Automated Software Engineering (2022) 29:36
https://doi.org/10.1007/s10515-022-00335-z

`DICE simulation`: a tool for *software performance* assessment at the design stage

Simona Bernardi[1] · Abel Gómez[2] · José Merseguer[1] · Diego Perez-Palacin[3] · José I. Requeno[4]

Received: 20 April 2021 / Accepted: 15 February 2022

Abstract
In recent years, we have seen many performance fiascos in the deployment of new systems, such as the US health insurance web. This paper describes the functionality and architecture, as well as success stories, of a tool that helps address these types of issues. The tool allows assessing software designs regarding quality, in particular performance and reliability. Starting from a UML design with quality annotations, the tool applies model-transformation techniques to yield analyzable models. Such models are then leveraged by the tool to compute quality metrics. Finally, quality results, over the design, are presented to the engineer, in terms of the problem domain. Hence, the tool is an asset for the software engineer to evaluate system quality through software designs. While leveraging the Eclipse platform, the tool uses UML and the MARTE, DAM and DICE profiles for the system design and the quality modeling.

✉ José Merseguer
jmerse@unizar.es

Simona Bernardi
simonab@unizar.es

Abel Gómez
agomezlla@uoc.edu

Diego Perez-Palacin
diego.perez@lnu.se

José I. Requeno
jrequeno@ucm.es

[1] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain

[2] Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya (UOC), Barcelona, Spain

[3] Department of Computer Science, Linnaeus University, Växjö, Sweden

[4] Dpto. de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Madrid, Spain

Keywords Software performance · Reliability · UML · Software tool

1 Introduction

Recently, many industrial-grade software systems have exposed significant performance issues when they were already deployed. For example, the US health insurance web[1] reported major outages when it was launched, and finally it crashed due to the heavy traffic of users. The Royal Bank of Scotland and other financial institutions were fined £42 millions by the Financial Conduct Authority (FCA[2]) in 2014 for IT failures related to performance, in particular pointing "risks related to the design of the software system". Most of these performance problems were due to a software deployed with no special concerns about software bottlenecks, resources provision, scalability or even without a users workload analysis.

The *software performance engineering* (SPE) field (Smith and Lloyd 2002, 2003; Cortellessa et al. 2011) deals with the above issues by assessing the quantitative behavior of the software systems. SPE was defined by Smith and Lloyd (2002) as "a systematic, quantitative approach to the cost-effective development of software systems to meet performance requirements. SPE is a software-oriented approach that focuses on architecture, design, and implementation choices. SPE gives the information needed to build software that meets performance requirements on time and within budget.". In consequence, the idea is to identify performance flaws, even before the system is deployed, hence comprehensively analyzing the structure and behavior of the software system, from design to code. As stated by Cortellessa et al. (2011), the performance analysis should be common practice within the software development process, then introducing performance concerns in the scope of software models. But for this to be real, we need methodologies and tools. Methodologies for properly fitting the performance practice in the software life cycle, and tools for ensuring the automation of such methodologies.

For automation, the European DICE project[3] (Casale et al. 2015) developed a tool chain for systems quality assessment. DICE used model-driven techniques (The Object Management Group 2018), from design to deployment, aimed at the quality assurance in the software engineering process. This tool chain addresses not only performance, but also reliability and safety quality aspects. The tools in the chain automatically generate performance, scalability and reliability predictions for different deployment scenarios. Although DICE is particularly focussed on data-intensive applications, most of their tools can be used in more general software contexts, by leveraging the Unified Modeling Language (UML OMG 2007).

The aim of this paper is to present one of the tools developed within the DICE project, the DICE Simulation tool, called "Simulation tool" from now on. This tool is for software engineers to carry out quality assessment. In particular, performance

[1] https://www.healthcare.gov/.
[2] https://www.fca.org.uk/.
[3] https://cordis.europa.eu/project/id/644869.

and reliability assessment of software systems, early in the life-cycle. Although the Simulation tool follows SPE principles, it does not prescribe a specific SPE methodology. The approach for assessment is *scenario-based*, as it is common practice in the performance evaluation field. Scenario-based means that the engineer defines the scenarios of interest and computes metrics for each of them, e.g. throughput or resource utilization. From the metrics, assessment regarding bottlenecks, scalability or fulfillment of requirements can be carried out. The Simulation tool agnostically focuses on the software design stage, and as scenarios it uses UML sequence and activity diagrams. The deployment diagram is for representing and quantifying system resources. The tool introduces the performance view in the software design by means of UML profile annotations (OMG 2007). By design, decisions can be made, such as replicating threads or servers or sizing repositories, to meet performance and reliability requirements. It is well known that the earlier the requirements are verified the more economical it is to fix them if necessary.

The tool contributes to automate some advances in the SPE field, it specifically:

- Allows to improve UML system models with quality profile annotations (The Eclipse Foundation 2012). In particular, the MARTE (OMG 2013), DAM (Bernardi et al. 2011) and DICE (Perez-Palacin et al. 2019) profiles.
- Transforms UML-profiled models into analyzable models (Woodside et al. 2014), i.e, Petri nets and reliability models.
- Allows to assess performance metrics (Cortellessa et al. 2011). Concretely, response time, throughput and resource utilization.
- Allows to assess reliability metrics (Bernardi et al. 2013) . Concretely, mean time to failure, availability, reliability and probability of failure.

The rest of the paper is organized as follows. Section 2 revises the related work. Section 3 explains the methodological approach followed to develop the tool. Section 4 introduces necessary scientific context. Section 5 presents the tool. Section 6 tells how to use the tool. Section 7 recalls successful stories of the tool. Section 8 discusses our final concers on the tool. Section 9 concludes the paper.

2 Related work

The Simulation tool, as the rest of the DICE tools, has been developed on top of the Eclipse Platform (The Eclipse Foundation 2021). Concretely, our tool leverages Eclipse Papyrus (The Eclipse Foundation 2012) for the modeling of UML sequence and activity diagrams. The work in Ozkaya et al. (2019) analyses 58 UML tools, from many different points of views. None of these tools explicitly address the topics of our tool, that is, the analysis of system performance and reliability requirements. However, 18 (31%) of the tools support some kind of model analysis, such as the simulation of some UML diagram or the checking of well-formedness rules.

Regarding system formal verification, only Reactive Blocks[4] and Umple[5] perform it. But, these tools address functional aspects, such as the verification of the correct system behaviour, while the Simulation tool addresses non-functional ones. Besides, different to the Simulation tool, none of these 18 tools consider specific technologies or current practices in software engineering, such as DevOps or data-intensive applications.

There are tools, not reviewed in Ozkaya et al. (2019), that transform UML-annotated diagrams into analyzable models. Very few of them target performance or reliability models, as the Simulation tool does. For performance models, Tulsa Li et al. (2017) transforms UML diagrams, annotated with the DICE profile, into Layered Queuing Networks (Neilson et al. 1995), but it is exclusively focused on data-intensive applications. For reliability models, OpenMADS (Andrade et al. 2013) transforms SysML diagrams and MARTE annotations into deterministic and stochastic Petri nets, but they only address the availability analysis.

Likewise the DICE Simulation tool, the Palladio Tool (Ralf et al. 2016) has been developed for the Eclipse platform. Palladio Tools[6] implement an integrated modeling environment for computing performance, reliability, maintainability, and cost metrics. One main difference is that its modeling language is the Palladio Component Model Becker et al. (2009) (PCM) rather than UML. PCM for performance assessment fits smoothly in component-based software development. However, for other approaches, PCM may require to model a subset of the system characteristics twice, and in two different languages, i.e., in the design language chosen by software engineers and in PCM for the performance view. Moreover, considering that performance evaluation is usually scenario-based, then the component-based approach presents a disadvantage. At this regard, the Simulation tool is scenario-based and it is useful when engineers use UML for the system design. Then, they only need to add relevant performance information to such design models. The XMI Object Management Group (2006) format also helps the Simulation tool, since a UML model can be created with tools supporting this format and then feed our tool with it. PerTract (Kroß and Krcmar 2019) is a tool that also uses PCM, in this case for the specific topic of extraction of performance models for stream applications.

Finally, some tools implement very specific aspects of SPE. For example, *Filling-the-gap* (Wang et al. 2015) implements routines to estimate parameters of performance models using monitoring information. MLOS (Curino et al. 2020) is an infrastructure that uses data science to automate performance tuning. The guest editorial *Automation in software performance engineering* (Merseguer et al. 2017) gathers recent advances in SPE tools at the time of publication.

[4] https://iot.eclipse.org/community/resources/videos/2017-03-08-virtual-iot-recording/

[5] https://cruise.umple.org/umple/

[6] https://www.palladio-simulator.com/tools/

3 Methodology

The Simulation tool aims at supporting performance and reliability assessment of software systems. More general, it is widely recognized that assessing non-functional properties of software systems, early in the lifecycle, i.e., before implementation, increases the quality of the delivered product (Smith and Lloyd 2003; Dependability Management 2003; McGraw 2016). In particular, early performance and reliability assessment can be facilitated by transformations of software design models into formal models. The latters are amenable for analysis with mathematical techniques.

To deal both, with models from a software design standpoint and to manipulate them from a mathematical standpoint, we use two complementary *technical spaces*. The concept of *technical space* was introduced by Kurtev et al. when discussing the problem of bridging different technologies (Ivanov et al. 2002). It is a common abstraction when dealing with this interoperability problem in MDE[7]-based developments (Brambilla et al. 2017). A technical space is a working context, with a set of concepts, a body of knowledge, tools, required skills, and possibilities (Bézivin et al. 2006). For example, we use UML and Generalized Stochastic Petri Nets (GSPN) (Ajmone Marsan et al. 1995) as technical spaces for our Simulation tool. UML is characterized by its wide support in industrial modeling tools—such as Papyrus UML—for solving actual Software Engineering problems. While GSPN is a well-know formalism for the stochastic modeling of systems, that constitutes the formal backbone for our simulation approach.

Thus, as proposed by Bézivin and Kurtev (2006), and as it has been common practice in the MDE field in the last decades, we have bridged both technical spaces using model transformations (e.g., Boronat et al. (2006); Cabot et al. (2008); Gómez et al. (2018); Esther et al. (2009)). The Simulation tool implements automatic *forward* transformations from UML models to GSPN. The obtained GSPN models are then analyzed, transparently to the user, with event-driven Monte Carlo simulation techniques (Rodríguez et al. 2020). The simulation enables to get estimates of the performance/reliability metrics at GSPN model level, i.e., throughput of transitions and number of tokens in places. Such estimates are then post-processed by the Simulation tool, in a *backward* transformation, to map them at UML model level.

The Simulation tool can leverage the best from both, UML and GSPN, by making an extensive use of traceability links. Specifically, the Simulation tool follows a *loosely coupled traceability approach* (Galvao and Goknil 2007), which allows you to use traceability links as any other modeling asset. Such traceability links maintain all the information about which elements in the UML technical space correspond to those in the GSPN space and vice versa.

The techniques above explained allow the engineer to detect performance and reliability issues, e.g., software bottlenecks or low expected mean time to failure, in the design and, thus, to improve such software design before implementation. But

[7] Model Driven Engineering.

it is noteworthy that, thanks to technical spaces, model transformations, and traceability, engineers do not need to apply a specific SPE methodology. The intuition for the Simulation tool is that it automates and makes transparent to the user some of the steps that are necessary to follow for a model-based software performance evaluation activity. In this way, the engineers are relieved of error-prone tasks, such as manually creating the formal model and extracting the appropriate values from the large set of outputs that a model-simulation can offer. All the required knowledge remains in the software modeling technical space, hence, engineers only have to model their software as they are used to.

Figure 1 sketches the rationale for the tool development, where the following decisions about methodological aspects have been made:

- The technical space for software modeling is UML, profiled with MARTE (OMG 2013), DAM (Bernardi et al. 2011) or DICE (Perez-Palacin et al. 2019). In particular, a UML model represents *usage scenarios* of the software system at design stage, a UML scenario includes a behavioral diagram, activity or sequence, and a deployment diagram. A UML performance or reliability scenario is a UML scenario profiled with MARTE (performance), DAM (reliability) or DICE (performance/reliability in data-intensive applications context) to specify input parameters and metrics as *non-functional properties* (NFPs).
- The technical space for performance/reliability analysis consists of GSPN and event-driven Monte Carlo simulation for the analysis of GSPN models (Rodríguez et al. 2020). Although several analysis techniques are available for GSPN, we opted for simulation ones since they have a wider applicability than state-based and bound-based approximation techniques, as it will be explained in Sect. 4.2.
- The forward transformations map the software models with NFP annotations to formal models in two sequential steps, which rely on Model-2-Model (M2M) and Model-2-Text (M2T) (Brambilla et al. 2017) transformations. The M2M is applied in the first step to derive a tool-independent GSPN model specification, based on the ISO/IEC standard Petri Net Markup Language (Billington et al. 2003). The M2T is applied in the second step to get the tool-specific GSPN model suitable for the analysis. This two-step approach facilitates the tool extensibility for what concerns the use of the backend GSPN-tools for the analysis, since a new GSPN-tool solver can be supported by implementing only the second step.
- The backward transformation synthetizes the GSPN metrics estimation, computed by the tool-specific GSPN simulator, to produce the metrics estimation in the software modeling technical space, i.e., NFP metrics, and to present them in a “software-developer-friendly” manner. On the one hand, we selected the most commonly used NFP metrics in performance/reliability assessment, they are detailed in Sect. 4.1. On the other hand, the presentation of the results is both textual and graphical, the latter option is useful in case of sensitivity analysis.

