

**Developing Data-Intensive Cloud
Applications with Iterative Quality
Enhancements**



DICE simulation tools - Final version

Deliverable 3.4

Deliverable:	D3.4
Title:	DICE simulation tools - Final version
Editor(s):	Diego Perez (ZAR)
Contributor(s):	Simona Bernardi (ZAR), José Merseguer (ZAR), José Ignacio Requeno (ZAR), Giuliano Casale (IMP), Lulai Zhu (IMP)
Reviewers:	Andrew Phee (FLEXI), Ismael Torres (PRO)
Type (R/P/DEC):	Report
Version:	1.0
Date:	31-July-2017
Status:	Final version
Dissemination level:	Public
Download page:	http://www.dice-h2020.eu/deliverables/
Copyright:	Copyright © 2017, DICE consortium – All rights reserved



The DICE project (February 2015-January 2018) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869

Executive summary

This document presents the final results of the development of the *Simulation tool*. It details the advances of this tool with respect to the intermediate version reported at M24. Main achievements in this final version are in the functionality offered by the tool. Nevertheless, it also contains new implementations in the graphical interaction with users. This deliverable is related to D1.2 (*Requirement specification*) as it evaluates the level of accomplishment of requirements achieved by this final version of the *Simulation tool*. New achievements on the functionality and graphical user interface allow the final version of the tool to fulfill all its requirements.

All the artifacts presented in this document are publicly available in the so-called *DICE-Simulation Repository* [36], whose structure and components are described in the Appendix A of this document.

Glossary

AWS	Amazon Web Services
DAM	Dependability Analysis and Modeling
DIA	Data-Intensive Application
DICE	Data-Intensive Cloud Applications with iterative quality enhancements
DPIM	DICE Platform Independent Model
DTSM	DICE Technology Specific Model
EC2	Elastic Cloud Computing
GUI	Graphical User Interface
IDE	Integrated Development Environment
JMT	Java Modelling Tools
M2M	Model-to-model Transformation
M2T	Model-to-text Transformation
MARTE	Modeling and Analysis of Real-time Embedded Systems
MDE	Model-Driven Engineering
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
OSGi	Open Services Gateway initiative
PNML	Petri Net Markup Language
QVT	Meta Object Facility (MOF) 2.0 Query/View/Transformation Standard
QVTo	QVT Operational Mappings language
RBD	Reliability Block Diagram
UML	Unified Modelling Language

Contents

Executive Summary	3
Glossary	4
Table of Contents	5
List of Figures	7
List of Tables	8
1 Introduction and Context	9
1.1 Structure of the Document	10
2 Summary of achievements in the intermediate version of the Simulation Tool	11
2.1 Requirements coverage summary	11
2.2 Summary of functionality and GUI of the intermediate version of the <i>Simulation tool</i>	12
3 Functionality enhancements to interpret user inputs	18
3.1 Enhancements on the models to simulate: Apache Spark	18
3.2 Enhancements on the quality properties that can be simulated	18
3.2.1 New performance metrics	19
3.2.2 New reliability metrics	21
3.3 Enhancements on the specification of SLA	22
4 Functionality enhancements to produce simulation results	24
4.1 Enhancements on the computation of performance results: Apache Spark	24
4.2 Enhancements on the computation of reliability results for Apache Hadoop, Storm, and Spark	25
4.2.1 Computation of MTTF	25
4.2.2 Computation of Availability	26
4.2.3 Computation of reliability $R(t)$	26
4.3 Enhancements on the solvers implementation	27
4.3.1 Solving with JMT	29
5 Updates in the Graphical User Interface	30
5.1 Graphical representation of specified SLA	30
5.2 Graphical configuration of quality properties solver	31
6 Simulation tool for containerized DIA deployments	34
7 Conclusion	35
References	37
Appendix A. The DICE-Simulation Repository	40
Appendix B. Modeling and transformations to Analysis Models of Apache Spark DIAs	41
B.1 Introduction of model-driven performance assessment of Spark applications	41
B.2 Spark and Performance	41
B.3 Modelling Spark applications with UML	44
B.4 A UML Profile for Spark	46
B.5 Transformation of the UML Design	49
Appendix C. JMT Petri Net Extension	54

Appendix D. Validation of Apache Spark	61
Appendix E. GSPN	63

List of Figures

1	Sequence diagram depicting the interaction between the components of the <i>Simulation tool</i>	12
2	Create a new <i>Simulation launch configuration</i> from icon in menu.	13
3	Create a new <i>Simulation launch configuration</i> from a workspace model.	13
4	A <i>Simulation launch configuration</i> window showing the Main tab.	14
5	A <i>Simulation launch configuration</i> window showing the Filters tab.	15
6	A <i>Simulation launch configuration</i> window showing the Parameters tab.	16
7	View of the <i>Invocations Registry</i> with the results of a <i>what-if</i> analysis composed of nine concrete simulations	16
8	Simulation result of DIA throughput	16
9	New option <i>Plot Results</i> within the <i>Invocations Registry</i> view	17
10	Plot showing <i>what-if</i> analysis results with the correlation of how the arrival rate affects the response time.	17
11	Sequence diagram highlighting the functionality enhancement with respect to the <i>enhancements on the models to simulate</i>	18
12	Activity diagram depicting a Spark execution scenario that is now accepted by the <i>Simulation tool</i>	19
13	Deployment Diagram modeling the resources used by a DIA based Spark technology. . .	19
14	Sequence diagram highlighting the functionality enhancement with respect to the <i>quality properties that can be simulated</i>	20
15	Definition of response time and throughput metrics in Spark scenarios	20
16	Definition of Utilization metric in Spark scenarios	20
17	Configuration of metrics to simulate	21
18	Definition of availability in a Hadoop scenario	21
19	Definition of MTTF in a Storm scenario	21
20	Definition of reliability in a Spark scenario	22
21	Definition of the response time metric to calculate and its SLA of 1 second	22
22	Sequence diagram highlighting the functionality enhancement with respect to the production of new simulation results.	24
23	Computed MTTF in a Storm application	26
24	Computed Availability in a Hadoop application	26
25	<i>what-if</i> analysis over variable MTTF to obtain a required Availability	27
26	Computed $R(t)$ in a Spark application	27
27	Computed $R(t)$ in a Spark application in function of the number of resources	28
28	Computed $R(t)$ in a Spark application in function of the mission time t	28
29	Sequence diagram highlighting the GUI enhancement with respect to the representation of results.	30
30	Computed $R(t)$ and limit SLA in a Spark application in function of the number of resources. .	31
31	Computed $R(t)$ and limit SLA in a Spark application in function of the mission time t . .	31
32	Sequence diagram highlighting the GUI enhancement with respect to the choice of solvers. .	32
33	Choice of simulation solver during the simulation configuration in <i>LaunchConfiguration</i> window.	32
34	Choice of simulation solver in the general preferences of the DICE IDE.	33
35	DAG representing a Spark application	42
36	Example of an activity diagram for Spark with profile annotations	44
37	Example of deployment diagram for Spark with profile annotations	45
38	GSPN for the Spark design in Figures 36 and 37	51
39	Example of a simple PN model with two closed classes.	54
40	Storage section panel of Place 1.	55
41	Enabling section panel of Transition 1.	55
42	Timing section panel of Transition 1.	56
43	Firing section panel of Transition 1.	56

