Specification and Simulation of Queuing Network Models using Domain-Specific Languages

Article *in* Computer Standards & Interfaces · September 2014

DOI: 10.1016/j.csi.2014.01.002

2 authors:

Javier Troya
University of Malaga
64 PUBLICATIONS **712** CITATIONS
SEE PROFILE

Antonio Vallecillo
University of Malaga
248 PUBLICATIONS **3,953** CITATIONS
SEE PROFILE

Specification and Simulation of Queuing Network Models using Domain-Specific Languages

Javier Troya*, Antonio Vallecillo

Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga, Bulevar Louis Pasteur, 35. (29071) Málaga, Spain

Abstract

Queuing Network Models (QNMs) provide powerful notations and tools for modeling and analyzing the performance of many different kinds of systems. Although several powerful tools currently exist for solving QNMs, some of these tools define their own model representations, have been developed in platform-specific ways, and are normally difficult to extend for coping with new system properties, probability distributions or system behaviors. This paper shows how Domain Specific Languages (DSLs), when used in conjunction with Model-driven engineering techniques, provide a high-level and very flexible approach for the specification and analysis of QNMs. We build on top of an existing metamodel for QNMs (PMIF) to define a DSL and its associated tools (editor and simulation engine), able to provide a high-level notation for the specification of different kinds of QNMs, and easy to extend for dealing with other probability distributions or system properties, such as system reliability.

Keywords: Domain-Specific Languages, Queuing Network Models, PMIF

1. Introduction

The specification and analysis of the non-functional properties of software systems, such as QoS usage and management constraints (performance, re-

*Corresponding author, telephone +34.95.213.2846, fax +34.95.213.1397

Email addresses: javiertc@lcc.uma.es (Javier Troya), av@lcc.uma.es (Antonio Vallecillo)

liability, etc.), is critical in most distributed application domains, such as embedded systems, multimedia applications or cloud computing. In fact, the development of methods and tools for performance evaluation and modeling has been an active area of research since the early days of software engineering.

Queuing Network Models (QNMs) provide powerful notations and tools for modeling and analyzing the performance of many different kinds of systems [1]. There are currently several tools for solving QNMs. However, some of these tools define their own model representations, have been developed in platform-specific ways, and are normally difficult to extend for coping with new system properties, probability distributions or system behaviours. A performance model interchange format, PMIF [2], was intended as a standard for defining and exchanging QNMs between tools, although only a few tools support it.

Domain Specific Languages (DSLs) provide intuitive notations, closer to the languages of the domain experts, in a compact and precise way, and at the right level of abstraction. When used in conjunction with Model-driven engineering (MDE) techniques [3], they become easy to develop, and allow the resulting models to be manipulated, analyzed and executed using standard tools.

This paper shows how a DSL for QNMs can be defined and built, providing a high-level and very flexible approach for the specification and execution of QNMs at a high-level of abstraction, and enabling the development of end-user tools in a flexible and cost-effective manner. We also show how an existing de-facto standard for QNM representation and interchange (PMIF) can be integrated into the MDE domain, being also extended and improved to cope with new required features and system properties.

Following the usual MDE process, the DSL is defined in terms of its *abstract syntax*, *concrete syntax* and *semantics*. The abstract syntax defines the domain concepts that the language is able to represent, and is defined by a metamodel. Given that the performance engineering community has already defined a common metamodel for QNMs, we have adopted PMIF as the base of our abstract syntax. The concrete syntax defines the notation of the language, and it is defined by a mapping from the concepts of the language into their textual or graphical representation. In this case this is defined using the Eclipse Graphical Modeling Framework (GMF [4]). Finally, the semantics describe the meaning of the models represented in the language, and in case of models of dynamic systems (such as ours) the semantics of a

model describe the effects of executing the models. Here, the semantics is given by a semantic bridge [5] from QNMs to in-place behavioral rules, and supported by the *e-Motions* toolkit [6, 7].

The resulting DSL, called xQNM, has been integrated in a tool, provides a notation for the specification of different kinds of QNMs, is easy to extend for dealing with other probability distributions or system properties—such as reliability—and is comparable to other existing QNM tools.

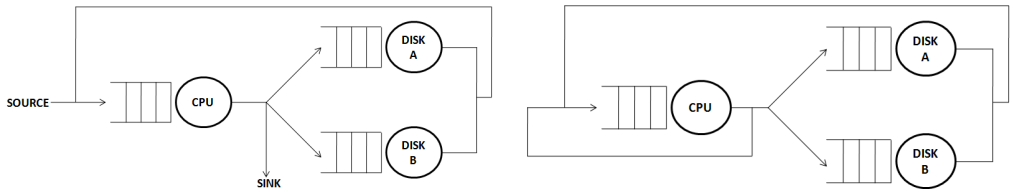
The rest of the paper is organized as follows. After this introduction, Section 2 presents the state of the art regarding QNMs, several tools and PMIF. Then, Section 3 introduces the abstract syntax of xQNM, in terms of an extension of PMIF 2 [2]. Section 4 presents the basic MDE concepts and mechanisms that we have used in our proposal. Section 5 presents an overview of the components of the xQNM language, describing its semantics in terms of a generic behavioral model for QNMs, its concrete syntax, and the graphical editor we have built to create and input queueing network models. Then, Section 6 explains how we deal with QNMs behavioral simulations, it compares them with other tools and presents the extensions needed to consider failures in servers. Finally, Section 7 concludes and outlines some lines of future work.

2. State of the Art

2.1. Queuing Network Models

In computer systems, many jobs share the system resources such as CPU, disks, and other devices. Since generally only one job (or some of them) can use the resource at any given time, all other jobs wanting to use that resource wait in queues. Systems where jobs may be serviced at one or more queues before leaving the system are modeled with queuing networks. Queuing theory helps in determining the time that jobs spend in various queues in the system [8]. These times can then be combined to predict the system response time, which is basically the total time that a job spends inside the system, and other non-functional features such as throughput, idle-times, etc.

There are two main types of queuing networks: *open* and *closed*. The former has external arrivals and departures. The jobs enter the system at a source and depart at a sink (Fig. 1(a)). The number of jobs in the system varies with time. Closed networks have no external arrivals or departures: the jobs in the system keep circulating from one queue to the next. The



(a) An Open Queuing Network (b) A Closed Queuing Network

Figure 1: Examples of an Open and a Closed Queuing Networks.

total number of jobs in the system is constant. It is possible to view a closed system as a system where the sink is connected back to the source (Fig. 1(b)), and jobs leaving the system immediately re-enter it. There are also *mixed* networks, which behave as open for some workloads and closed for others. All jobs of a single class have the same service demands and transition probabilities.

2.2. QNM tools

There are several commercial packages to queuing network modeling, like QNAP2 [9], the PDQ analyzer [10], SPE·ED [11], RESQME [12], BEST/1 [13], CSIM [14]. There are also many academic tools including TANGRAM-II [15], SHARPE [16], JINQS [17, 18], qnetworks [19] and JMT [20] (for a very complete list, see [21]).

Table 1 presents several relevant features of some of the existing packages and tools for solving QNMs (xQNM has also been included for comparison with the rest). They are listed according to their approximate chronological appearance. For each tool we list the evaluation technique it uses (analytical methods, simulation or both), the specific model representation needed, the probability distributions it accepts and the types of QNMs it can analyze. Most of these tools were developed some years ago, and each of them specifies a queuing network model in a different way and with a different language. To address the problem of exchanging models among tools, a performance model interchange format (PMIF) was proposed [2, 24, 25, 26]. PMIF provides a common representation for system performance model data that can be used to exchange models among QNM modeling tools. However, still most of the existing tools are not able to receive a PMIF model as input. It is true that some tools tried to define common formats for tool interoperability purposes, with goal similar to PMIF. This is the case of MOSEL-2 [27], a tool that provides means for specifying QNMs and carrying out some performance

Table 1: Features of some packages and tools for QN modeling and analysis

Tool	Evaluation technique	(Input) Model Format	Probability Distributions admitted	Types of QNM supported
RESQME (1986)	Discrete event simulation	Graphical environment with textual information to draw input models	Erlang, Exponential, Normal, Uniform, etc.	Extended QNMs of resource connection systems
SHARPE (1987)	Analysis	Graphical user interface for drawing input models	It allows s- independent random variables and mixing of distributions. It cannot handle Weibull distributions [22]	QNMs and also multiple model types (Fault Tree, Markov Chain, Semi-Markov Chain, MRGP, GSPN, PFQN, MPFQN, Trask graph, etc.)
QNAP2 (1992)	Both discrete event simulation and analysis	Programmatical. The analytical solvers need to be invoked	Erlang, Exponential, Normal, Uniform, etc.	Open, closed and mixed queuing networks
QSIM (1995, Release 6.11)	Discrete event simulation	Graphical user interface for drawing the input models	Exponential, Gamma, Erlang, Uniform, Deterministic, Non-Homogeneous Poisson, etc.	Open and closed networks
SPE·ED (1996)	Analysis and Simulation	Graphical user interface for drawing the input models	Various (for simulation)	Any QNM as well as SPE models as defined in Connie U. Smith's books
PEPSY-QNS (1996)	Both analysis and discrete event simulation	Graphically (with XPEPSY), or textually	Various (for simulation)	Open, closed and mixed networks
TANGRAM-II (1997)	Analysis and simulation	Programmatical (models are composed of objects that interact by exchanging messages)	Exponential, Pareto, Deterministic, Uniform, Erlang, Gaussian, Log-normal, FARIMA, FBM	Models of communication systems (computer networks, traffic systems, etc.)
PDQ (1998)	Analysis	Programmatical (using C)	Exponential distribution	Open and closed networks
MQNA (2003)	Analysis	Textually	Exponential distribution	Open and closed product-form QNs and finite capacity QNs.
WinPEPSY-QNS (2006)	Analysis and simulation (closed queuing systems with capacity and phase type distributions cannot be simulated)	Graphical user interface for drawing input models	Phase-type distributions (approximations of long-tail distributions achieved by finite mixtures of exponentials [23])	Stochastic models based on queuing networks with phase-type distributions
JINQS (2006)	Discrete event simulation	Programmatical (in Java)	Exponential, Weibull, Cauchy, Deterministic, Erlang, Gamma, Geometric, Normal, Pareto, Uniform	Any queuing system and queuing network model
JMT (2007)	Analysis and discrete event simulation	Graphical user interface for drawing input models. Wizards are available. It also supports interoperability via XML	Pareto, Gamma, Hyperexponential, Erlang, etc.	Any queuing system and queuing network model
qnetworks (2009)	Analysis	Programmatical (in Octave)	Poisson distributions for arrival rates and Exponential distributions for service times	Open, closed and mixed networks with multiple job classes
xQNM (2012)	Discrete event simulation	Graphical user interface for drawing the input models. Importation of PIMF models is also allowed	Uniform, Exponential, Normal, Gamma, Weibull, Erlang, F, Log-normal, Pareto, Pascal, etc.	Open and closed networks

measurements over them. The tool is equipped with a set of model translators that allow the automatic transformation of MOSEL-2 models to several third-party performance evaluation tools. WEASEL [28] is an interesting client-server application in which the user can specify a PMIF 2 (see Sect. 2.3) model graphically and then solve it by using the following external solution tools: PDQ, SHARPE, MVACCKSW (MVA using different methods) and PEPSY. Furthermore, it offers the option to translate the PMIF 2 model to the specific notation of different tools, such as PDQ, SHARPE, PMVA, QNAP, OPENQN, CLOSEDQN, MVAQFP, MQNA1, MQNA2 and PEPSY.

