Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

A PAAS-BASED FRAMEWORK FOR AUTOMATED PERFORMANCE ANALYSIS OF SERVICE-ORIENTED SYSTEMS

Andrea D'Ambrogio
Paolo Bocciarelli
Antonio Mastromattei

Department of Enterprise Engineering
University of Rome Tor Vergata
Via del Politecnico, 1
00100, Rome, ITALY

ABSTRACT

Service-oriented systems are often at the core of mission- or business-critical systems, and thus advanced quantitative analysis techniques are needed to assess, from the early development stages, whether or not the system accomplishes the stakeholder requirements and constraints. In this respect, in order to take advantage of the distributed nature of the considered systems, the use of distributed simulation (DS) appears the most natural and effective simulation approach. Nevertheless, the integration of traditional system development processes with DS approaches can be cost- and time-demanding. This paper presents *SOAsim*, a highly automated framework that allows system designers to generate the executable DS code from the model-based specification of the system under study, by use of automated model transformations. Moreover, in order to reduce the costs of setting-up dedicated DS platforms, SOAsim also automates the DS deployment and execution over a cloud-based infrastructure, according to a Platform-as-a-Service (PaaS) paradigm.

1 INTRODUCTION

Service orientation has proven to be a valuable approach for designing and implementing heterogeneous and distributed systems throughout the orchestration of remotely available services, according to the Service Oriented Architecture (SOA) paradigm and the Web Services technology (Papazoglou et al. 2007).

Service-oriented systems are often at the core of mission- or business-critical systems, and thus advanced quantitative analysis techniques are needed to assess, from the early development stages, whether or not the system accomplishes the stakeholder requirements in terms of performance levels to be guaranteed. In this respect, simulation is an effective and widely adopted approach for carrying-out a quantitative early evaluation of the system behavior. Simulation-based techniques provide a valuable strategy both to cut the cost of developing experimental prototypes and to mitigate the risk of time/cost overrun due to re-design and re-engineering of a system that does not provide the required performance. Moreover, in order to take advantage of the distributed nature of the considered systems, the use of distributed simulation (DS) turns out to be a natural and effective solution.

Nevertheless, the integration of traditional development process with distributed simulation approaches constitutes a non-trivial issue. The adoption of DS-based performance analysis approaches indeed requires significant know-how about distributed simulation and performance engineering, which system developers are likely not familiar with. Moreover, the deployment and execution of a DS requires the availability of a distributed infrastructure that can be time- and cost-demanding to be setup and maintained.

In this respect, this paper propose *SOAsim*, a model-driven framework that overcomes the aforementioned limitations and allows system developers to effectively carry out an early DS-based performance analysis of service-oriented systems.

The SOAsim design and implementation is based on standards and technologies provided by the OMG's Model Driven Architecture, which aims at automating, by use of model transformations, the generation of the executable DS code starting from the abstract specification of the system under study. Specifically, the SOAsim's model transformation chain takes as input a UML model of the system under study and generates as output a performance-oriented simulation model in the form of an Extended Queueing Network (EQN) model (Bolch et al. 2006). The EQN model is then mapped to the corresponding implementation code specified by use of jEQN (D'Ambrogio et al. 2006, Gianni and D'Ambrogio 2007), a domain-specific language for defining and executing EQN models onto either local or distributed environments.

Finally, in order to save the investments for developing and maintaining expensive distributed platforms, SOAsim automates the DS deployment and execution over a cloud-based infrastructure, according to the PaaS (Platform-as-a-Service) paradigm (Hayes 2008).

The reminder of the paper is structured as follows: Section 2 reviews related work. Section 3 summarizes the main concepts at the basis of this paper. Section 4 introduces the proposed SOAsim framework. Section 5 illustrates an overview of the SOAsim implementation. Section 6 discusses an example application and, finally, Section 7 gives concluding remarks.

2 RELATED WORK

To the best of our knowledge, no contributions can be found that specifically address both the model-driven performance analysis of SOA systems and the adoption of cloud computing infrastructures for executing relevant simulation models. In this respect, this section first reviews relevant contributions dealing with the performance analysis of SOA-based systems and then investigates the use of the cloud computing paradigm in the modeling and simulation domain.

Concerning the analysis of service-oriented systems it is worth noting that the queueing network formalism is widely used in the performance analysis domain as it is supported by several tools and framework, such as JMT (Bertoli et al. 2009) or Omnet++ (OpenSim 2014). Specifically, the use of queueing-based models to analyze performance-related issues in the SOA domain has been widely investigated in (Bardhan and Menascé 2013, Haddad et al. 2013, Mokdad and Youcef 2012). In (Bardhan and Menascé 2013) an approximate two-level single-class queueing network model is proposed to predict the execution time of applications executed on multicore systems. In (Haddad et al. 2013) and (Mokdad and Youcef 2012) the service response time of single and composite Web services is estimated by use of bounding models, so as to obtain an appropriate trade-off between accuracy and computational complexity. response time. Both single and multiple composite Web service execution instances are studied.

The above-mentioned contributions propose approaches to predict performance-related characteristics (e.g., the service time) of composite Web services by use of analytical-based techniques which provide valuable solutions but exhibit limitations due to the effort required to specify analytical performance models and to the computational costs needed to solve the models. Differently, the proposed approach overcomes such limitations by making of the jEQN language for implementing and executing the EQN simulation model. Moreover, the proposed method does not require specific skills of performance modeling theory as the adoption of model-driven standards and tools allows to effectively support the automated development of the jEQN code from system specification.