44	Structure of a queueing place.	58
----	--	----

List of Tables

1	Level of compliance with requirements of the intermediate version of the <i>Simulation tool</i> , at M24. Data brought from D3.3 [15]	11
2	Level of compliance with requirements of the final version of the <i>Simulation tool</i>	35
3	Spark concepts that impact in performance	43
4	Spark profile extensions	48
5	Transformation Patterns for Spark-profiled UML Activity Diagrams	50
6	Transformation Patterns for SparkMap stereotype in Spark-profiled UML Activity Diagrams	52
7	Transformation Patterns for Spark-profiled UML Deployment Diagrams	53
8	Results of the SparkPrimes experiment	62

1 Introduction and Context

The goal of the DICE project is to define a quality-driven framework for developing data-intensive applications (DIA) that leverage Big Data technologies hosted in private or public clouds. DICE offers a novel profile and tools for data-aware quality-driven development. In particular, the goal of WP3 of the project is to develop a quality analysis tool-chain that will be used to guide the early design stages of the data intensive application and guide quality evolution once operational data becomes available. Therefore, the main outputs of tasks in WP3 are: tools for simulation-based reliability and efficiency assessment; tools for formal verification of safety properties related to the sequence of events and states that the application undergoes, and tools for numerical optimization techniques for searching the optimal architecture designs. Concretely, Task T3.2 in WP3 is in charge of developing the *DICE Simulation tool*, a simulation-based tool for reliability and efficiency assessment of DIA.

This deliverable describes the final version, at M30 of the project, of the *DICE Simulation tool*. The intermediate version of this *Simulation tool* was already reported, at M24, in deliverable D3.3 [15], while its initial version was reported in deliverable D3.2 [14] at M12. Simulations carried out by the *Simulation tool* are model-based simulations. To perform its model-based simulation task, the tool takes as input UML models annotated with the DICE profile developed in Task T2.2. Then, it uses Model-to-model (M2M) transformations, that transform the DIA execution scenarios represented in these profiled UML models into Petri net models, evaluates these Petri nets, and finally uses the results of the Petri net evaluation to obtain the expected performance and reliability values in the domain of DIA software models.

Compared with the intermediate version of the tool, released at M24, the final version has enriched its functionality and its Graphical User Interface (GUI) to accomplish all its requirements defined in [12,13].

The functional enhancements extend the capabilities of the tool both to accept and interpret different inputs, and to produce results of the quality of the DIA for the users. Regarding the extensions in accepted inputs, the final version of the tool is able to accept DIAs models based on Apache Spark technology. It is also able to read specifications of reliability properties at the technology specific level for Hadoop, Storm and Spark technologies, and to accept the specification of limit values for the Service Level Agreements (SLA) of quality properties.

Regarding the extensions in produced outputs, the final version of the tool is able to compute performance and reliability metrics for Spark technology and reliability metrics for Hadoop and Storm technologies (the ability to compute performance metrics for Hadoop and Storm was already present in the intermediate version of the tool at M24). This final version is also equipped with a new solver of the quality properties for which exists a direct analytical solution, and an additional third solver based on Java Modelling Tools (JMT) has been extended to evaluate the Petri nets generated by the output of the M2M transformations developed for the *Simulation tool*. JMT was extended to handle Petri net models as it allows to efficiently simulate certain classes of models that cannot be handled by GreatSPN, such as hybrid models mixing queueing networks and Petri nets. Moreover, with JMT it is possible to run experiments that read traces of monitoring data, allowing a more realistic parameterization of complex models, where needed.

Regarding the extensions on the graphical interaction of the tool with the user, the final version of the tool has enhanced the representation of simulation results in *what-if* analysis with a graphical representation of the SLA and has also enhanced its configuration windows to allow to graphically choose the quality properties solver to use.

The implementation of the previously mentioned M2M transformations used by the *Simulation tool* is also one of the objectives of Task T3.2. This objective is achieved in close interaction with WP2, since Tasks T2.1 and T2.2 of WP2 define the languages and notations that are used to express the input design models. Therefore, the M2M transformations that lacked in the intermediate version have been defined

and implemented. Transformation rules for Apache Hadoop and Storm were reported in Deliverable D3.1 [16], while transformation rules for the simulation of Apache Spark are reported in this document in Appendix B. The Apache Spark transformation reported in this document completes the set of M2M transformations implemented by the *Simulation Tool*.

The tool is implemented as a set of intercommunicated Eclipse plugins and it is integrated in DICE IDE. The code of the tool is published as an open source software that can be downloaded from the *DICE-Simulation* repository [36].

1.1 Structure of the Document

The structure of this deliverable is as follows:

- Section 2 presents a summary of the achievements already included in the intermediate version of the *Simulation tool* and the fulfillment of requirements of that intermediate version.
- Section 3 details the new functionalities of the final version the tool to accept inputs from users.
- Section 4 details the new functionalities of the final version of the tool to produce simulation results for users.
- Section 5 presents the new updates of the the final version of the tool with respect to the GUI.
- Section 6 discusses the utilization of the tool for DIAs that are deployed on containers.
- Section 7 presents the updates on the fulfillment of requirements by the final version of the *Simulation tool* and concludes the deliverable.

2 Summary of achievements in the intermediate version of the Simulation Tool

This section provides an overview of the intermediate version of the *Simulation tool* reported in D3.3 on M24, and a summary of the requirements accomplished by such intermediate version.

2.1 Requirements coverage summary

Deliverable D1.2 [12, 13], released at M6, presented the requirements analysis for the DICE project. The outcome of the analysis was a consolidated list of requirements and the list of use cases that define the project's goals that guide the DICE technical activities. Deliverables D3.2 [14] and D3.3 [15] gathered the requirements on Task T3.2 and reported the level of accomplishment of each of these requirements of the initial and intermediate versions of the *Simulation tool*, respectively. Table 1 brings from D3.3 [15] the level of accomplishment that the intermediate version of the tool already achieved for each requirement. It provides a schematic view of their **ID**, **Title** and **Priority**. The meaning of the labels used in column **Level of fulfillment** is the following: (i) ✗ (unsupported: the requirement is not fulfilled); (ii) ✓ (partially-low supported: a few of the aspects of the requirement are fulfilled); (iii) ✓ (partially-high supported: most of the aspects of the requirement are fulfilled); and (iv) ✓ (supported: the requirement is fulfilled and a solution for end-users is provided).