Only some of the tools mentioned provide a graphical interface for the definition of QNMs (namely RESQME, SHARPE, SPE·ED, PEPSY, JMT, QSIM and xQNM), in the rest the input models have to be introduced textually or programmatically. And in most cases, all these formats are proprietary and cannot be easily ported to other tools.

Analytical methods do not allow the exact evaluation of the performance of QNMs with arbitrary probability distributions for arrival and service times, only if they use Exponential and Uniform distributions. This is why many packages also offer solutions based on simulation for dealing with other distributions: TANGRAM-II, SPE·ED, QNAP2, WinPEPSY-QNS and the JMT suite. Our tool belongs to this group.

Among the tools described in Table 1, there are tools written in FORTRAN (QNAP2), C++ (TANGRAM-II and WinPEPSY-QNS), C (PEPSY-QNS, PDQ Analyzer), GNU Octave (qnetworks) and Java (JINQS, JMT). This is one aspect in which our tool significantly differs from the rest, because it has been developed using MDE techniques, and is defined in terms of DSLs and model transformations between them, at a higher level of abstraction. This allows us the possibility to modify or improve one of its parts and keep the rest untouched, and provides us with a very organized and modular architecture. Consequently, it makes the tool easier extensible for future versions and improves its maintainability. jEQN [29] is a DSL for the specification and implementation of distributed simulators for extended queueing networks. Although it also uses MDE techniques and provides a DSL for specification and simulation, it builds on Java while our approach relies on an existing DSL for the specification of real-time systems. Besides, jEQN focuses on the development of distributed simulators from local ones for extended QNMs while our tool focuses on the definition and management of QNMs (definition, importation, exportation) as well as on their simulation.

Most of the works about QNMs do not consider failures. This is, the

servers that compose the network can fail, being unable to process jobs for some time and contributing to system delay. In this sense, these works consider that the networks have an "ideal" behavior, where nothing can go wrong. But this is far from reality, since in many systems modeled with queuing networks many things can go wrong. For example, in manufacturing systems, the machines that make up the system can fail, or the actual servers that compose any kind of network modeled with a QNM can have failures too (hardware failures, failures due to wear out, random failures, etc.).

There are some works that do take into account failures of this type. For example, Das and Murray Woodside [30] consider that any of the entities in a model can undergo a failure, which is independent of the failures of other entities in the model. Each entity i has its own component state, S_i (0 or 1), corresponding to its working state or failed state, and is governed by a separate Markov chain with a working state ($s_i = 1$) and a failed state ($s_i = 0$), with rates of failure and repair. We have applied this idea of networks' components having two states to extend the behavior of ordinary QNMs (see Section 6.4). Altiok [31] has reviewed in detail literature pertaining to queues with service breakdowns due to failures of service stations. S. Kumar and P. R. Kumar consider machine's failures in manufacturing systems [32], and assign exponential times for times to failure and times to repair. Govil and Fu survey in [33] contributions and applications of queuing theory in the field of discrete part manufacturing, where they reference other works dealing with failures in manufacturing flow lines [34, 35, 36].

2.3. Evolution of PMIF

PMIF was conceived as a common representation for system performance model data that could be used to move models between modeling tools [26]. Its creators were interested in tool interoperability for *Software Performance Engineering* [37]. Its structure represents the software processing steps and other information for workloads that execute in the system performance level. PMIF, however, was born for system performance models that represent computer platforms and network interconnections with a network of queues and servers. Its representation technique had to be appropriate to express the interchange format and it needed to be capable of expressing a wide range of system execution models: those containing a small number of servers to a very large number of them, from one to many workloads, both open and closed models, that may be solved using either analytical or simulation solution techniques. It also had to be usable with existing tools, include modeling

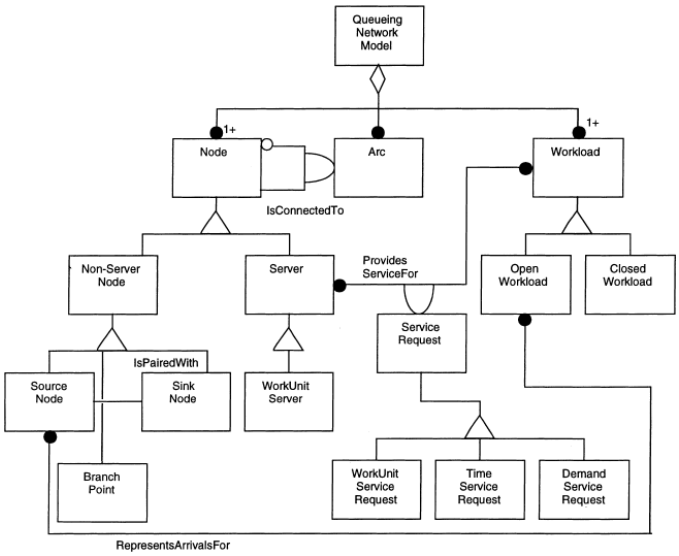


Figure 2: PMIF 1.0 Metamodel

features that tools provide, support the modeling paradigms prevalent in tools, and use terminology common in tools and modeling research. So the first version of PMIF (1998), as explained in [26], addressed a specific type of performance model: Queuing Network Models that may be solved using exact analytical solution algorithms. The resulting metamodel is shown in Figure 2. In this version, the use of the operational analysis term *visits* rather than the stochastic modeling *probability* among servers was proposed.

A new version of the PMIF metamodel and its XML schema specification (called PMIF 2.0, and later PMIF 2) was then presented in [24, 38, 2]. An XML-based approach was used to tackle the complexity and amount of effort required to create the PMIF interface. It uses the previous PMIF (PMIF 1.0) metamodel as a starting point because it is a good description of the information requirements for performance model interchange, but uses XML to implement the transfer format. As previously mentioned, the PMIF 1.0 metamodel uses number of visits instead of routing probabilities, assuming that from the number of visits, and with the knowledge of the queuing network topology, routing probabilities can be calculated. This assumption is true for many of the queuing networks that model computer systems. However, it is not true for the general case. This is why the routing probability was added as a transit element which specifies where a job has to transit and

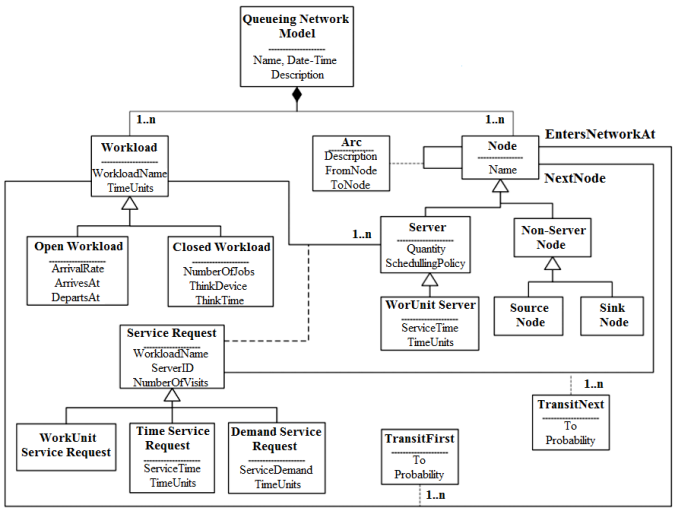


Figure 3: PMIF 2 Metamodel (from [2])

with what probability. One of the advantages of PMIF is that it can be used by web services to export and import QNMs among different modeling tools. In [39], PMIF 2 is used as the exchange format of QNMs among SPE·ED and QNAP by means of a web service. First, the software model created in the SPE·ED performance modeling tool is exported to the PMIF 2 format. Then, it is transformed to the QNAP notation, after which the model is ready to be analyzed by QNAP. The PMIF 2 metamodel is shown in Fig. 3.

In this paper, we take a step forward because our aim is not just to be able to describe models in XML, but to integrate them into the MDE tool chain. Thus, we have used Ecore [40] as meta-metamodel, and so Ecore models representing queuing network models expressed in PMIF can be defined. Furthermore, several probability distributions for arrival and service times can be specified in the models. This is further explained in Section 3.

3. Expressing PMIF in Ecore

The metamodel conforming to Ecore [40] that we propose for defining QNMs, named ePMIF (for *Ecore*-PMIF), is shown in Figure 4. It can be seen as the MDE version of the PMIF 2 metamodel presented in [2] (Fig. 3), with some minor changes.

A QueuingNetworkModel is composed of one or more Workloads, zero or more Arcs, one or more Nodes and one or more ServiceRequests. The Arc class

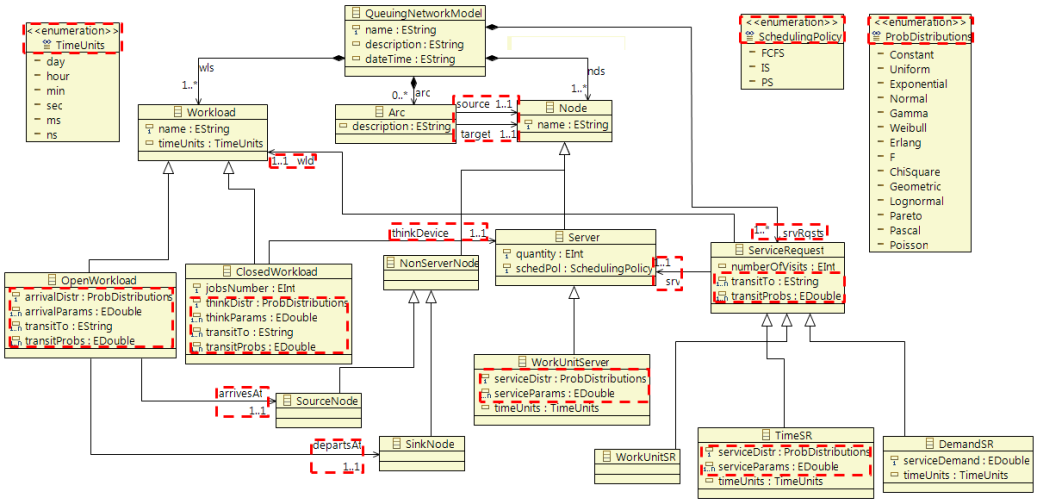


Figure 4: ePMIF metamodel (conforming to Ecore)

connects Nodes between them. In a queuing network, jobs flow from node to node. There are two types of nodes, Servers and NonServerNodes. The former provide a processing service, while the latter show the topology of the network and represent the origin (SourceNode) and exit point (SinkNode) of OpenWorkloads. The Server class has a specialization class, named WorkUnitServer, that represents resources with a fixed processing service for each Workload that makes a request for service.

A Server provides service for different Workloads, where a Workload represents a collection of jobs that make similar ServiceRequests from Servers. Depending on the type of queuing network (open or closed), there are two types of Workloads:

- OpenWorkload. It represents a set of jobs which arrive from the outside world, are serviced, and leave the system. The number of jobs belonging to an OpenWorkload at any given time is variable. A job represented by an OpenWorkload arrives at a SourceNode and departs at a SinkNode.

- ClosedWorkload. It represents a fixed population of jobs that circulates among the Servers. A ClosedWorkload has a Server which acts as thinkDevice and which is characterized by a think time.

