parameters. We evaluate alternative scenarios by analysing their performance, and discuss on its consequences in practice. As currently done for static-architecture systems, this facilities will allow detecting errors in configurations and wrong decisions, and providing optimal parameters for them, without requiring expensive on-site experimentation.

The remainder of this paper is structured as follows. Section 2 provides some background on the Palladio and e-Motions frameworks, also introducing a motivating example to illustrate our approach. Section 3 presents the adaptation rules defined in e-Motions and shows how they are woven with the Palladio system to enrich its capabilities to analyse dynamic systems. Section 4 presents some analysis results and a discussion on them. Section 5 presents some related work. We wrap up with some conclusions and future work in Section 6.

2 Palladio and e-Motions

This work is based on the Model-Driven Engineering (MDE) frameworks Palladio and e-Motions. We use Palladio [12] for the system modeling, and e-Motions [18] to specify Palladio's operational semantics and the adaptation of Palladio systems over time, what allows us to simulate and analyse Palladio-like adaptive systems. In this section, we provide some background on both frameworks to ground the discussion that will follow. We introduce our motivation example to illustrate the main ideas of Palladio, and illustrate the use of e-Motions on its definition of Palladio.

2.1 The Palladio Component Model

Palladio [12] is a mature modeling language for modeling component-based and service-oriented software systems, with a focus on the prediction of extra-functional properties of systems based on their constituting components. Palladio relies on model-driven software development techniques for its definition, and uses automated transformations into different prediction models or simulation systems. The meta-model of Palladio—its language description—is provided the Palladio Component Model (PCM) [3]. The semantics of the models, and of the non-functional properties to be analysed, is encapsulated in the respective transformations.

Palladio models are composed of four different artefacts, provided by corresponding developer roles involved in a Component-Based Software Engineering development process [4]: component developers provide component specifications, software architects provide assembly models, system developers provide allocation models, and business domain experts provide usage models. Let us illustrate these artefacts on a very simple case study, originally published in [1], that specifies a load balancer.

Component developers specify and implement parametric descriptions of components and their behaviour. Figure 1 shows the Palladio Component repository for our example. It depicts three components and corresponding interfaces: ApplicationServer implements IApplicationServer, ApplicationServer2 implements IServer2, and LoadBalancer implements ILoadBalancer. There are two Requires relations from the LoadBalancer component, one to the IApplicationServer interface and another to IServer2. Both IServer2 and IApplicationServer offer the processRequest() operation. As we will see in the assembly model, the front-end node, containing the LoadBalancer component, will invoke the processRequest() operation provided by these two interfaces.

Components' services are described with service effect specifications (SEFF), which abstractly model the externally visible behaviour of a service with resource demands and calls to required services. Figure 2 shows the SEFFs of these three components. Figure 2(a) shows the processRequest SEFF that models the behaviour of the servers components. Such processing is very simple in our example, it just consists in an internal action that consumes 300 units of CPU (CPU cycles). Figure 2(b) shows balancer SEFF and indicates that the control flow in the LoadBalancer component may branch into one of two flows, each of them with an external call action to a different node. Each branch can be associated with a particular branch probability to indicate the likelihood of a particular branch being taken. This is the kind of information required to perform execution-time analysis on the component's behaviour as is standard in software performance engineering (see, e.g., [20]).

Software architects assemble components from the repository to build applications. Figure 3 shows how the services of the LoadBalancer, ApplicationServer and ApplicationServer2 components are composed. The biggest square surrounding the boxes represents the entire environment. For each 'provides' relation in the repository model, a provided role is created for the container containing such component.

System deployers model the resource environment and the allocation of components from the assem-

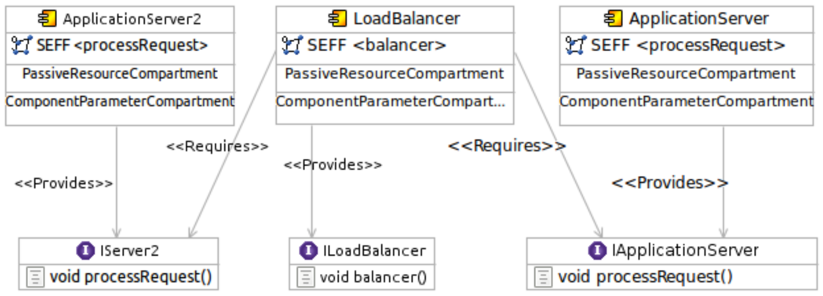
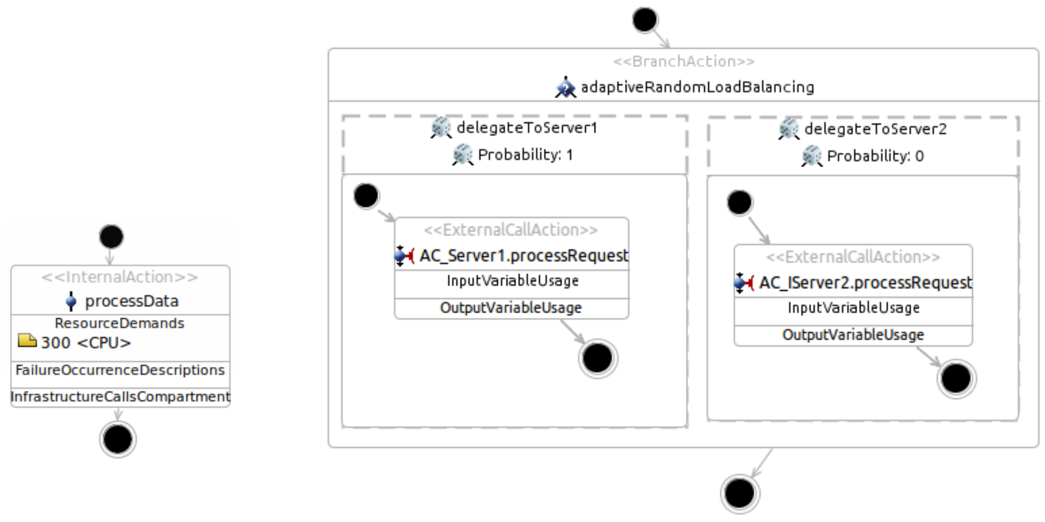


