Towards the predictive analysis of cloud systems with e-Motions

Patrícia de Oliveira, Antonio Moreno-Delgado,
Francisco Durán, and Ernesto Pimentel

University of Málaga, Spain
{patricia,amoreno,duran,ernesto}@lcc.uma.es

Abstract. Current methods for the predictive analysis of software systems are not directly applicable on self-adaptive systems as cloud systems, mainly due to their complexity and dynamism. To tackle the difficulties to handle the dynamic changes in the systems and their environments, we propose using graph transformation to define an adaptive component model and analysis tools for it, what allows us to carry on such analyses on dynamic architectures. Specifically, we use the e-Motions system to define the Palladio component model, and simulation-based analysis tools for it. Adaptation mechanisms are then specified as generic adaptation rules. This setting will allow us to study different mechanisms for the management of dynamic systems and their adaptation mechanisms, and different QoS metrics to be considered in a dynamic environment.

Keywords: Cloud Systems; Predictive analysis; Graph-Transformation Systems

1 Introduction

Performance analysis is becoming one of the most important processes for the development of information systems. In fact, having the capability of predicting problems related with performance constraints, scalability issues or reliability risks when the system is being modeled is highly appreciated. 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. 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.

On the other hand, Cloud Computing [1] has experienced a high growth in the last few years, and the development of cloud applications is now presenting new challenges. In this context, the Cloud can be considered as an environment providing different and heterogeneous services which can be combined and reused to build new applications. This opens up countless possibilities, but also includes a very high level of uncertainty. In fact, the intrinsic dynamism of the Cloud has immediate consequences on the QoS of these applications. The physical infrastructure (computational resources and network capabilities) on which applications are executed directly influence their non-functional requirements, and therefore it also influence on their QoS.

Currently, cloud providers offer their services guaranteeing certain QoS levels by means of Service Level Agreements (SLAs), which can be seen as a unilateral contract from the provider to the customer. As a consequence, cloud users typically choose the most popular provider, even when it is not the most adequate alternative for their applications. Thus, having the possibility of carrying out a rigorous analysis and predicting, at design time, the QoS offered by different providers could be crucial to make a convenient selection among the wide existing vendors offer. Moreover, it may lead to a radical change in the mechanisms for cloud applications management, with effective options to improve the QoS of the applications while reducing costs. Thus, predictive analysis may provide information about the behaviour of an application even before being deployed on a given environment, and this knowledge may be essential to achieve an actual and efficient deployment, and a rapid adaptation to workload changes [6, 15].

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. Among these, 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 acceptable. Unfortunately, current predictive analysis tools do not deal in a satisfactory way with dynamic architectures, which characterize cloud platforms. This is the case of Palladio Architecture Simulator [11], a predictive software analysis which predicts QoS properties (performance and reliability) from software architecture models, and widely used in both industry and academia. Palladio only supports the modeling of static architectures, thus providing no support for systems which dynamically adapt to context changes, which is the case in cloud scenarios.

In order to overcome the limitations previously mentioned of current performance analysis tools, we propose to extend the approach presented in [14], where authors use the e-Motions [16] system to implement the Palladio behavior. This specification is executable and introduces the possibility of adding new features and capabilities to Palladio through the definition of new e-Motions rules representing the dynamic behaviour of self-adaptation mechanisms, and in particular, cloud features related with elasticity (scalability, QoS, and cost). 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.

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 [6]. 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 [3], and also a taxonomy of different sources of uncertainty present in the models of self-adaptive systems and the different ways of managing them [15].

The proposed approach is based on the definition of a number of transformation rules, modeling different mechanisms of cloud policies for elasticity, where different QoS metrics are considered in a dynamic environment. In particular, we focus on two scalability strategies usual in cloud environments, such as increasing CPU replicas (resource provisioning) and load balancing (reacting to workloads).

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. We wrap up with some conclusions and future work in Section 5.