Besides decisions related to methodological aspects, design decisions concerning the modeling and analysis framework were taken, some of them detailed in Sect. 5.

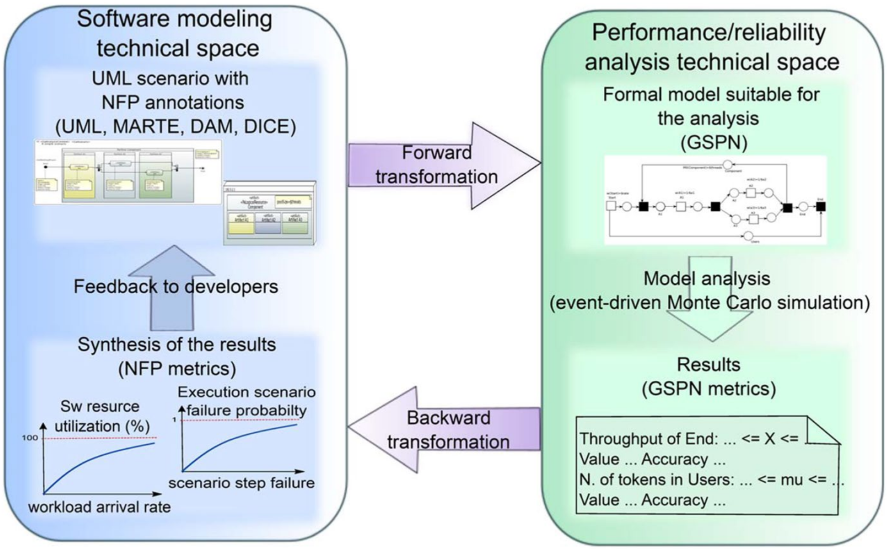

Software modeling technical space
UML scenario with NFP annotations (UML, MARTE, DAM, DICE)
Feedback to developers
Synthesis of the results (NFP metrics)
Sw resurce utilization (%)
100
workload arrival rate
Execution scenario failure probabilty
1
scenario step failure
Forward transformation
Backward transformation
Performance/reliability analysis technical space
Formal model suitable for the analysis (GSPN)
Model analysis (event-driven Monte Carlo simulation)
Results (GSPN metrics)
Throughput of End: ... <= X <= Value ... Accuracy ...
N. of tokens in Users: ... <= mu <= .. Value ... Accuracy ...

Fig. 1 Model-driven performance and reliability assessment of software systems

Regarding the modeling framework, the DICE consortium decided to develop the tool chain within the Eclipse platform (The Eclipse Foundation 2021) and its *Eclipse Modeling Framework* (EMF) (Steinberg et al. 2009). Document (The DICE Consortium 2015) provides details about the rational behind this decision, which was taken after careful analyses of many different UML modelling frameworks. EMF provides the Ecore language, which can be regarded as the reference implementation of the Meta-Object Facility (OMG 2016) standard proposed by the OMG[8]. Moreover, on top of EMF, we find a plethora of tools supporting model-driven development (MDD). Among them, general-purpose modeling languages such as UML, domain specific languages -either based on MOF/Ecore or on UML by using profiles-, query languages and M2M and M2T transformation languages. It is noteworthy that Eclipse is, to the best of our knowledge, the only platform providing tools in the same integrated environment, covering all phases of an MDD process, and using OMG standards.

Thus, from the modeling point of view, our open-source Simulation tool presents a consistent ecosystem not only for its users, but also for engineers extending or customizing the tool itself. The Simulation tool, among other OMG standards, relies on: (a) Papyrus (The Eclipse Foundation 2010) as front-end UML modeling tool, (b) UML profiles (MARTE, DAM and DICE) to specify NFPs also with Papyrus (The Eclipse Foundation 2012), (c) QVTo (OMG 2011) and Acceleo (The Eclipse Foundation & Obeo 2015) (an implementation of the MOFM2T (OMG 2008)) to specify M2M and M2T transformations, respectively, and (d) OCL to specify model

[8] https://www.omg.org/

constraints and model queries. Even considering that the Simulation tool relies upon GreatSPN (Dipartimento di informatica 2015) as back-end for the analysis, its integration is achieved by using Ecore (i.e. MOF) similarly to other intermediate or internal models.

4 Estimation of performance and reliability metrics

This section introduces the performance and reliability metrics offered by the Simulation tool and the analysis techniques used to estimate them.

4.1 Metrics

The tool allows to compute the most commonly used performance and reliability metrics (Jain 1991) on UML scenarios. In particular, the tool supports the following basic metrics:

Response time of the usage scenario. It measures the time between the arrival of a request, to be processed by the system, and the completion of the scenario.
Throughput of the usage scenario. It measures the number of processed requests per time unit.
Utilization of a software resource. It measures the percentage of "busy" time of a software artifact, that collaborates in the execution of the usage scenario, over the observation period.
MTTF of the execution environment. It measures the expected time until the infrastructure resources fail. Its computation requires the MTTF values of the each infrastructure resource.
Availability of the execution environment. It measures the percentage of time that the infrastructure resources is ready for utilization. It is calculated by using a mean time to repair (MTTR) value provided by the user and the previously mentioned MTTF.
Probability of failure of a concrete execution of the usage scenario. It measures the probability that an execution of the usage scenario workflow fails. It requires the definition of the probability of failure of some actions in the workflow.
Reliability of the execution environment. It measures the probability of the infrastructure resources to continuously work up to a user defined time value, called *mission time*.

The first three are performance metrics, whereas the other four are reliability metrics. Table 1 summarizes the UML profile extensions to be applied to define the performance and reliability metrics in a UML scenario. Concerning performance, all the usage scenario metrics have to be annotated in the behavioral diagram. Concretely, by stereotyping either the Activity model element (activity diagram) or the Interaction model element (sequence diagram) as *GaScenario*. The utilization metric must be annotated in the deployment diagram. In particular, each software artifact of interest for the performance analysis, needs to be stereotyped as *PaLogicalResource*. Depending on the type of metric (fourth column), the table indicates the corresponding tag (last column) to be used. Concerning reliability, the properties to be modeled are: (a) the probability of failure of software operations and, (b) the mean time to failure and mean time to repair of infrastructure elements. Whereas, the metrics to be annotated are summarized in Table 1.

4.2 Analysis techniques

The estimation of metrics is carried out, transparently to the practitioner, using GSPN simulation techniques. Several GSPN solution techniques are available in the literature and have been implemented in GSPN tools. We can roughly classify them in state-based, bound-based approximation and simulation techniques (Balbo and Silva 1998). We opted for event-driven Monte Carlo simulation (Rubinstein and Kroese 2008), since it has a wider applicability than both, state-based and bound-based, approximation techniques. On the one hand, simulation can be used also when the state space of the GSPN model is too large to be analyzed with state-based techniques, or even infinite. Moreover, unlike bound-based approximation techniques, the quality of the results does not depend on the topological structure of the GSPN model. On the other hand, unlike state-based techniques, which provide "exact" values for performance metrics, simulation computes a confidence interval, by generating randomly chosen paths through the state space.

Concretely, the Simulation tool relies upon the steady state simulator implemented in GreatSPN (Dipartimento di informatica 2015), which estimates two types of basic metrics at GSPN level. Concretely, mean throughput of transitions and mean marking of places. The correctness and performance of the GreatSPN simulator has been extensively tested in Rodríguez et al. (2020), through a comparative analysis, based on a GSPN benchmark, with other two simulation techniques implemented in different tools. For each basic metric, the simulator provides the estimated value, the confidence interval and the precision error, i.e., the real (unknown) value falls into the confidence interval with a certain probability (i.e., confidence level) and with a precision error (Rubinstein and Kroese 2008). The confidence level and the maximum threshold of precision error are input parameters to be set for a simulation experiment. The simulator implements the *batch means method*, thus measurements of statistics variables associated to the GSPN model, such as the number of transition firings or token residence time in places, are taken across successive epochs or "batches" of a single simulation run. After the termination of an epoch, the simulator estimates each basic metric. The simulation terminates when the precision error of the confidence interval of all the basic metrics is lower than, or equal to, the established maximum threshold.[9]

[9] A minimum number of 10 batches is executed, in each simulation run, before computing the confidence intervals the first time.

Table 1 List of performance and reliability metrics for a UML scenario

Diagram	Model	Stereotype	Metric	Tag
Performance metrics (MARTE profile)				
Activity	Activity	GaScenario	Throughput	throughput
			Response time	respT
Sequence	Interaction	GaScenario	Throughput	throughput
			Response time	respT
Deployment	Artifact	PaLogicalResource	Utilization	utilization
Reliability metrics (DAM profile)				
Deployment	Node	DaComponent	Availability	ssAvail
			Reliability	reliability
			MTTF	failure.MTTF
Activity	Activity	DaService	Prob. of failure	failure.occurrenceProb

Regarding the computation of performance metrics. The throughput of the GSPN transition that represents the scenario termination is mapped directly to the throughput of the usage scenario. The former is also used, together with the mean marking of the GSPN place that models the requests arrived to the system, to compute the response time of the usage scenario, by applying the Little's formula (Jain 1991). The mean marking of the GSPN places representing software resources are used to compute the utilization of the corresponding software resources.

Regarding the computation of reliability metrics. They are calculated using the Petri net simulation results, as explained above, and the principles for analyzing Reliability Block Diagrams in series/parallel systems.[10]

The simulation results are then post-processed, in a *backward* transformation, by the Simulation tool to present the performance and reliability metrics results at UML model level, as explained in Sect. 3.

5 Software framework

The Simulation tool can be used as a standalone software, directly running on Eclipse, or it can be integrated, as an Eclipse plugin, with another third-party tool. The latter scenario has been successfully addressed by integrating the tool into the DICE platform[11] (The DICE Consortium 2015). For both cases, the functionalities, software architecture and execution workflow remain the same, as explained in the following.

[10] Document (The DICE Consortium 2017), Sect. 4.2, provides details about the formulas used for calculating reliability.

[11] https://github.com/dice-project

5.1 Software functionalities

The functionalities are organised in three groups: main functionalities, additional functionalities, and user facilities.

Three **main functionalities** allow to carry out a quantitative analysis of a software system, in particular, performance and reliability analysis:

MF1 — The tool allows the **annotation of UML diagrams** with system quantitative properties (e.g., host demands, routing rates or workloads). The annotations follow the standard MARTE UML profile (OMG 2013), as well as the DAM profile (Bernardi et al. 2011). The UML diagrams considered are activity diagrams, sequence diagrams and deployment diagrams.
MF2 — The tool allows to **compute performance metrics** (response time, throughput and resource utilization) on a UML scenario[12] that represents a particular system execution.
MF3 — The tool allows to **compute reliability metrics** (MTTF, availability and reliability) on a UML scenario that represents a particular system execution. Performance and reliability metrics (**MF2** and **MF3**) are detailed in Sect. 4.

The tool offers **additional functionalities**, also from the end user point of view, that improve the main ones:

AF1 — The tool allows the annotation of UML diagrams with quantitative properties which are data-intensive applications specific (i.e., Apache Hadoop, Spark, Storm and Tez applications). The annotations conform to the DICE profile (Perez-Palacin et al. 2019). This functionality improves **MF1** for addressing specific big data applications.
AF2 — The tool can perform *what-if* or *sensitivity* analysis for **MF2** and **MF3**. So, the tool allows the user to see multiple output results, i.e., quantitative metrics, in a user-friendly format, showing the result values in a 2D plot.
AF3 — The tool transforms a UML scenario into a Generalized Stochastic Petri net (Ajmone Marsan et al. 1995). The output format can be: PNML (Petri Net Markup Language (Billington et al. 2003)), GreatSPN (Dipartimento di informatica 2015) format or DOT (Gansner 2015) (only graphical). GreatSPN format is in fact used for computing the metrics in **MF2**, but all formats are made available outside the tool, in case the engineer requires other kind of analyses.

Other functionalities of the tool are useful **facilities** for the end user:

FF1 — The tool allows to set parameters of the simulation carried out by **MF2**. Rationale: GreatSPN enables configuring the computation of performance metrics

[12] By UML scenario, we mean the combination of an activity diagram plus a deployment diagram, or a sequence diagram plus a deployment diagram.

in terms of the confidence interval and error of the results. Our tool offers widgets in the user interface to set them.

FF2 — The tool allows to stop long run simulations carried out by **MF2**. Rationale: Some simulations may last for long time periods, depending on the model and parameters used (**FF1**). Our tool offers a simple user interface button to stop these simulations, while the results computed so far are presented, also indicating the accuracy reached.