The adoption of a cloud infrastructure to deploy and execute a distributed simulation is a novel issue that has just recently started to be addressed, such as in (Fujimoto et al. 2010, Tolk and Mittal 2014, Rak et al. 2012). In (Fujimoto et al. 2010) the authors argue that cloud computing eases the use of distributed simulation techniques as it eliminates the need to purchase, operate and maintain the needed computational infrastructure at the local site. In (Tolk and Mittal 2014) it is argued how the adoption of cloud infrastructures to support distributed simulation is a challenging issue in the Modeling and Simulation domain and thus requires a further exploration. The contribution proposes a new paradigm to support simulation by use of composable and cloud-based simulation services. In (Rak et al. 2012) mJades, a simulation environment that allows users to execute multiple simulations in the cloud according to a SaaS paradigm, is presented. mJades exploits the mOSAIC cloud middleware and is built on top of JADES, a simulation library for building agent-based discrete event simulation systems. According to such contributions, this paper proposes the use of a cloud-based infrastructure to ease the deployment of a distributed simulation. Differently, in this paper approach the adoption of a cloud-computing paradigm is proposed in a wider perspective: the cloud infrastructure is part of a complete framework which enables the effortless DS-based analysis of SOA systems. The combined use of a model-driven approach, the jEQN library and a cloud-based infrastructure brings several benefits and contributes to overcome most limitations of existing contributions.

3 BACKGROUND

3.1 Model-Driven Architecture

Model-driven engineering (MDE) is an approach to system design and implementation that addresses the raising complexity of execution platforms by focusing on the use of formal models (Schmidt 2006). According to this paradigm, a system is initially specified by use of high-level models. Such models are then used to generate other models at a lower level of abstraction, which are used in turn to generate other models, until stepwise refined models can be made executable. One of the most important initiatives driven by MDE principles is MDA (Model Driven Architecture), the OMG's incarnation of MDE (OMG 2014). MDA-based development is founded on the principle that a system can be built by specifying a set of model transformations, which allow to obtain models at lower abstraction levels from models at higher abstraction levels. To achieve such an objective, MDA introduces the following key standards:

- Meta Object Facility (MOF): for specifying technology neutral metamodels (i.e., models used to describe other models) (OMG 2015b);
- Query/View/Transformation (QVT): for specifying *model-to-model transformations* (OMG 2015a);
- MOF Model to Text Transformation Language (Mof2Text or MOFM2T): for specifying *model-to-text transformations* (OMG 2008)

The relationship among MDA standards can be summarized as follows. Let M_A and M_B be two models, instances of the corresponding metamodels MM_A and MM_B, respectively. Both MM_A and MM_B are defined in terms of MOF constructs. A *model-to-model* transformation can be specified by use of QVT to generate M_B models from M_A models. MOF2MT-based *model-to-text* transformations are instead specified to obtain, e.g., executable code from models. In this work MDA has been used to design and develop the framework that enacts the automated performance analysis of service-oriented systems.

3.2 Distributed Simulation, SimArch and jEQN

A simulation program can be executed according to a local, parallel or distributed execution paradigm. A *local* (i.e., sequential) *simulation* is deployed and executed onto a single host. A *parallel simulation* is concerned with the execution on multiprocessor computing platforms containing multiple CPUs that interact frequently. A *distributed simulation (DS)* is concerned with the execution on a distributed platform consisting of a set of hosts interconnected through a computer network infrastructure, in which interactions take much more time (Fujimoto et al. 2010).

This work specifically addresses the DS case. The adoption of DS approaches bring several benefits, such as potentially reduced execution time, geographical distribution, fault tolerance, interoperability and reusability. Despite such advantages, the use of DS approaches is a complex and effort-comsuming task, which requires significant skills and know-how. In this respect, this work exploits the *SimArch* architecture

(Gianni et al. 2011) and the *jEQN* language (D'Ambrogio et al. 2006), which have been specifically proposed to reduce the DS development effort.

SimArch is a layered architecture that eases the development of local and distributed simulation systems by raising the developers from all the details concerning the execution environment, which can be either a conventional local execution platform or a distributed execution platform, e.g., one based on *HLA (High Level Architecture)*, the reference DS standard (IEEE 2010).

The simulation model is specified in terms of a domain-specific language (DSL) defined at the simulation components layer of SimArch. As aforementioned, this work exploits the jEQN language, a DSL for the specification of the EQN models (D'Ambrogio et al. 2006). The language provides several simulation components (i.e., jEQN components), which have been implemented on top of the underlying SimArch layers. Such jEQN components are classified into simulation entity components and support components. The simulation entity components identify the simulation logic and are named using the EQN standard taxonomy (e.g., user sources, waiting systems, service centers, routers and special nodes). The support components identify all the objects that do not affect the simulation logic and provide the structures for the entity components parameterization, e.g., policy frameworks, and for the data definition, e.g., users and queues. For a detailed description of SimArch and jEQN the reader is sent to (Gianni et al. 2011) and to the SimArch project web site.

3.3 Cloud-based Infrastructure

A pillar of the proposed approach is the adoption of a *cloud-based infrastructure* for DS deployment and execution. Cloud computing refers to the use of hardware or software computing resources that are delivered as a service over a network. This new paradigm basically makes the computing infrastructure available on demand over a the network, thus resulting in a substantial reduction of the costs associated with the management of both hardware and software resources (Vaquero et al. 2008, Hayes 2008).