2 Palladio and its e-Motions Specification

This work is based on two Model-Driven Engineering (MDE) frameworks, namely Palladio and e-Motions. We use Palladio [11] for the modelling and performance analysis of component-based systems, and e-Motions [16] to specify Palladio's operational semantics and the adaptation of Palladio systems over time, what allows us to simulate and analyze Palladio-like adaptive systems. In this section, we provide some background on both frameworks to ground the discussion that will follow. We introduce our case study to illustrate the main ideas of Palladio, and illustrate the use of e-Motions on its definition of Palladio.

2.1 Palladio Architecture Simulator

Palladio [11] 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 DSL used by Palladio is provided by its metamodel, the Palladio Component Model (PCM) [4]. 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 CBSE development process [5]: Component specifications (by component developers), assembly model (by software

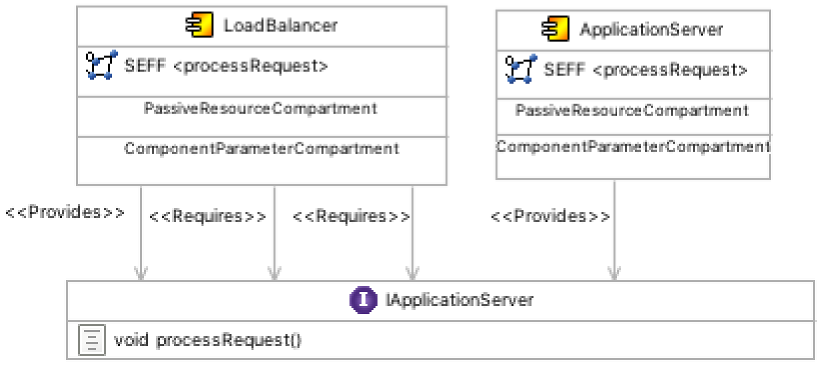
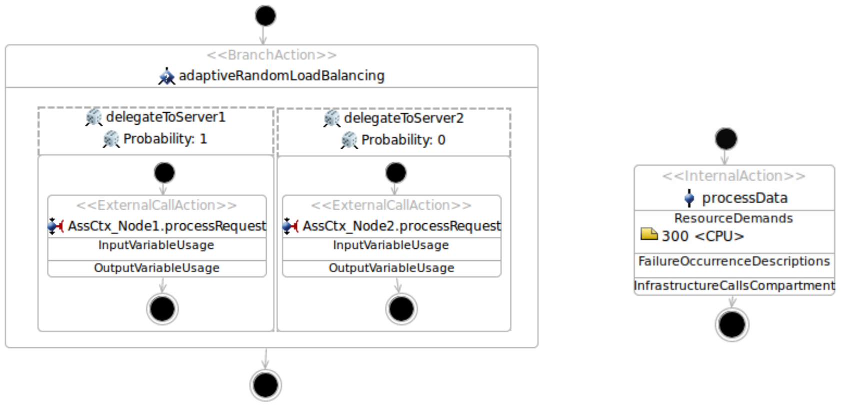


Fig. 1. Components repository



(a) Load balancer (b) Server node

Fig. 2. Components' SEFFs

architects), allocation model (by system developers), and usage model (by business domain experts). We illustrate these artefacts on a very simple case study, originally published in [2], that specifies a load balancer.

Component specifications. Component developers specify and implement parametric descriptions of components and their behavior. Fig. 1 shows the Palladio Component repository for our example, with components `LoadBalancer` and `ApplicationServer`, and their dependencies. In it we can observe that there is an interface implemented by two different components. The first one represents the load balancer and the second one the component itself. Note that there are two `Requires` relations between `LoadBalancer` and `IApplicationServer`, which means that in the system model, the node containing such component (the front-end node) has to be connected to two nodes.