Figure 1: Components repository



(a) Process Request SEFF (b) Balancer SEFF

Figure 2: Components' SEFFs

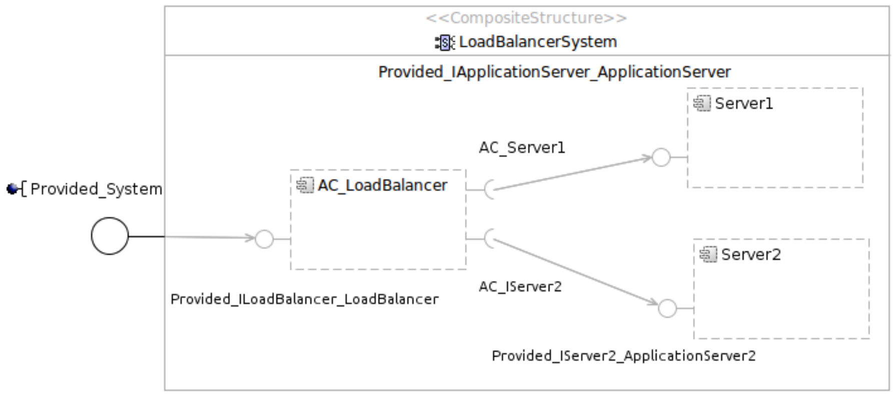

<<CompositeStructure>>
LoadBalancerSystem
Provided_IApplicationServer_ApplicationServer
Server1
AC_Server1
Provided_System
AC_LoadBalancer
Server2
AC_IServer2
Provided_ILoadBalancer_LoadBalancer
Provided_IServer2_ApplicationServer2

Figure 3: Assembly model

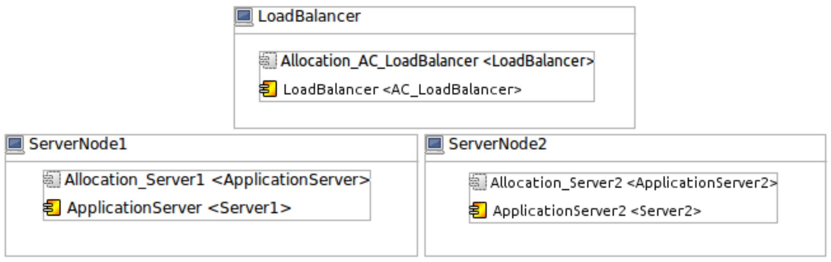

LoadBalancer
Allocation_AC_LoadBalancer <LoadBalancer>
LoadBalancer <AC_LoadBalancer>
ServerNode1
Allocation_Server1 <ApplicationServer>
ApplicationServer <Server1>
ServerNode2
Allocation_Server2 <ApplicationServer2>
ApplicationServer2 <Server2>

Figure 4: Allocation Model

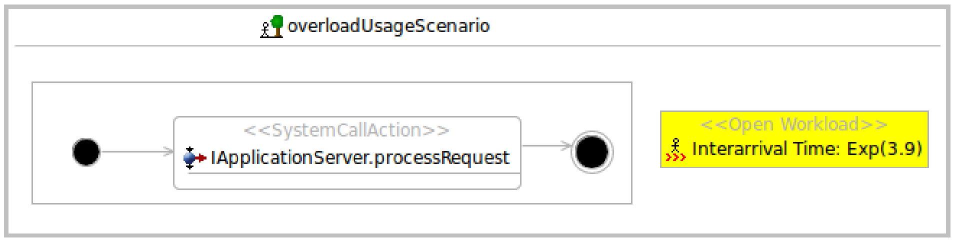

overloadUsageScenario
<<SystemCallAction>>
IApplicationServer.processRequest
<<Open Workload>>
Interarrival Time: Exp(3.9)

Figure 5: Usage Model

bly model to different resources of the resource environment. Figure 4 shows the allocation model for our case study, where we can see how each of the components is allocated in a different node.

Domain experts specify a systems usage in terms of workload (i.e., the number of concurrent users), user behaviour (i.e., the control flow of user system calls), and parameters (i.e., abstract characterisations of the parameter instances). Given the usage model definition in Figure 5, in our example, tasks will arrive following an exponential probability distribution with rate parameter 3.9 time units (Exp(3.9)), which means that tasks will arrive every ≈ 0.256 time units in average.

In the specification of the Palladio models, the parameters specified are fixed, and cannot be changed along executions. For instance, the arrival rate for work arrivals has been established in Exp(3.9), the demand of CPU for the processing of the internal action in the servers is set to 300, and the number of CPU replicas in each server is 1, the CPU processing rate is 1000. With our proposal, these parameters can be changed, as the architecture or allocation of components, at simulation run time.