A ServiceRequest associates Workloads with Servers. According to the relation from ServiceRequest to Workload and Server (one to one in both cases), a

ServiceRequest associates one (and only one) Server with one (and only one) Workload. In this way, when a job which is represented by a workload arrives at a server, the service request associated to the workload and the server specifies how the job will be treated in that server. There are three types of ServiceRequests (for all of them, the numberOfVisits is an optional attribute):

- WorkUnitServiceRequest. They are ServiceRequests associated to WorkUnitServers, so nothing about the service time has to be specified, since it is already in the WorkUnitServer.
- TimeServiceRequest. It specifies the service time that the jobs representing the Workload associated to the ServiceRequest will have in the associated Server.
- DemandServiceRequest. Similar to TimeServiceRequest, but service time is now specified in terms of service demand (service time multiplied by number of visits).

All these elements are equivalent to those in PMIF 2 (Fig. 3), apart from the following differences (they are marked with dotted boxes in Fig. 4). First, in PMIF 2, probabilities are specified as classes in the metamodel, while in our approach they have become attributes (to reduce the number of elements in the resulting models, mainly for performance reasons). Second, ServiceRequest is no longer an association class, and we have also unified the way to specify transitions in Workload and ServiceRequest classes (this will be very useful when specifying the behavior). Thus, in the PMIF 2 metamodel an element of type Transit was needed for each path in a fork; in our case, no matter how many paths a Workload may follow, we only need two attributes: transitTo and transitProbs. The former contains a sequence with the names of the Nodes where the Workload can transit. The latter contains a sequence with the probabilities of these transitions. Note that the order of the elements in both attributes has to match.

For example, suppose that in the network shown in Fig. 1(a), the probability of a job to transit from the CPU server to DISKA is 0.4375, to DISKB is 0.5, and to leave the system is 0.0625 (example taken from [41, page 572]). This is represented in our approach by setting the values of attributes transitTo and transitProbs of the ServiceRequest associated to the CPU server, to the sequences {DISKA, DISKB, SINK} and {0.4375, 0.5, 0.0625} respectively.

Another difference between ePMIF and PMIF 2 is how service and arrival times are specified. In PMIF 2, they are specified by attributes ArrivalRate and ServiceTime respectively. PMIF 2 assumes that these values represent parameters of Poisson and Exponential distributions, respectively. Given that we want to accept different probability distributions for service and arrival times, we have defined a new type (ProbDistributions) which is an enumeration with literals Constant, Uniform, Exponential, Normal, Gamma, Weibull, Erlang, F, ChiSquare, Geometric, Lognormal, Pareto, Pascal and Poisson. If the distribution is Constant, it means that the arrival/service time is constant.

The last difference between PMIF 2 and ePMIF is that we use references instead of attributes to refer to other objects. This has the advantage that references cannot be incorrectly specified. However, if objects are referred to by their names, it is easy to mistakenly write a String with a name that refers to a non-existent object. Furthermore, a change in the name of an object would result in an inconsistent reference.

4. MDE Essentials

MDE [3] is becoming a widely accepted approach for developing complex distributed applications. MDE advocates the use of models as the key artifacts in all phases of development, from system specification and analysis, to design and implementation. Each model usually addresses one concern, independently from the rest of the issues involved in the construction of the system. Model transformations define relationships between models, either at the same or at different level of abstraction. Thus we may have different kinds of transformations: correspondences, refinements, abstractions, development relations, etc. Domain Specific Languages (DSLs) are key in MDE for representing models. The benefits of using DSLs is that they provide intuitive notations, closer to the languages of the domain experts, in a compact and precise way, and at the right level of abstraction.

A DSL is defined in terms of three basic components: abstract syntax, concrete syntax and semantics.

The *abstract syntax* of a DSL is normally specified by a metamodel, which describes the concepts of the language, the relationships between them, and the structuring rules that constrain the model elements and their combinations in order to represent the domain rules. In our case, it is the ePMIF metamodel shown in Fig. 4.

The goal of the *concrete syntax* of a DSL is to provide users with a notation close to the one they normally use, in this case to specify queuing network models. The concrete syntax is normally defined as a mapping between the metamodel concepts and their textual or graphical representation. For visual representations, as in our case, it is necessary to establish links between these concepts and the visual symbols that represent them. We chose those visual symbols which are as intuitive as possible for representing QNM concepts (server, workload, service request, etc.). Some of them, like the icon used to represent servers, are widely used in the QNM literature. Others, like service requests, are new concepts that have recently appeared, e.g, with the definition of PMIF.

The concrete syntax of xQNM will be described in Section 5.1. It has been realized using the Eclipse Graphical Modeling Framework (GMF) [42]. GMF provides a generative component and runtime infrastructure for developing graphical model editors for DSLs. It automatically generates an Eclipse plugin with a DSL diagram editor from (1) the DSL metamodel (abstract syntax); (2) a graphical definition (concrete syntax); (3) a tooling definition (i.e., the buttons that enable the creation of the model elements); and (4) a mapping model relating the three previous artifacts.

Finally, the *semantics* of a DSL describes the precise meaning of its models, and in case of DSLs for dynamic systems, it defines their behavior. One way of specifying the behavior of a DSL is by describing the evolution of the modeled artifacts along a time model. In MDE, this can be done using model transformations supporting in-place update [43]. The behavior of the DSL is then specified in terms of the permitted actions, which are in turn modeled by the transformation rules.

4.1. e-Motions

The environment we have used for specifying the behavior of queueing network models, named *e-Motions* [6, 7], is a Domain Specific Language supported by a graphical framework developed for Eclipse that supports the specification, simulation, and formal analysis of real-time systems. Thanks to the use of MDE techniques in our approach, we have been able to reuse the *e-Motions* environment and integrate it in our tool architecture, as we shall see in next section.

e-Motions extends in-place transformation rules with a quantitative model of time and with mechanisms that allow designers to specify action-based properties, thus facilitating the design of real-time systems. While there are

several approaches that propose in-place model transformation rules to deal with the behavior of DSLs (see [44] for a brief summary of such approaches), *e-Motions* provides a very intuitive and natural way to specify behavioral semantics, close to the language of the domain expert and the right level of abstraction [45]. These transformations are composed of a set of rules, each of which represents a possible *action* of the system. Similar to Graph Grammars [46], these rules are of the form $l : [\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$, where l is the rule's label (its name); and LHS (left-hand side), RHS (right-hand side), and NAC (negative application conditions) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the precondition for the rule to be applied, whereas the RHS one represents its postcondition, i.e., the effect of the corresponding action. Thus, a rule can be applied, i.e., triggered, if an occurrence (or match) of the LHS is found in the model and none of its NAC patterns occurs. Generally, if several matches are found, one of them is non-deterministically selected and applied, producing a new model where the match is substituted by the appropriate instantiation of its RHS pattern (the rule's *realization*). The model transformation proceeds by applying the rules in a non-deterministic order, until none is applicable — although this behavior can be usually modified by some execution control mechanism [44].

e-Motions allows attributes to be added to rules to represent features like duration or periodicity. OCL [47] expressions can be used to specify values for attributes, variables, conditions, etc. *e-Motions* also implements a reflective mechanism that allows to explicitly represent *action executions*, which are model elements that describe actions that have been performed, or are currently executing. They specify the type of the action (i.e., the name of the atomic rule), the identifier of the action execution, its starting and ending time, and the set of objects involved in the action. This provides a useful mechanism when we want to check whether an object is participating in an action, or if an action has already been executed, for instance.

Finally, a special kind of object, named Clock, represents the current global time elapse. Designers are allowed to use it in their rules (using its attribute time) to know the amount of time that the system has been working. Further clocks can be specified by users, according to the requirements of their systems (to model, e.g., systems with several distributed clocks).

The semantics of xQNM will be then defined by a set of rules in *e-Motions*, which specify the behavior of xQNM models. Note that this is nothing but the *generic* behavior of Queueing Network Models. In fact, the behavior

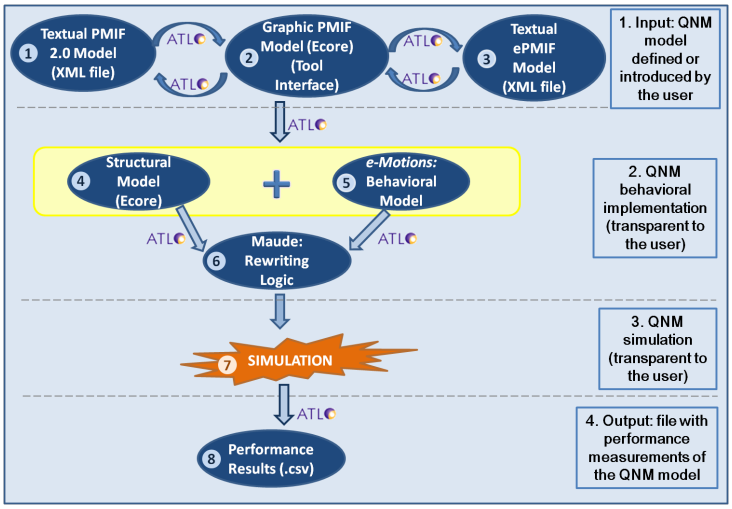


Figure 5: xQNM Architecture

described in next section has been defined once, and serves for any QNM.

5. xQNM overview

Fig. 5 shows the basic elements of xQNM, and how to use them to specify and simulate QNMs.

The concrete syntax of the language and the tool support we have built for editing xQNM models (that is, QNMs) is described first in Section 5.1. It corresponds to the ovals numbered 1, 2 and 3 in the figure. Section 5.2 explains how the behavior of queuing network models can be specified with *e-Motions*. It provides the *semantics* of the xQNM language, and corresponds to the ovals 4 and 5 in the figure.

Once we have an initial model of a QNM and the behavioral dynamics of QNMs specified in *e-Motions*, we can simulate it. In fact, *e-Motions* translates its specifications (using ATL transformations from ovals 4 and 5 to oval 6 in Fig. 5) into the corresponding formal specifications in Real-Time Maude [48], which in turn provides semantics to the visual *e-Motions* specifications of the system. Maude specifications are executable, and therefore they can be used to run simulations. Section 6 describes the simulations that are possible with xQNM, how they are realized, and how results are returned to the user.

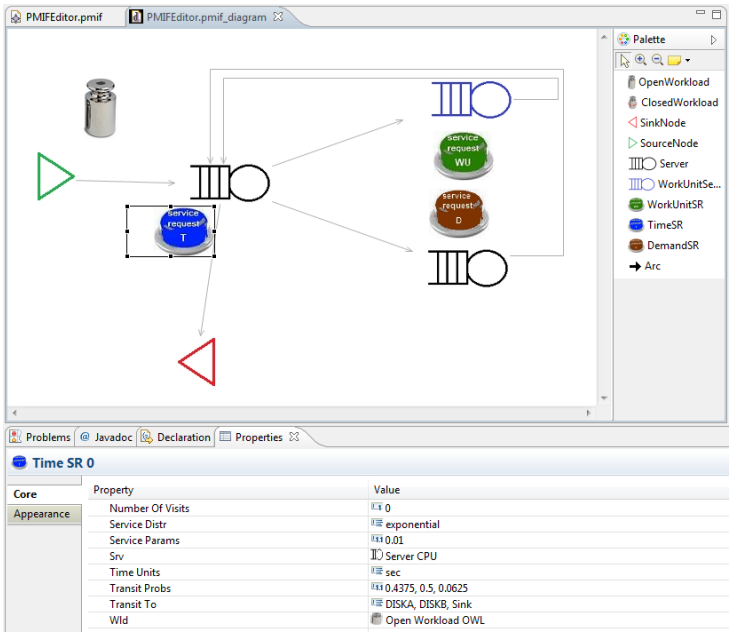


Figure 6: xQNM Graphical Editor

5.1. A Tool for Drawing and Simulating QNMs

Our DSL is supported by a tool which provides a graphical editor for creating queueing networks conforming to PMIF or ePMIF metamodels. It means that it can be defined open, closed and mixed network models in the graphical interface. At this moment, only open and closed networks can be simulated in xQNM. This section explains the capabilities provided by this tool.