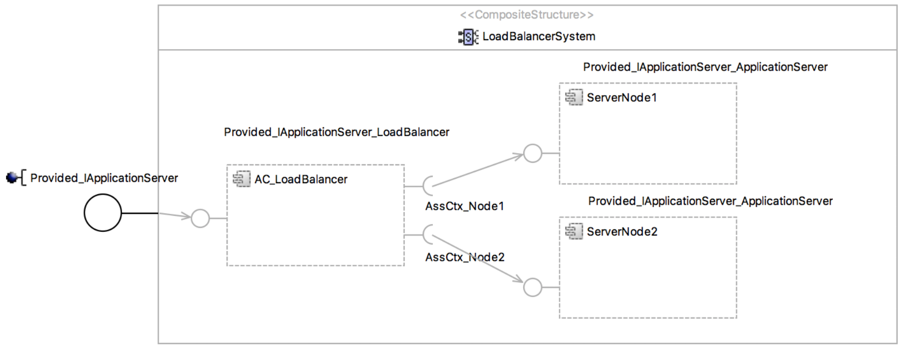

<<CompositeStructure>>
LoadBalancerSystem
Provided_IApplicationServer_ApplicationServer
ServerNode1
Provided_IApplicationServer_LoadBalancer
Provided_IApplicationServer
AC_LoadBalancer
AssCtx_Node1
Provided_IApplicationServer_ApplicationServer
ServerNode2
AssCtx_Node2

Fig. 3. Assembly model

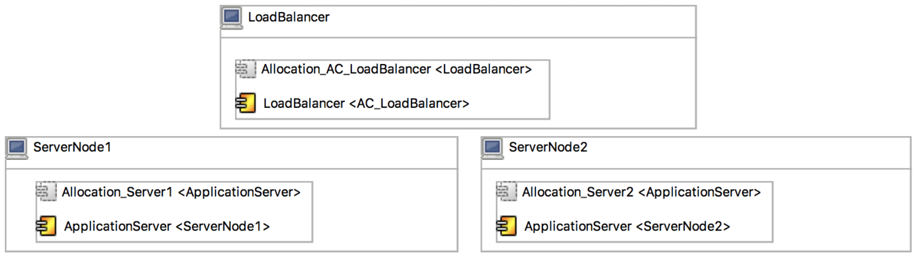

LoadBalancer
Allocation_AC_LoadBalancer <LoadBalancer>
LoadBalancer <AC_LoadBalancer>
ServerNode1
Allocation_Server1 <ApplicationServer>
ApplicationServer <ServerNode1>
ServerNode2
Allocation_Server2 <ApplicationServer>
ApplicationServer <ServerNode2>

Fig. 4. Allocation Model

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. Fig. 2 shows the SEFFs of these components. Fig. 2(a) shows 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., [18]).

Assembly model. Software architects assemble components from the repository to build applications. Fig. 3 shows how the components `LoadBalancer` and `ApplicatonServer` 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.

Allocation model. System deployers model the resource environment and the allocation of components from the assembly model to different resources of the resource environment. Fig. 4 shows the allocation model for our case study, where we can see how each of the components is allocated in a different node.

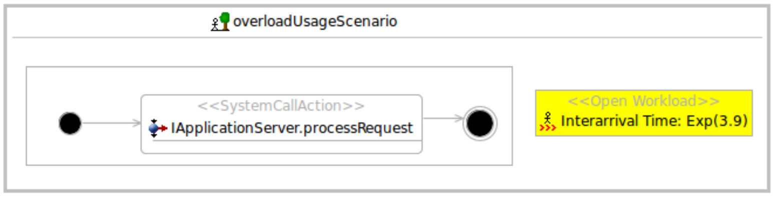

overloadUsageScenario
<<SystemCallAction>>
IApplicationServer.processRequest
<<Open Workload>>
Interarrival Time: Exp(3.9)