Table 1: Level of compliance with requirements of the intermediate version of the *Simulation tool*, at M24. Data brought from D3.3 [15]

Requirement	Title	Priority	Level of fulfillment intermediate
R3.1	M2M Transformation	MUST	✓
R3.2	Taking into account relevant annotations	MUST	✓
R3.4	Simulation solvers	MUST	✓
R3.6	Transparency of underlying tools	MUST	✓
R3.10	SLA specification and compliance	MUST	✓
R3.13	White/black box transparency	MUST	✓
R3IDE.1	Metric selection	MUST	✓
R3IDE.4	Loading the annotated UML model	MUST	✓
R3.3	Transformation rules	COULD	✓
R3.14	Ranged or extended what if analysis	COULD	✓
R3IDE.2	Timeout specification	SHOULD	✓
R3IDE.3	Usability	COULD	✗
R3IDE.7	Output results of simulation in user-friendly format	COULD	✓

Since M24, there has been an update in the DICE requirements that affect the work to carry out in Task 3.2 and in the *Simulation tool*. The update refers to the deprecation of R3IDE.3 “Usability”. The rationale for deprecating R3IDE.3 is that the *Simulation tool* is implemented as a set of Eclipse plugins to be incorporated either in the DICE IDE or to work as a standalone in a standard Eclipse installation. Therefore, the general usability and Look&Feel of the tool is strictly attached to the guidelines of Eclipse views, perspectives and editors. Nevertheless, within the remaining degree of freedom to compose the perspectives, views and configuration pages, the development of *Simulation tool* has highly taken into account its usability and user experience. Graphical interfaces and process interruptions have been implemented as much as possible to make easier the user interaction and to avoid unnecessary waiting times; albeit a requirement on the formal process of analyzing the Usability of the tool has been discarded.

2.2 Summary of functionality and GUI of the intermediate version of the *Simulation tool*

The high-level view of the behavior of the intermediate version of the tool and the interactions of their components is depicted in Figure 1 (brought from D3.3 [15]). The user interacts with the tool through a *simulate* step, whose characteristics have been set by a precedent configuration step. Once the *Simulator GUI* component receives the model to simulate and the simulation configuration characteristics, it invokes the *Simulator* component, which orchestrates the rest of steps until it gets the results of the simulation. These steps comprise: model-to-model M2M transformations of the input profiled UML model to create a Petri net, model-to-text M2T transformations to save the Petri net in files in the format accepted by *GreatSPN* simulation engine, the invocation to *GreatSPN* simulator and the processing of its results –which are in the domain of Petri nets– to transform them to the domain of the UML model and the quality properties to evaluate.

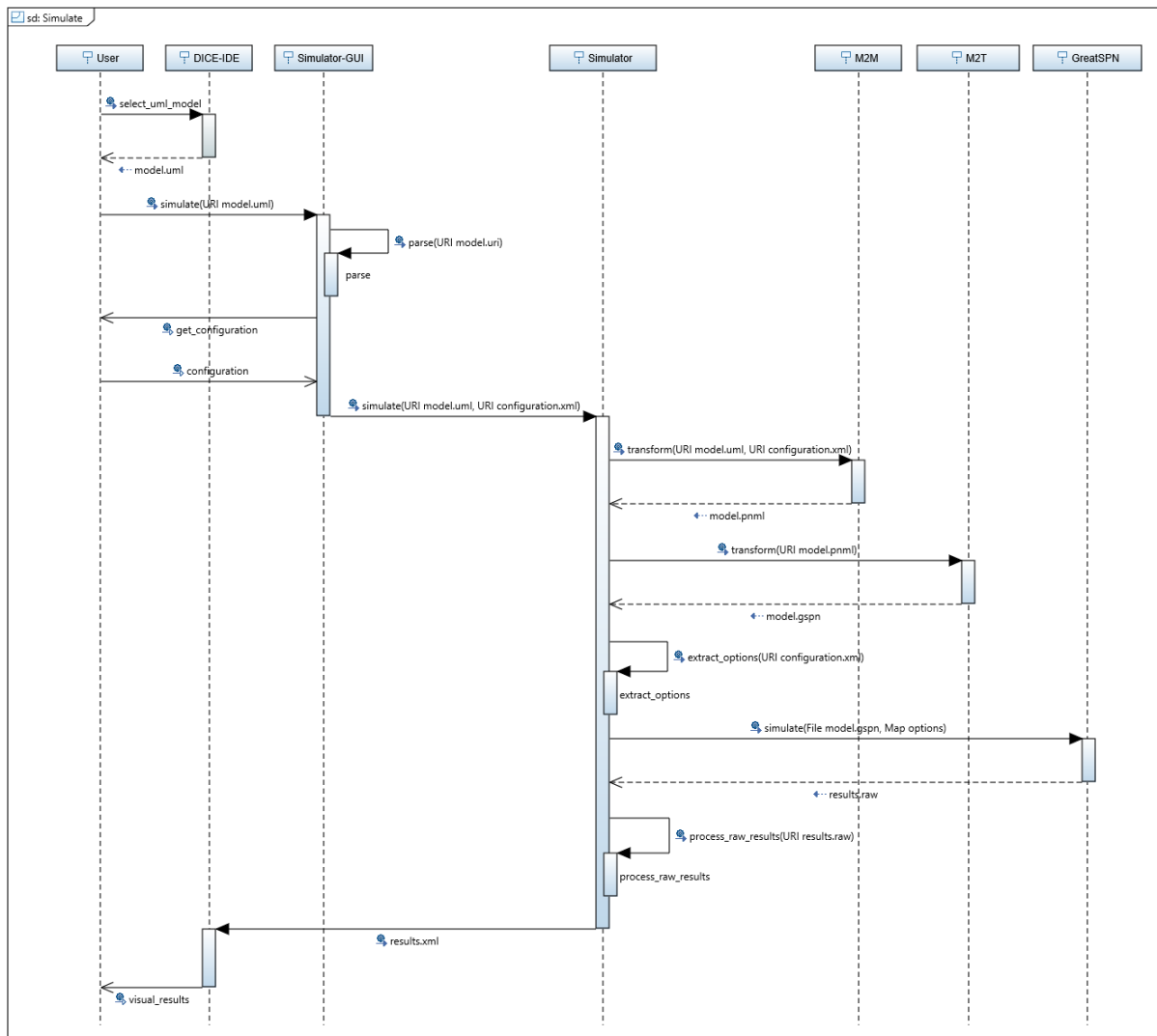


Figure 1: Sequence diagram depicting the interaction between the components of the *Simulation tool*

The functionality of the intermediate version of the tool accepted platform independent models (i.e., at DPIM level) for evaluating their performance and reliability, and also accepted Apache Hadoop and Storm technology specific model (i.e., at DTSM level) for evaluating their performance. Moreover, the tool accepted several configuration parameters, specially remarking its possibility to choose a *what-if* analysis over intervals of values, and already returned the simulation results in the domain of the quality metrics.

The GUI of the intermediate version was composed of:

- Two different ways to launch a simulation configuration: through the icon in the menu bar (see Figure 2), and through a shortcut menu when a UML model was selected in the IDE (depicted in Figure 3).