2.2 The e-Motions System

The e-Motions system [18] is a graphical framework that supports the specification, simulation, and formal analysis of real-time systems. It provides a way to graphically specify the dynamic behaviour of DSLs using their concrete syntax, making this task very intuitive. The abstract syntax of a DSL is specified as an Ecore metamodel, which defines all relevant concepts and their relations in the language. Its concrete syntax is given by a GCS (Graphical Concrete Syntax) model, which attaches an image to each language concept. Then, its behaviour is specified with (graphical) in-place model transformations.

e-Motions provides a model of time, supporting features like duration, periodicity, etc., and mechanisms to state action properties. From a DSL definition e-Motions generates an executable Maude [6] specification which can be used for simulation and analysis.

The in-place model transformations used to specify the behaviour of systems are defined by rules, each of which represents a possible *action* of the system. These rules are of the form $[\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$, where LHS (left-hand side), NAC (negative application conditions) and RHS (right-hand side) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the conditions for the rule to be applied, whereas the RHS represents the effect of the corresponding action if its conditions are satisfied. Thus, the action described in RHS can be applied, i.e., a rule can be triggered, if a match of the LHS is found in the model and none of its NAC patterns occurs. A LHS may also have positive conditions, which are expressed, as any expression in the RHS, using OCL (Object Constraint Language). If several matches are found, one of them is non-deterministically chosen and applied, giving place to a new model where the matching objects are substituted by the appropriate instantiation of its RHS pattern. The transformation of the model proceeds by applying the rules on sub-models of it in a non-deterministic order, until no further transformation rule is applicable.

In e-Motions, a DSL is defined by providing a metamodel, which defines its syntax, and a set of graph transformation rules, which define its behaviour. These DSL definitions can then be used for simulation and analysis. For instance, we can perform reachability analysis, model checking, and statistical model checking of the DSLs defined using e-Motions (see [19] and [8]).

Palladio is a DSL, and has been specified in [16] using the visual facilities of the e-Motions system [18]. As for any DSL, the e-Motions definition of Palladio includes its abstract syntax (the PCM), its concrete syntax, and its behaviour. Its concrete syntax is provided in e-Motions by a GCS model in which each concept in the abstract syntax being defined is linked to an image. These images are used to graphically represent Palladio models in e-Motions, which uses the same images that the PCM Bench to represent these concepts. Its behaviour is defined by graph transformation rules, thus becoming explicit at a very high level of abstraction.

The operational semantics of Palladio, i.e., its behaviour, is given as a token-based execution model, where each work that enters the system is modelled as a token that moves around the different services of the system, and inside each service description, around the different tasks (start, stop, branch, loop, etc.) in its SEFF descriptions. Each of the actions that may occur in the system are then specified by e-Motions transformation rules. For example, Figure 6 shows the e-Motions rule that specifies the execution of an InternalAction, like the one shown in Figure 2(a). This rule represents a generic execution of an internal activity by a component service, possibly using some resources, like HDD or CPU. In Palladio, these executions present a high-level of abstraction, and the resource demands are expressed as stochastic expressions. In the e-Motions rule, the LHS indicates if there is an internal action not completed in the system, the RHS will execute in time rTime. The duration of this action depends on the corresponding Palladio elements, specifically on the Parameter Resources Demanded (PRD) and on the Processing Resource Specified (PRS). For example, a PRS may have an initial specification of 300 work units per second (PRS.processingRate) and 1 CPU replica (PRS.numberOfReplicas). Tokens are served following an FCFS strategy by using a queue associated to each resource type. Only the first PRS.numberOfReplicas tokens in the queue PRT.queue get to be executed. Once an internal action is executed, its token is removed from the queue (PRT.queue→excluding(t)), and marked as completed, being then ‘moved’ to the following task in the service description.

The behaviour of Palladio’s core features has been specified by some 30 time-aware in-place transformation rules, corresponding to the possible model changes. Once the whole DSL has been defined, and given a model as initial state, it may be simulated by applying the rules describing its behaviour. However, this model does not collect information on non-functional properties (NFPs), and therefore is not ready for performance analysis. For this, an observer mechanism [21] is used to measure the non-functional properties of each of the components in the system. Since the PCM is used as metamodel in the e-Motions definition of Palladio, models developed using the Palladio Bench can be directly

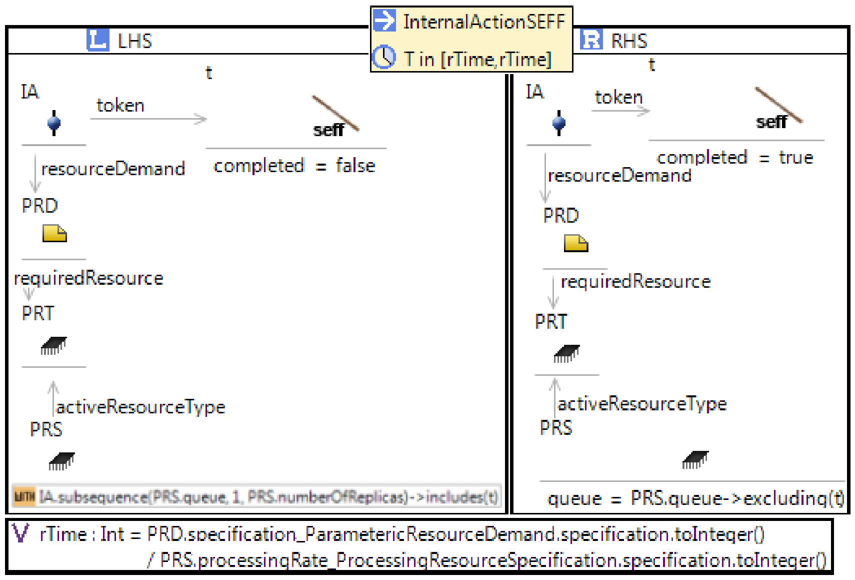

InternalActionSEFF
T in [rTime,rTime]
LHS
RHS
IA
token
t
seff
completed = false
completed = true
resourceDemand
PRD
requiredResource
PRT
activeResourceType
PRS
IA.subsequence(PRS.queue, 1, PRS.numberOfReplicas)->includes(t)
queue = PRS.queue->excluding(t)
rTime : Int = PRD.specification_ParametericResourceDemand.specification.toInteger()
/ PRS.processingRate_ProcessingResourceSpecification.specification.toInteger()

Figure 6: Internal Action SEFF rule

loaded into the e-Motions tool. The complete e-Motions definition of the Palladio DSL is available at `http://atenea.lcc.uma.es/e-Motions`.