Fig. 5. Usage Model

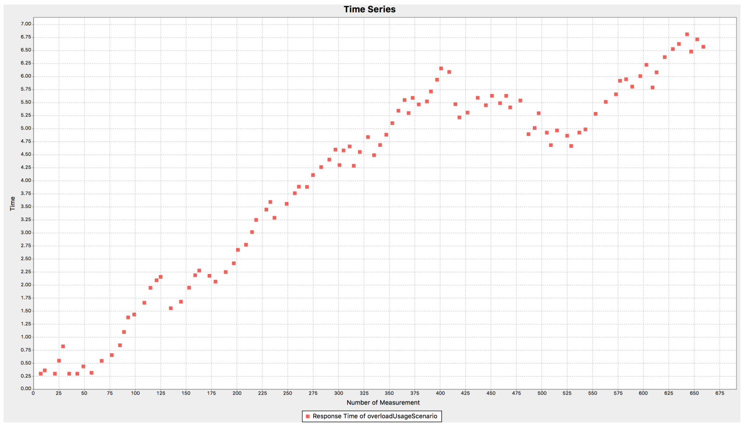


Fig. 6. Response time analysis by Palladio

Usage model. 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 users utilise). Given the usage model definition in Fig. 5, 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.

The analysis of this system with the Palladio Bench produces the graph in Fig. 6, producing a mean response time 4.0197 for 100 observations. Notice that with the workload used the system get overloaded, producing increasingly bigger response times as time passes. 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. With our proposal, these parameters can be changed, as the architecture or allocation of components, at run time.

2.2 The e-Motions System

e-Motions [16] 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 (Domain-Specific Language) 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 [7] specification which can be used for simulation and analysis.

The in-place model transformations used to specify the behavior of systems are defined by rules, each of which represents a possible *action* of the system. These rules are of the form $[\text{NAC}]^* \times \text{LHS} \to \text{RHS}$, 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 behavior. 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 [17] and [8]).

Palladio is a DSL, and has been specified in [14] using the visual facilities of the e-Motions system [16]. As for any DSL, the e-Motions definition of Palladio includes its abstract syntax (the PCM), its concrete syntax, and its behavior. 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 behavior is defined by graph transformation rules, thus becoming explicit at a very high level of abstraction.

The operational semantics of Palladio, i.e., its behavior, is given as a token-based execution model, where each work that enters the system is modelled

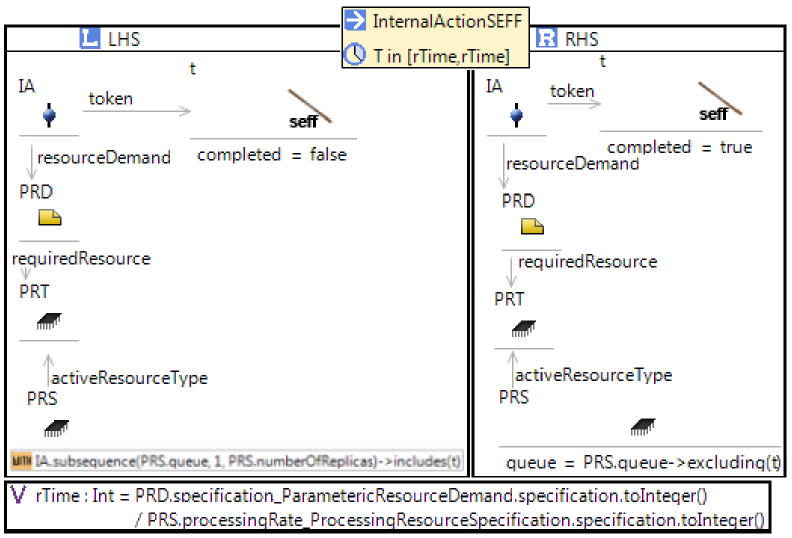

InternalActionSEFF
T in [rTime,rTime]
LHS
RHS
IA
token
t
seff
completed = false
resourceDemand
PRD
requiredResource
PRT
activeResourceType
PRS
IA.subsequence(PRS.queue, 1, PRS.numberOfReplicas)->includes(t)
IA
token
t
seff
completed = true
resourceDemand
PRD
requiredResource
PRT
activeResourceType
PRS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

Fig. 7. Internal Action SEFF rule

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, ...) in its SEFF descriptions. Each of the actions that may occur in the system are then specified by e-Motions transformation rules. For example, Fig. 7 shows the e-Motions rule that specifies the execution of an `InternalAction`, like the one shown in Fig. 2(b). 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 [19] 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 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 Adaptive Palladio Systems with 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 behavior 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 [13]. 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 consider the adaptation rules for CPU scale up and adjustment of parameters at the infrastructure level. Some of these adaptation mechanisms may dynamically become in conflict. For example, a failure in the satisfaction of a response 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. To simplify our presentation, and specifically for our case study, we assume that rules are specified so that one adaptation mechanism is attempted after another. More precisely, we use a token element with the name of the active adaptation mechanism, which can be either `CPU`, `LB`, or `noAdapt`.