5.1.1. QNMs graphical definition

The graphical editor of our tool has been developed using GMF. Fig. 6 contains a snapshot of our editor, with the graphical representation of the QNM model showed in Fig. 1(a). The different kinds of network objects (OpenWorkloads, ClosedWorkloads, Servers, WorkUnitServers, etc.) can be selected from the menu on the right and be placed on the main panel.

The properties of objects (attributes and references) are specified in the lower panel. To assign values to the sequences of transitions, the user has to select the object and click on the attribute in the lower panel. A new window where the values can be introduced is shown in Fig. 7. Probability

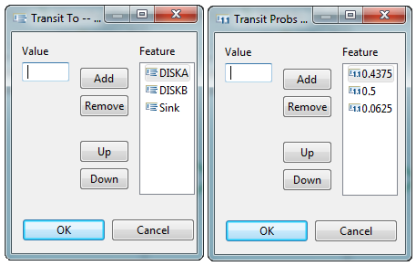
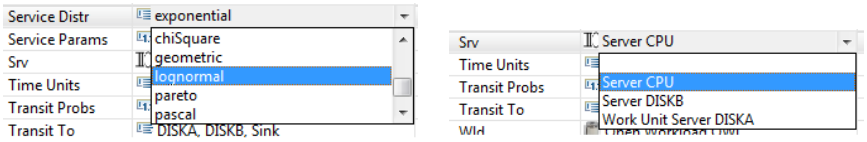


Figure 7: Assigning values to sequences



(a) Specifying the distribution.

(b) Selecting the Server associated to the ServiceRequest.

Figure 8: Drop down lists in the lower panel.

distributions are specified as attributes of type ProbDistribution (Fig. 8(a)). References to objects (that model for example transitions) are indicated using drop down lists (Fig. 8(b)).

As in any GMF project, xQNM models admit two representations, each one stored in a different file. One contains the graphical information, and can be edited with our graphical tool. The second one is plain XML file that contains the model elements, and can be edited with the standard Eclipse tree-view model editor. The user can select either of them in the left panel.

5.1.2. Exporting QNMs

Once a queuing network model is defined with the graphical editor, it can be exported to an XML file with its ePMIF representation. The XML is similar to the PMIF 2 XML file, with the corresponding extensions for transitions and probability distributions. Thus, there are no ArrivalRate, ServiceTime and ThinkTime attributes anymore; but ArrivalDistr, ServiceDistr and ThinkDistr. Objects containing any of these attributes also contain one or more attributes named Param that specify the parameters of the distributions. For instance, let us consider the example shown in Fig. 1(a) and described in Section 3 of an open QNM with a CPU and two disks: A and B. Distributions for service times are supposed to be Gamma (for disk A) and Exponential (for disk B),

and Poisson for arrival times. Listing 1 shows the XML file that has been exported from the definition of this open network model using our tool.

Listing 1: ePMIF XML File

```
<QueueingNetworkModel Name="Jain572" Description=
 "Ecore XML PMIF" Date-Time="040711">
  <Workload>
    <OpenWorkload WorkloadName="OWL" ArrivesAt="Source"
     DepartsAt="Sink" ArrivalDistr="Poisson"
     TimeUnits="sec">
       <Transit Probability="1.0" To="CPU"/>
       <Param Value="3.0"/>
    </OpenWorkload>
  </Workload>
  <Node>
    <Server Name="CPU" Quantity="1"
     SchedulingPolicy="FCFS"/>
    <Server Name="DISKB" Quantity="1"
     SchedulingPolicy="FCFS"/>
    <WorkUnitServer Name="DISKA" Quantity="1"
     SchedulingPolicy="FCFS" TimeUnits="sec"
     ServiceDistr="Gamma">
       <Param Value="0.5"/>
       <Param Value="2.0"/>
    </WorkUnitServer>
    <SourceNode Name="Source"/>
    <SinkNode Name="Sink"/>
  </Node>
  <ServiceRequest>
    <DemandServiceRequest ServiceDemand="2592.0"
     TimeUnits="sec" WorkloadName="OWL"
     ServerID="DISKB" NumberOfVisits="86400">
       <Transit Probability="1.0" To="CPU"/>
    </DemandServiceRequest>
    <WorkUnitServiceRequest WorkloadName="OWL"
     ServerID="DISKA">
       <Transit Probability="1.0" To="CPU"/>
    </WorkUnitServiceRequest>
    <TimeServiceRequest TimeUnits="sec"
     WorkloadName="OWL" ServerID="CPU"
     ServiceDistr="Exponential">
       <Param Value="0.01"/>
       <Transit Probability="0.4375" To="DISKA"/>
       <Transit Probability="0.5" To="DISKB"/>
       <Transit Probability="0.0625" To="Sink"/>
    </TimeServiceRequest>
  </ServiceRequest>
  <Arc FromNode="Source" ToNode="CPU"/>
  <Arc FromNode="CPU" ToNode="DISKA"/>
  <Arc FromNode="CPU" ToNode="DISKB"/>
  <Arc FromNode="CPU" ToNode="Sink"/>
  <Arc FromNode="DISKA" ToNode="CPU"/>
  <Arc FromNode="DISKB" ToNode="CPU"/>
</QueueingNetworkModel>
```

Our tool also supports the exportation to standard PMIF 2 XML format, as long as the distributions are those supported by PMIF 2.

5.1.3. Importing QNMs

The xQNM tool offers the possibility to import PMIF 2 files. These files are transformed into the corresponding ePMIF files, modifying the attributes as required. Thus, attributes ServiceTime and ThinkTime are automatically translated into the corresponding serviceDistr and thinkDistr attributes. New attributes serviceParams and thinkParams are created with the values of ServiceTime and ThinkTime attributes, respectively. The same happens with the ArrivalRate attribute in PMIF 2, which is translated to an arrivalDistr attribute (of type Poisson in this case).

It is also possible to import plain ePMIF XML files to the xQNM tool (i.e., with no graphical information). The ATL transformation used for this is the opposite to the one used for the exportation of ePMIF models, explained above.

When plain XML files containing either PMIF or ePMIF models are imported into our tool, a file containing the new model generated is created. Having no graphical information about the model, this file only accepts the visualization using the Eclipse tree-view editor. From this file, the user can generate another file so that the model can be deployed in the graphical editor. The elements will initially appear in a random position in the graphical editor (since no graphical information is available), and the user is then free to arrange them as preferred.

5.2. A Generic Behavioral Model for QNMs

The *generic* behavioral model for open and closed QNMs is defined in terms of a set of *e-Motions* rules. Note that users of the xQNM tool do not need to be aware of such behavioral model. Mixed queuing network models defined using the graphical user interface of xQNM cannot be simulated at this moment.

5.2.1. QNMs structural model and Observers addition

PMIF models describe the structure of the QNM, and can be used to specify the dynamics of QNMs in terms of job flows. However, we also need to specify, record and manage additional information to deal with the performance properties of the system. For these tasks we use *observers*.

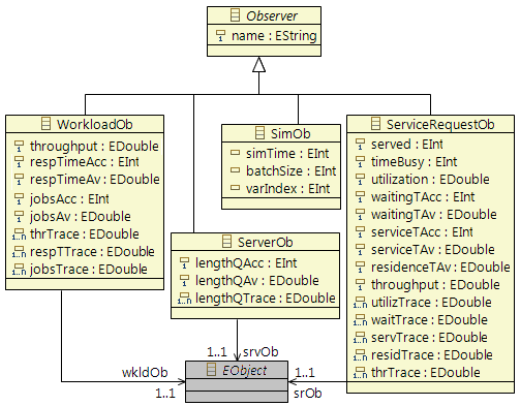


Figure 9: *Observers* metamodel

Observers were introduced in [49, 50] as an effective means to specify the non-functional properties of systems described by high-level DSLs. An observer is an object whose purpose is to monitor the state of the system objects and actions. Observers, as any other objects, have a state and a well-defined behavior. The attributes of the observers capture their state, and are used to store the variables that we want to monitor.

To introduce observers into the behavioral rules of xQNM (in order to specify and measure the performance properties of QNMs), we need to specify a metamodel for them. This is shown in Fig. 9. The idea is to combine both metamodels (Figs. 4 and 9) so that observers can be used in our behavioral rules. In fact, since *e-Motions* allows users to merge several metamodels in the definition of a DSL, we can define the observers metamodel in a non-intrusive way, i.e., we do not need to modify the system metamodel to add attributes that store the values of the non-functional properties we want to monitor.

In the observers metamodel we can see that there are three types of observers for monitoring the performance metrics of a different type of object. We have WorkloadOb for monitoring Workloads, ServerOb to monitor Servers, and ServiceRequestOb to monitor TServiceRequests. These three have a reference to class EOject, which points to the object they monitor. In addition, we have the SimOb observer, which stores the simulation run parameters introduced by the user (see Section 6.1).

The aim of WorkloadOb observers is to monitor performance properties of workloads. The idea is to associate one observer of this type to each workload.

Its attributes are used to measure the average throughput (thoughputAv), response time (respTimeAv) and jobs (jobsAv) of the associated workload. It also contains three sequences (thrTrace, respTTrace and jobsTrace) that store the traces with the values for throughput, response time and jobs average, respectively, at different times of the simulation.

ServerOb observers monitor servers. They store the average queue length in their attribute lengthQAv, and keep the traces in attribute lengthQTrace. Attribute lengthQAcc is used to compute lengthQAv. As explained in [41], the queue length of a server considers the jobs in the queue and the jobs being served.

Each service request in the model will have a ServiceRequestOb observer associated to it. Considering that a service request is the relationship between a server and a workload that requests its service, the data monitored by this observer represents the performance relationship between them. In this way, when we mention workloads (or jobs belonging to them) and servers in the explanation of the attributes, we mean the workloads (or jobs) and servers associated to the service request. ServiceRequestOb observers have several attributes:

- served. Number of jobs processed by the server.
- timeBusy. Time that the server has been busy (processing jobs).
- utilization. Percentage of the time that the server has been busy.
- waitingTAcc and waitingTAv. Sum and average waiting times in the queue of the jobs processed by the server, respectively (the waiting time of a job is the time between the arrival of the job to the server queue until it starts being processed).
- serviceTAcc and serviceTAv. Sum and average service time of all jobs processed by the server.
- residenceTAv. Average residence time of all jobs processed by the server (the residence time of a job is the time between the job enters the server queue and leaves the server).
- throughput. Number of jobs processed by the server per unit of time.
- utilizTrace, waitTrace, servTrace, residTrace and thrTrace. These attributes keep the traces of the corresponding values throughout the simulation.