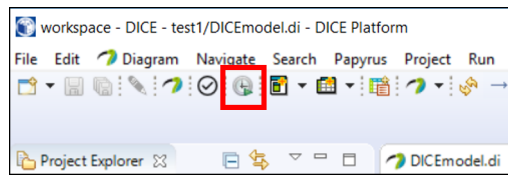


Figure 2: Create a new *Simulation launch configuration* from icon in menu.

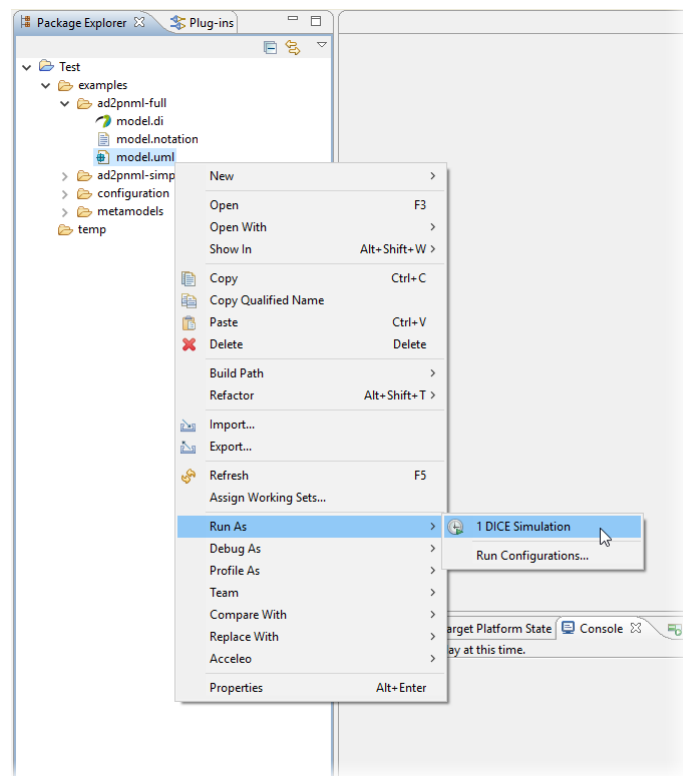


Figure 3: Create a new *Simulation launch configuration* from a workspace model.

- Configuration windows that allow to characterize the evaluation of the model. The available configuration options were: choose the scenario to simulate among the multiple possible scenarios in the model, choose the type of the quality evaluation, give value to variables in the model (enabling *what-if* analysis when multiple values are given to some variables), check the set of quality properties to evaluate, set maximum simulation execution time through a Timeout value. Figures 4-6 show these configuration windows.

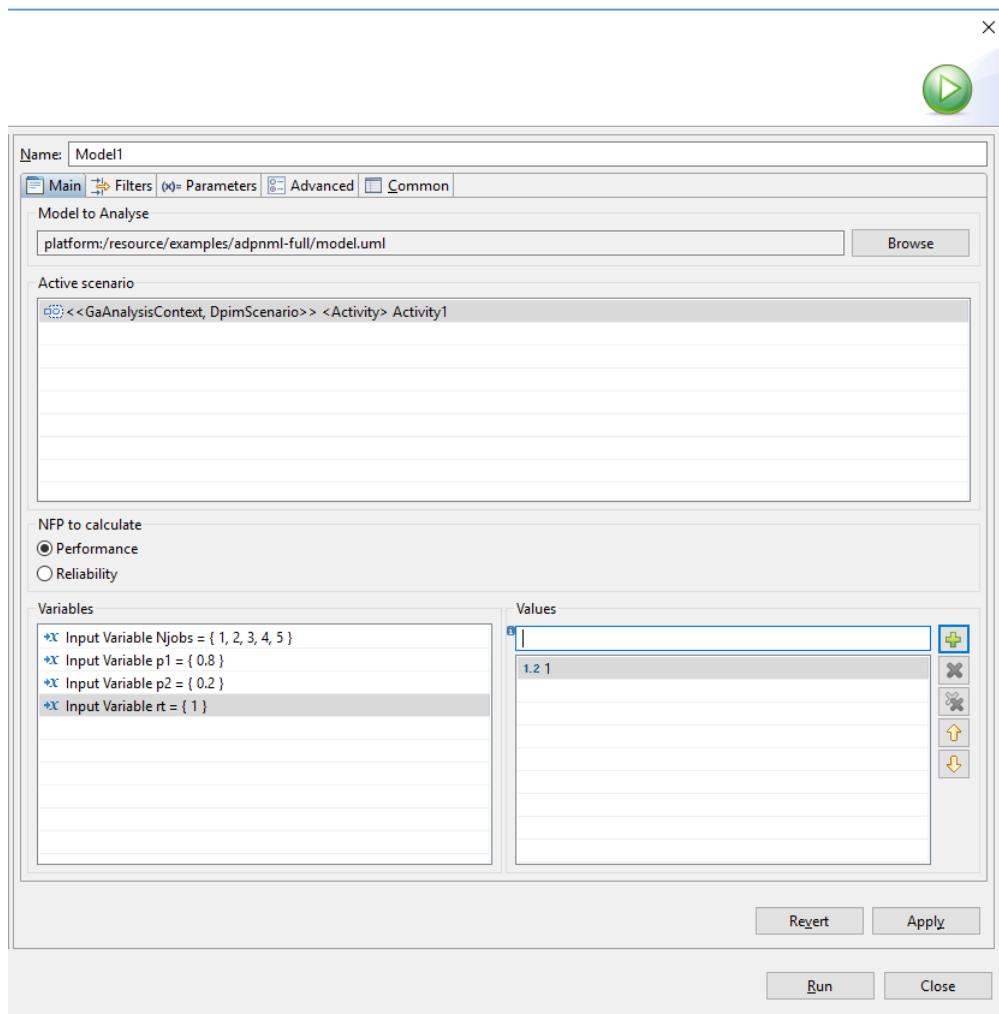


Figure 4: A *Simulation launch configuration* window showing the Main tab.