3 Palladio Model Analysis with Adaptation Mechanisms in e-Motions

The e-Motions specification of the Palladio DSL is an executable model, with which we can experiment and develop new features. Specifically, we have modelled different types of adaptation mechanisms as transformation rules, representing possible changes in the behaviour of systems and their environments. Having adaptation rules as part of the e-Motions specification of Palladio and the system under analysis, enables the predictive analysis of dynamic systems, and specifically Cloud systems.

The different adaptation mechanisms available in cloud systems are classified in [14]. Some adaptation dimensions are identified for cloud systems, and they correspond to the resources that will be adapted, the adaptation objectives, the adaptation techniques used, whether the adaptation is reactive or proactive, the architecture of the adaptation engine, and the managed infrastructure. Specifically, they consider three types of resources: Compute (CPU, CPU cache and primary storage memory); Storage (non-volatile secondary storage memory); and Network (network cards and other infrastructure that allow components to connect into servers). These resources may be considered both for real and virtual machines, and be continuously monitored to provide information to be taken into account in the adaptation actions. Indeed, the workloads information and the use of resources lead to decisions to increase or decrease their allocation.

Since the Palladio models the e-Motions tool operates on represent entire system states, we can specify system adaptations in exactly the same way we model their evolution. To illustrate this idea, we will show adaptation rules for two of the above adaptation mechanisms. Specifically, we present here adaptation rules for CPU scale up and adjustment of parameters at the infrastructure level. Figures 7 and **??** show e-Motions rules modelling possible instances of these two adaptation mechanisms. As for other rules in eMotions definitions, rules' LHSs model the state of a sub-model on which the rule

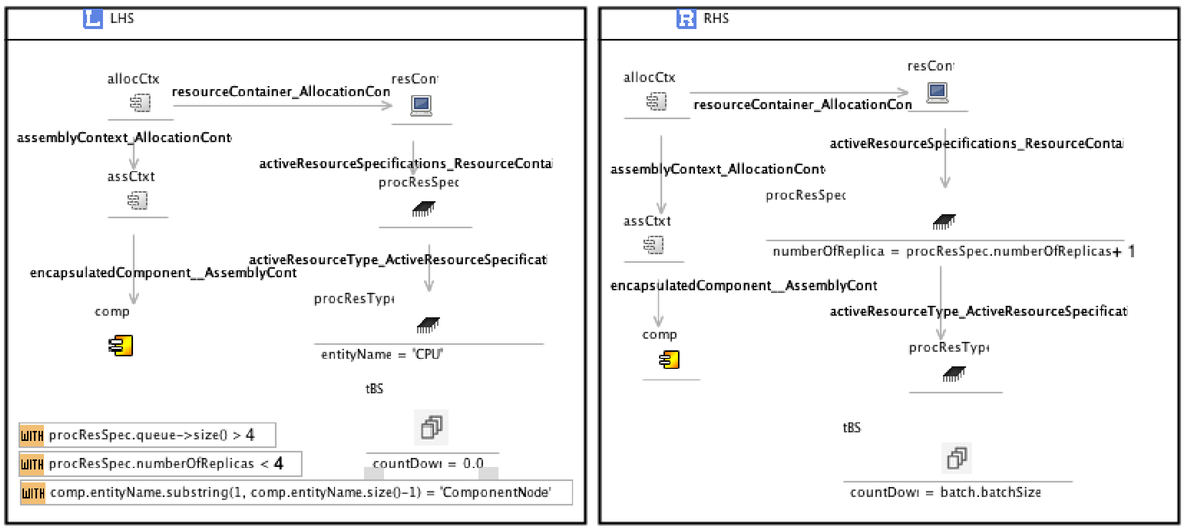

LHS
allocCtx
resourceContainer_AllocationCon
resCon
assemblyContext_AllocationCont
assCtxt
activeResourceSpecifications_ResourceConta
procResSpec
encapsulatedComponent__AssemblyCont
activeResourceType_ActiveResourceSpecificat
procResType
comp
entityName = 'CPU'
tBS
WITH procResSpec.queue->size() > 4
WITH procResSpec.numberOfReplicas < 4
countDown = 0.0
WITH comp.entityName.substring(1, comp.entityName.size()-1) = 'ComponentNode'
RHS
allocCtx
resCon
resourceContainer_AllocationCon
activeResourceSpecifications_ResourceConta
assemblyContext_AllocationCont
procResSpec
assCtxt
numberOfReplica = procResSpec.numberOfReplicas+ 1
encapsulatedComponent__AssemblyCont
activeResourceType_ActiveResourceSpecificat
comp
procResType
tBS
countDown = batch.batchSize

Figure 7: Increase CPU Replicas Rule

will take place; RHSs represent the effects of the application of such a rule on such a sub-model. For adaptation rules, we need a reason to adapt, that will trigger the application of the rule, and an effect, a change in parameters or reconfiguration of the system. To avoid continuous adaptations of the system, a minimum time elapse is waited until a new adaptation is attempted, letting the system to stabilize after each operation. For that, there is a batch object (tBS) which attributes countDown and batchSize. When the count down is zero, the rule can be fired, and when it does the batch is reset. For our example, batch size is 2.