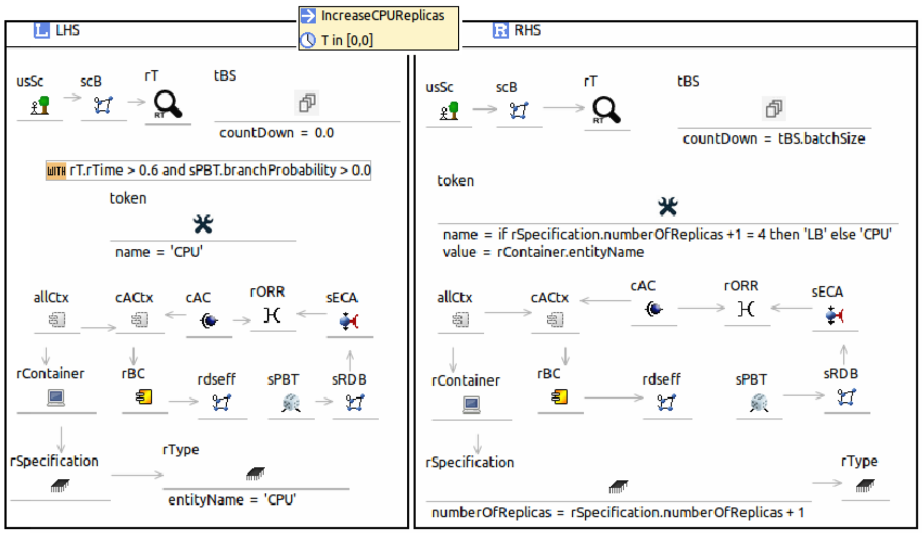

IncreaseCPUReplicas
T in [0,0]
LHS
RHS
usSc
scB
rT
tBS
countDown = 0.0
rT.rTime > 0.6 and sPBT.branchProbability > 0.0
token
name = 'CPU'
allCtx
cACtx
cAC
rORR
sECA
rContainer
rBC
rdseff
sPBT
sRDB
rSpecification
rType
entityName = 'CPU'
countDown = tBS.batchSize
name = if rSpecification.numberOfReplicas +1 = 4 then 'LB' else 'CPU'
value = rContainer.entityName
numberOfReplicas = rSpecification.numberOfReplicas + 1

Fig. 8. Increase CPU Replicas Rule

Figs. 8 and 9 show the e-Motions rules modelling these two adaptation mechanisms. In these rules, three elements control the application of an adaptation action: a threshold for the average response time of the system (`rT.rTime`), a batch size used as the minimum time interval between adaptation actions (`tBS.batchSize`), and the token that indicates the adaptation rule to be triggered (`token.name`). These parameters are configurable and can be established by the designer according to the system requirements.

Fig. 8 shows the `IncreaseCPUReplicas` rule that models the scale up adaptation mechanism by increasing the number of CPU replicas in a node (`rContainer`). If the adaptation mechanism is set to CPU (`token.name = 'CPU'`), the batch countdown reaches zero (`tBS.countDown = 0.0`), and the average response time of the last batch-size elements is greater than the threshold (`rt.rTime > 0.6`), then the number of replicas is increased. Note that if the number of replicas reaches its maximum value (5), then the adaptation mechanism is set to `LB`.

Fig. 9 shows the rule that changes the probability branch of a `LoadBalancer` component (see Fig. 2(a)). Again, if the adaptation mechanism is set to `LB`, the batch countdown reaches zero, and the average response time of the last batch size elements is greater than the threshold, then the probabilities assigned to the branches are changed.