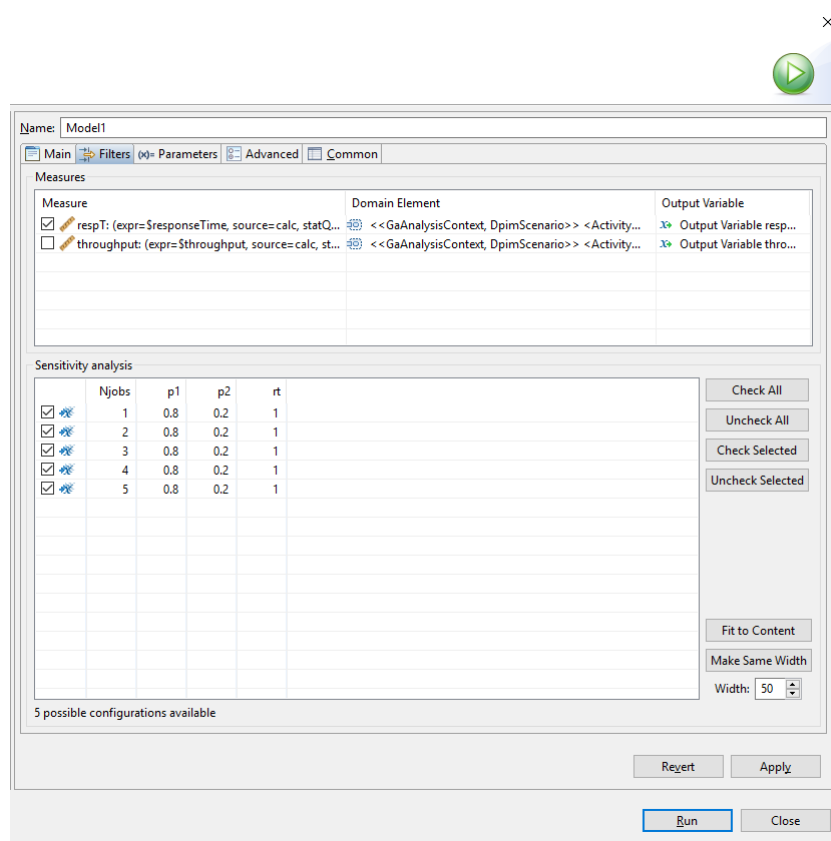


Figure 5: A *Simulation launch configuration* window showing the Filters tab.

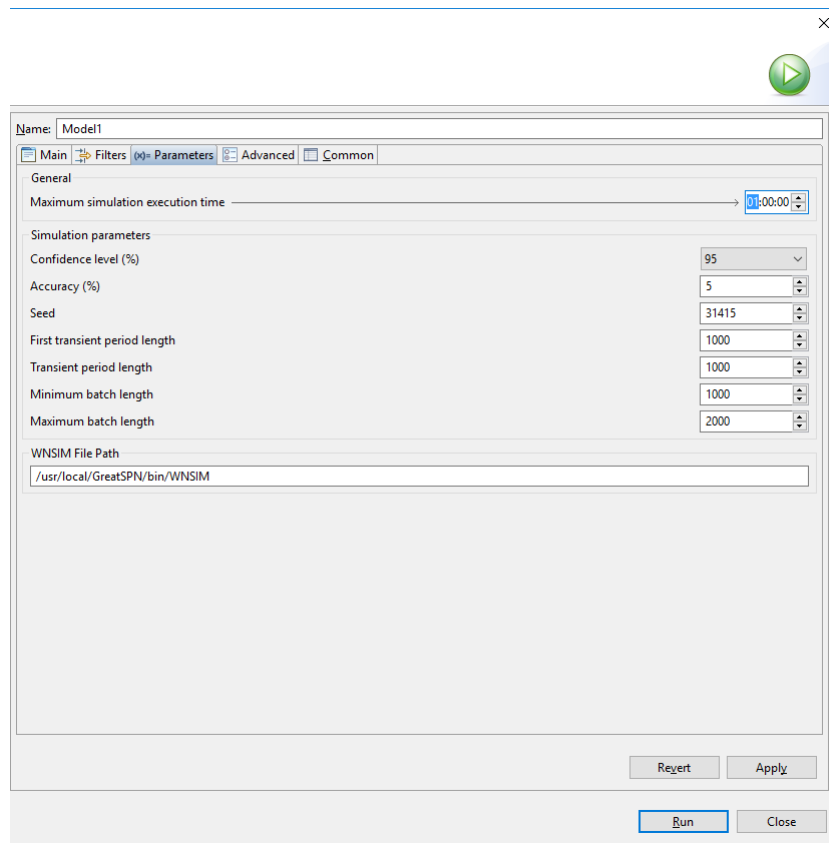


Figure 6: A Simulation launch configuration window showing the Parameters tab.

- A view for the simulation results, called *Invocation Registry* to keep a list of all the simulations executed and to see easily their quality results, also hierarchically grouped when a set of invocations where launched within a single *what-if* analysis. Figure 7 depicts this view *Invocation Registry* with lists of simulations and Figure 8 depicts an example of the visualization of a single result of the list for throughput evaluation.

Simulation	Started	Finished	Status
<<GaAnalysisContent.DpinScenario>> <Activity> Activity1 [91519d]	Mon Dec 19 12:14:13	Mon Dec 19 12:14:20	Finished
82c4928a-303c-48b2-bce3-38ab94c4da77	Mon Dec 19 12:14:13	Mon Dec 19 12:14:14	Finished
a11d8ceb-0eab-4ead-b9ba-e00c5242d062	Mon Dec 19 12:14:14	Mon Dec 19 12:14:15	Finished
28fec3f0-8b69-40a4-97f2-dfa44e53ef18	Mon Dec 19 12:14:15	Mon Dec 19 12:14:15	Finished
1bf39fad-32ea-41f4-bd10-34ca478c1e1a	Mon Dec 19 12:14:15	Mon Dec 19 12:14:16	Finished
bb45fa59-33d9-4ffe-a7a6-9998e0167eac	Mon Dec 19 12:14:16	Mon Dec 19 12:14:17	Finished
125f18ee-3744-4951-a001-945dd6a0555e	Mon Dec 19 12:14:17	Mon Dec 19 12:14:17	Finished
5b3f9f6c-6e5f-4968-ace6-292601e53df7	Mon Dec 19 12:14:18	Mon Dec 19 12:14:18	Finished
cac57539-a1a8-4acb-b283-2c97214ab58f	Mon Dec 19 12:14:18	Mon Dec 19 12:14:19	Finished
e609aa4f-e9ee-45e3-bccc-dcac30f9f233	Mon Dec 19 12:14:19	Mon Dec 19 12:14:20	Finished

Figure 7: View of the *Invocations Registry* with the results of a *what-if* analysis composed of nine concrete simulations

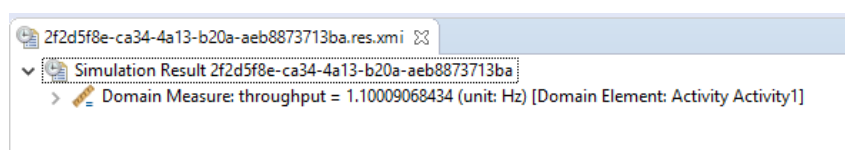


Figure 8: Simulation result of DIA throughput

- a Wizard to plot the results of *what-if* analysis. The wizard is launched from the container of

the *what-if* analysis into the *Invocation Registry* view, as depicted in Figure 9. After a three-step interaction process with the wizard, where it is set the location of the resulting plot, the independent variable to be placed in the x-axis of the plot and the variable to be placed in the y-axis, the result is a plot of the type illustrated in Figure 10 for the variables arrival rate (in x-axis) and response time (y-axis).