Figure 7 models the increase in the number of replicas of CPU associated to a particular container node (rContainer). Each resource has associated a queue of works waiting to use such a resource. With this rule, we model an increase of the number of CPU replicas when the number of works in its associated queue goes over some threshold. Note that an allocation context (allocCtx) has some component (comp) deployed on it, and some resources associated (procResSpec), of some type (procResType). For the specific sub-model in the LHS of the rule, if the resource's queue has at least 4 works (procResSpec.queue -> size() $>$ 4), the number of CPU replicas has not reached its limit (procResSpec.numberOfReplicas $<$ 4), and the component is one of the nodes, then the number of replicas gets increased (procResSpec.numberOfReplicas$+$1).

Figure 8 shows the rule that changes the probability branch of a LoadBalancer component (see Figure 2(b)). Assuming that the system models the distribution of load between a local server and a remote server which use we want to minimized, the system is initially set with probabilities 1 and 0, changing the distribution of load with the LoadBalancer rule if the expected response time value is not reached. If the batch countdown reaches zero, and the average response time of the last batch size elements is greater than the threshold (1), then the probabilities assigned to the branches are changed. Note that the probabilities are progressively changed until they become equal.

With these rules, we have tried to show the modelling of different adaptation mechanism, triggered by different criteria. Any adaptation mechanism you can express with transformation rules, which can moreover be combined, synchronized and iterated, can be modelled. Some of these adaptation mechanisms may dynamically become in conflict. For example, a failure in the satisfaction of a response

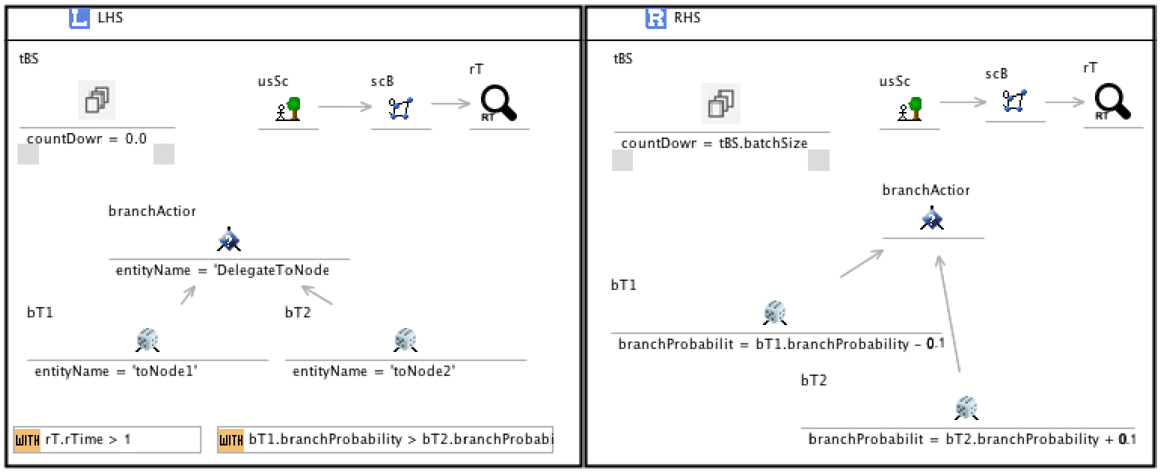

LHS
tBS
countDowr = 0.0
usSc
scB
rT
branchActior
entityName = 'DelegateToNode
bT1
entityName = 'toNode1'
bT2
entityName = 'toNode2'
WITH rT.rTime > 1
WITH bT1.branchProbability > bT2.branchProbabi
RHS
tBS
countDowr = tBS.batchSize
usSc
scB
rT
branchActior
bT1
branchProbabilit = bT1.branchProbability - 0.1
bT2
branchProbabilit = bT2.branchProbability + 0.1

Figure 8: LoadBalancer Rule

time requirement may be handled by scaling up or scaling down, what requires some way of managing the adaptation strategies. Given our approach, managing adaptation strategies means managing the e-Motions adaptation rules. There are several alternatives for this, going from a distributed control, with intelligence in each of the rules, to a centralized approach, in which a central controller gathers all the information on the environment and decides how to proceed. We have not provided any such mechanism here, and therefore, as we will see in the next section, when several rules are applicable, the rewrite engine picks one.

4 Analysis

In this section, we show that despite the simplicity of the adaptation rules implemented in the previous sections, we may already draw some conclusions on it usefulness, and make some estimations on the parameters to use. The Palladio Model example shown in Figures 1-5 (Section 2) is our initial model. As said in Section 2.1, the model was developed in the Palladio Benchmark and then loaded into the e-Motions tool.

Despite its simplicity, there are already a number of parameters we may need to provided: the internal operations performed require 300 CPU cycles; CPUs used were both assumed of capacity 1000 operations per time unit; the branches controlling the load balancer are initialized with values 1 and 0 (all works are executed in the local server). Then, there are other parameters with control the adaptation mechanisms: the batch size (set to 2), the maximum number of CPU units per contained (4), the thresholds for the number of elements in the queues (4) and the expected response time, which we assume is the threshold for firing the LoadBalancer.