FF3 — The tool allows to configure and use simulators installed in remote computers. Rationale: In **MF2**, the simulation of complex systems or the simulation of *what-if* scenarios, that depend on multiple parameters, may demand significant computing resources. Our tool provides a friendly user interface for remotely configure and use the GreatSPN tool, that can be running anywhere in the world. This characteristic brings special benefit when a developer runs the tool on a battery-powered laptop.

5.2 Software architecture

Figure 2 depicts a simplified view of the architecture of the Simulation tool. It represents the dependencies with other tools and the integration into the DICE platform[11]. The dependencies are with:

1. Eclipse **Papyrus** modeling environment (The Eclipse Foundation 2010). It is an open source UML modeling tool that supports the MARTE profile natively. It also supports the DAM and DICE profiles. Its main role is to enable to create profiled UML models. So, it offers support to **MF1** and **AF1**.
2. GreatSPN (Dipartimento di informatica 2015) is a third-party tool used for Petri nets performance analysis. Since it does not provide a specific Application Programming Interface (API), its integration is made through native `ssh` calls and standard input/output. It gives support to **MF2**.

The tool is made of two layers, a *GUI layer* and a *model-based simulation layer*, as follows.

5.2.1 GUI layer

This layer contributes with a set of graphical interfaces to provide the user with a transparent and intuitive interaction. The main responsibilities are to collect information for the simulation of the UML scenario, and to display the results. It comprises the following components.

– The **Simulation configuration** asks the user for: (i) the UML scenario to be simulated[13]; (ii) the metrics to be computed; (iii) the simulation parameters; (iv)

[13] A UML model can include multiple scenarios.

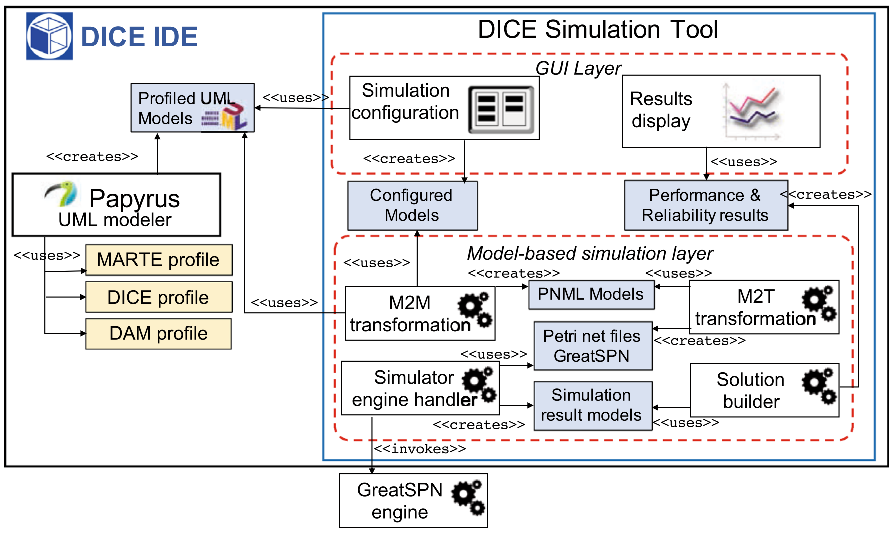


Fig. 2 High-level architecture of the DICE Simulation tool

the user's choice on performing *what-if* analysis; and (v) the location (local or remote) of the GreatSPN tool. Therefore, this module gives support to **AF2**, **FF1** and **FF3**. It saves the choices in a *configured model*.

– The **Results display** is implemented as an Eclipse view and it depicts the results in a user-friendly mode. When the *what-if* analysis is configured, this module plots results in 2D graphics. So, it helps for **AF2**.

Moreover, this layer also implements a button to abort long-run simulations (**FF2**).

5.2.2 Model-based simulation layer

This layer comprises OSGi components, called by the GUI layer, that run in background. It has been specifically designed to orchestrate the interaction among the different tools that perform the actual analysis and simulation. This layer receives UML models and configured models, with the user's expectations of the simulation, and it produces the performance and reliability results. Concretely, it is made of:

- **M2M transformation** component, which executes a model-to-model transformation of the UML scenario to a Petri net model. The transformation selected depends on the metric that the user needs to compute. Transformations are implemented using QVTo (OMG 2011), being the resulting Petri net an instance of the PNML meta-model.
- **M2T transformation** component, which executes model-to-text transformations from the PNML to create Petri nets in the GreatSPN tool format. The trans-

formations are developed using the Acceleo (The Eclipse Foundation & Obeo 2015) MOFM2T[14] implementation.
- **Simulator engine handler** component, which orchestrates all interactions with GreatSPN. It invokes GreatSPN commands, receives a stream of (partial) final results, parses them and creates the *simulation result models*. These results are in terms of Petri net concepts, i.e., they are statistical information about the throughput of each transition and the number of tokes in each place.
- **Solution builder** component, which transforms the *simulation results* into proper performance and reliability results. Hence, it receives statistical information, about throughput and tokens, and computes results in terms of the UML scenario response time, throughput, utilization of resources, mean time to failure or availability.

The **M2M** and **M2T transformation** components implement **AF3**. The **Simulator engine handler** and the **Solution builder** implement the part of **AF2** not dealt by the GUI layer.

5.3 Execution workflow

Figure 3 depicts a UML activity diagram that represents an execution workflow of the tool. The partitions in the diagram refer by name to the layers and tools in Fig. 2.

First of all, the annotated UML model must be created (1). Then, the simulation must be configured (2). After that, the tool proceeds with the performance or reliability evaluation of the configured UML scenario without requiring any other input from the user. To do that, it executes a sequence of operations composed of: the model-to-model transformations (3), model-to-text transformations (4), the simulation and evaluation of the Petri nets (5) (6), the generation of the performance and reliability results from the Petri nets simulation results (7), and the display of results to the user (8).

When the user configures the simulation for a *what-if* evaluation, setting ranges of values for some model variables, then the model transformation creates several Petri nets that have to be simulated. Then, the simulation engine handler (5) needs to sequentially invoke the GreatSPN tool (6), once per Petri net, as indicated by the decision node.

6 Illustrative example

The wiki of the tool in the GitHub repository offers a complete information about:

- How to install the tool[15]

[14] MOF Model to Text Transformation Language. https://www.omg.org/spec/MOFM2T/

[15] https://github.com/dice-project/DICE-Simulation/wiki/Installation.

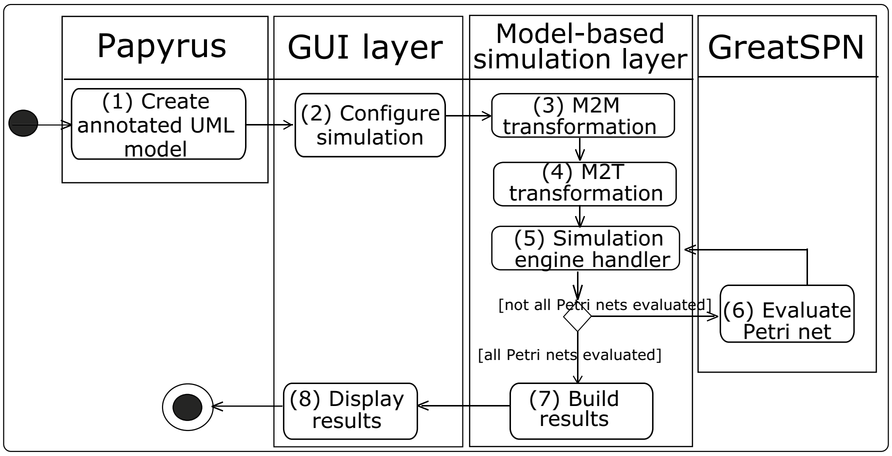


Fig. 3 Execution workflow of the DICE Simulation tool

- How to install tool updates[16]
- How to set up the tool[17]
- How to use the tool[18]

Once the tool has been installed and the environment set up, according to the guidelines in the Wiki[17], the practitioner can use the *DICE Simulation Examples* shortcut to download the *PosidoniaSimplified* model. Alternatively, this model can be downloaded from GitHub[19] (*PosidoniaSimplified* folder). This model is a simplified version of the Posidonia case study, that will be described in Sect. 7.1.

6.1 UML modeling and MARTE profiling with the tool

The simplified UML model includes only one performance scenario, consisting of an activity diagram and a deployment diagram (Fig. 4). In the following paragraph, we describe the activity diagram, which corresponds to the workflow of the *Complex Event Processing* (CEP) component.

Two components of the CEP are involved in the scenario: *AIS Sentence Listener* and *Stateful Knowledge Session*. The former continuously monitors a message queue, and when a new AIS statement occurs it adds the statement to the database and delegates the rest of the operations to the *Stateful Knowledge Session*. This latter component updates the list of active rules. A rule is made of two parts: the trigger

16 https://github.com/dice-project/DICE-Simulation/wiki/Installing-Updates-And-Utils.

17 https://github.com/dice-project/DICE-Simulation/wiki/First-Steps.

18 https://github.com/dice-project/DICE-Simulation/wiki/Getting-Started.

19 https://github.com/dice-project/DICE-Simulation/tree/master/bundles/es.unizar.disco.pnml.m2m/examples.

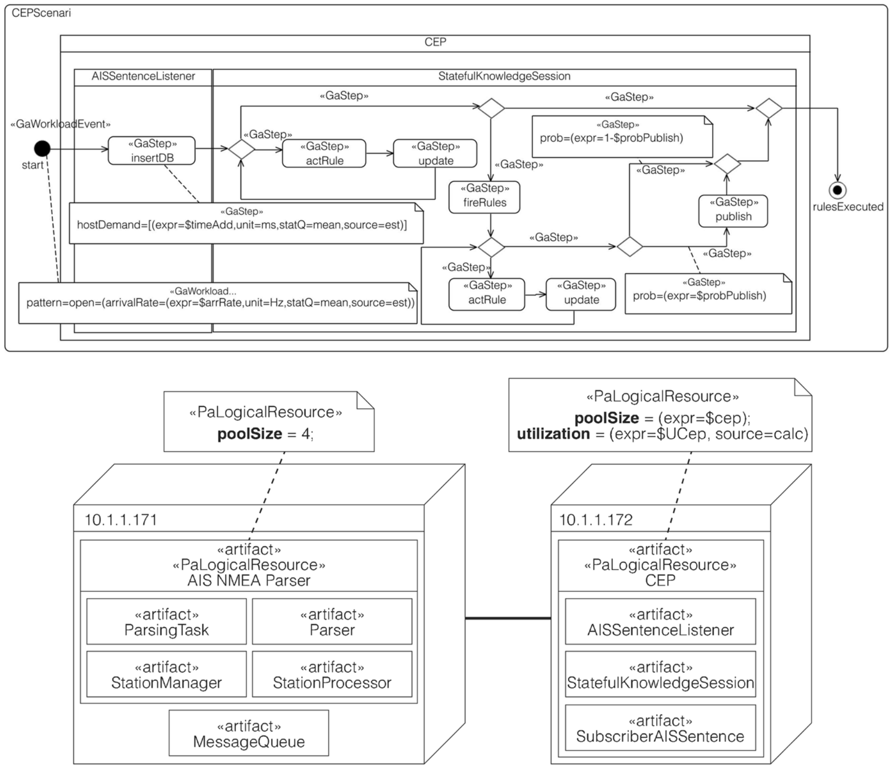

CEPScenari
CEP
AISSentenceListener
StatefulKnowledgeSession
«GaWorkloadEvent»
start
«GaStep» insertDB
«GaStep»
«GaStep» actRule
«GaStep» update
«GaStep» fireRules
«GaStep» prob=(expr=1-$probPublish)
«GaStep» publish
«GaStep» hostDemand=[(expr=$timeAdd,unit=ms,statQ=mean,source=est)]
«GaWorkload... pattern=open=(arrivalRate=(expr=$arrRate,unit=Hz,statQ=mean,source=est))
«GaStep» prob=(expr=$probPublish)
rulesExecuted
«PaLogicalResource» poolSize = 4;
«PaLogicalResource» poolSize = (expr=$cep); utilization = (expr=$UCep, source=calc)
10.1.1.171
«artifact» «PaLogicalResource» AIS NMEA Parser
«artifact» ParsingTask
«artifact» Parser
«artifact» StationManager
«artifact» StationProcessor
«artifact» MessageQueue
10.1.1.172
«artifact» «PaLogicalResource» CEP
«artifact» AISSentenceListener
«artifact» StatefulKnowledgeSession
«artifact» SubscriberAISSentence

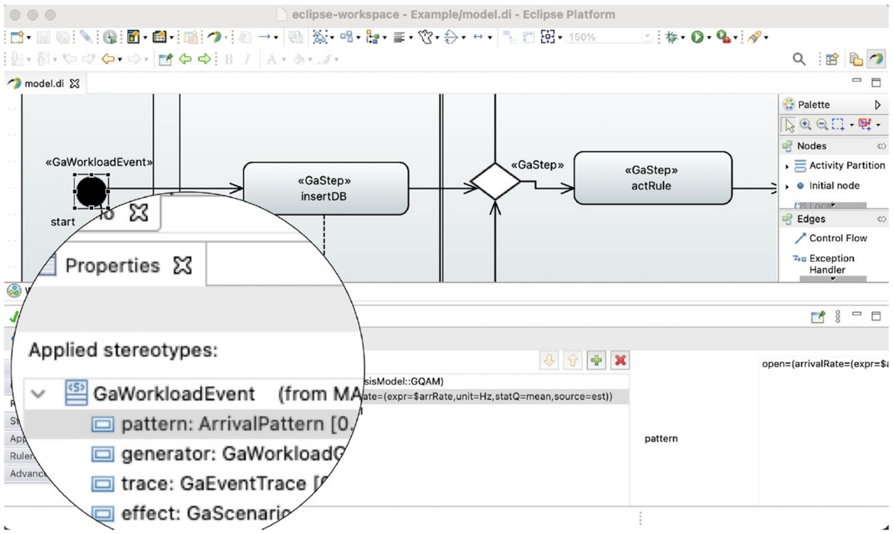
Fig. 4 Activity and deployment diagrams

condition and the body. Then, it decides whether to execute the active rules. If so, a second phase starts by firing all active rules and their management. If not, the execution cycle ends. Since the firing of a rule can cause the activation of other rules, then the component updates the list of active rules again. Finally, this component decides whether to publish the identified AIS events in the message queue.