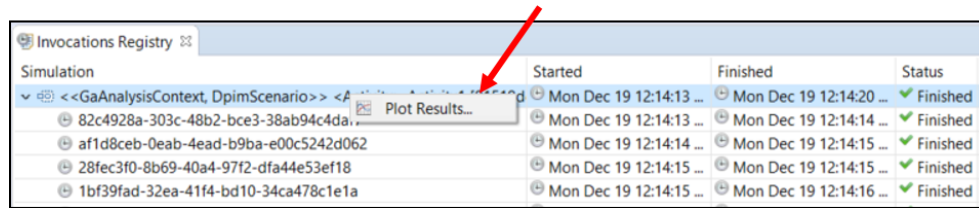


Figure 9: New option *Plot Results* within the *Invocations Registry* view

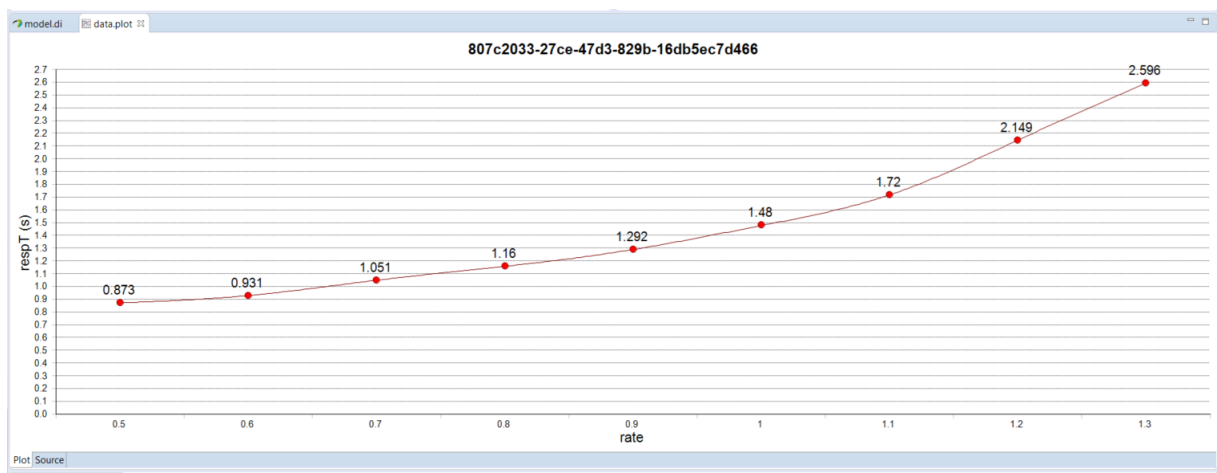


Figure 10: Plot showing *what-if* analysis results with the correlation of how the arrival rate affects the response time.

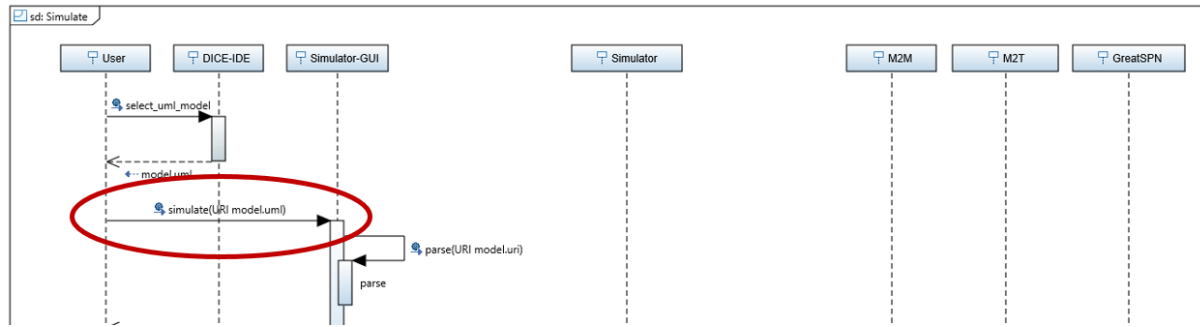


Figure 11: Sequence diagram highlighting the functionality enhancement with respect to the *enhancements on the models to simulate*.

3 Functionality enhancements to interpret user inputs

This section provides the enhancements in the functionality offered by the final version of the *Simulation tool* regarding type of information from users that the tool is able to accept and appropriately interpret. Section 3.1 provides the information for the new Big Data technology that the Simulation tool is able to interpret, Apache Spark; Section 3.2 provides information for the new quality metrics, both for performance and reliability evaluation, whose computation is accepted by the Simulation tool; and Section 3.3 provides information for the definition of SLA that is accepted by the final version of the tool.

3.1 Enhancements on the models to simulate: Apache Spark

The final version of the *Simulation tool* accepts DIAs at DPIM level and DIAs at DTSM based on technologies Apache Hadoop, Apache Storm, and Apache Spark.; i.e., it accepts UML models that are annotated with the DICE DTSM:Hadoop, DTSM:Storm or DTSM::Spark profiles. The intermediate version of the tool already accepted DTSM models for DIAs based on Hadoop and Storm. Therefore, the novel enhancement of the final version of the tool regarding its accepted models is the new capability on Apache Spark. Figure 14 highlights the part of the behavior of the tool where the user perceives this new enhancement; i.e., the user is allowed to execute `simulate` with Apache Spark UML models. While guidelines for the UML modeling with DTSM profiles and transformation from UML models to analyzable models for these Hadoop and Storm technologies were reported in Deliverable D3.1 [16], the primary source of the DICE report for modeling and transformation of DIAs based on Apache Spark to analyzable Petri net models is in Appendix B of this document.

With this new functionality, the user can request for simulation UML designs of DIAs based on Spark, such as the example depicted in Figure 12. The goal of this Spark-based DIA is to compute the prime numbers within a interval $[0, N]$. More details about the example model are provided in Appendix B.3. This UML activity diagram is helped by the UML deployment diagram in Figure 13 to represent the existence and allocation of resources.

In order to offer this enhancement on the models, it has been necessary to extend the definition of plugins to recognize UML models stereotyped with Spark profile and to implement new M2M transformations to generate analyzable models. For a detailed description of the UML modeling of DIAs based on Spark technology, the DTSM::Spark UML profile, and the M2M transformations, readers can consult Appendix B.

This enhancement increments the fulfillment level of requirements R3.1, R3.2 and R3IDE.4.

3.2 Enhancements on the quality properties that can be simulated

The final version of the tool is equipped with the possibility to study additional performance and reliability metrics of DIAs. The intermediate version of the tool allowed to define:

- Performance and reliability metrics for DIAs at DPIM level.