5.2.2. QNMs behavioral model

This section introduces the *e-Motions* rules that describe the behavior of QNMs. Basically there is one rule for jobs entering the network (EnterOpenWLFnT), one for specifying how jobs transit between servers (TransitJobsnT), and a third one for jobs leaving the network (ExitOpenWLF). For efficiency reasons there are variations of these rules when there is only one server to which the jobs can transit to (so no decisions are to be made). In addition, two rules are in charge of specifying how the values of global observers are updated.

These rules are briefly described here. For a complete description of all the rules, the interested reader can consult [51]. In any case, the rules are completely transparent to the xQNM user, they just specify the behavior of the system, and allow to simulate it.

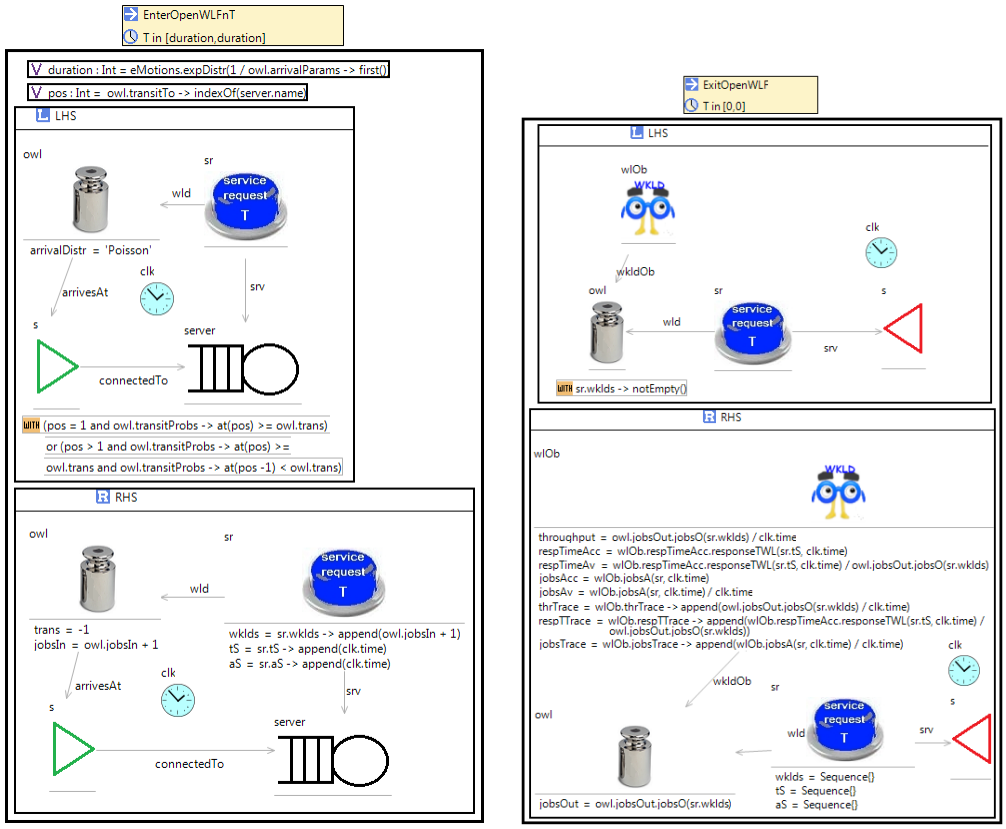
a) A set of jobs enter the network. Rule EnterOpenWLFnT (Fig. 10(a)) models how OpenWorkload objects enter the network, when they can transit to more than one server. The rule has in both LHS and RHS patterns the OpenWorkload to which the job belongs (owl), the Server to which the job transits to (server), the ServiceRequest that relates both of them (sr), and the Source node at which jobs belonging to the OpenWorkload enter (s). There are also the relationships between these objects (wld, arrivesAt, srv and connectedTo).

The destination server is determined by variable pos and the OCL condition in the LHS. It uses the transition probabilities. A new job entering the system is modeled by the addition of a new identifier to the wklds sequence of the TServiceRequest, and the addition of the current time elapse to the tS and aS sequences. Variable duration specifies the duration of the rule: in this example it follows a Poisson distribution (see the variable duration declaration in the top left corner of the rule).

There is also a similar rule for OpenWorkloads whose jobs always transit to the same Server when they enter the Source node. That rule, called EnterOpenWLF1T [51], is a simplified version of the EnterOpenWLFnT rule that we have developed for performance reasons (because no OCL expressions or conditions need to be computed in this case).

b) Transition of jobs between servers. Rule TransitJobsnT (Fig. 11) models the transition of jobs between servers (and also from a Server to a Node of type SinkNode). Jobs can belong to either OpenWorkloads or ClosedWorkloads, so this rule is used for both.

The LHS of rule TransitJobsnT contains all the objects needed for this



(a) EnterOpenWLFnT rule (b) ExitOpenWLF Rule

Figure 10: Rules for packets entry and leaving.

rule to be triggered: the source Server (s), the target Node (n, which is either a Server or a SinkNode), the Workload (wl) to which jobs belong, the TServiceRequests associated to the mentioned elements (srS and srT), and the Observers (srSOb and sOb) whose attributes are to be updated in the RHS of the rule. The three sequences representing the jobs in the source TServiceRequest (srS) are also updated in the RHS by eliminating the corresponding jobs and adding them to the sequences of the target TServiceRequest (srT). The attributes of the two observers are also updated.

c) Jobs leave the network. Rule ExitOpenWLF (Fig. 10(b)) models how jobs leave the network. Consequently, it is applied only over OpenWorkloads. When the TServiceRequest (sr) contains jobs, this rule is fired and the corresponding attributes in the OpenWorkload (owl) and the observer associated

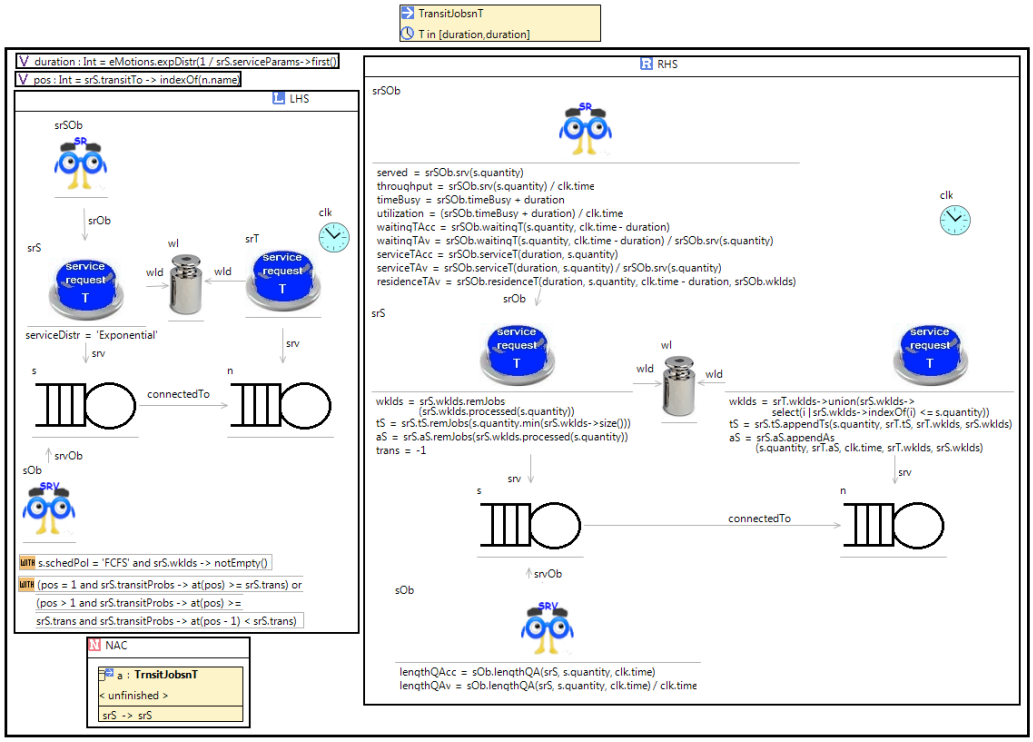


Figure 11: TransitJobsnT rule

to it (wlOb) are updated. The jobs present in the TServiceRequest (sr) are deleted, modeling that they have left the network.

This rule updates the attributes of the observers, namely thrTrace, RespTTrace and jobsTrace, every time a job leaves the system. The new values correspond to the calculated throughput, mean response time and jobs average, which are appended to the sequences with the traces.

Similar to rule ExitOpenWLF, another rule is in charge of updating the attributes of observers associated to ClosedWorkloads. The attributes are the same, apart from the one for the average number of jobs, which is no longer necessary.

Finally, another rule, UpdateTraces (not shown here for brevity), is defined to update the attributes for traces in the other observers. They are updated either when jobs leave the system (in OpenWorkloads) or when jobs arrive at the centralSrv (for ClosedWorkloads).

It is important to recall that users do not need to write these rules,

they have been defined once and apply to all QNMs. In fact, they can be seen as providing a behavioral semantics of QNMs by explicitly specifying the behavior of QNMs in a language with well-defined semantics [52]. In addition, they are all automatically configured and generated according to the type of network defined by the user, and to the probability distributions used.

5.2.3. Generating the behavioral rules

Once the user inserts a model within the xQNM tool either by drawing it with the graphical interface or by importing it, it is automatically translated to its structural and behavioral models. This is done by the ATL transformation shown in Fig. 5 from oval 2 to ovals 4 and 5.

The transformation has two main parts, the generation of the structural model and the generation of the behavioral model. For the former, the transformation takes the ePMIF model conforming to the ePMIF metamodel and transforms it into an more compact representation of ePMIF that we use internally with *e-Motions*. Although the first version of xQNM used ePMIF directly, we realized that for performance reasons we could optimize this representation to make it more compact and efficient. This was very important for conducting the simulations. Such new representation is internal to our tool and transparent to users, who still use ePMIF models to describe their QNM models. That metamodel and the changes with respect to ePMIF are described in detail in [51].

For generating the behavioral model, the transformation identifies the type of queuing network used (open or closed) and selects the appropriate rules among the ones presented in Section 5.2.2, which are available in a repository. Those rules, as well as the ePMIF model are the input parameters of the ATL transformation. The transformation also adjusts some features of the rules according to the probability distributions used in the model, which is reflected in the rules duration, or the performance metrics that the user wants to monitor. Regarding the latter, only those metrics are filtered by the ATL transformation and appear as objects attributes in the final rules (Figure 12 shows how such parameters are specified by the user). It means that simulations where less parameters are to be monitored are faster.

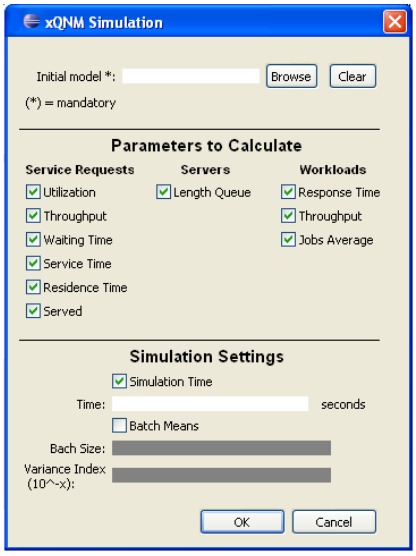


Figure 12: xQNM Simulation Window

6. Experimentation

6.1. Simulation in xQNM

Once we have the behavioral dynamics of a QNM specified in *e-Motions*, we are ready to simulate it. Our environment supports the translation of the specifications (ATL transformations from ovals 4 and 5 to oval 6 in Fig. 5) into the corresponding formal specifications in Real-Time Maude [48].