The UML diagrams in Fig. 4 contain MARTE profile annotations needed for performance analysis. The Simulation tool supports a subset of the standard MARTE annotations, those needed to specify model parameters and metrics to be computed[20]. This subset of annotations is the most commonly used, in practice, for performance analysis (Smith and Lloyd 2003). For example, to mention few of them, the mean arrival rate for open workloads, the mean time and probability associated to execution steps, and common metrics such as the system response time and resource utilization.

These annotations are easily accessible in the tool, just by selecting the stereotyped model element and the *Profile* tab of the *Properties* view, in the *Papyrus*

[20] A comprehensive list of supported annotations is available at: https://github.com/dice-project/DICE-Simulation/wiki/Profiles-Reference



GaWorkloadEvent (from MA…sisModel::GQAM)
pattern: ArrivalPattern [0… …ate=(expr=$arrRate,unit=Hz,statQ=mean,source=est))
generator: GaWorkloadG…
trace: GaEventTrace […
effect: GaScenari…

Fig. 5 UML activity diagram with MARTE profile annotations (tagged values of the *GaWorkloadEvent* stereotype)

perspective. The screenshot in Fig. 5 shows, as an example, the tagged values associated to the *start* initial node, in the activity diagram, that has been stereotyped as a *GaWorkloadEvent*.

The practitioner can navigate, in the tool, both diagrams and browse all the stereotyped model elements already defined. Table 2, first and second columns, summarizes all stereotypes and tags used in the example, they define the performance parameters and the metrics to compute. The tagged-values are expressed using the Value Specification Language (VSL)[21] and most of them include variables (third column).

The tagged-values representing metrics must include output variables, which are prefixed with *out*\$. Their values are unknown before the analysis and they will be estimated with the simulation.

On the other hand, the tagged-values representing performance parameters can include values or input variables. The latter need to be set to values for conducting a performance analysis experiment. The choice of the values for the input variables is up to the practitioner. In fact, it is the practitioners experience and knowledge in the problem domain what certainly gives the insights for the choice. For example, in the Posidonia project, we used, for setting the performance parameters, already existing system logs. In particular, these logs were useful to make accurate estimates regarding activities durations and their execution probabilities. Finally, the input variables can also be used for sensitivity analysis. Concretely, when one aims to define different analysis experiments for the same annotated model.

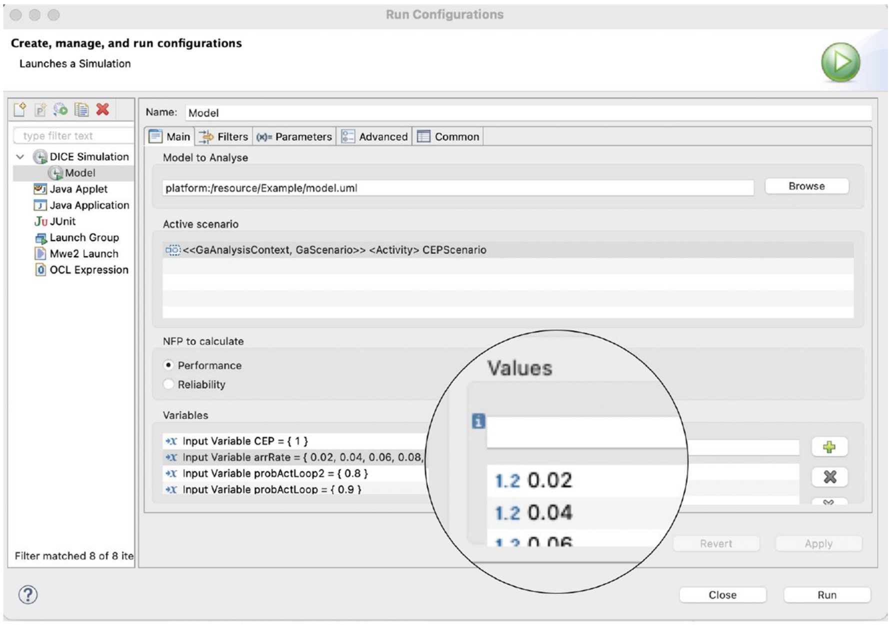
[21] VSL is defined in (OMG 2013).

Table 2 MARTE Profile annotations, model parameters and basic configuration of a performance analysis experiment

Activity diagram			
Model element (*stereotype*)	NFP (*tag*)	Variable	Value
Activity (GaAnalysisContext)			
CEP scenario	variable list (*context*)		[...]
Activity (GaScenario)			
CEP scenario	throughtput (*throughput*)	out$XCep	
	response time (*respT*)	out$RTCep	
	used resources (*usedResources*)		CEP
Initial node (GaWorkloadEvent)	*open workload (pattern=open)*		
Start	mean arrival rate (*arrivalRate*)	$arrRate	[0.02–0.2]
Action (GaStep)	*mean duration (hostDemand)*		
InsertDB		$timeAdd	590
ActRule (first phase)		$timeActRule	400
Update (first phase)		$timeUpdate	100
FireRules		$timeFire	500
ActRule (second phase)		$timeActRule	400
Update (second phase)		$timeUpdate	100
Publish		$timePubEvent	240
Transitions (GaStep)	*execution probability (prob)*		
First activation loop–cont		$probActLoop	0.9
First activation loop–end		1-$probActLoop	0.1
Fire rules?–yes		$probFireRules	0.5
Fire rules?–no		1-$probFireRules	0.5
Second activation loop–cont		$probActLoop2	0.8
Second activation loop–end		1-$probActLoop2	0.2
Publish events?–yes		$probPublish	0.5
Publish events?–no		1-$probPublish	0.5
Deployment diagram			
Model element (*stereotype*)	NFP (*tag*)	Variable	Value
Artifact (PaLogicalResource)			
CEP	Number of instances (*poolSize*)	$CEP	1
	Utilization (*Utilization*)	Out$UCep	

6.2 Performance analysis with the tool

A performance analysis experiment consists in creating and running a new configuration.



Run Configurations

Create, manage, and run configurations

Launches a Simulation

type filter text

- DICE Simulation
  - Model
- Java Applet
- Java Application
- JUnit
- Launch Group
- Mwe2 Launch
- OCL Expression

Filter matched 8 of 8 ite

Name: Model

Main | Filters | Parameters | Advanced | Common

Model to Analyse

platform:/resource/Example/model.uml

Active scenario

<<GaAnalysisContext, GaScenario>> <Activity> CEPScenario

NFP to calculate

[✓ Performance]
Reliability

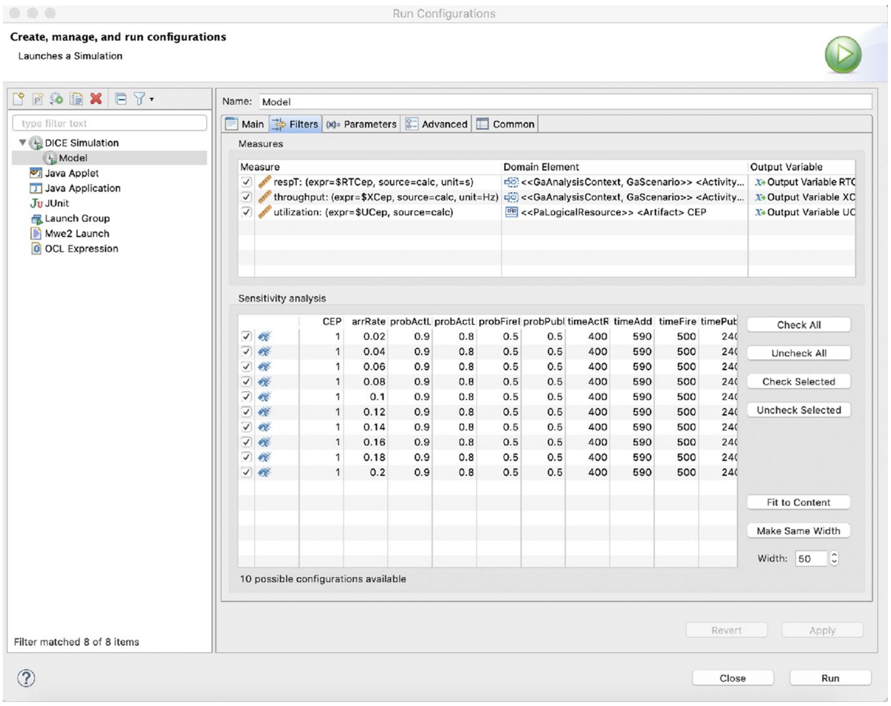
Variables

Input Variable CEP = { 1 }
Input Variable arrRate = { 0.02, 0.04, 0.06, 0.08,
Input Variable probActLoop2 = { 0.8 }
Input Variable probActLoop = { 0.9 }

Values

1.2 0.02
1.2 0.04
1.2 0.06

Fig. 6 Selection of the model, analysis scenario, and type of analysis, and model input variables setting (*Main* tab)



Run Configurations

Create, manage, and run configurations

Launches a Simulation

type filter text

- DICE Simulation
  - Model
- Java Applet
- Java Application
- JUnit
- Launch Group
- Mwe2 Launch
- OCL Expression

Filter matched 8 of 8 items

Name: Model

Main | Filters | Parameters | Advanced | Common

Measures

	Measure	Domain Element	Output Variable
✓	respT: (expr=$RTCep, source=calc, unit=s)	<<GaAnalysisContext, GaScenario>> <Activity...	Output Variable RTC
✓	throughput: (expr=$XCep, source=calc, unit=Hz)	<<GaAnalysisContext, GaScenario>> <Activity...	Output Variable XC
✓	utilization: (expr=$UCep, source=calc)	<<PaLogicalResource>> <Artifact> CEP	Output Variable UC

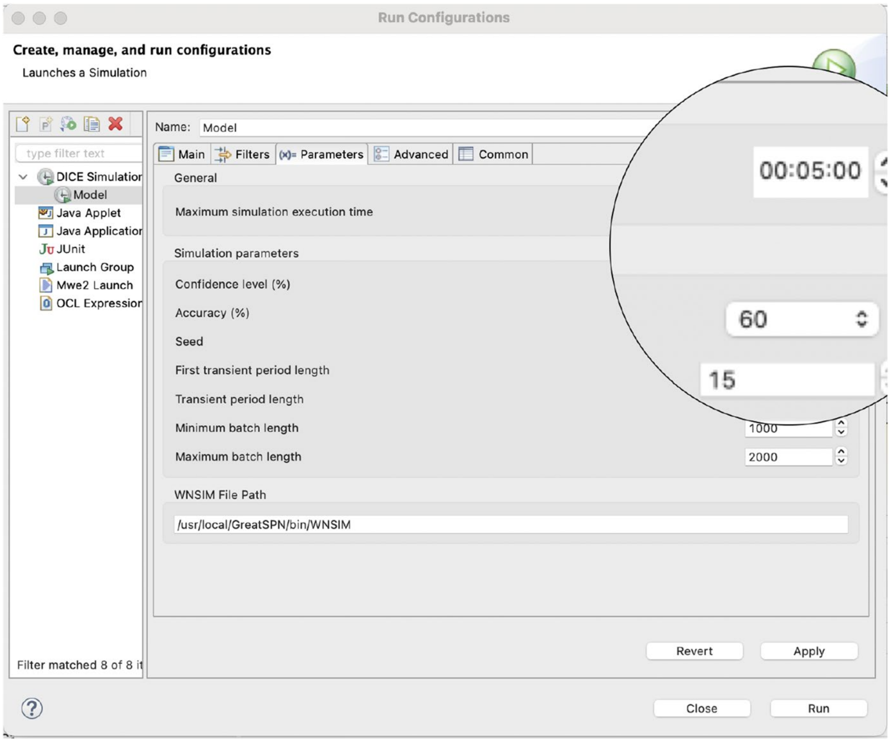
Sensitivity analysis

	CEP	arrRate	probActL	probActL	probFirel	probPubl	timeActR	timeAdd	timeFire	timePub
✓	1	0.02	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.04	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.06	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.08	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.1	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.12	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.14	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.16	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.18	0.9	0.8	0.5	0.5	400	590	500	240
✓	1	0.2	0.9	0.8	0.5	0.5	400	590	500	240

