Computer Standards & Interfaces 36 (2014) 863–879

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

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

ARTICLE INFO

Article history:
Received 3 November 2012
Accepted 2 January 2014
Available online 17 January 2014

Keywords:
Domain-Specific Languages
Queuing network models
PMIF

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.

1. Introduction

The specification and analysis of the non-functional properties of software systems, such as QoS usage and management constraints (performance, reliability, etc.), are critical in most distributed application domains, such as embedded systems, multimedia applications and 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 behaviors. A performance model interchange format, PMIF citePMIF2 [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 describes 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 describes 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

* Corresponding author. Tel.: +34 95 213 2846; fax: +34 95 213 1397.
E-mail addresses: javiertc@lcc.uma.es (J. Troya), av@lcc.uma.es (A. Vallecillo).

0920-5489/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.csi.2014.01.002

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 and idle-times.

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 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] and 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 Ref. [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 [2,24–26]. 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 a 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 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 Section 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 and 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

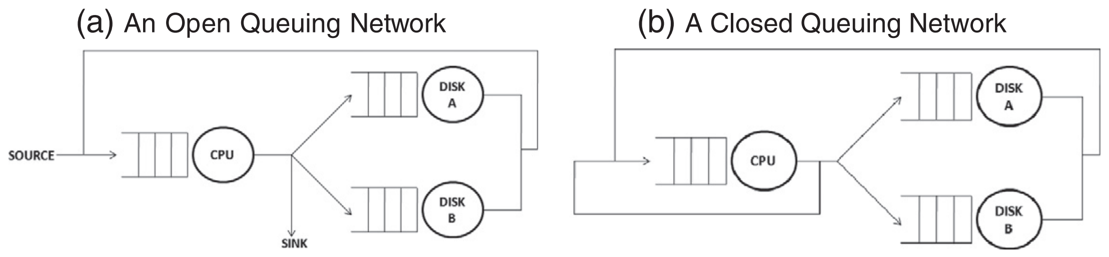

Fig. 1. Examples of an open and a closed queuing networks.