A cloud-based infrastructure typically provides its services according to different paradigms, i.e, *IaaS* (Infrastructure as a Service), *PaaS* (Platform as a Service) and *SaaS* (Software as a Service). The proposed SOAsim framework specifically exploits the PaaS paradigm, according to which a cloud service provider offers the platform to build and launch applications, with significant benefits in terms of deployment and scalability.

In this respect, SOAsim makes use of the cloud infrastructure to deploy and execute the various jEQN federates (i.e., simulation components according to the HLA terminology) that implement the distributed simulation for the performance analysis of the system under study.

4 The SOAsim Framework

In order to effectively support the cloud-based and performance-oriented simulation analysis of SOA systems, this work introduces SOAsim, a framework tailored to the needs of service-oriented system developers. As aforementioned, the SOAsim framework supports the DS-based analysis by use of jEQN and makes use of model-driven standards and tools to automate the development of the jEQN simulation. The next subsections illustrate the architectural and the operational view of SOAsim, respectively.

4.1 SOAsim Architectural View

The architecture of the SOAsim framework is depicted in Figure 1. SOAsim includes four different components, which provide the various functions required for enacting the performance-oriented simulation analysis of SOA systems.

The *System Model* component provides a visual environment to specify a UML model of the SOA system under study. Such UML model is annotated by use of the MARTE and SoaML profiles (or standard UML extensions), in order to include both the performance properties and the required information to drive the development of the simulation systems.

The *Model-driven Performance Model Generation* component wraps the execution of the required *model-to-model* transformation for generating the adopted performance model, which is specified in terms of an extended queueing network (EQN) model.

The *DS Development and Deploy* component provides a visual environment for (i) partitioning the EQN model into a set of EQN submodels, (ii) executing the *model-to-text* transformation that yields the jEQN implementation of the EQN subnetwork federates and (iii) deploying the jEQN federates over the PaaS-based cloud infrastructure. In this respect, the partitioning of the EQN model conforms to the EQN metamodel introduced in (Bocciarelli et al. 2012), which includes the concept of *subnetwork* as an aggregation of *links* and *centers*.

The *DS Execution* component allows system developers to start the simulation execution and collect the related results.

It should be noted that the DS may include existing (third-party) jEQN federates, thus enhancing reusability. Moreover, the DS may include local/remote jEQN federates available on conventional LAN/WAN infrastructures.

4.2 SOAsim Operational View

This section illustrates how the SOAsim framework effectively enacts the performance-based analysis of service-oriented systems. The operational view of SOAsim is outlined in Figure 2, which details the four components introduced in the previous sections (i.e., system modeling, performance model generation, DS development & deployment and DS execution).

The system model is specified as a UML design model by use of the SOAsim visual environment. Such a model, which is annotated with stereotypes provided by the SoaML profile (OMG 2012), consists of the following set of packages, according to the modeling conventions for the application of the SoaML profile (Amsden 2010):

- *Service Interface Package:* specifies the capabilities and the related interfaces that the services must implement in order to be integrated in the service-oriented system;
- *Service Contract Package:* defines the Service Contracts (i.e., the specification of agreements between interacting parties, in SoaML terms) that specify how the interfaces are used;
- *Participant Package:* contains the Participants (i.e., a person, a system or an organization that provides or consumes a service, in SoaML terms) that implement the abstract interfaces;
- *Service Architecture Package*: specifies the Service Architecture (i.e., the network of interacting participants providing and consuming services, in SoaML terms), by means of a UML collaboration enriched with a nested UML activity diagram, which specifies the system behavior.

Besides the functional specification, the UML system model also includes the specification of the performance-related requirements. In this respect, the proposed method makes use of the MARTE profile (OMG 2011).

Since the service-oriented system is implemented as an orchestration of Web services, a service discovery is carried out to bind each service interface included in the UML model to one of the several concrete Web services matching the relevant service interface. The set of concrete services can be retrieved both from a public UDDI registry, which stores the specification of third-party services, and from a local repository, in which organizations store the description of ad-hoc services specifically provided by their known suppliers. The method has been designed to be compliant to standard WSDL-based service descriptions, which can be extended to include the specification of the performance properties (by use of, e.g., the Q-WSDL extension (D'Ambrogio and Bocciarelli 2007)). The resulting UML model thus contains both the performance requirements and the performance-related characterization of the orchestrated Web services.

Once the SOA system has been fully specified, a model-to-model transformation is executed to derive the EQN performance model of the system under study. Such EQN model is based on an enhanced EQN

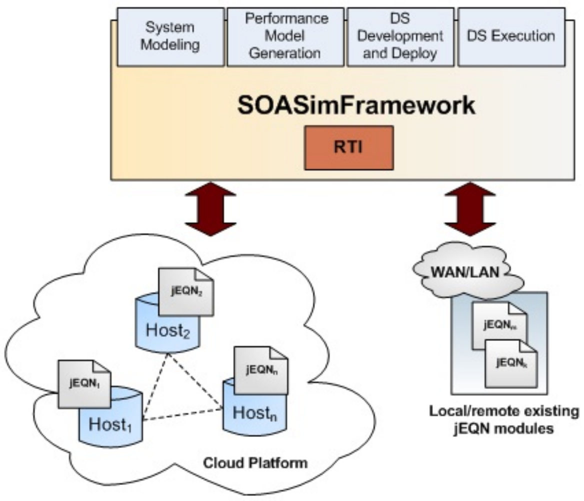

System Modeling
Performance Model Generation
DS Development and Deploy
DS Execution
SOASimFramework
RTI
WAN/LAN
Host2
Host1
Hostn
Cloud Platform
Local/remote existing jEQN modules

Figure 1: Architecture of SOAsim Framework.