10 possible configurations available

Width: 50

Fig. 7 Selection of the performance metrics and configurations (*Filters* tab)


Run Configurations

Create, manage, and run configurations

Launches a Simulation

type filter text

DICE Simulation
Model
Java Applet
Java Application
JUnit
Launch Group
Mwe2 Launch
OCL Expression

Filter matched 8 of 8 it

Name: Model

Main | Filters | Parameters | Advanced | Common

General

Maximum simulation execution time 00:05:00

Simulation parameters

Confidence level (%)

Accuracy (%) 60

Seed

First transient period length 15

Transient period length

Minimum batch length 1000

Maximum batch length 2000

WNSIM File Path

/usr/local/GreatSPN/bin/WNSIM

Revert | Apply

Close | Run

Fig. 8 Setting of the simulation parameters (*Parameters* tab)

6.2.1 Creating a new configuration

The practitioner needs to open the *Run Configurations* window, either by selecting the *Run as→Run Configurations...* option from the contextual menu associated to the model (*Project Explorer* view), or by clicking the clock button in the Eclipse icons topmost bar. The configuration is carried out through the following three main steps:

Step 1 In the *Main* tab of the *Model* configuration (screenshot in Fig. 6), the practitioner selects the UML annotated model, a scenario to analyse, and the type of analysis (i.e., performance or reliability) and configure the input variable values. When he/she selects the scenario to analyse, the tool retrieves all the related input variables annotated in the model, and show them in the *Variables* field, with their default values in the *Values* field. The practitioner can modify/add input variable values directly in the *Values* field. In particular, a range of values can be assigned to each input variable. For example, for the *arrRate* in Fig. 6, in the *Values* field it can be specified the lower, upper and increment step of the range of values. The syntax is available by positioning the mouse over the *i* symbol (see the zoomed part of the screenshot). The tool generates all the values in the range. In the example, a range of values between 0.02 and 0.2 Hz, with an incremental step of 0.02 Hz has been set up. Table 2 (last column) lists the variable values used in the *CEP analysis configuration*: observe that all the values are numeric, whereas the units of measure associated are defined with the MARTE annotations.

Step 2 In the *Filters* tab of the *Model* configuration (screenshot in Fig. 7), the practitioner selects the metrics to be computed and the experiments to carry out.

The *Measures* panel shows the metrics, that were annotated in the model. In the example, the CEP scenario response time and throughput (from the activity diagram), and the utilization of the CEP (from the deployment). Moreover, this panel shows the model elements the metrics belong to, as well as the output variables used in the annotations. The *Sensitivity analysis* panel shows the ten possible experiments or configurations. Each experiment corresponds to a combination of the input variables set in **Step 1**. In fact, when a range of values is assigned, the tool generates all possible variable configurations (i.e., it generates the product space $D = \prod_{i=1}^{n} D_i$, where D_i is the value domain of the i^{th} variable).

Both, the performance metrics and the experiments configurations, have associated a checkbox, that can be used to select/deselect each of them.

Step 3 In the *Parameters* tab of the *Model* configuration (screenshot in Fig. 8), the practitioner can modify the default setting.

The *General* parameter enables to set a maximum execution time for a simulation run. The *Simulation parameters* are specific of the GreatSPN engine (Dipartimento di informatica 2015). In particular, *confidence level* and *accuracy*, which are also relevant for the duration of a simulation run. The zoomed part of the screenshot in Fig. 8 shows the values set in the example for these parameters, i.e., maximum 5 minutes of execution time, 60% level for the confidence interval and 15% of accuracy. The accuracy is interpreted as a percentage of the approximation error, thus lower the value higher is the accuracy of the estimated metrics. The changes made in the configuration are saved with the *Apply* button (see at the bottom right of Fig. 8).

6.2.2 Running a configuration

The practitioner can finally launch the simulation experiment with the *Run* button, located below the *Apply* button (Fig. 8). Then, the tool will ask to switch to the *DICE Simulation* perspective. In the example, the simulation experiment consists of the 10 experiments previously commented.

In the *DICE Simulation* perspective (screenshot in Fig. 9), there are three key views that allow monitoring the simulation runs:

- The *Debug* view (top left), it shows the information about the simulation process.
- The *Console* view (top right), it shows the messages that the simulation process dumps into the standard output and error streams. If an errror occurs during a simulation run, it will be reported in this view.
- The *Inspector* view (bottom), it shows the starting/ending times and the status of the simulation runs.

In this perspective, it is also possible to stop the simulation process by pushing the *Stop* button (red rectangle icon) located above the *Console* view. As soon as a simulation run finishes, it is possible to view the estimated performance metrics by right clicking on a particular simulation run in the *Inspector* view (see Fig. 9 applied on the third simulation run) and choosing *Open Simulation Results* option: a new view will pop-up that enables to navigate the tree and see the metric results.

When sensitivity analysis is carried out, then it is possible to see 2D plots showing trends of the metrics against an input variable, in the range of values set during the first configuration step. To generate a 2D plot, the practitioner needs to right click on the simulation experiment in the *Invocation Registry* view (see screenshot in Fig. 10), then choose the *Plot Results...* option from the contextual menu and follow the wizard provided by the tool. Observe that the simulation experiment in Fig. 10 includes one simulation run that was killed. However, it is still possible to generate the 2D plot from the results of the rest of simulation runs.

Figure 11 depicts the plot of the utilization of the CEP (cfr. Fig. 4, deployment diagram) against the mean arrival rate of the input stream (cfr. Fig. 4, activity diagram, *GaWorkloadEvent* annotation). The system is clearly not stable for mean arrival rates greater than 0.14 Hz, this is the reason for the possible long simulation runs that may occur for such variable configurations[22]. The user can modify the plot characteristics by editing the *Source* tab (zoomed part in the figure).

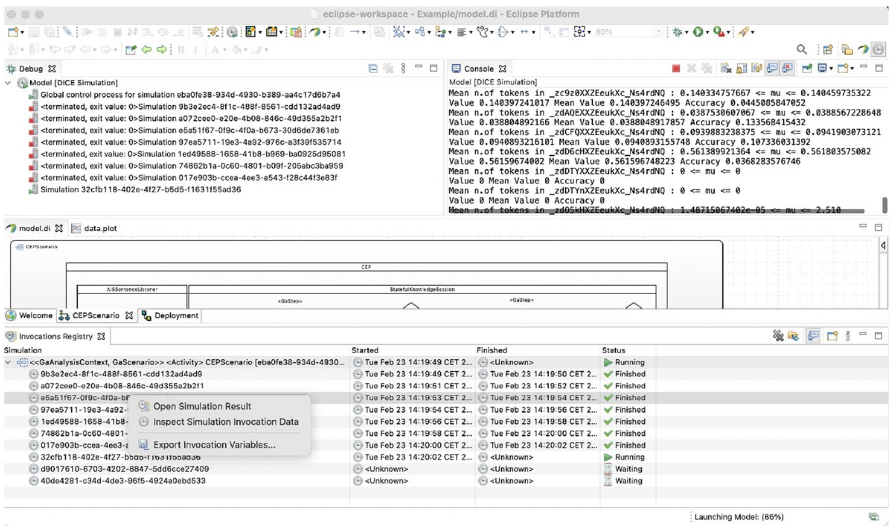
7 Empirical results

The tool has been applied to two industrial case studies. The first one carried out for the performance assessment of Posidonia Operations system[23]. The second one for the redesign of the NewsAssets suit[24], which automates the editorial workflow and analyzes millions of media items, in real time, from the social network, then providing services for journalists. Both case studies have been reported by journal articles Bernardi et al. (2018) and Requeno et al. (2019), respectively. However, these papers exclusively focus on the product goals, methodology applied and

[22] Indeed, the simulation run that failed in the experiment, was killed since it reached the maximum of 5 minutes execution time set in step 3, Fig. 8.

[23] https://www.prodevelop.es/en/ports/posidonia/posidonia-operations-2

[24] https://www.atc.gr/#InnovationLab



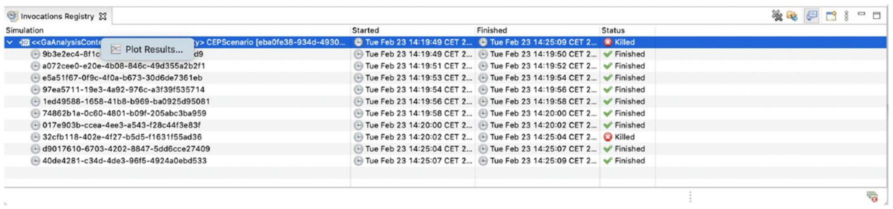
Model [DICE Simulation]
Global control process for simulation eba0fe38-934d-4930-b389-aa4c17d6b7a4
<terminated, exit value: 0>Simulation 9b3e2ec4-8f1c-488f-8561-cdd132ad4ad9
<terminated, exit value: 0>Simulation a072cee0-e20e-4b08-846c-49d355a2b2f1
<terminated, exit value: 0>Simulation e5a51f67-0f9c-4f0a-b673-30d6de7361eb
<terminated, exit value: 0>Simulation 97ea5711-19e3-4a92-976c-a3f39f535714
<terminated, exit value: 0>Simulation 1ed49588-1658-41b8-b969-ba0925d95081
<terminated, exit value: 0>Simulation 74862b1a-0c60-4801-b09f-205abc3ba959
<terminated, exit value: 0>Simulation 017e903b-ccea-4ee3-a543-f28c44f3e83f
Simulation 32cfb118-402e-4f27-b5d5-f1631f55ad36

```
Model [DICE Simulation]
Mean n.of tokens in _zc9z0XXZEeukXc_Ns4rdNQ : 0.140334757667 <= mu <= 0.140459735322
Value 0.140397241017 Mean Value 0.140397246495 Accuracy 0.0445085847052
Mean n.of tokens in _zdAQEXXZEeukXc_Ns4rdNQ : 0.0387530607067 <= mu <= 0.0388567228648
Value 0.038804892166 Mean Value 0.0388048917857 Accuracy 0.133568415432
Mean n.of tokens in _zdCFQXXZEeukXc_Ns4rdNQ : 0.0939883238375 <= mu <= 0.0941903073121
Value 0.0940893216101 Mean Value 0.0940893155748 Accuracy 0.107336031392
Mean n.of tokens in _zdD6cHXZEeukXc_Ns4rdNQ : 0.561389921364 <= mu <= 0.561803575082
Value 0.56159674002 Mean Value 0.561596748223 Accuracy 0.0368283576746
Mean n.of tokens in _zdDTYXXZEeukXc_Ns4rdNQ : 0 <= mu <= 0
Value 0 Mean Value 0 Accuracy 0
Mean n.of tokens in _zdDTYnXZEeukXc_Ns4rdNQ : 0 <= mu <= 0
Value 0 Mean Value 0 Accuracy 0
Mean n.of tokens in _zdO5kHXZEeukXc_Ns4rdNQ : 1.48715067402e-05 <= mu <= 2.510
```

Invocations Registry

Simulation	Started	Finished	Status
<<GaAnalysisContext, GaScenario>> <Activity> CEPScenario [eba0fe38-934d-4930...	Tue Feb 23 14:19:49 CET 2...	<Unknown>	Running
9b3e2ec4-8f1c-488f-8561-cdd132ad4ad9	Tue Feb 23 14:19:49 CET 2...	Tue Feb 23 14:19:50 CET 2...	Finished
a072cee0-e20e-4b08-846c-49d355a2b2f1	Tue Feb 23 14:19:51 CET 2...	Tue Feb 23 14:19:52 CET 2...	Finished
e5a51f67-0f9c-4f0a-b6	Tue Feb 23 14:19:53 CET 2...	Tue Feb 23 14:19:54 CET 2...	Finished
97ea5711-19e3-4a92-	Tue Feb 23 14:19:54 CET 2...	Tue Feb 23 14:19:56 CET 2...	Finished
1ed49588-1658-41b8-	Tue Feb 23 14:19:56 CET 2...	Tue Feb 23 14:19:58 CET 2...	Finished
74862b1a-0c60-4801-	Tue Feb 23 14:19:58 CET 2...	Tue Feb 23 14:20:00 CET 2...	Finished
017e903b-ccea-4ee3-a	Tue Feb 23 14:20:00 CET 2...	Tue Feb 23 14:20:02 CET 2...	Finished
32cfb118-402e-4f27-b5d5-f1631f55ad36	Tue Feb 23 14:20:02 CET 2...	<Unknown>	Running
d9017610-6703-4202-8847-5dd6cce27409	<Unknown>	<Unknown>	Waiting
40de4281-c34d-4de3-96f5-4924a0ebd533	<Unknown>	<Unknown>	Waiting