In Maude, the result of a simulation is the final configuration of objects reached after completing the rewriting steps, which is nothing but a model. The semantic mapping as well as the transformation process back and forth between the *e-Motions* and Real-Time Maude specifications is described in detail in [44], although it is completely transparent to the *e-Motions* (and so xQNM) user. The user, consequently, is completely unaware of the Maude rewriting engine performing the simulation.

A very important advantage of our approach is that observers are also objects of the system, and therefore we can retrieve the values of their attributes after the simulation is conducted to know how the system behaved. In fact, this is crucial for the approach we are presenting here.

When the user wants to launch a simulation in xQNM, the window shown in Figure 12 is displayed. Users have to specify the input queuing network model to be simulated. Moreover, they have to specify which performance

measures want to obtain as output for each service request, server and workload. Since the behavioral rules presented in Section 5.2 are available in a repository, the ATL transformation from oval 5 to oval 6 filters only those attributes that the user wants to monitor. Besides, depending if the network model is open or closed, the transformation filters the appropriate rules to be used in the simulation. Finally, the stopping criteria has to be determined. It can be established either by the desired simulation time or the method of the batch means. The settings for the stopping criteria are stored in the simOb observer (Figure 9).

One important issue of any simulation in any kind of system is the stopping criteria. The simulation should stop at a certain point where the performance parameters are accurate enough, and the system is stable. To be able to reach that point, an important requirement is Little's Law [41], which in a queueing network applies as long as, in average, the number of jobs entering the system are less or equal to those leaving it. It also implies that the average arrival rate of jobs should be lower than the service time, if we do not want the network to overflow.

Related to the stopping criteria of a simulation, we also need to determine its length, that is, for how long it should run. Thus, if the simulation is too short, the results will probably be unreliable. But if the simulation is too long, computing time and resources will be unnecessarily wasted. In most simulations, only the performance after the system reaches a stable state is of interest. The initial part, also called transient state or warm-up period, should not be included in the final computations. The problem of identifying the end of the transient state is termed as transient removal. Some of the common heuristic methods for transient removal are: long runs, proper initialization, truncation, initial data deletion, moving average of independent replications and batch means.

In our approach, we allow *long runs* and the *batch means* methods. We discarded the *proper initialization* method because it requires starting the simulation in a state close to the expected steady state. We cannot follow this approach because in principle we do not know the expected steady state. The *truncation* method is based on the assumption that the variability during the steady state is less than during the transient state, which is normally true. It considers that the data in the transient state is monotonous, i.e., continuously increasing or decreasing. However, we found out that our simulations of QNMs may have significant peaks during their transient states. Methods *initial data deletion* and *moving average of independent replications* require

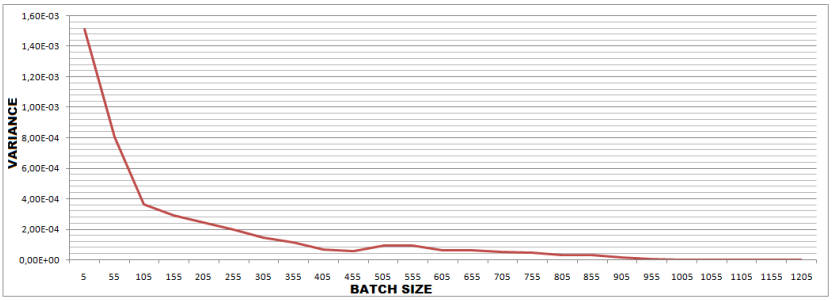


Figure 13: Method of the Batch Means

studying the overall average after some of the initial observations are deleted from the sample. These methods apply over several replications, which differ only in the seed values used in the random number generators, of a fixed size. The problem, again, is how to determine a priori the length of the replications. This makes them unappropriate for our proposal.

To determine when a simulation has to stop, the original method of *batch means* requires running a long simulation and later dividing it up into several parts of equal duration, which are called batches. The mean of the observations within each batch is called the batch mean. This method requires studying the variance of these batch means as a function of the batch size. But instead of running a very long simulation and later dividing it, what we do is to apply the method at certain points during the simulation as it moves forward. For that we store the values of the performance parameters in traces, and apply this method over them every time that N new jobs leave the system (or, in the case of closed networks, when N jobs complete a cycle). We consider that a simulation has reached the steady state when the variance of the batch means is in the order of 10^{-x}.

Fig. 13 shows a chart with the variance (Y-axis) plotted as a function of batch sizes (X-axis), using $N = 50$ and $x = 6$. In this example, the variance goes below 10^{-6} between steps 905 and 955, i.e, when 955 jobs have been processed. This means that the steady state of the system starts when 955 jobs have left the system (or completed a cycle in a closed network). Based on the experiments we have conducted with different networks, the default values we have currently assigned to these two variables are $N = 100$ and $x = 6$. Of course, these parameters can be easily configured by the user (Figure 12).

We run independent simulations, each one being stopped when it reaches

SERVICE REQUESTS

ServiceRequest	owl, CPU
Utilization	0,47365
Throughput	47,46225
Waiting Time	0,00823
Service Time	0,00992
Residence Time	0,01815
Served	38755

ServiceRequest	owl, DISKA
Utilization	0,41726
Throughput	20,76636
Waiting Time	0,01347
Service Time	0,02012
Residence Time	0,03359
Served	16850

ServiceRequest	owl, DISKB
Utilization	0,71737
Throughput	23,83625
Waiting Time	0,07586
Service Time	0,03017
Residence Time	0,10603
Served	19405

SERVERS

Server	CPU
Length queue	1,62958

Server	DISKA
Length queue	1,48269

Server	DISKB
Length queue	2,86924

WORKLOADS

OpenWorkLoad	owl
Response Time	1,37598
Throughput	3,06168
Jobs Average	5,96572

Figure 14: Results obtained for the QNM of Fig. 1(a)

its steady state as detected by the batch method. The performance values returned by each simulation are the values of the attributes of the observers defined for the model, at the end of the simulation. Given that we have reached the steady state, the values are stable. To compute the final result we take the average of these values, and the result is returned to the user.

When all the simulations have stopped and the performance results are ready to be returned to the user, a final ATL transformation is applied to the results. This transformation is shown between ovals 7 and 8 in Fig. 5. Its goal is to return the data in a format that can be easily consulted, managed and manipulated by the user. This is why we have chosen the *csv* (comma-separated values) format, which is readable by most spreadsheet applications. Figure 14 shows the results returned by our approach for our open queue network example.

6.2. Simulation in other tools

Table 2 displays the relevant features of some packages and tools regarding analysis and simulation of QNMs. For each one we explain how the performance results are shown to the user, the stopping criteria used by the tool (for tools that perform simulations) and the accuracy or confidence interval of the results.

Table 2: Simulation features of some packages and tools for QN modeling and analysis

Tool	Results presentation	Stopping criteria	Accuracy / Confidence interval
RESQME	Graphical and tabular results, and animation of the original diagram	Offers several methods to determine simulation run lengths: simulated time, number of departures from specified queues or nodes, etc.	User-defined confidence interval
SHARPE	Collection of visualization routines to analyze output results; results can be plotted. Excel spreadsheets can also be generated.	Steady state and transient computations	User-defined precision level (number of digits).
QNAP2	Screen textual output. Results can be saved in files	Users can simulate until some confidence interval is reached or a given simulation time expires	User-defined confidence interval
QSIM	Graphical interface	Simulation length control	Up to 95% confidence interval
SPE·ED	Graphical-interface	Batch means, simulation length control, number of jobs	User-defined confidence interval
PEPSY-QNS	Textual files	Offers several methods for identifying steady states, including *batch means*	Some methods require users to input the desired accuracy
TANGRAM-II	Textual files generated during the simulation	Offers several methods for identifying steady states, including *batch means*	User-defined confidence interval
PDQ	Results shown textually, by means C code (displayed in console or saved into a file	N/A (it only performs analysis)	Up to six decimal digits of precision
MQNA	Results are shown textually	Product-form QNs are solved analytically, and non product-form QNs are solved outside MQNA (PEPS and SMART use iterative methods)	Up to 10 digits precision in non-congested models
WinPEPSY-QNS	Textual files	Batch means	Errors smaller than 0.15% compared to the exact solution in several experiments
JINQS	Textual files	Warm-up period specified by the user; there is no built-in mechanism for detecting an approximate steady state	User-defined confidence interval
JMT	Graphical interface. Results can be exported in XML format	Implements transient detection using the R5 heuristic [53] and the MSER-5 [54] stationarity rule. Then it uses variable batch sizes. It can also perform long-run simulations for the case of models with heavy-tail distributions	User-defined confidence interval
qnetworks	Results returned as GNU Octave vectors or matrices, with values shown programmatically	Steady state and transient computations	Exact results for product-form QNs
xQNM	Textual files (*cvs* format)	Batch means or user-defined number of jobs	User defined confidence level

There are tools based on simulation that need to know when the steady state of the simulation starts, i.e., the warm-up period must be specified by the user and will typically be based on observations of pilots of the model. An example of such tools is JINQS, where there is no built-in mechanism for detecting when a simulation is close to a steady state. In this tool, when the approximate warm-up period is known, the simulation method can be optionally parameterized by this warm-up period.

Other tools implement some sophisticated methods to detect when the steady state is reached. In this way, JMT implements transient detection using the R5 heuristic [53] and the MSER-5 [54] stationarity rule. Then it uses variable batch sizes and a fixed amount of memory until the confidence intervals are generated with enough accuracy. It can also perform long-run simulations for the case of models with heavily-tail distributions. Some other tools also use the batch means methods, like TANGRAM-II [15], or WinPEPSY-QNS [55], and so does ours.

Some tools offer different ways to determine the accuracy and confidence level of the solution. For example, JINQS offers an optional parameter called confidence interval. If none is supplied, a value of 0.05 is assumed. If the logged measures are dependent and/or are not normally distributed, the computed confidence interval will be inaccurate. If the replications are independent, mean values will be approximately normal, but variances and other measures may not be.

Normally, tools that carry out analytical methods offer a great accuracy. This is the case of QNAP2 [9], which satisfies the confidence interval introduced by the user; the PDQ Analyzer [56, 10], which offers up to six decimal digits of precision; SHARPE [16, 57, 22], whose output precision is determined by the option "Number of digits printed" in the output; or MQNA [58], whose outputs have up to ten digits precision in non-congested models. As explained in the previous section, our tool uses the batch means method in each replication. Simulations start with a very small batch size and then they increment it until the variance of the batch means goes below 10^{-6}.

Results are presented in very different ways depending on the tool. Many tools display the results in plain text following some template, like QNAP2, TANGRAM-II, JINQS, PDQ and MQNA. Others are provided with a graphical user interface, which shows the results (SHARPE, JMT and WinPEPSY-QNS). RESQME [12, 59, 60] is even capable of animating the models as the discrete event simulations progress. Our tool outputs the results in a textual format readable by spreadsheet applications (*csv* format, Figure 14).