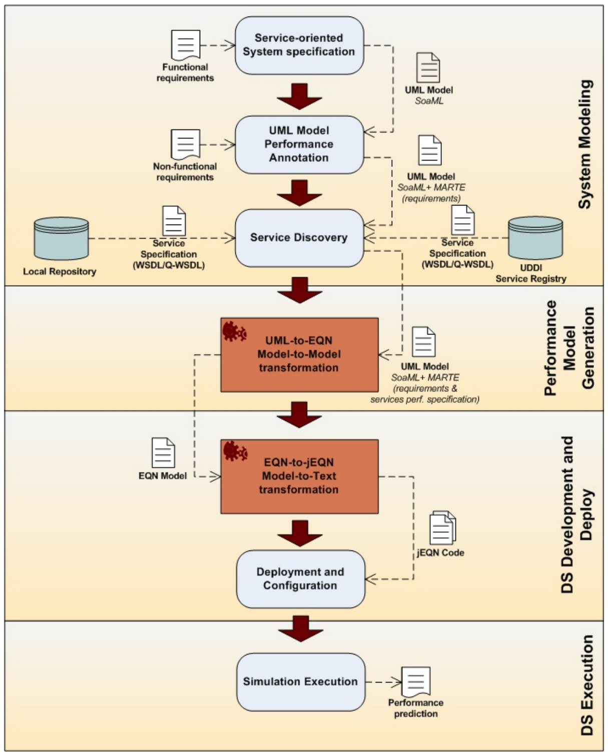

Service-oriented System specification
Functional requirements
UML Model SoaML
UML Model Performance Annotation
Non-functional requirements
UML Model SoaML+ MARTE (requirements)
Service Discovery
Local Repository
Service Specification (WSDL/Q-WSDL)
UDDI Service Registry
System Modeling
UML-to-EQN Model-to-Model transformation
UML Model SoaML+ MARTE (requirements & services perf. specification)
Performance Model Generation
EQN-to-jEQN Model-to-Text transformation
EQN Model
jEQN Code
Deployment and Configuration
DS Development and Deploy
Simulation Execution
Performance prediction
DS Execution

Figure 2: SOAsim Framework Operational View.

metamodel, which allows to represent the partitioning of the EQN model into a set of submodels (or subnetworks).

In this respect, the visual environment provided by SOAsim allows one to specify the partitioning of the EQN model and the execution of a model-to-text transformation, which yields as output the jEQN executable code of the DS federates. Finally, SOAsim supports the deployment of the jEQN federates onto the underlying cloud-based infrastructure.

Once the distributed simulation has been deployed, SOAsim allows the system developer to execute the performance-oriented DS and evaluate whether or not the system under study meets the specified performance requirements.

The next section gives an implementation overview of the SOAsim framework.

5 SOAsim Implementation Overview

The core of the proposed framework consists of two model transformations. The first one, namely `UML-to-EQN`, is the *model-to-model* transformation that takes as input the UML system model and yields as output the EQN performance model. Specifically, it maps an input UML activity diagram to an output EQN model. The second one, namely `EQN-to-jEQN`, is the *model-to-text* transformation that takes as input the EQN model and yields as output the jEQN-based DS executable code. The next two sections give a detailed view of such model transformations, respectively.

5.1 Generation of the EQN Performance Model

The `UML-to-EQN` model transformation has been specified by use of QVT (OMG 2015a) and has been implemented and executed by use of the Eclipse MMT plugin (Eclipse Foundation 2016). QVT prescribes that both the source and the target models used in a transformation must be instances of a MOF compliant metamodel (OMG 2015b).

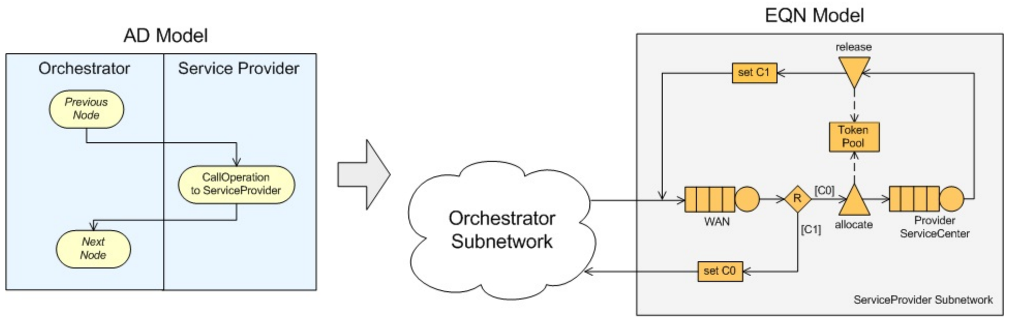


Figure 3: EQN mapping of the UML-based service invocation pattern.

The service-oriented system has been designed as an orchestration of services implemented by use of Web Services technology. It is assumed that the UML model includes an activity diagram that gives a detailed view of the party which acts as the coordinator of the service composition (the *orchestrator*, in SOA terminology), while other parties, which act as black-box service providers, are represented by swimlanes including an activity for each provided operation. In this perspective, a service invocation is represented in the UML model by an activity edge that connects a pair of activity nodes, the first one belonging to the swimlane associated to the orchestrator, and the second one belonging to the swimlane associated to the service provider. In order to give the rationale of the model transformation process, Figure 3 outlines the mapping of the UML fragment representing a service invocation into the corresponding portion of the EQN model.