Fig. 9 Running a simulation experiment



Invocations Registry

Simulation	Started	Finished	Status
<<GaAnalysisConte … y> CEPScenario [eba0fe38-934d-4930...	Tue Feb 23 14:19:49 CET 2...	Tue Feb 23 14:25:09 CET 2...	Killed
9b3e2ec4-8f1c … d9	Tue Feb 23 14:19:49 CET 2...	Tue Feb 23 14:19:50 CET 2...	Finished
a072cee0-e20e-4b08-846c-49d355a2b2f1	Tue Feb 23 14:19:51 CET 2...	Tue Feb 23 14:19:52 CET 2...	Finished
e5a51f67-0f9c-4f0a-b673-30d6de7361eb	Tue Feb 23 14:19:53 CET 2...	Tue Feb 23 14:19:54 CET 2...	Finished
97ea5711-19e3-4a92-976c-a3f39f535714	Tue Feb 23 14:19:54 CET 2...	Tue Feb 23 14:19:56 CET 2...	Finished
1ed49588-1658-41b8-b969-ba0925d95081	Tue Feb 23 14:19:56 CET 2...	Tue Feb 23 14:19:58 CET 2...	Finished
74862b1a-0c60-4801-b09f-205abc3ba959	Tue Feb 23 14:19:58 CET 2...	Tue Feb 23 14:20:00 CET 2...	Finished
017e903b-ccea-4ee3-a543-f28c44f3e83f	Tue Feb 23 14:20:00 CET 2...	Tue Feb 23 14:20:02 CET 2...	Finished
32cfb118-402e-4f27-b5d5-f1631f55ad36	Tue Feb 23 14:20:02 CET 2...	Tue Feb 23 14:25:04 CET 2...	Killed
d9017610-6703-4202-8847-5dd6cce27409	Tue Feb 23 14:25:04 CET 2...	Tue Feb 23 14:25:07 CET 2...	Finished
40de4281-c34d-4de3-96f5-4924a0ebd533	Tue Feb 23 14:25:07 CET 2...	Tue Feb 23 14:25:09 CET 2...	Finished

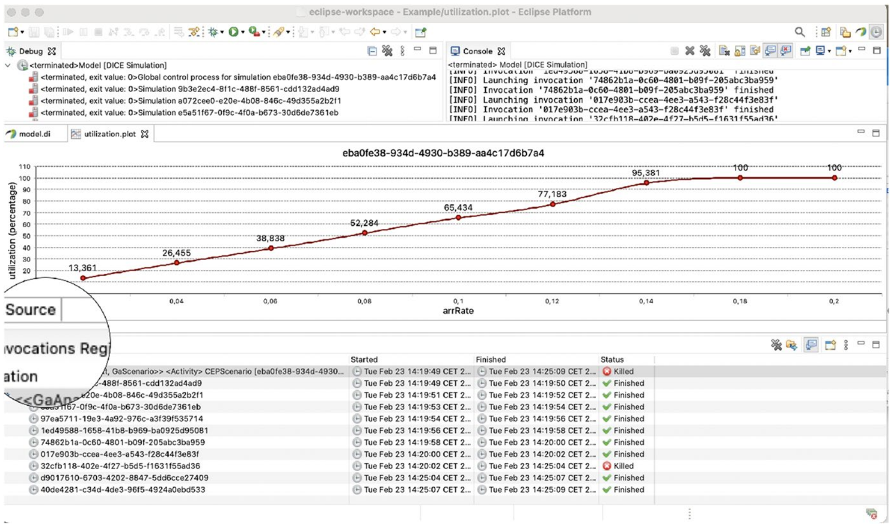
Fig. 10 Generating a 2D plot of the performance results

performance goals addressed, but none of them report, as this one, about the tool architecture, its usage nor implementations specifically.

The models and results of both cases studies are publicly available. The models are provided with the installation of the tool, also they can be downloaded from GitHub[19]. The results are in the Zenodo[25][26] repository. We are aware of the difficulty of reproducing these empirical results, mainly due to the inherent complexity of the models and experiments and due to the need for experience in the application domains. Therefore, we do not expect a practitioner to reproduce them, that is the purpose of Sect. 6. The purpose of this section is only to reveal two cases where the Simulation tool was successful.

In the following, we illustrate the major functionalities of the Simulation tool by describing how it contributed to these case studies.

[25] http://doi.org/10.5281/zenodo.1010446
[26] http://doi.org/10.5281/zenodo.1134267



```
<terminated>Model [DICE Simulation]
  <terminated, exit value: 0>Global control process for simulation eba0fe38-934d-4930-b389-aa4c17d6b7a4
  <terminated, exit value: 0>Simulation 9b3e2ec4-8f1c-488f-8561-cdd132ad4ad9
  <terminated, exit value: 0>Simulation a072cee0-e20e-4b08-846c-49d355a2b2f1
  <terminated, exit value: 0>Simulation e5a51f67-0f9c-4f0a-b673-30d6de7361eb
```

```
<terminated> Model [DICE Simulation]
[INFO] Launching invocation '74862b1a-0c60-4801-b09f-205abc3ba959'
[INFO] Invocation '74862b1a-0c60-4801-b09f-205abc3ba959' finished
[INFO] Launching invocation '017e903b-ccea-4ee3-a543-f28c44f3e83f'
[INFO] Invocation '017e903b-ccea-4ee3-a543-f28c44f3e83f' finished
[INFO] Launching invocation '32cfb118-402e-4f27-b5d5-f1631f55ad36'
```

	Started	Finished	Status
t, GaScenario>> <Activity> CEPScenario [eba0fe38-934d-4930...	Tue Feb 23 14:19:49 CET 2...	Tue Feb 23 14:25:09 CET 2...	Killed
c-488f-8561-cdd132ad4ad9	Tue Feb 23 14:19:49 CET 2...	Tue Feb 23 14:19:50 CET 2...	Finished
e20e-4b08-846c-49d355a2b2f1	Tue Feb 23 14:19:51 CET 2...	Tue Feb 23 14:19:52 CET 2...	Finished
f67-0f9c-4f0a-b673-30d6de7361eb	Tue Feb 23 14:19:53 CET 2...	Tue Feb 23 14:19:54 CET 2...	Finished
97ea5711-19e3-4a92-976c-a3f39f535714	Tue Feb 23 14:19:54 CET 2...	Tue Feb 23 14:19:56 CET 2...	Finished
1ed49588-1658-41b8-b969-ba0925d95081	Tue Feb 23 14:19:56 CET 2...	Tue Feb 23 14:19:58 CET 2...	Finished
74862b1a-0c60-4801-b09f-205abc3ba959	Tue Feb 23 14:19:58 CET 2...	Tue Feb 23 14:20:00 CET 2...	Finished
017e903b-ccea-4ee3-a543-f28c44f3e83f	Tue Feb 23 14:20:00 CET 2...	Tue Feb 23 14:20:02 CET 2...	Finished
32cfb118-402e-4f27-b5d5-f1631f55ad36	Tue Feb 23 14:20:02 CET 2...	Tue Feb 23 14:25:04 CET 2...	Killed
d9017610-6703-4202-8847-5dd6cce27409	Tue Feb 23 14:25:04 CET 2...	Tue Feb 23 14:25:07 CET 2...	Finished
40de4281-c34d-4de3-96f5-4924a0ebd533	Tue Feb 23 14:25:07 CET 2...	Tue Feb 23 14:25:09 CET 2...	Finished

Fig. 11 Plot of utilization vs. mean arrival rate

7.1 Posidonia case study

PRODEVELOP[27], located in Valencia (Spain), is a medium-sized enterprise employing more than 80 engineers, with high expertise in advanced geospatial technologies. PRODEVELOP developed Posidonia Operations, a commercial product deployed by many *port authorities* across Europe, Africa and America. Posidonia processes streamed data from *Automatic Identification System* (AIS) receivers, a system that gets vessels position in real time. A *complex event processing* (CEP) engine correlates the AIS messages in time and space by means of a set of geospatial rules to identify events produced by the vessels in the port, e.g. port enter and leaving, berth change, anchoring, tugs or repairs.

In particular, the Simulation tool helped evaluating three non functional requirements for the upcoming version of the system, as reported in Bernardi et al. (2018):

NFR1 — *Scalability* of the product under different velocity and volume of data to be processed.
NFR2 — *Bottlenecks* in the processing of the CEP rules, as well as in the AIS data parsing implementation.
NFR3 — *Performance* impact of a business rule on the CEP.

System scenarios, in terms of UML sequence and activity diagrams were modeled using the Simulation tool. Moreover, a UML deployment diagram, see Fig. 4, modeled the distribution of system components onto processing nodes. The UML

27 https://www.prodevelop.es

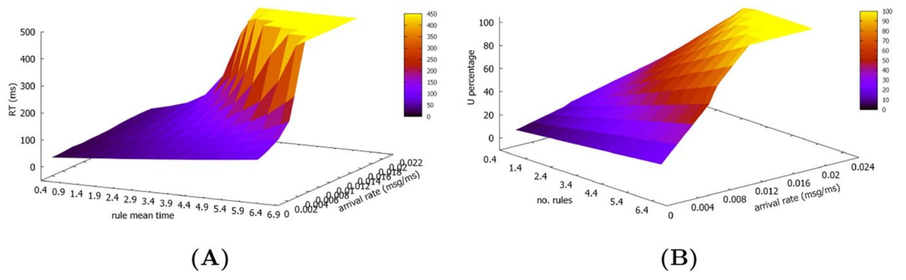


Fig. 12 CEP response time (left) and utilization (right) vs/ rule mean time (ms) and arrival rate (msg/ms) (taken from Bernardi et al. (2018))

MARTE profile was used to annotate, in the model, performance input and output parameters. Input parameters were the system open-workload, the duration of the system activities and the load of the messages, the probabilities associated to decision steps and the number of logical resources allocated in physical ones. Regarding output parameters (i.e., metrics), they were necessary for reasoning about the non functional requirements. So, for NFR1, we explicitly annotated response times and throughput in each performance scenario. For NFR2, we annotated the utilization of the logical resources. For assessing NFR3, all the output parameters (response time, throughput and utilizations) were used. Specific details about how the performance assessment, for each NFR, was carried out with the support of the Simulation tool were reported in Sect. 6 of Bernardi et al. (2018). Notably, for NFR3 we analyzed the impact of the mean execution time of a business rule activation on the CEP execution cycle scenario and the CEP resource utilization. The performance results, computed by the tool through sensitivity analysis, were synthetized in 3D plots (Fig. 12) that revealed, from the one hand, the exponential grow of the CEP execution cycle time (plot A) as the mean execution time of the rule activation increased and, on the other hand, the saturation of the CEP resource (plot B) for high values of the mean execution time and arrival rate parameters in the considered range of values (i.e., mean execution time of rule activation greater than 5.4 ms and arrival rate of 0.014 messages/ms).

7.2 NewsAsset case study

NewsAsset[24] is a commercial product developed by the Athens Technological Center, ATC (Greece)[28]. Technically, it is a distributed multi-tier engine that provides services to journalist in handling large volumes of information from heterogeneous sources, like social or sensor networks that feed the Internet continuously. News agencies realized the importance for journalists to access and handling, in real time, the vast source of data provided by Twitter, Facebook or Instagram or by sensors capturing traffic information on roads or air pollution levels. Managing such

[28] https://www.atc.gr/.

volume of sensitive information of text, images, reports or videos requires to adopt technologies, such as Apache Storm[29], for collecting, processing and aggregating big streams of data and converting them in useful services that help journalists to produce new stories.

Services of NewsAsset need to continuously evolve while they must unceasingly operate satisfying high quality requirements in terms of reliability and performance. The work in Requeno et al. (2019) described the modernization of NewsAsset for introducing Apache Storm, so to address three major challenges:

Ch1 — Refactoring of the old-fashioned engine related to cloud processing and Big Data technologies.
Ch2 — Reconfiguration of the obsolete architecture with respect to quality-driven metrics.
Ch3 — Managing the complexity real-time responsiveness for temporal peaks of high computational demand.

The refactoring (Ch1) was focused on changing the batch processing core, in the News Orchestrator application, with stream processing capabilities. Then, optimizing processing time and maximizing the crawling capacity for analyzing as much social networks content as possible. In particular, the topic-detector module of the News Orchestrator was refactored using Apache Storm to increase data intensive computation for extracting abnormal increase of frequent features (e.g., hashtags), i.e., for detecting trending topics. The topic-detector was modeled using the Simulation tool, as an Apache Storm application (see functionality **AF1** of the tool). Fig. 13 depicts the workflow with the DICE profile annotations[30]. The profile annotations introduced the system performance parameters, as well as the metrics to be computed (**MF1**). Among the former, the tool allowed to model the parallelism and expected mean execution times of the computational activities, i.e., spouts and bolts, grouping policies, weights and probabilities of the communication streams and also the characteristics of the computational resources in a deployment diagram. The UML activity diagram, the deployment diagram and the profile annotations enabled the Simulation tool to compute the quality-driven metrics (**MF2** and **MF3**) that assessed the reconfiguration of the architecture (Ch2).

The ATC engineers decided to evaluate the challenges addressed by carrying out two studies with the Simulation tool:

S1 — Scalability of the current topology.
S2 — Performance of alternative architectures.