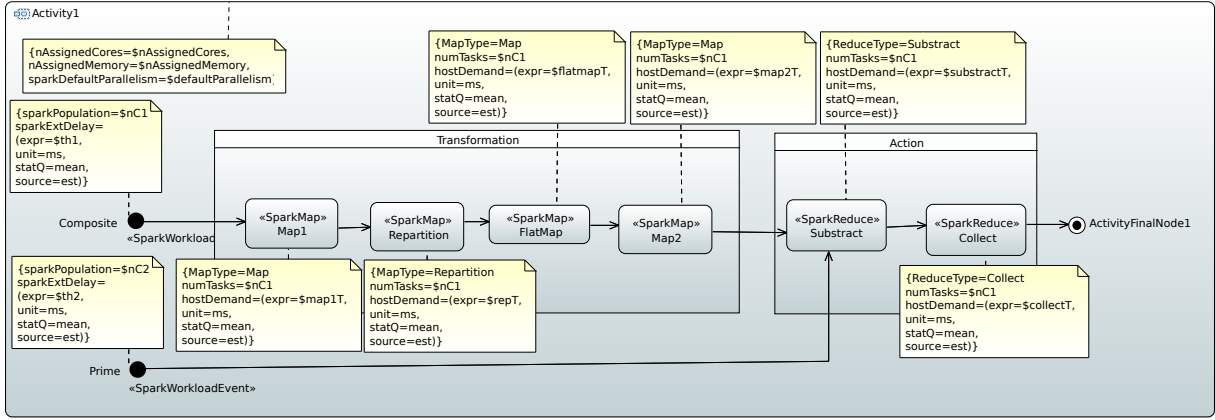


Figure 12: Activity diagram depicting a Spark execution scenario that is now accepted by the *Simulation tool*.

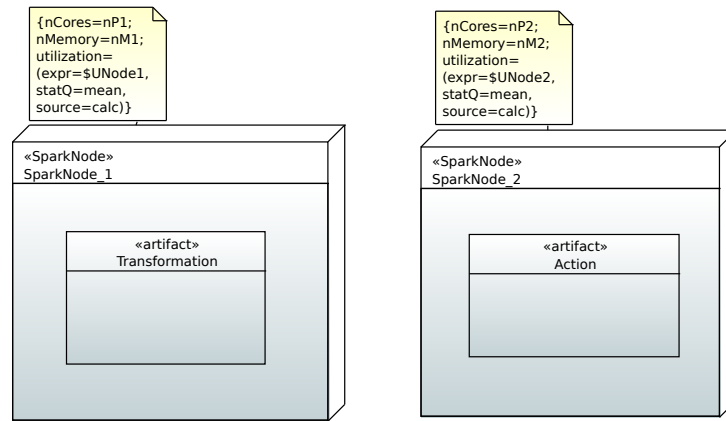


Figure 13: Deployment Diagram modeling the resources used by a DIA based Spark technology.

- Performance metrics for DIAs at DTSM level, concretely for DIAs based on Apache Hadoop and Apache Storm technologies.

The final version of the tool has been enhanced to also allow to define:

- Performance metrics for DIAs at DTSM level that are based on Apache Spark technology. Concretely, the mean response time of Spark applications, their throughput and the utilization of hardware resources.
- Reliability metrics for DIAs at DTSM level, for all three technologies Apache Hadoop, Apache Storm and Apache Spark. Concretely, it can be defined the Mean Time To Failure (*MTTF*) of the DIA, the expected *availability* of the DIA –for DIAs that execute in preemptable resources– and the probability of the DIA to continuously offer correct service during a given period of time (this definition conforms to the pure definition of *reliability* ($R(t)$) property of computing systems [8]).

During the interaction of the user with the simulation tool, these new functionalities are perceived by the user in the interactions depicted in Figure 14. The following subsections detail how the user has now the possibility to define these new quality properties to be sent within the `simulate` message, and how to configure the simulation process.

3.2.1 New performance metrics

The new performance metrics that can be requested to compute by users are related to Apache Spark technology. Users can define their profiled UML models with variables to store the average response

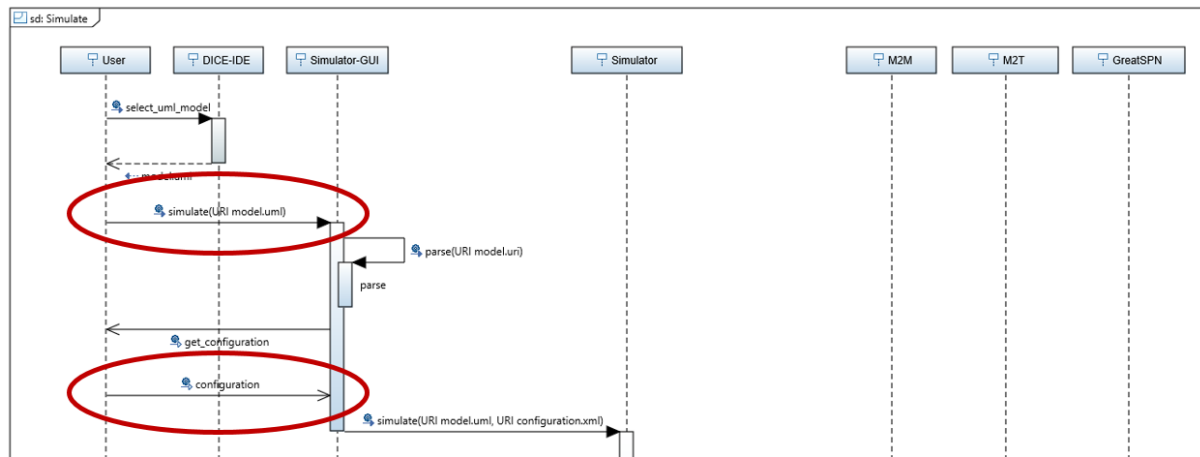


Figure 14: Sequence diagram highlighting the functionality enhancement with respect to the *quality properties that can be simulated*

time of Spark application, their throughput and the utilization of hardware resources. Figure 15 depicts how the response time and throughput are defined within <<SparkScenario>> stereotype, and Figure 16 depicts how the utilization of resources is defined for computation resources stereotyped as <<SparkNode>>. This information is included in the highlighted message *simulate* in Figure 14 between *User* and *Simulator-GUI*.

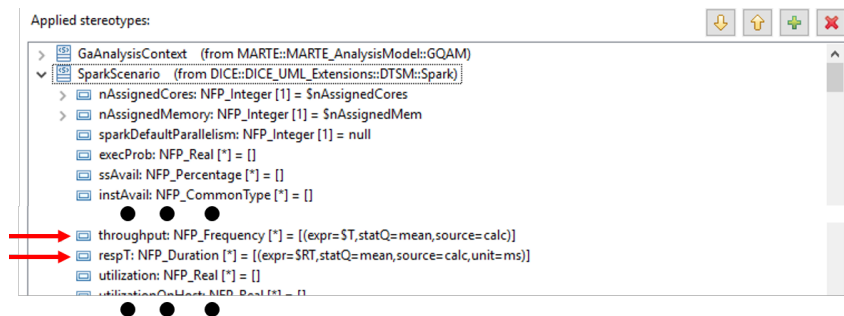


Figure 15: Definition of response time and throughput metrics in Spark scenarios

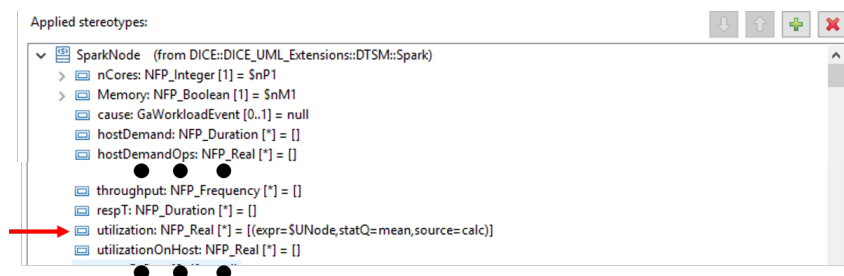


Figure 16: Definition of Utilization metric in Spark scenarios

Later, at the simulation configuration step (i.e., the highlighted message *configuration* in Figure 14 between *User* and *Simulator-GUI*), the users can choose which metrics, among the previously defined, they wish to simulate. Figure 17 shows a screenshot of this step where, for instance, the user has decided to compute response time and utilization of resources, but not the throughput.