The adaptation mechanisms implemented by these rules are very simple, but show the potential of the approach. The average response time for execution of the Palladio example presented, under ideal conditions, is $0,3$ time units. We define the average system response time threshold is the `rT.rTime` $< 0,6$. The time interval between adaptation actions defined was `tBS.batchSize` $= 10$ and, to establish adaptation control, we understand that increasing the number

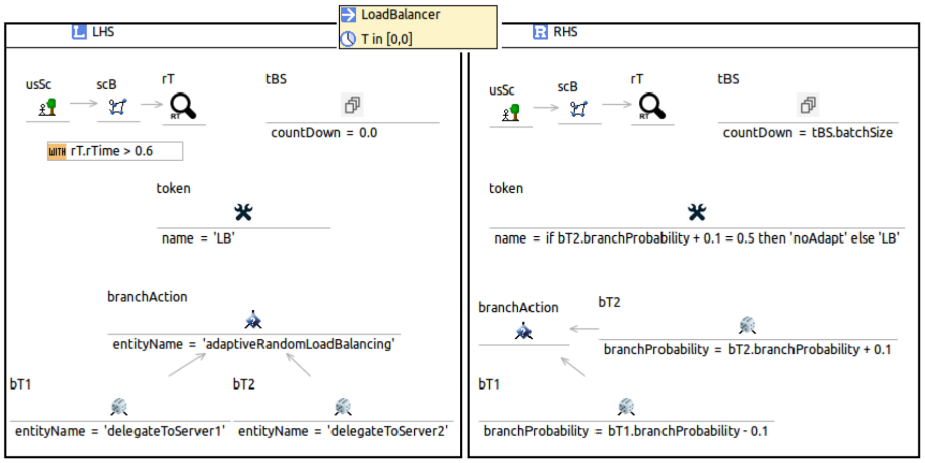

LoadBalancer
T in [0,0]
LHS
RHS
usSc
scB
rT
tBS
countDown = 0.0
rT.rTime > 0.6
token
name = 'LB'
branchAction
entityName = 'adaptiveRandomLoadBalancing'
bT1
bT2
entityName = 'delegateToServer1'
entityName = 'delegateToServer2'
countDown = tBS.batchSize
name = if bT2.branchProbability + 0.1 = 0.5 then 'noAdapt' else 'LB'
branchProbability = bT2.branchProbability + 0.1
branchProbability = bT1.branchProbability - 0.1

Fig. 9. `LoadBalancer` Rule

of CPU replicas is less costly than a load balancer adaptation. Thus, our system was modeled to trigger a CPU adapted as a first alternative to unwanted system changes, i.e, the system starts with a `token.name = 'CPU'`. With this parameters, the results shown by the system with and without the adaptation mechanism on are depicted in Fig. 10. The applications of the adaptation rules are represented by the vertical lines. The `IncreaseCPUReplicas` rule is applied by the first time at time 1.2. After that, it is applied again at times 11.2 and 21.2, when the batch countdowns reach 0. While without the adaptation mechanism the response times shown by the system grows very rapidly, with the adaptation mechanism its behavior gets stable quite soon.

4 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 [12] a DSL to describe the behavior 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 [2] 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.