[29] https://storm.apache.org.

[30] The image only depicts the profile stereotypes. However, the tool allows to specify for each stereotyped element all the tagged values that detail its properties. For example, for a bolt or a spout, its parallelism, host demand or probability to execute can be specified.

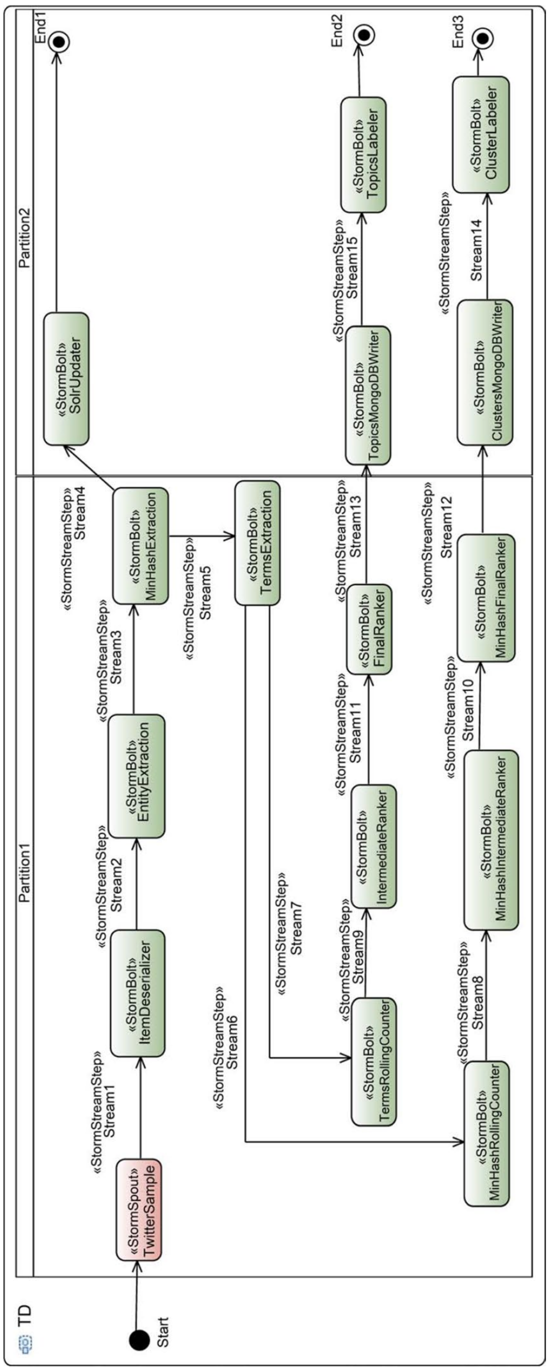

TD
Start
Partition1
Partition2
«StormSpout» TwitterSample
«StormStreamStep» Stream1
«StormBolt» ItemDeserializer
«StormStreamStep» Stream2
«StormBolt» EntityExtraction
«StormStreamStep» Stream3
«StormBolt» MinHashExtraction
«StormStreamStep» Stream4
«StormBolt» SolrUpdater
End1
«StormStreamStep» Stream5
«StormBolt» TermsExtraction
«StormStreamStep» Stream6
«StormStreamStep» Stream7
«StormBolt» TermsRollingCounter
«StormStreamStep» Stream9
«StormBolt» IntermediateRanker
«StormStreamStep» Stream11
«StormBolt» FinalRanker
«StormStreamStep» Stream13
«StormBolt» TopicsMongoDBWriter
«StormStreamStep» Stream15
«StormBolt» TopicsLabeler
End2
«StormBolt» MinHashRollingCounter
«StormStreamStep» Stream8
«StormBolt» MinHashIntermediateRanker
«StormStreamStep» Stream10
«StormBolt» MinHashFinalRanker
«StormStreamStep» Stream12
«StormBolt» ClustersMongoDBWriter
«StormStreamStep» Stream14
«StormBolt» ClusterLabeler
End3

Fig. 13 UML activity diagram with profile annotations (taken from Requeno et al. (2019))

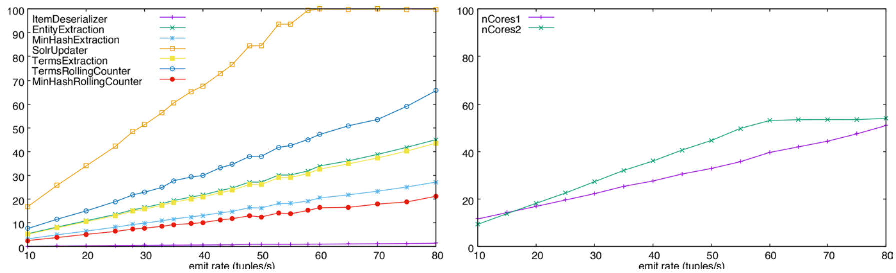


Fig. 14 Utilization (%) of bolts on the left, and utilization (%) of cores on the right (taken from Requeno et al. (2019))

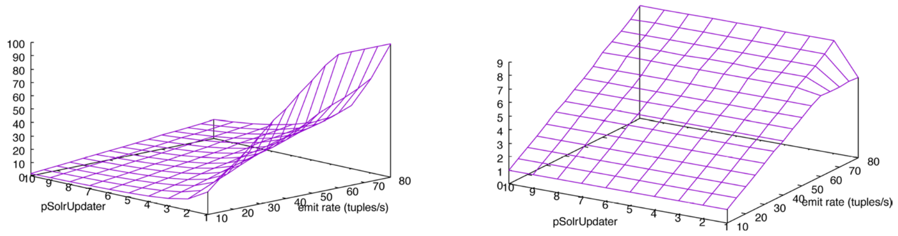


Fig. 15 Utilization (%) of *SolrUpdater* on the left, and throughput (tuples/ms) of *SolrUpdater* on the right (taken from Requeno et al. (2019))

For S1 the software architects were interested in figuring out whether the system was linearly scalable. The Simulation tool was used to conduct a bottleneck analysis, in order to identify the type of resources that needed to be increased. Starting from the UML diagrams that modeled the refactored architecture, the utilizations of different software and hardware resources were computed, see Fig. 14. It was confirmed that the increase in the number of computation cores will not improve the system performance. The engineers were assessed to redesign certain algorithms in the workflow implying the parallelism of some bolts that were identified as bottlenecks.

For S2 the software quality engineers were interested in measuring the impact of different architecture alternatives based on performance and cost. The engineers used the Simulation tool for modeling Apache Storm performance features (**AF1**) on different architectures for the News Orchestrator. In particular, multiple combinations of system workloads were considered, including high peak demands (Ch3), bolts parallelism and hardware configurations. Then, they conduct *what-if analyses* (**AF2**) of such architectures by computing resource utilizations and system throughputs. Figure 15 depicts the utilizations and throughputs obtained for a critical component. Finally, the optimal configuration was assessed, even detailing the specific configuration for each bolt in the system.

8 Discussion

The paper has so far presented the most important aspects of the Simulation tool. However, we consider to discuss some final concerns, that will allow to better understand about the suitability, scalability and limitations of the tool.

8.1 On the tool correctness

The Simulation tool was proposed in the H2020 contract of the DICE project as a functional demonstrator, i.e., Technology readiness level 5 (TRL 5[31]). Therefore, we developed an extensive plan of software tests, guided by examples, with the aim of validating the tool functionalities described in the contract. The idea was that the tool continually passes through validation and verification stages. Regarding the verification, the tool repository includes groups of tests, both unit tests and integration tests between modules. For instance, an integration test involves the GreatSPN engine and all the components in the *Model-based simulation layer* in Fig. 2. These tests can be found in the GitHub project[32] of the tool. Regarding the validation, we claim it since the tool has been extensively tested in real projects, then ensuring that it does the “right thing". Last paragraph of this subsection elaborates this aspect.

We also consider important to point out that the correctness of the tool mainly resides on the implementation of the components that carry out the model transformations. Notably, on the correctness of the theoretical proposals to transform software designs into formal models. At this regard, we identified two components, the **M2M transformation** and **Solution builder**, see Fig. 2. Concretely, the first one implements transformations from UML software models to Petri net models, following the theory developed in Woodside et al. (2014). All the design decisions and details on the implementations of these model-to-model transformations are available in The DICE Consortium (2016). The second component is to interpret the Petri nets results in the UML domain, *backward* transformation, Sect. 4.2 already explained how to compute the performance and reliability metrics. Although the development of the above components required deep theoretical knowledge on the specific topic of the model-transformation, we have made extensive use of traceability links between the assets exploited by both components to ensure consistency and to minimize the chances of failure.

Another important aspect that impacts on the correctness of the Simulation tool refers to the correctness of the third-party dependencies. The most prominent one is the GreatSPN tool. Section 4.2 presented a broad discussion on the correctness of GreatSPN. Despite the extensive tests passed by GreatSPN in the last decades, if new bugs in the GreatSPN simulation engine were discovered, they would possibly be inherited by the Simulation tool.

[31] TRL 5 denotes technologies validated in relevant environments (industrially relevant environments in the case of key enabling technologies). See Annex G of the HORIZON 2020 – Work Programme 2014–2015: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

[32] https://github.com/dice-project/DICE-Simulation/tree/master/tests.

The final aspect that deserves to be discussed regarding the correctness of the Simulation tool refers to its use in real projects. The tool has been applied in two industrial case studies. The results of both projects confirm that the simulations of the tool provide accurate results, that are very close to the ones obtained by experimentation, as reported in Sect. 7. Despite all these efforts, we can never claim to be completely bug free.

8.2 On the tool scalability

Large-scale software, such as micro-services or distributed systems in general, are today mainstream, as well as one of the main concerns in the field of software development. The Simulation tool was developed in the context of the DICE project, that aimed to address data intensive applications, an important kind of distributed systems. In fact, the case studies reported in this paper, Posidonia and NewsAsset, are both examples of very large distributed systems, where the Simulation tool has been successfully applied. Nevertheless, we consider important to discuss about the ability of the Simulation tool to handle models, that design large systems. since it will identify the components involved.

One of the main aspects that affects the execution time of the Simulation tool is the simulation of the GSPN models. The **GreatSPN** engine and **Simulator engine handler** component are those active during simulation time. The simulation engine will keep running until the results it produces reach a previously configured confidence interval and error. But, depending on certain parameters of the model, specially the probabilities associated to decision nodes, the simulation may last for long time. For example, if the scenario is represented by a UML activity diagram and it contains operations rarely executed, then the simulation will require longer execution times than if no rare operations are in the model. This will happen irrespective of the topology of the scenario, i.e., even though the number of operations, decision nodes and fork/joins are the same. By rare operations we mean those reached only after passing a number of decision nodes with very low probability. Certainly, this is a common issue in simulation-based system evaluations. To mitigate this situation and to avoid that the Simulation tool freezes for long time, we implemented two solutions. First, the simulation can be manually aborted using a button in the GUI (**FF2**). Second, the tool allows to configure a maximum execution time for the simulation run.

Another issue could be in the generation of the formal models. However, the size of the obtained GSPN model, that represents the software system, is linear with the size of the corresponding UML model. Therefore, the **M2M transformation** or **M2T transformation** components are not affected by combinatory or state space explosion problems, which avoids any scalability issue when generating the models. In turn, the **Solution builder** component may need to traverse the GSPN simulation output files in order to extract the appropriate values to construct the performance or reliability result. Since there is only one simulation output for each GSPN element, i.e., for each place and transition, this component does not show scalability issues either.

8.3 Limitations of the tool

The tool has limitations from both the modeling and analysis point of views. On the modeling side, the system under analysis has to be represented by a UML scenario, consisting of exactly one behavioral diagram, either activity or sequence, and one deployment diagram. The engineer can create different behavioral diagrams, but the analysis can be performed considering each behavioral diagram separately. This is because the forward transformations work on UML scenario basis. Another limitation is inherent to the front-end tool, i.e., the Papyrus UML modeler. Definitely, it provides a modest support for the modeling with sequence diagrams through the GUI, which limits its usability to the case of simple scenarios. In the case of complex scenarios, although it is possible to create combined fragments, the modeling task becomes cumbersome due to problems of synchronization of the graphical elements with the model ones.

On the analysis side, the tool can compute a limited set of performance and reliability metrics. This limitation is due to both the backward transformations and the GreatSPN simulation engine. The backward transformations extract only a subset of the results produced by the back-end tool and map them to the metrics at UML scenario level, whereas the latter computes steady state estimations, basically, mean values.

9 Conclusion

Systems need to meet functionalities, but quality attributes differentiate usable from non-usable software. Indeed, performance and reliability are among the most important quality attributes. The Simulation tool, presented in this paper, allows to analyze software models, according to the SPE principles, for assessing these quality attributes. As a result, the tool is an asset for software engineers, since it allows them to take informed decisions for sizing the software infrastructure to meet response times, throughputs or to minimize probabilities of failure.

In fact, applying SPE principles is a very good choice for addressing quality problems before the software is deployed. At this regard, our simulation tool is just a small step ahead to contribute to the automation of the SPE methodologies.