The EQN model includes a subnetwork (or submodel) associated to each participant (e.g., the orchestrator and the set of service providers). Each subnetwork can be simulated by different simulation components of the DS infrastructure, as detailed later on. Two classes of jobs are introduced: the fist one, named `toServe`, is used to represent jobs that have to be served by a participant, the second one, named `Served`, is used to model a job just served by a participant. The pattern shown in the figure is related to a job that, once processed by the orchestrator, has to be forwarded to the next service center, according to the orchestration paradigm. The job class is initially set to `toServe`, briefly denoted as `C0`. A request to the next service center is structured as follows:

- the job passes through the WAN service center, to model the request message that the orchestrator sends to the service provider;
- the job passes through the router R and the through the Provider Service Center, to model the service execution performed by the participant. The access to the Provider Service Center is controlled by an Allocate/Release node, in order to model the capacity of the server that provides the requested service. Finally, as the job leaves the Release node, its class is changed to `Served`, briefly denoted as `C1`;
- the job passes through the WAN Service Center, to model the response message that the service provider sends to the orchestrator;
- the job returns to the Orchestrator Subnetwork. It should be noted that the router `R` forwards all jobs belonging to class `C1` to the `Set C0` node and then to the `Orchestrator Subnetwork`.

The parameterization of the EQN model has been carried out by implementing an algorithm based on the one specified in (D'Ambrogio 2005), which takes into proper account the MARTE annotations included in the input UML model. A complete outline of the mapping rules between UML elements and EQN elements is provided by Table 1.

The next section describes some implementation issues regarding the `EQN-to-jEQN` *model-to-text* transformation.

Table 1: Mapping of AD elements to EQN elements

UML Element	**EQN element**
MARTE annotation *(associated to swimlane)*	Users and think time parameters *(for closed EQN)*
MARTE annotation *(associated to swimlane)*	Distribution of interarrival time *(for open EQN)*
Start/Final Node	Terminal node *(for closed EQN)* Source/Sink node *(for open EQN)*
Opaque Action Node	request to Orchestrator Service Center
Call Operation Node	see Figure 3
Fork Node	Fork Node
Join Node	Join Node
Decision Node	Router Node
Control Flow	used for defining the routing within the EQN

5.2 Generation of the Executable DS Code

Once the EQN model has been obtained from the `UML-to-EQN` transformation, the `EQN-to-jEQN` transformation is executed to derive the jEQN executable code. The `EQN-to-jEQN` transformation, which has been specified as a MOFM2T *model-to-text* transformation (OMG 2008), has been implemented and executed by use of the Acceleo transformation language (Eclipse Foundation 2015), which is itself provided as an Eclipse plugin. Such a transformation also provides the required set of property files for configuring the simulation environment.

The transformation consists of the following steps: i) generation of both the software components managing the DS execution and the data structures exchanged by simulation components, ii) generation of simulation scenario settings and iii) generation of jEQN federates, according to the EQN partitioning;

As the currently adopted jEQN implementation makes use of the HLA standard (IEEE 2010), the software components created at step i) refer to the HLA Federation Manager, which coordinate the simulation lifecycle (Kuhl et al. 1999). This steps also produces the data definition files for data exchange among federates. Step ii) deals with the production of the configuration files that specify the HLA environment (e.g., hostname and port number of the HLA server) and the simulation scenario (e.g., number of jEQN federates, simulation length, etc.). Finally, at step iii) the code of the required jEQN federates is derived.

The mapping of local EQN elements to jEQN classes, for each subnetwork, is straightforward and is outlined in Table 2. Differently, the interaction among elements that belong to different subnetworks (e.g., to different jEQN federates) is managed by a mapping algorithm that generates local stubs for remote entities. Once all the stubs for the remote entities are defined, the `EQN-to-jEQN` transformation generates the jEQN code that implements the connections among jEQN elements, according to the EQN topology.

5.3 DS Deployment and Execution

SOAsim provides a specific facility to support the deployment and the execution of the jEQN-based DS over the cloud infrastructure. Specifically, SOAsim exploits the PaaS paradigm and adopts the Docker

Table 2: Mapping of EQN elements to jEQNclasses

EQN UML Class	**jEQN Class**
Terminal	Source Class + InfiniteServer Class
Source	Source Class
Sink	Sink CLass
Queue	WaitingSystem Class
Passive Queue	PassiveQueue Class
Split/Join Node	SplitNode/JoinNode Class
Router Node	Router Class

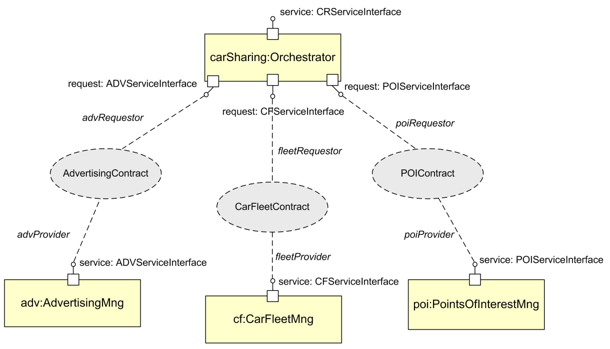


Figure 4: Composite Structure Diagram.