D-Klaper [10] 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 lan-

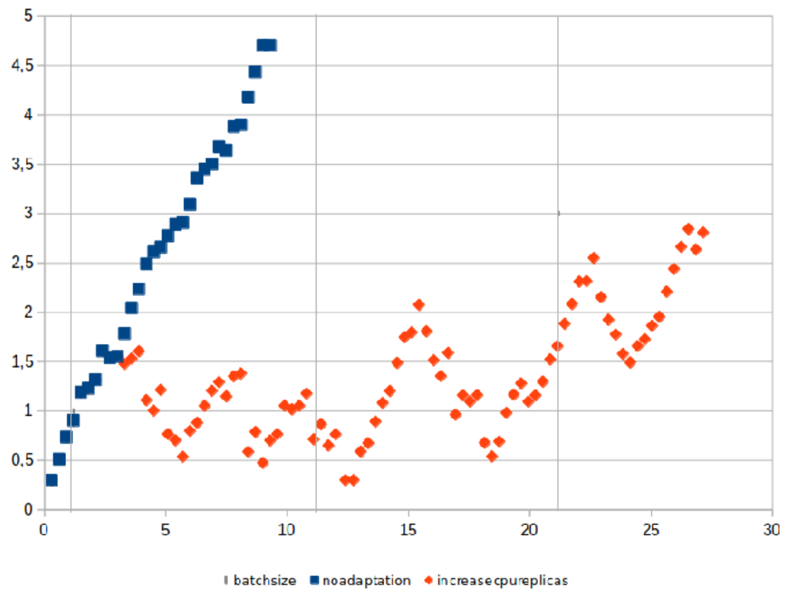


Fig. 10. Internal Action SEFF rule

guage 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.

5 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.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (Apr 2010)
2. Becker, M., Becker, S., Meyer, J.: Simulizar: Design-time modeling and performance analysis of self-adaptive systems. Software Engineering 213, 71–84 (2013)
3. Becker, M., Lehrig, S., Becker, S.: Systematically deriving quality metrics for cloud computing systems. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering. pp. 169–174. ACM (2015)
4. Becker, S., Koziolek, H., Reussner, R.: Model-based performance prediction with the Palladio component model. In: Proc. 6th Int'l Workshop on Software and Performance (WOSP'07). ACM (2007)
5. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3 – 22 (2009)
6. Chinneck, J., Litoiu, M., Woodside, M.: Real-time multi-cloud management needs application awareness. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. pp. 293–296. ICPE '14, ACM (2014)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude, LNCS, vol. 4350. Springer (2007)

8. Durán, F., Moreno-Delgado, A., Álvarez-Palomo, J.M.: Statistical model checking of e-Motions domain-specific modeling languages. In: Stevens, P., Wasowski, A. (eds.) 19th International Conference Fundamental Approaches to Software Engineering (FASE). Lecture Notes in Computer Science, vol. 9633, pp. 305–322. Springer (2016)
9. Falkner, K., Szabo, C., Chiprianov, V.: Model-driven performance prediction of systems of systems. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 44–44. ACM (2016)
10. Grassi, V., Mirandola, R., Randazzo, E.: Model-driven assessment of QoS-aware self-adaptation. In: Software Engineering for Self-Adaptive Systems, pp. 201–222. Springer (2009)
11. Happe, J., Koziolek, H., Reussner, R.: Facilitating performance predictions using software components. IEEE Software 28(3), 27–33 (2011)
12. Huber, N., van Hoorn, A., Koziolek, A., Brosig, F., Kounev, S.: S/t/a: Metamodeling run-time adaptation in component-based system architectures. In: e-Business Engineering (ICEBE), 2012 IEEE Ninth International Conference on. pp. 70–77. IEEE (2012)
13. Hummaida, A.R., Paton, N.W., Sakellariou, R.: Adaptation in cloud resource configuration: a survey. Journal of Cloud Computing 5(1), 1–16 (2016)
14. Moreno-Delgado, A., Durán, F., Zschaler, S., Troya, J.: Modular DSLs for flexible analysis: An e-Motions reimplementation of Palladio. In: European Conference on Modelling Foundations and Applications (ECMFA). pp. 132–147. Springer (2014)
15. Pérez-Palacín, D., Mirandola, R.: 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. pp. 3–14. ICPE '14, ACM (2014)
16. Rivera, J.E., Durán, F., Vallecillo, A.: A graphical approach for modeling time-dependent behavior of DSLs. In: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 51–55. IEEE (2009)
17. Rivera, J.E., Durán, F., Vallecillo, A.: Formal specification and analysis of domain specific models using Maude. Simulation 85(11-12), 778–792 (2009)
18. Smith, C.U., Williams, L.G.: Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Object-Technology Series, Addison-Wesley (2002)
19. Troya, J., Vallecillo, A., Durán, F., Zschaler, S.: Model-driven performance analysis of rule-based domain specific visual models. Information & Software Technology 55(1), 88–110 (2013)