Table 3: Analysis comparison. RP: Routing Probabilities, NV: Number of Visits

Tool	Utilization	Throughput	Wait T.	Serv T.	Res T.	Queue L.
PDQ (RP)	0.09	03.0	0.003	0.03	0.0330	0.099
PDQ (NV)	0.72	24.0	0.056	0.03	0.0857	2.571
PEPSY (RP)	0.72	24.0	0.077	0.03	0.107	1.851
PEPSY (NV)	0.72	24.0	0.077	0.03	0.107	1.851
JMT (RP)	0.69	23.4	0.158	0.03	0.187	2.597
xQNM (RP)	0.72	23.8	0.076	0.03	0.106	2.869
Theoretical	0.72	24.0	0.077	0.03	0.107	2.571

Regarding the time that these packages and tools take to get the performance metrics, analytical methods are of course much faster than simulations. For example, QNAP2, PDQ, SHARPE, qnetworks or MQNA take less than a few seconds to obtain the results. On the contrary, RESQME, PEPSY-QNS and WinPEPSY-QNS may take from some minutes up to several hours to obtain the results, depending on the complexity of the input model. Our tool also uses simulation, and thus it may take from a few seconds to several hours depending on the size of the model.

6.3. Analysis comparison among tools

Once we have shown simulation and analysis features of some tools, in this section we run our case study in some of them to see the differences between them and our tool. The queuing network model is that of Figure 1(a). In this analysis comparison, we are going to focus on the performance measures obtained for DISK B.

For the analysis comparison, we have used WEASEL [28] and JMT [20]. WEASEL is based on PMIF and, consequently, the available elements to be drawn and the configuration parameters for those elements are very similar to those in xQNM. On the other hand, JMT is much more powerful in terms of available elements and configuration parameters; it offers the possibility to include in the model many elements not specified in PMIF: forks, joins, delays, routing stations, etc. The configuration parameters for the elements are also much larger: different load strategies for servers, many routing strategies available, etc. Since we are only dealing with the PMIF capabilities, we only use a small subset of JMT. The results for each tool run are shown in Table 3. The references used to compare the results of the different runs to check their accuracy were the theoretical results available in Jain's book [41].

In the table, *RP* stands for *routing probabilities* and *NV* for *number of visits*. These correspond with the routing criteria followed in the runs. The PDQ Analyzer, executed by means of WEASEL, does not accept routing probabilities. Thus, when we run the experiment with routing probabilities (because it is possible to define routing probabilities with the graphical user interface of WEASEL, independently of the tool used afterwards to solve the models), the results were erroneous. This is because it only considers number of visits, so it considered that the number of visits in every server was 1. We then changed the criteria to number of visits, and the results were all the same as the reference apart from the waiting and residence times, which significantly differed. WEASEL offers the possibility to solve the models with PDQ using exact solutions, approximate solutions and canonical solutions. Our experiment was run with the canonical solution because the other two do not accept open networks.

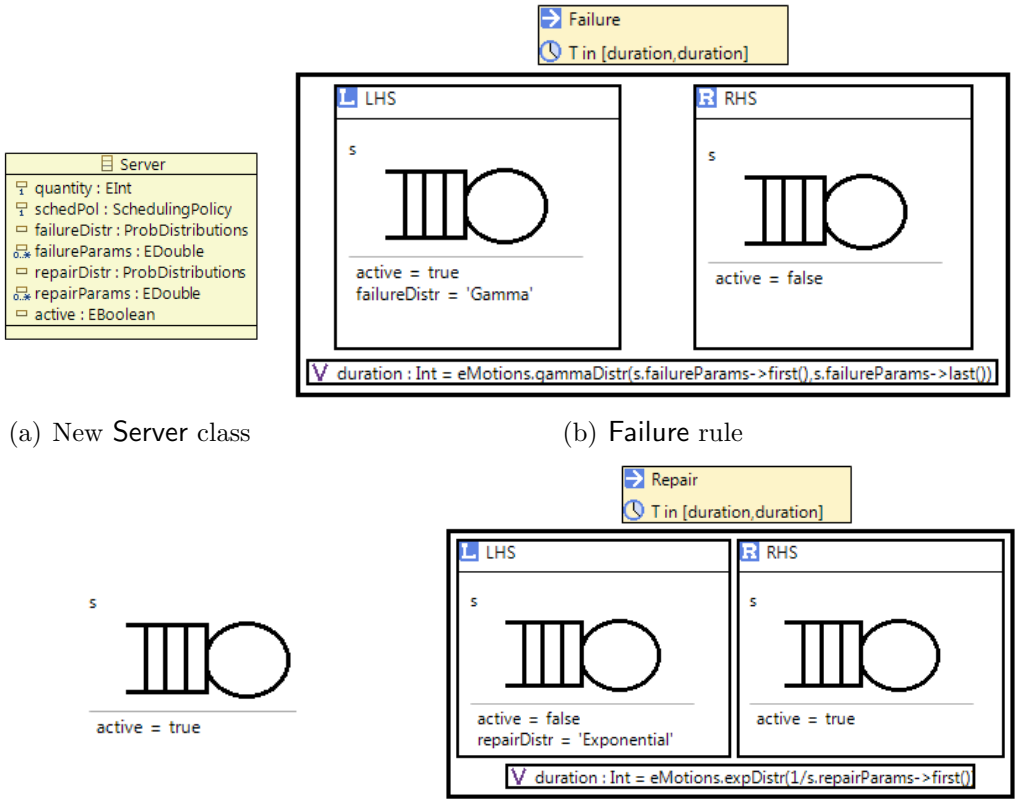
The results with PEPSY-QNS [61] were also obtained by means of WEASEL. PEPSY-QNS offers different solving methods in WEASEL, and we used sopenpfn. We run the experiment with both number of visits and routing probabilities and they were the same, so this tool is capable of dealing with both. All the measures obtained with this tool coincided with the reference except for the queue length, which was smaller.

In WEASEL, it is not possible to specify the performance metrics to be monitored. JMT accepts probabilities as routing strategy, apart from random, round robin, join the shortest queue, shortest R time, least utilization and fastest service. The output presented by the JMT tool is very intuitive, complete and easily readable. It offers statistics, including a chart, for each metric of each server. Furthermore, the user can specify which metrics he/she wants to monitor for which server.

The penultimate row of the table contains the results obtained with xQNM. The accuracy of the results is very good, which shows that the behavioral model that defines for QNM is faithful and accurate.

6.4. Considering failures

Once we have modeled the behavior of QNMs and are able to analyze their performance metrics, we are interested in extending their behavior in order to consider more realistic situations. In this regard, we want to take into account failures and repairs in networks' servers, as they happen in real life. Thus, after starting all the servers operative, they can fail at some point and be inactive for a while before they are repaired and back to service. We



(a) New Server class

(b) Failure rule

(c) Server in TransitJobsnT's LHS

(d) Repair rule

Figure 15: Extensions for considering failures.

have to consider times to failure and times to repair. These are normally modeled with exponential distributions [32], so that analytical calculations are possible. However, since we can include many probabilistic distributions to model this behavior, the modeler can choose any of them.

In order to extend the behavior of our DSL for modeling and analyzing QNMs with failures, we simply need to do two things: extend the ePMIF metamodel and add a couple of very simple behavioral rules. Only the Server class needs to be extended in order to include rates for failures and repairs in servers (Fig. 15(a)). The new attribute active is true whenever the server is operating, and false when it is not. Attributes failureDistr and repairDistr dictate the distribution followed by the time failures and repairs happen, respectively, while attributes failureParams and repairParams contain the pa-

rameters of such distributions.

Rule Failure (Fig. 15(b)) models the failure of a server. It is the only rule that needs to be added for modeling such failures. In this case, the distribution followed by the time to failure is Gamma. It can follow any distribution in the ProbDistributions enumeration type (Fig. 4). In the rule's RHS the active attribute is turned to false, modeling the inactivity of the server. A slight modification needs to be carried out in the LHS of rule TransitJobsnT (Fig. 11) to launch it only if the server s is active (Fig. 15(c)). A similar rule is included for repairing a server, which takes a server which is inactive and activates it. Such rule is shown in Fig. 15(d), where it considers a repair rate that follows an Exponential distribution.

We have included these modifications and have carried out some experiments. We have made the times to failure and repair follow exponential distributions with rates 10 and 5, respectively. In general, jobs take longer in being processed and leaving the system, since they may need to wait in queues whose server is inactive, and have to wait until it is repaired. Furthermore, for the same arrival and services times of our case study [41, page 572], the network does not satisfy Little's Law anymore, so analytical calculation becomes very hard and complex. The reason is that this example was created so that Little's Law was satisfied for the arrival and service times established, and considering that servers never fail. This is, the number of incoming and outgoing jobs per time unit (throughput) with the servers being active all the time was the same, 3, once the steady state was reached. However, since Little's Law is no longer satisfied in this example when server failures are taken into account, no steady state is reached, and the performance measures for our network depend now on the number of incoming jobs. Simulation becomes crucial in this case. Thus, we have carried out an experiment where 100 jobs enter (and leave after being processed) the network, and have checked that the performance measures significantly change, even for such a small number of jobs. The throughput value is now 2.22, and it will decrease as the number of incoming jobs increases due to contention in queues. The theoretical response time for the network without failures is 1.41, while the new response time considering failures is 2.33.

Although in the example we have considered the same failure and repair rates for every server, each one could have been modeled to have different rates, since every server can have its own characteristics (as it happens in reality). Similarly, different probabilistic distributions can be used and more realistic values for failure and repair times could also be set. This flexibility

is one of the benefits that can be obtained by the use of appropriate DSLs for modeling complex systems.

7. Conclusions and Future Work

In this paper we have surveyed several tools for analyzing QNMs. We have shown how QNMs can be interpreted in another modeling domain, in this case the one provided by *e-Motions* for specifying and simulating real-time systems. Having a representation of QN models in that domain has allowed the easy definition of a DSL for the specification and simulation of general QNMs, and the use of the tools available in that domain. In particular, our proposal has provided several interesting advantages and results.

First, a generic behavioral model for QNMs has been defined by means of six *e-Motions* rules. They provide a behavioral semantics for QNMs, expressed in a high-level language with precise semantics and execution facilities. Such behavioral model has been easily extended with two more rules in order to model failures and repairs in servers, which allows to analyze more realistic situations. This also shows how simple and flexible the behavioral model of the QNM can be changed when it is defined by means of a DSL, incorporating new features by simply adjusting some high-level rules.

Second, we have obtained a prototype tool that allows to draw QNMs, automatically translate them to their behavioral representation and finally simulate them. Models can be depicted graphically in xQNM, and they can be exported to PMIF 2 and ePMIF models. PMIF 2 models can also be imported to our tool in order to simulate them or to represent them graphically. The tool, together with a set of examples, is available from [62]. The use of MDE techniques has enabled a modular architecture, which can be easily maintained and extended in future versions, since each of its parts can be independently improved. We have also shown how the existing de-facto standard metamodel for QNM representation and interchange can be incorporated into the MDE domain, and easily extended to take into consideration more powerful and flexible possibilities and system properties.

As future work, some new features could be added in new versions of xQNM. For example, it could return, as result, not the average of the different simulations, but a mixture of probability distributions (in case the behavior of the system is composed of several independent behaviors). We also plan on automatically distributing the simulations across several machines, by means of a concurrent and distributed solution that would use a

task farm approach [63], so that results would be collected faster. We are also considering to extend the behavior of our generic behavioral model for QNMs. For example, since we now take failures in servers into account, we could include some rules for re-adapting the network when jobs are waiting in queues whose server is inactive.

Acknowledgements. This work has been supported by Spanish Research Project TIN2011-23795.