The paper has identified and analyzed, in Sect. 8, the limitations of the tool. Most of them are already addressed by the tool, others will be taken into account in upcoming versions, but some of them depend on third-party tools, which is beyond our possibilities. Although the tool addresses main performance and reliability metrics, others could be integrated, probably new solvers would be needed to implement or existing ones could be connected.

Appendix

The manuscript has not been submitted to other journal for consideration. The submitted work is original and has not been published elsewhere. Regarding results in Sect. 7, it is clearly identified the sources of the results.

Code availability

Table 3.

Table 3 Code metadata

Nr.	Code metadata description	
C1	Current code version	v1.1.0
C2	Permanent link to code/repository	https://doi.org/10.5281/zenodo.4694078
C3	Legal Code License	Eclipse Public License - v 1.0
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Java
C6	Compilation requirements, operating environments & dependencies	Maven 3.6.3, Java 11, Eclipse Platform 4.18 (2020-12) (only required for debugging), and GreatSPN 3.0 (only required for executing integration tests)
C7	Developer documentation/manual	https://github.com/dice-project/DICE-Simulation/wiki/Developer-Resources
C8	Support email for questions	jmerse@unizar.es

Executable software version

Table 4.

Table 4 Software metadata

Nr.	(Executable) Software metadata description	
S1	Current software version	1.1.0
S2	Permanent link to executables of this version	https://github.com/dice-project/DICE-Simulation/wiki/Installation
S3	Legal Software License	Eclipse Public License - v 1.0
S4	Computing platform / Operating System	Eclipse Platform (cross-platform)
S5	Installation requirements & dependencies	Java 11, Eclipse Platform 4.18 (2020-12), and GreatSPN 3.0
S6	Link to user manual	https://github.com/dice-project/DICE-Simulation/wiki/
S7	Support email for questions	jmerse@unizar.es

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The Simulation tool was developed under DICE project, EU H2020 Grant agreement ID: 644869. J. Merseguer and S. Bernardi have been funded by project PID2020-113969RB-I00 of the Spanish Ministry of Science and Innovation. A. Gomez by FAME-RTI2018-093608-B-C31). J. Requeno by FORTE-CM ref. S2018/TCS-4314.

Declarations

Conflict of interest The research leading to these results received the funding declared in Sect. A.1. The authors have no other relevant financial or non-financial interests to disclose. The authors have no conflict of interest.

References

Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. John Wiley and Sons (1995)

Andrade, E.C., Alves, M., Matos, R., Silva, B., Maciel, P.: Openmads: An open source tool for modeling and analysis of distributed systems. In International Conference on Computer Safety, Reliability, and Security, pages 277–284. Springer, (2013) Balbo, G., Silva, M. (eds.): Performance Models for Discrete Event Systems with Synchronizations: Formalisms and Analysis Techniques. Editorial KRONOS, Zaragoza, Spain (1998)

Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software **82**(1), 3–22 (2009). https://doi.org/10.1016/j.jss.2008.03.066. (Special Issue: Software Performance - Modeling and Analysis.)

Bernardi, S., Dominguez, J.L., Gomez, A., Joubert, C., Merseguer, José, Perez-Palacin, D., Requeno, J.I., Romeu, A.: A systematic approach for performance assessment using process mining. Empirical Software Engineering, 23(6):3394–3441, 2018. https://doi.org/10.1007/s10664-018-9606-9

Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. Softw. Syst. Model. **10**(3), 313–336 (2011). https://doi.org/10.1007/s10270-009-0128-1

Bernardi, S., Merseguer, J., Petriu, D.C.: Model-Driven Dependability Assessment of Software Systems. Springer Publishing Company, Berlin (2013)

Bézivin, J., Devedzic, V., Djuric, D., Favreau, J.-M., Gasevic, D., Jouault, F.: An M3-Neutral Infrastructure for Bridging Model Engineering and Ontology Engineering. In D. Konstantas, J.-P. Bourrières, M. Léonard, and N. Boudjlida, editors, Interoperability of Enterprise Software and Applications, pages 159–171, London, (2006). Springer London

Bézivin, J., Kurtev, I.: Model-based Technology Integration with the Technical Space Concept, (2006). Metainformatics Symposium. URL: https://hal.archives-ouvertes.fr/hal-00483587

Billington, J., Christensen, S., Van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The petri net markup language: Concepts, technology, and tools. In Applications and Theory of Petri Nets 2003. ICATPN 2003. Lecture Notes in Computer Science, vol 2679, ICATPN'03, page 483-505, Berlin, Heidelberg, (2003). Springer-Verlag

Boronat, A., Carsí, J., Ramos, I.: Algebraic Specification of a Model Transformation Engine. In Luciano Baresi and Reiko Heckel, editors, Fundamental Approaches to Software Engineering, pages 262–277, Berlin, Heidelberg, (2006). Springer Berlin Heidelberg

Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice: Second Edition. Synthesis Lectures on Software Engineering. Morgan & Claypool, (2017). https://doi.org/10.2200/S00751ED2V01Y201701SWE004

Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In 2008 IEEE International Conference on Software Testing Verification and Validation Workshop, pages 73–80, (2008). https://doi.org/10.1109/ICSTW.2008.54

Casale, G., Ardagna, D., Artac, M., Barbier, F., Di Nitto, E., Henry, A., Iuhasz, G., Joubert, C., Merseguer, J., Munteanu, V. I., Perez, J. F., Petcu, D., Rossi, M., Sheridan, C., Spais, I., Vladuic, D.: DICE: Quality-driven development of data-intensive cloud applications. In 2015 IEEE/ACM 7th International Workshop on Modeling in Software Engineering, pages 78–83, (2015). https://doi.org/10.1109/MiSE.2015.21

Cortellessa, V., Di Marco, A., Inverardi, P.: Model-Based Software Performance Analysis. Springer (2011). https://doi.org/10.1007/978-3-642-13621-4

Curino, C., Godwal, N., Kroth, B., Kuryata, S., Lapinski, G., Liu, S., Oks, S., Poppe, O., Smiechowski, A., Thayer, E., Weimer, M., Zhu, Y.: MLOS: An infrastructure for automated software performance engineering. In Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning, DEEM'20, pages 1–5, New York, NY, USA, (2020). ACM. https://doi.org/10.1145/3399579.3399927

Dipartimento di informatica, Università di Torino. GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets, Dec., (2015). URL: www.di.unito.it/~greatspn/index.html

Galvao, I., Goknil, A.: Survey of traceability approaches in model-driven engineering. In 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pages 313–313, 2007. https://doi.org/10.1109/EDOC.2007.42

Gansner, E.R., Eleftherios, K., Stephen, N.: Drawing graphs with *dot*, (2015). URL: https://www.graphviz.org/pdf/dotguide.pdf

Gómez, A., Smith, C.U., Spellmann, A., Cabot, J.: Enabling Performance modeling for the masses: Initial experiences. In F. Khendek and R. Gotzhein, editors, System Analysis and Modeling. Languages, Methods, and Tools for Systems Engineering, pages 105–126, Cham, (2018). Springer International Publishing

Guerra, E., de Lara, J., Orejas, F.: Pattern-based model-to-model transformation: Handling attribute conditions. In R.F. Paige, editor, Theory and Practice of Model Transformations, pages 83–99, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg

Dependability Management (2003). Part 3-1: Application Guide: Analysis Techniques for dependability: Guide on methodology

Ivanov, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In 4th International Symposium on Distributed Objects and Applications, DOA 2002 - University of California, Irvine, United States, pages 1–6, October (2002). URL: https://research.utwente.nl/en/publications/technological-spaces-an-initial-appraisal

Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley Professional Computing, (1991)

Kroß, J., Krcmar, H.: PerTract: model extraction and specification of big data systems for performance prediction by the example of apache spark and hadoop. Big Data Cogn. Comput. **3**(3), 47 (2019)

Li, C., Altamimi, T., Zargari, M. H., Casale, G., Petriu, D.: Tulsa: a tool for transforming UML to layered queueing networks for performance analysis of data intensive applications. In International Conference on Quantitative Evaluation of Systems, pages 295–299. Springer, (2017)

McGraw, G.: Software Security: Building Security In. Addison Wesley Professional, (2006)

Merseguer, J.: Binder, Walter, Murphy, John: Guest editorial: Automation in software performance engineering. Autom. Softw. Eng. **24**(1), 71–72 (2017). https://doi.org/10.1007/s10515-016-0201-2

Neilson, J.E., Woodside, C.M., Petriu, D.C., Majumdar, S.: Software bottlenecking in client-server systems and rendezvous networks. IEEE Trans. Software Eng. **21**(9), 776–782 (1995). https://doi.org/10.1109/32.464543

Object Management Group (OMG). XML Metadata Interchange, Version 2.5.1. OMG Document Number formal/2015-06-07 (http://www.omg.org/spec/XMI/2.5.1), (2006)

OMG. Modeling and Analysis of Real-time Embedded Systems (MARTE), Ver. 1.1. (2013) URL: http://www.omg.org/spec/MARTE/1.1/

OMG. Unified Modeling Language (UML), Ver. 2.5. (2007) URL: http://www.omg.org/spec/UML/2.5/

OMG. MOF Model to Text Transformation Language (MOFM2T), 1.0, Jan. (2008). URL: http://www.omg.org/spec/MOFM2T/1.0/

OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1, January (2011). URL: http://www.omg.org/spec/QVT/1.1/

OMG. Meta Object Facility (MOF) Core, Version 2.5.1, November (2016). URL: http://www.omg.org/spec/MOF/2.5.1/

Ozkaya, M.: Are the UML modelling tools powerful enough for practitioners? A literature review. IET Softw. **13**(5), 338–354 (2019)

Perez-Palacin, D., Merseguer, J., Requeno, J., Guerriero, M., Di Nitto, E., Tamburri, D. A.: A UML profile for the design, quality assessment and deployment of data-intensive applications. Software and Systems Modeling, 18(6):3577–3614, (2019). https://doi.org/10.1007/s10270-019-00730-3

Requeno, J.I., Merseguer, J., Bernardi, S., Perez-Palacin, D., Giotis, G., Papanikolaou, V.: Quantitative analysis of apache storm applications: The newsasset case study. Inf. Syst. Front. **21**(1), 67–85 (2019). https://doi.org/10.1007/s10796-018-9851-x

Reussner, R. H., Becker, S., Happe, J., Heinrich, R., Koziolek, A., Koziolek, H., Kramer, M., Krogmann, K.: The Palladio Approach. The MIT Press, Modeling and Simulating Software Architectures (2016)

Rodríguez, R.J., Bernardi, S., Zimmermann, A.: An evaluation framework for comparative analysis of generalized stochastic petri net simulation techniques. IEEE Trans. Syst. Man Cybern. Syst. **50**(8), 2834–2844 (2020). https://doi.org/10.1109/TSMC.2018.2837643

Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method. Wiley, (2008)

Smith, C.U., Williams, L.G.: Performance solutions: A practical guide to creating responsive. Addison-Wesley, Scalable Software (2002)

Smith, C.U., Lloyd, G.W.: Software performance engineering. In L. Lavagno, G. Martin, and B.V. Selic, editors, UML for Real: Design of Embedded Real-Time Systems, pages 343–365. Springer, (2003)

Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edition, (2009)

The DICE Consortium. State of the art analysis. deliverable d1.1. Technical report, European Union's Horizon 2020 research and innovation programme, (2015). URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.1_State-of-the-art-analysis1.pdf

The DICE Consortium. Transformations to analysis models. Deliverable D3.1, (2016). URL: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5abac3ce2&appId=PPGMS

The DICE Consortium. DICE simulation tools - Final version. Deliverable D3.4, (2017). URL: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5b419ef47&appId=PPGMS

The Eclipse Foundation. Designing and using UML profiles with Papyrus, Juny, (2012). URL: https://eclipse.org/papyrus/usersTutorials/resources/PapyrusUserGuideSeries_AboutUMLProfile_v1.0.0_d20120606.pdf

The Eclipse Foundation. Eclipse Platform, March, (2021). URL: https://projects.eclipse.org/projects/eclipse.platform

The Eclipse Foundation. A slide-ware tutorial on Papyrus usage for starters, Oct, (2010). URL: https://eclipse.org/papyrus/usersTutorials/resources/TutorialOnPapyrusUSE_d20101001.pdf

The Eclipse Foundation & Obeo. Acceleo, Dec., (2015). URL: https://eclipse.org/acceleo/

The Object Management Group (OMG). Model-Driven Architecture Specification and Standardisation, (2018). URL: http://www.omg.org/mda/

Wang, W., Pérez, J. F., Casale, G.: Filling the gap: A tool to automate parameter estimation for software performance models. In Proceedings of the 1st International Workshop on Quality-Aware DevOps, QUDOS 2015, page 31-32, New York, NY, USA, (2015). ACM. https://doi.org/10.1145/2804371.2804379

Woodside, M., Petriu, D., Merseguer, J., Petriu, D., Alhaj, M.: Transformation challenges: from software models to performance models. Software and Systems Modeling, 13:1529–1552, 10 2014. https://doi.org/10.1007/s10270-013-0385-x