Given these parameters, the analysis of the system shows a continuously growing sequence of response times for the works handled by the system. The capacity of a CPU is not enough to handle all the works that arrive to the system, and the resource's queue gets an increasing number of works. Note that the load balancer sends all works to one of the servers, which has one single CPU. This trend may be observed in the chart in Figure **??**. The x-axis represents the total time of the simulation and the y-axis the response times for the works that leave the system. We expect that We expect that with both

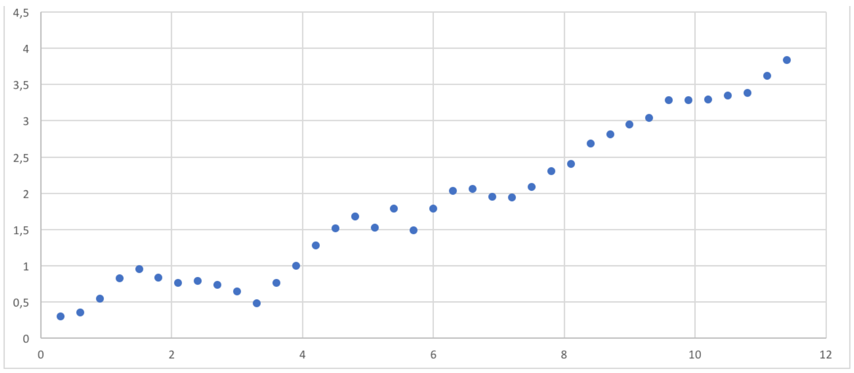


Figure 9: Response times with no adaptation

adaptations the response time will decrease in comparison to the response time of a simulation without adaptation.

The chart in Figure 10 shows the response times for the system with the LoadBalancer rule activated. As soon as the average response time goes over 1, the rule gets applied for the first time in time 2,7. After that, since the response time is always over 1, the rule is applied once every 2 time units, until the probabilities of the branches become equal, and the response time continue growing.

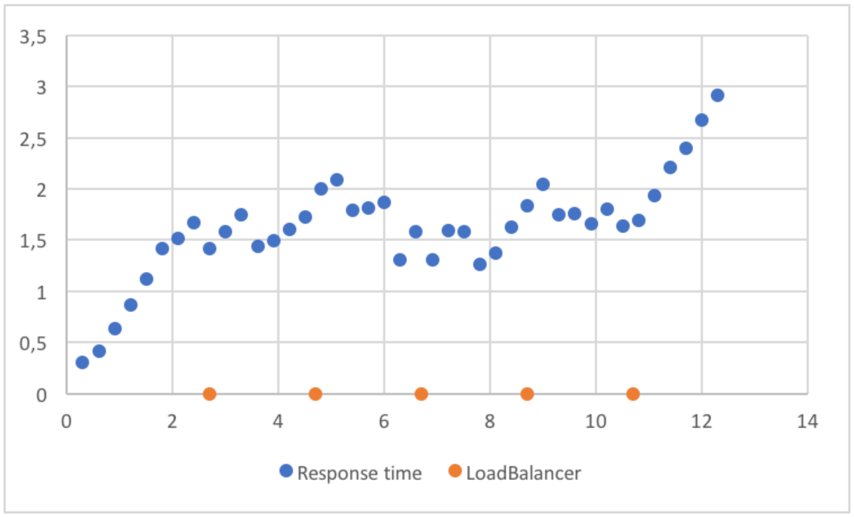


Figure 10: Adaptation with the LoadBalancer, batch size 2

The chart in Figure 11 shows the response times for the system with the QueueIncCPUReplicas rule activated. The rules gets applied at times 2 and 12,34. After a quick grow in the response times at the begining of the simulations, the adaptation is applied at time 2, producing a significant reduction in the response times of the rest of the works. Indeed, the addition of a second CPU make the system to stabilize with response times under 0.6. The flow of works into the system gets higher after time 11, producing an increased in the number of works in the queues, and leading to a second application of the rule in time 12,34. This suppose an excess of computational power that results in a flat line after time 15 with response times of 0.6.

Given the observation of a possible excess of computational power, and to take advantage of our