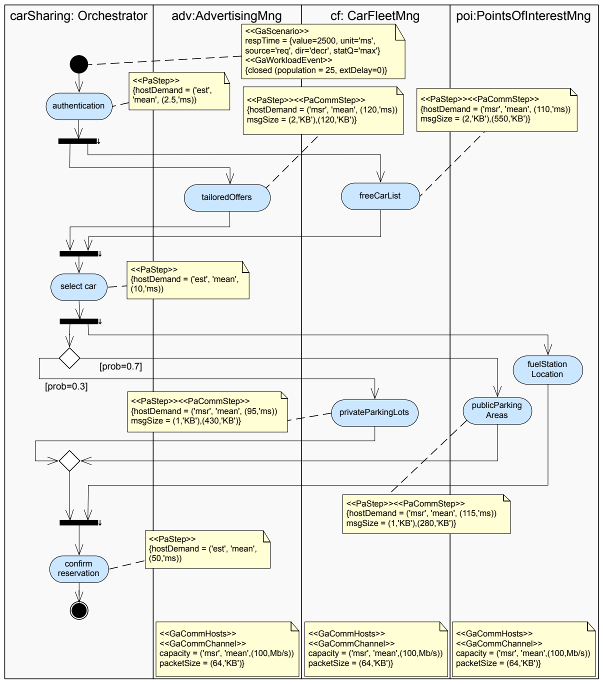

carSharing: Orchestrator
adv:AdvertisingMng
cf: CarFleetMng
poi:PointsOfInterestMng
<<GaScenario>> respTime = {value=2500, unit='ms', source='req', dir='decr', statQ='max'} <<GaWorkloadEvent>> {closed (population = 25, extDelay=0)}
<<PaStep>> {hostDemand = ('est', 'mean', (2.5,'ms))
authentication
<<PaStep>><<PaCommStep>> {hostDemand = ('msr', 'mean', (120,'ms)) msgSize = (2,'KB'),(120,'KB')}
<<PaStep>><<PaCommStep>> {hostDemand = ('msr', 'mean', (110,'ms)) msgSize = (2,'KB'),(550,'KB')}
tailoredOffers
freeCarList
<<PaStep>> {hostDemand = ('est', 'mean', (10,'ms))
select car
[prob=0.7]
[prob=0.3]
fuelStation Location
<<PaStep>><<PaCommStep>> {hostDemand = ('msr', 'mean', (95,'ms)) msgSize = (1,'KB'),(430,'KB')}
privateParkingLots
publicParking Areas
<<PaStep>><<PaCommStep>> {hostDemand = ('msr', 'mean', (115,'ms)) msgSize = (1,'KB'),(280,'KB')}
<<PaStep>> {hostDemand = ('est', 'mean', (50,'ms))
confirm reservation
<<GaCommHosts>> <<GaCommChannel>> capacity = ('msr', 'mean',(100,Mb/s)) packetSize = (64,'KB')}
<<GaCommHosts>> <<GaCommChannel>> capacity = ('msr', 'mean',(100,Mb/s)) packetSize = (64,'KB')}
<<GaCommHosts>> <<GaCommChannel>> capacity = ('msr', 'mean',(100,Mb/s)) packetSize = (64,'KB')}

Figure 5: Activity Diagram with MARTE annotations (service characteristics).

open source project (Docker 2016) to manage the deployment of the distributed simulation over the related virtual containers. This is achieved by realizing the *Deployment* use case, which is specified as follows:

Primary actor: User (system developer)
Pre-conditions:

- the system developer has executed the EQN-to-jEQN model transformation and the jEQN code of the simulation is available.
- the working directory has been selected.

Flow of actions:

1. the user selects the `Deploy` item from the SOAsim menu;
2. the user provides the required information (code package, host name, username, password);
3. the SOAsim system copies the jEQN modules over the remote container by use of the SSH/SCP protocol;
4. the SOAsim system manages the execution of the jEQN code;
5. the SOAsim system retrieves the simulation results from the remote host;
6. the simulation output is shown to the user.

6 EXAMPLE APPLICATION

This Section presents an example method application to a service-oriented application for reserving cars provided by a car sharing company. Let us suppose that the reservation process includes the following main steps:

1. the user provides the authentication credentials;

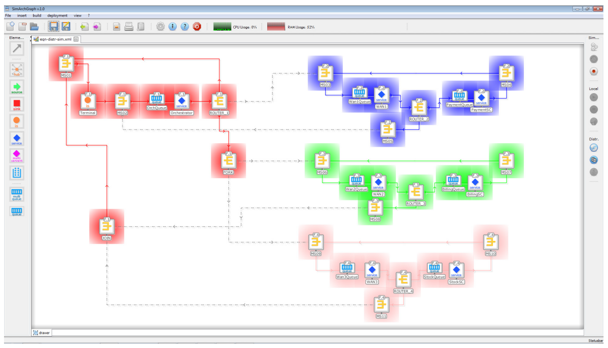
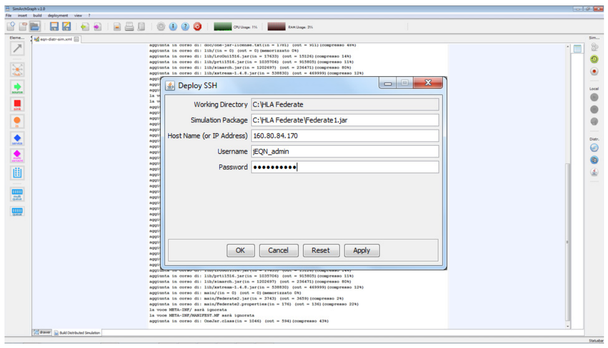