References

[1] P. J. Denning, J. P. Buzen, The Operational Analysis of Queueing Network Models, ACM Comput. Surv. 10 (1978) 225–261.

[2] Smith, Connie U. and Lladó, Catalina M. and Puigjaner, Ramon, Performance Model Interchange Format (PMIF 2): A comprehensive approach to Queueing Network Model interoperability, Performance Evaluation 67 (7) (2010) 548–568.

[3] T. Stahl, M. Völter, Model-Driven Software Development: Technology, Engineering, Management, Wiley, 2006.

[4] Eclipse, Graphical Modeling Framework, `http://www.eclipse.org/modeling/gmf` (2008).

[5] D. Djuric, D. Gasevic, S. Fraser, V. Devedzic, The tao of modeling spaces, Journal of Object Technology 5 (2006) 125–147.

[6] J. E. Rivera, F. Durán, A. Vallecillo, A Graphical Approach for Modeling Time-Dependent Behavior of DSLs, in: Proc. of VL/HCC'09, 2009.

[7] Atenea, The e-Motions tool, `http://atenea.lcc.uma.es/E-motions` (2009).

[8] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, Quantitative System Performance: Computer System Analysis Using Queueing Network Models, Prentice-Hall, Inc., 1984.

[9] M. Veran, D. Potier, QNAP2: A portable environment for queueing system modelling, in: D. Potier (Ed.), Proc. of the International Conference on Modelling Techniques and Tools for Performance Analysis, 2004, pp. 5–24.

[10] N. J. Gunther, Analyzing Computer System Performance with Perl::PDQ, Springer, 2005.

[11] LS Computer Technology Inc., SPE·ED, `http://www.spe-ed.com` (2010).

[12] K. C. Chang, R. F. Gordon, P. G. Loewner, E. A. MacNair, The Research Queuing Package Modeling Environment (RESQME), in: Proc. of the 25th conference on Winter simulation (WSC'93), ACM, 1993, pp. 294–302.

[13] BGS Systems, BEST/1 Product Description, BE77-010-2 (Jan. 1977).

[14] H. Schwetman, CSIM: a C-based process-oriented simulation language, in: Proc. of the 18th conference on Winter simulation (WSC'86), ACM, 1986, pp. 387–396.

[15] E. de Souza e Silva, R. Leo, The TANGRAM-II Environment, in: Computer Performance Evaluation. Modelling Techniques and Tools, Vol. 1786 of LNCS, Springer, 2000, pp. 366–369.

[16] C. Hirel, R. A. Sahner, X. Zang, K. S. Trivedi, Reliability and Performability Modeling Using SHARPE 2000, in: Proc. of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, TOOLS'00, Springer, London, UK, 2000, pp. 345–349.

[17] T. Field, JINQS: An Extensible Library for Simulating Multiclass Queuing Networks V1.0 User Guide, `www.doc.ic.ac.uk/~ajf/Research/manual.pdf` (october 2010).

[18] T.-C. Horng, N. Anastasiou, T. Field, W. Knottenbelt, LocTrackJINQS: An Extensible Location-aware Simulation Tool for Multiclass Queueing Networks, Electronic Notes in Theoretical Computer Science 275 (2011) 93 – 104.

[19] M. Marzolla, The `qnetworks` Toolbox: A Software Package for Queueing Networks Analysis, in: Proc. of Analytical and Stochastic Modeling Techniques and Applications (ASMTA'10), Vol. 6148 of LNCS, Springer, 2010, pp. 102–116.

[20] M. Bertoli, G. Casale, G. Serazzi, JMT: performance engineering tools for system modeling, SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 10–15.

[21] M. Hlynka, List of Queueing Theory Software, `http://web2.uwindsor.ca/math/hlynka/qsoft.html` (2011).

[22] R. A. Sahner, K. S. Trivedi, Reliability Modeling Using SHARPE, IEEE Transactions on Reliability R-36 (2) (1987) 186–193.

[23] A. Feldmann, W. Whitt, Fitting mixtures of exponentials to long-tail distributions to analyze network performance models, in: Proc. of INFOCOM'97, Vol. 3, 1997, pp. 1096–1104.

[24] Smith, Connie U. and Lladó, Catalina M., Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation, in: Proc. of the First International Conference on Quantitative Evaluation of Systems, 2004, pp. 38–47.

[25] D. García, C. M. Lladó, C. U. Smith, R. Puigjaner, Performance Model Interchange Format: Semantic Validation, in: Proc. of ICSEA'06, 2006, pp. 47–52.

[26] C. U. Smith, L. G. Williams, A performance model interchange format, Journal of Systems and Software 49 (1) (1999) 63–80.

[27] P. Wüchner, H. de Meer, J. Barner, G. Bolch, A brief introduction to MOSEL-2, in: Proc. of the 13th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems, MMB, VDE Verlag, Nürnberg, Germany, 2006, pp. 469–472.

[28] SEALAB Quality Group, WEASEL, `http://sealabtools.di.univaq.it/toolWeasel.php` (2012).

[29] D. Gianni, A. D'Ambrogio, A language to enable distributed simulation of extended queueing networks, Journal of Computers 2 (4) (2007) 76–86.

[30] O. Das, C. Murray Woodside, The fault-tolerant layered queueing network model for performability of distributed systems, in: Computer Performance and Dependability Symposium, 1998. IPDS '98. Proceedings. IEEE International, 1998, pp. 132 –141.

[31] T. Altiok, Performance Analysis of Manufacturing Systems, Springer, 1997.

[32] S. Kumar, P. Kumar, Performance bounds for queueing networks and scheduling policies, Automatic Control, IEEE Transactions on 39 (8) (1994) 1600 –1611.

[33] M. K. Govil, M. C. Fu, Queueing theory in manufacturing: A survey, Journal of Manufacturing Systems 18 (3) (1999) 214 – 240.

[34] J. Keilsen, Queues Subject to Service Interruptions, Annals of Mathematical Statistics 33 (1962) 1314–1322.

[35] A. Federgruen, L. Green, Queueing systems with service interruptions, Oper. Res. 34 (5) (1986) 752–768.

[36] T. Altiok, Queueing Models of a Single Processor with Failures, Performance Evaluation 9 (1989) 93–102.

[37] C. U. Smith, Performance engineering of software systems, Addison-Wesley, 1990.

[38] C. Smith, C. Llado, Performance model interchange format (pmif 2.0): XML definition and implementation, `www.perfeng.com/paperndx.htm` (Apr. 2004).

[39] J. Rosselló, C. M. Lladó, R. Puigjaner, C. U. Smith, A web service for solving queuing network models using PMIF, in: Proc. of WOSP'05, ACM, 2005, pp. 187–192.

[40] F. Budinsky, E. Merks, D. Steinberg, EMF: Eclipse Modeling Framework (2nd Edition), Addison-Wesley Longman, Amsterdam, 2006.

[41] R. Jain, The Art of Computer Systems Performance Analysis: techniques for experimental design, measurement, simulation, and modeling, Wiley, 1991.

[42] R. Gronback, Introduction to the Eclipse Graphical Modeling Framework, in: Proc. of EclipseCon'06, 2006.

[43] K. Czarnecki, S. Helsen, Classification of Model Transformation Approaches, in: OOPSLA'03 Workshop on Generative Techniques in the Context of MDA, 2003.

[44] J. E. Rivera, A. Vallecillo, F. Durán, Formal Specification and Analysis of Domain Specific Languages using Maude, Simulation: Transactions of the Society for Modeling and Simulation International 85 (11/12) (2009) 778–792.

[45] J. de Lara, H. Vangheluwe, Translating Model Simulators to Analysis Models, in: Proc. of FASE'08, no. 4961 in LNCS, Springer, 2008, pp. 77–92.

[46] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, World Scientific, 1997.

[47] OMG, Object Constraint Language (OCL) Specification. Version 2.2, Object Management Group, document formal/2010-02-01 (Feb. 2010).

[48] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude – A High-Performance Logical Framework, Vol. 4350 of LNCS, Springer, Heidelberg, Germany, 2007.

[49] J. Troya, J. E. Rivera, A. Vallecillo, Simulating Domain Specific Visual Models by Observation, in: Proc. of the 2010 Spring Simulation Multiconference, SpringSim'10, ACM, New York, NY, 2010, pp. 128:1–8.

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

[51] J. Troya, A. Vallecillo, Behavioral Approach for QNMs, `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions/QNMs` (2011).

[52] J. E. Rivera, F. Durán, A. Vallecillo, On the behavioral semantics of real-time domain specific visual languages, in: Proc. WRLA'10, Vol. 6381 of LNCS, Springer, 2010, pp. 174–190.

[53] G. Fishman, Statistical analysis for queuing simulations, Management Science 3 (20) (1973) 363–369.

[54] K. P. White, Jr., M. J. Cobb, S. C. Spratt, A comparison of five steady-state truncation heuristics for simulation, in: Proc. of the 32nd conference on Winter simulation (WSC'00), 2000, pp. 755–760, http://dl.acm.org/citation.cfm?id=510378.510486.

[55] P. Bazan, R. German, Approximate transient analysis of large stochastic models with WinPEPSY-QNS, Computer Networks 53 (8) (2009) 1289–1301.

[56] N. J. Gunther, PDQ, http://www.perfdynamics.com/Tools/PDQ.html (2009).

[57] K. S. Trivedi, SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator, in: International Conference on Dependable Systems and Networks (DSN), Bethesda, MD, USA, 2002, p. 544.

[58] L. Brenner, P. Fernandes, A. Sales, MQNA - Markovian Queueing Networks Analyser, in: Proc. of MASCOTS'03, Orlando, FL, 2003, pp. 194–199.

[59] E. A. MacNair, R. F. Gordon, An introduction to the RESearch Queueing Package for modeling contention systems, SIGSIM Simul. Dig. 24 (1994) 40–70.

[60] A. Aggarwal, K. J. Gordon, J. F. Kurose, R. F. Gordon, E. A. MacNair, Animating simulations in RESQME, in: Proc. of the 21st conference on Winter simulation (WSC'89), ACM, 1989, pp. 612–620.

[61] G. Bolch, M. Kirschnick, The Performance Evaluation and Prediction SYstem for Queueing NetworkS - PEPSY-QNS, Tech. Rep. TR-I4-94-18, University of Erlangen-Nuremberg, Germany (Jun. 1994).

[62] J. Troya, A. Vallecillo, xQNM: A domain-specific language to specify and simulate queuing network models, http://atenea.lcc.uma.es/index.php/Main_Page/Resources/xQNM (2012).

[63] M. Danelutto, Task farm computations in Java, in: Proc. of the 8th International Conference on High-Performance Computing and Networking (HPCN Europe 2000), Vol. 1823 of LNCS, Springer, London, UK, 2000, pp. 385–394.

Javier Troya is a PhD student at the Department of Computper Science at the University of Málaga, where he received a MSc Degree in Computer Science in 2009. His research interests include Model-Driven Software Development and its industrial applications, as well as the formal semantics of model transformation languages. For further information, please visit `http://www.lcc.uma.es/~jtc` or contact him at `javiertc@lcc.uma.es`.

Antonio Vallecillo is Professor of Computer Science at the University of Málaga. His research interests include Open Distributed Processing, Model-Based Engineering, Componentware, Software Quality, and the industrial use of formal methods. For further information about his research projects and publications, please visit `http://www.lcc.uma.es/~av` or contact him at `av@lcc.uma.es`.