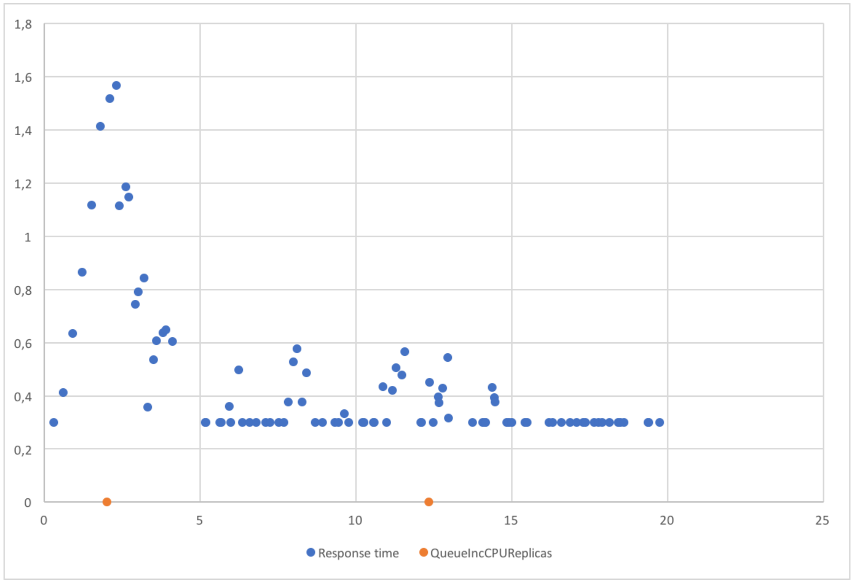


Figure 11: Adaptation with the QueueIncCPUReplicas, batch size 2

capacity of adaptation according to the processing needs, we may consider the option of reducing costs by using slower, and hence cheaper, machines. Figure 12 shows the chart for the response times obtained for a system in which CPU have speeds of 500 cycles per time unit. In this case, both the LoadBalancer and QueueIncCPUReplicas are activated. In this case, the response times for the first works arriving the system get deteriorated quite fast, reaching a peak of 2.5 time units despite the application of adaptation rules. Indeed, the first application of an adaptation rule takes place as soon as possible, at time 2, when the batch's countdown reaches 0. The QueueIncCPUReplicas rules gets applied two more times, at times 4 and 6, reaching the maximum of 4 replicas for the the local server, which is still getting all the works. The LoadBalancer rules gets applied every 2 times units until the branches get equal probabilities. Note that the values used for firing the rule are the average of the works in the last batch.

The reason why the LoadBalancer rule is applied once the QueueIncCPUReplicas rule cannot be executed is completely accidental. The rewrite engine picks one rule between all the activated ones. However, once one adaptation rule is executed, the batch countdown is reset, thus preventing the execution of further adaptation rules until the next batch is completed.

5 Related Work

In this section, we discuss other approaches that use predictive strategies, most of them for performance prediction at runtime or at design time.

Huber et al. propose in [13] a DSL to describe the behaviour of self-adaptive systems based on strategies, tactics and actions. This work is part of the Descartes project, which uses Palladio PCM for their design time phases. However, they focus on runtime performance analysis, not in predictive analysis of applications at design time.

SimuLizar [1] is an extension of Palladio for the performance analysis of self-adaptive systems at design time. However, the simulation scope is limited to only a set of rules that are triggered between the static environment models, which prevents from testing all possible reachable states of the systems.

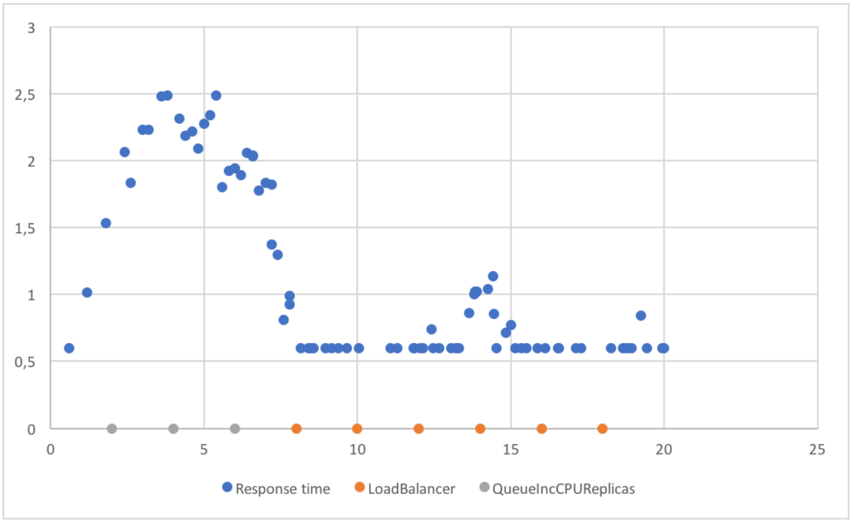


Figure 12: Adaptation with both adaptation rules, batch size 2, CPU capacity 500

D-Klaper [11] is a tool for model-driven performance engineering which can be applied to self-adaptive systems. It uses an intermediate language to provide software design models, which can then be analyzed. However, the D-Klaper language does not support the modeling of adaptation rules, nor the transformation of input models.

MEDEA [9] is an approach that proposes the performance prediction at the beginning of its life cycle for this, modeled the workloads with the resource consumption, capturing the CPU, memory and disks, and this is used to generate executable code for real hardware and middleware deployments. The results of the executions are presented to the expert through specific context views that indicate whether the design meets the performance requirements.

Johnsen et al. present in [15] an approach model-based prediction to compare the effect, in terms of performance and accumulated cost, of selecting different instance types for virtual servers from Amazon Web Services (AWS), for this their used a highly configurable modeling framework for applications running on Apache YARN, the ABS-YARN, which using the executable semantics of Real-Time ABS, defined in Maude, as a simulation tool. However, they are limited to the environment model and do not simulate the application model into environment.

6 Conclusions and Future Work

We have presented two adaptations mechanisms by providing appropriate adaptation rules as graph transformation rules. We used a simple case study to apply our adaptation rules developed in e-Motions to change the amount of resources used during the operation of the system depending on its state. By using a preliminary adaptation controller distributed in the adaptation rules, we were able to perform simulation-based predictive analysis of adaptive systems.

For our case study we only observed the response time of the system, but we were able to operate simulations and perform predictive analysis taking into account different adaptation mechanisms, showing the feasibility of the approach. So far, we have been able to model the dynamic adaptation of the system in accordance to its continuous monitoring by creating rules for scale up CPU and load balancing. As future work we intend to model different workflows, variable usage, the scale in/out nodes, the scale up/down resources capacity (specifically CPU), power on/off inactive resources, and commute some operations of the application.

Building on the knowledge we have gathered so far, we will specify other mechanisms of adaptation available for cloud systems in more ambitious case studies. We will consider other QoS metrics in addition to response time, including not only performance metrics, such as throughput or resource usage, but also others related to feasibility, costs, and security.

To perform the analysis of a dynamic system, it is necessary to consider their capacity to process and manage different workflows, react through variations of the usage, and to carry on the necessary changes when components have assigned different workloads. Following standard techniques, we will model workloads based on real uses of systems, and will use this information to perform our simulations.