Figure 6: Example EQN model partitioning.



```
aggiunta in corso di: doc/one-jar-license.txt(in = 1781) (out = 911)(compresso 48%)
aggiunta in corso di: lib/(in = 0) (out = 0)(memorizzato 0%)
aggiunta in corso di: lib/LrcGui1516.jar(in = 17633) (out = 15126)(compresso 14%)
aggiunta in corso di: lib/prti1516.jar(in = 1035706) (out = 915805)(compresso 11%)
aggiunta in corso di: lib/simarch.jar(in = 1202697) (out = 236471)(compresso 80%)
aggiunta in corso di: lib/xstream-1.4.8.jar(in = 538830) (out = 469999)(compresso 12%)
```

Deploy SSH

Working Directory: C:\HLA Federate

Simulation Package: C:\HLA Federate\Federate1.jar

Host Name (or IP Address): 160.80.84.170

Username: jEQN_admin

Password: ••••••••••

```
aggiunta in corso di: lib/prti1516.jar(in = 1035706) (out = 915805)(compresso 11%)
aggiunta in corso di: lib/simarch.jar(in = 1202697) (out = 236471)(compresso 80%)
aggiunta in corso di: lib/xstream-1.4.8.jar(in = 538830) (out = 469999)(compresso 12%)
aggiunta in corso di: main/(in = 0) (out = 0)(memorizzato 0%)
aggiunta in corso di: main/Federate2.jar(in = 3743) (out = 3659)(compresso 2%)
aggiunta in corso di: main/Federate2.properties(in = 176) (out = 136)(compresso 22%)
la voce META-INF/ sarà ignorata
la voce META-INF/MANIFEST.MF sarà ignorata
aggiunta in corso di: OneJar.class(in = 1046) (out = 594)(compresso 43%)
```

Figure 7: Example federate deployment.

2. according to the existing user profile, the system retrieves and shows to the user a set of tailored commercial offers. The system also shows on a map the location of available cars ready for renting;
3. the user selects the car to be rented;
4. the system retrieves and shows the list of fuel stations and parking areas;
5. the user confirms the reservation.

As regards step 4, it is assumed that with a 70% probability the user prefers a public parking area. Depending to the user choice the system retrieves and shows the desired type of parking. According to the operational behaviour of SOAsim, shown in Figure 2, at the beginning the specification of functional and non-functional requirements of the service-oriented system is provided by use of a UML description annotated with stereotypes provided by SoaML and MARTE profiles. The system is designed as an orchestration of the following services: a service providing commercial services, denoted as *AdvertisingManager* service, a service managing the car fleet, denoted as *CarFleetManager* service and a service managing points of interests, denoted as *PointsOfInterestsManager* service.

As discussed in Section 4.2, the UML model consists of four diagrams that provide the following views: service interfaces and capabilities, service participants, service contracts and service architecture. As an example of the SoaML specification of the system, the *composite structure diagram* which specifies the service architecture is shown in Figure 4.

A service discovery is carried out to find a set of concrete services that match the abstract service interfaces specified in the UML model. For the sake of simplicity, it is assumed that a single concrete service is found for each required service interface. After the execution of a service discovery, the performance characteristics of the candidate service are finally available and thus included in the UML model. The resulting annotated UML activity diagram is shown in Figure 5. The provided annotations specify both the performance characteristics of the selected concrete services, i.e., annotations with `source` attribute valued as *'msr'* (measured) or *'est'* (estimated), and the performance requirements, i.e., annotations with `source` attribute valued as *'req'* (required). As an example, the requirement associated to the *Orchestrator* participant specifies that the response time of the whole process shall be less than 2500 ms.

The SOAsim visual interface allows to automatically generate the EQN model and to specify its partitioning, as shown in Figure 6, where the various subnetworks are identified by use grouping elements in different colors.

Then, the `EQN-to-jEQN` model transformation is executed to obtain the jEQN code of each subnetwork and the visual interface of SOAsim is used to specify the deployment parameters, as shown in Figure 7. Finally, the simulation is executed to yield the results of interest. In the example case, the resulting mean response time is 2298 ms (with a confidence interval of 95%), which validates the system against the requirement shown in Figure 6.

7 CONCLUSIONS

This paper has introduced an automated method and a supporting framework, named SOAsim, to carry out the simulation-based performance analysis of service-oriented systems. SOAsim enables system developers to easily transform a system model into the executable code of the corresponding distributed simulation, which is executed on top of a cloud-based infrastructure according to the PaaS paradigm.

The proposed method assumes the service-oriented system is specified by use of a UML model appropriately annotated by use of MARTE and SoaML profiles. The system model is then transformed into a performance model of EQN type, which is partitioned and transformed into the jEQN implementation code. Finally, the various jEQN federates are deployed for execution on the cloud-based infrastructure.

SOAsim can also be used without partitioning the EQN model, in case, e.g., it is not convenient to make use of a distributed execution, as discussed in (Gianni et al. 2010). The cloud-based deployment facility can thne be used to execute the jEQN code of the corresponding sequential implementation.

Work is in progress to investigate the possibility of reducing the amount of code to be deployed by separating the simulation model specification from the simulation executive, thus exploiting the full potential of the Modeling & Simulation as a Service (MSaaS) paradigm (Cayirci 2013).

REFERENCES

Amsden, J. 2010. "Modeling with SoaML, the Service-Oriented Architecture Modeling Language". http://www.ibm.com/ developerworks/rational/library/09/ modelingwithsoaml-1/index.html.

Bardhan, S., and D. A. Menascé. 2013. "Analytic Models of Applications in Multi-core Computers". In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*, 318–322. IEEE.

Bertoli, M., G. Casale, and G. Serazzi. 2009. "JMT: Performance Engineering Tools for System Modeling". *SIGMETRICS Perform. Eval. Rev.* 36 (4): 10–15.