We will evaluate our proposal modeling real applications running in real cloud environments, and will verify that the results produced by our predictive analysis match the actual behavior of the real system executed in the cloud.

With this work, we try to offer the techniques and tools to allow the modeling of self-adaptive systems, and specifically cloud infrastructure, and their analysis, so that a better estimation of the satisfaction of the requirements of systems can be carried on, supporting a better selection of resources and a better calibration of the operational parameters.

The prediction of QoS metrics is one of the most relevant issues when gathering knowledge of applications and their environments, compared to other solutions presented in the literature [5]. However, existing prediction methods do not consider specific cloud metrics and, therefore, they are not capable of managing other particular cloud features, such as self-provisioning on demand, measured usage, network access, resource pooling, and elasticity. Properties related to scalability, elasticity and efficiency are essential to achieve a dynamic adaptation in a cloud scenario, specifically for resource allocation and pay-per-use. Thus, we need to take into account these new metrics [2], and also a taxonomy of different sources of uncertainty present in the models of self-adaptive systems and the different ways of managing them [17].

References

[1] Matthias Becker, Steffen Becker & Joachim Meyer (2013): *SimuLizar: Design-Time Modeling and Performance Analysis of Self-Adaptive Systems.* Software Engineering 213, pp. 71–84.

[2] Matthias Becker, Sebastian Lehrig & Steffen Becker (2015): *Systematically deriving quality metrics for cloud computing systems.* In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, ACM, pp. 169–174.

[3] Steffen Becker, Heiko Koziolek & Ralf Reussner (2007): *Model-Based Performance Prediction with the Palladio Component Model.* In: Proc. 6th Int'l Workshop on Software and Performance (WOSP'07), ACM.

[4] Steffen Becker, Heiko Koziolek & Ralf Reussner (2009): *The Palladio component model for model-driven performance prediction.* Journal of Systems and Software 82(1), pp. 3 – 22.

[5] John Chinneck, Marin Litoiu & Murray Woodside (2014): *Real-time Multi-cloud Management Needs Application Awareness.* In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ICPE '14, ACM, pp. 293–296.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn L. Talcott (2007): *All About Maude.* *LNCS* 4350, Springer.

[7] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong & Schahram Dustdar (2013): *Sybl: An extensible language for controlling elasticity in cloud applications.* In: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, IEEE, pp. 112–119.

[8] Francisco Durán, Antonio Moreno-Delgado & José M. Álvarez-Palomo (2016): *Statistical Model Checking of e-Motions Domain-Specific Modeling Languages*. In Perdita Stevens & Andrzej Wasowski, editors: *19th International Conference Fundamental Approaches to Software Engineering (FASE)*, *Lecture Notes in Computer Science* 9633, Springer, pp. 305–322.

[9] Katrina Falkner, Claudia Szabo & Vanea Chiprianov (2016): *Model-driven performance prediction of systems of systems*. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ACM, pp. 44–44.

[10] A. H. Ghamarian, M. J. de Mol, A. Rensink, E. Zambon & M. V. Zimakova (2012): *Modelling and analysis using GROOVE*. *IJSTTT* 14(1), pp. 15–40.

[11] Vincenzo Grassi, Raffaela Mirandola & Enrico Randazzo (2009): *Model-driven assessment of QoS-aware self-adaptation*. In: *Software Engineering for Self-Adaptive Systems*, Springer, pp. 201–222.

[12] Jens Happe, Heiko Koziolek & Ralf Reussner (2011): *Facilitating Performance Predictions Using Software Components*. *IEEE Software* 28(3), pp. 27–33, doi:10.1109/MS.2011.25.

[13] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig & Samuel Kounev (2012): *S/T/A: Meta-modeling run-time adaptation in component-based system architectures*. In: *e-Business Engineering (ICEBE), 2012 IEEE Ninth International Conference on*, IEEE, pp. 70–77.

[14] Abdul R Hummaida, Norman W Paton & Rizos Sakellariou (2016): *Adaptation in cloud resource configuration: a survey*. *Journal of Cloud Computing* 5(1), pp. 1–16.

[15] Einar Broch Johnsen, Jia-Chun Lin & Ingrid Chieh Yu (2016): *Comparing AWS Deployments Using Model-Based Predictions*. In Tiziana Margaria & Bernhard Steffen, editors: *Proceedings 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, *Lecture Notes in Computer Science* 9953, Springer, pp. 482–496.

[16] Antonio Moreno-Delgado, Francisco Durán, Steffen Zschaler & Javier Troya (2014): *Modular DSLs for flexible analysis: An e-Motions reimplementation of Palladio*. In: *European Conference on Modelling Foundations and Applications (ECMFA)*, Springer, pp. 132–147.

[17] Diego Pérez-Palacín & Raffaela Mirandola (2014): *Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation*. In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, ACM, pp. 3–14.

[18] Jose E Rivera, Francisco Durán & Antonio Vallecillo (2009): *A graphical approach for modeling time-dependent behavior of DSLs*. In: *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, pp. 51–55.

[19] José Eduardo Rivera, Francisco Durán & Antonio Vallecillo (2009): *Formal Specification and Analysis of Domain Specific Models Using Maude*. *Simulation* 85(11-12), pp. 778–792.

[20] Connie U. Smith & Lloyd G. Williams (2002): *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Object-Technology Series, Addison-Wesley.

[21] Javier Troya, Antonio Vallecillo, Francisco Durán & Steffen Zschaler (2013): *Model-driven performance analysis of rule-based domain specific visual models*. *Information & Software Technology* 55(1), pp. 88–110.