Bocciarelli, P., A. Pieroni, D. Gianni, and A. D'Ambrogio. 2012. "A Model-Driven Method for Building Distributed Simulation Systems from Business Process Models". In *Proceedings of the 2012 Winter Simulation Conference*, edited by O. Rose and A. M. Uhrmacher, WSC '12, 227–239. Berlin, Germany: Institute of Electrical and Electronics Engineers, Inc.

Bolch, G., S. Greiner, H. de Meer, and K. S. Trivedi. 2006. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons.

Cayirci, E. 2013. "Modeling and Simulation as a Cloud Service: a Survey". In *Proceedings of the 2013 Winter Simulation Conference*, edited by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, WSC '13, 389–400. San Diego, CA, USA: Institute of Electrical and Electronics Engineers, Inc.

D'Ambrogio, A. 2005. "A Model Transformation Framework for the Automated Building of Performance Models from UML Models". In *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, 75–86. New York, NY, USA: ACM.

D'Ambrogio, A., and P. Bocciarelli. 2007. "A model-driven Approach to Describe and Predict the Performance of Composite Services". In *Proceedings of the 6th International Workshop on Software and Performance*, WOSP'07, 78–89. New York, NY, USA: ACM.

D'Ambrogio, A., D. Gianni, and G. Iazeolla. 2006. "jEQN a Java-based Language for the Distributed Simulation of Queueing Networks". In *Proceedings of the 21st International Conference on Computer and Information Sciences*, ISCIS'06, 854–865. Berlin, Heidelberg: Springer-Verlag.

Docker 2016. "Docker". Website: https://www.docker.com.

Eclipse Foundation 2015. "Acceleo". Website: https://eclipse.org/acceleo/.

Eclipse Foundation 2016. "Model-to-Model Transformation (MMT)". Website: https://projects.eclipse.org/projects/modeling.mmt.

Fujimoto, R. M., A. W. Malik, and A. Park. 2010. "Parallel and Distributed Simulation in the Cloud". *SCS M&S Magazine* 3:1–10.

Gianni, D., and A. D'Ambrogio. 2007. "A Language to Enable Distributed Simulation of Extended Queueing Networks". *Journal of Computers* 2 (4): 76–86.

Gianni, D., A. D'Ambrogio, and G. Iazeolla. 2011. "A software Architecture to Ease the Development of Distributed Simulation systems". *Simulation* 87 (9): 819–836.

Gianni, D., G. Iazeolla, and A. D'Ambrogio. 2010, May. "A Methodology to Predict the Performance of Distributed Simulations". In *IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*. Atlanta, GA.

Haddad, S., L. Mokdad, and S. Youcef. 2013. "Bounding Models Families for Performance Evaluation in Composite Web Services". *Journal of Computational Science* 4 (4): 232–241.

Hayes, B. 2008, July. "Cloud Computing". *Commun. ACM* 51 (7): 9–11.

IEEE 2010. *IEEE 1516 - Standard for Modeling and Simulation High Level Architecture - Framework and Rules*.

Kuhl, F., R. Weatherly, and J. Dahmann. 1999. *Creating Computer Simulation Systems: an Introduction to the High Level Architecture*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Mokdad, L., and S. Youcef. 2012. "Stochastic Bounds for Composite Web Services Response Times". *Cluster Computing* 15 (4): 363–371.

OMG 2008. *MOF Model To Text Transformation Language, version 1.0*.

OMG 2011. *UML Profile For MARTE: Modeling And Analysis Of Real-Time Embedded Systems, v. 1.1*.

OMG 2012. *Service oriented architecture Modeling Language (SoaML), v. 1.0.1*.

OMG 2014. *MDA Guide, version 2.0*.

OMG 2015a. *Meta Object Facility (MOF) 2.0 Query/View/Transformation, version 1.2*.

OMG 2015b. *Meta Object Facility, version 2.5*.

OpenSim 2014. "OMNeT++ Discrete Event Simulator v.4.6". Website: http://www.omnetpp.org/.

Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann. 2007. "Service-Oriented Computing: State of the Art and Research Challenges". *IEEE Computer* 40 (11): 38–45.

Rak, M., A. Cuomo, and U. Villano. 2012. "mJADES: Concurrent Simulation in the Cloud". In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, 853–860.

Schmidt, D. C. 2006. "Model-driven Engineering". *IEEE Computer* 39 (2): 25.

Tolk, A., and S. Mittal. 2014. "A Necessary Paradigm Change to Enable Composable Cloud-based M&S Services". In *Proceedings of the 2014 Winter Simulation Conference*, edited by S. J. Buckley and J. A. Miller, WSC '14, 356–366. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Vaquero, L. M., L. Rodero-Merino, J. Caceres, and M. Lindner. 2008, December. "A Break in the Clouds: Towards a Cloud Definition". *SIGCOMM Comput. Commun. Rev.* 39 (1): 50–55.

AUTHOR BIOGRAPHIES

ANDREA D'AMBROGIO is associate professor at the University of Roma Tor Vergata (Italy). His research interests are in the fields of model-driven engineering, dependability engineering, distributed and web-based simulation. His email address is dambro@uniroma2.it.

PAOLO BOCCIARELLI is a postdoc researcher at the University of Rome Tor Vergata (Italy). His research interests include software and systems engineering, business process management and model-driven development. His email address paolo.bocciarelli@uniroma2.it.

ANTONIO MASTROMATTEI is a graduate student at the University of Rome Tor Vergata (Italy), currently developing a master thesis on cloud-based distributed simulation. His email address antonio.mastromattei@gmail.com.