This two-step process is useful because it relieves users from modifying the attributes of stereotypes

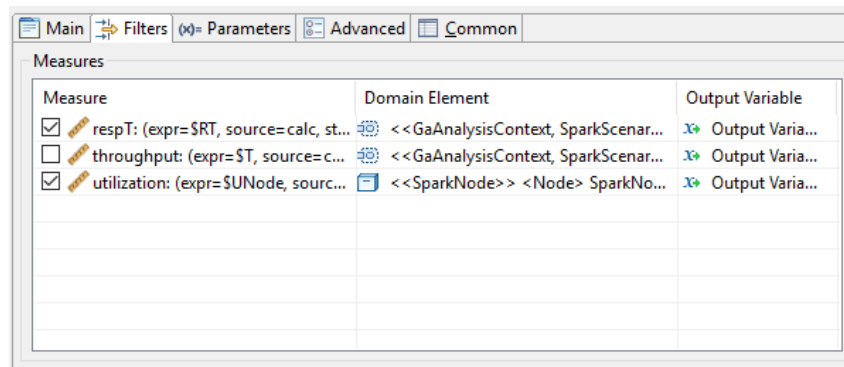


Figure 17: Configuration of metrics to simulate

in the model between different simulations. Following this process, the metrics of interest are defined only once in the model and do not need to be further modified, regardless the metric of interest in a concrete simulation. The user can alternate between different sets of metrics to simulate by just clicking on a check-box at the configuration step.

3.2.2 New reliability metrics

The new reliability metrics that users can request to compute affect all technologies covered by the DICE Simulation tool: Apache Hadoop, Apache Storm and Apache Spark. The new metrics that can be defined are MTTF, availability and probability of continuous correct operation. Figures 18-20 show three examples of such definition.

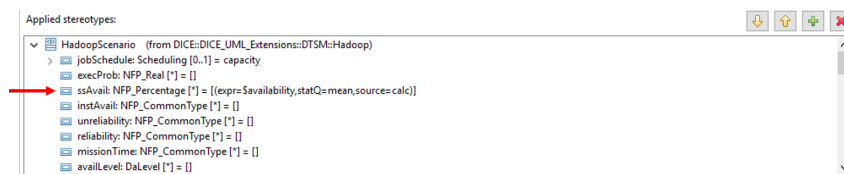


Figure 18: Definition of availability in a Hadoop scenario

Figure 18 highlights the definition of the availability definition in a DIA that uses Hadoop technology (see that the attribute “ssAvail” belongs to <<HadoopScenario>> stereotype). This metric provides the percentage of time that the DIA is reachable. According to [7], engineers may choose to rent large amounts of preemptable resources for their cloud deployments of Hadoop applications, for instance spot instances of Amazon AWS Elastic Cloud Computing [5, 20]. They are much cheaper than on-demand resources although their availability is not guaranteed. Anticipating some details of the computation of metrics in Section 4, this metric is calculated by using two values: the mean time that it is possible to use a resource between two consecutive preemptions, and the mean time to boot and set up a new resource. Therefore, this definition of availability metric is intended to be used when the DIA is deployed using an infrastructure of resources that can be preempted and relaunched.

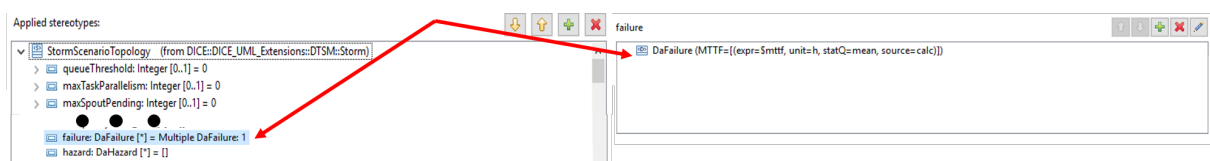


Figure 19: Definition of MTTF in a Storm scenario

Figure 19 highlights the definition of the MTTF definition in a DIA that uses Storm technology (see that the attribute “failure” belongs to <<StormScenarioTopology>> stereotype). This metric provides information of the expected working time of a Storm application until failure. Different from the previous availability metric, which was defined in DAM profile as NFP_Percentage type –and which, in turn, is defined in MARTE profile– MTTF definition is included in the complex type DaFailure defined in DAM profile. Therefore, to define this metric, the user fills the “failure” attribute of <<StormScenarioTopology>> by creating a new DaFailure element, and fills its MTTF field as represented on the right side in Figure 19.

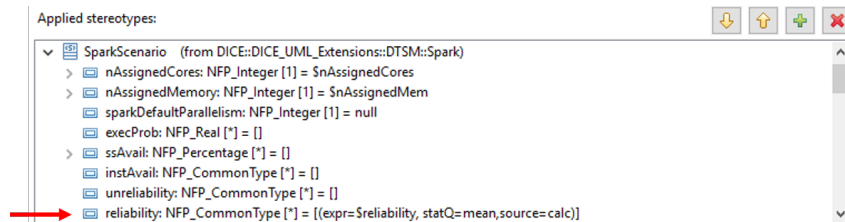


Figure 20: Definition of reliability in a Spark scenario

Figure 20 highlights the definition of the reliability definition in a DIA that uses Spark technology (see that the attribute “reliability” belongs to <<SparkScenario>> stereotype). This metric provides information of the probability that the application works correctly and continuously for a given period of time t . Anticipating some details of the computation of this metric that will be given in Section 4, this metric $R(t)$ requires as input from the user the MTTF value of Spark nodes and the target time t (also called *mission time*).

Later, the simulation *configuration* step highlighted in Figure 14 proceeds analogously as explained above for the new performance metrics for Spark scenarios: the user sees all the metrics defined in the model and can check the subset of metrics in which s/he is interested in each simulation.

These enhancements on the quality metrics that are accepted for evaluation by the Simulation tool increment the fulfillment level of requirements R3.2, and R3IDE.4.

3.3 Enhancements on the specification of SLA

The final version of the tool allows specifying an SLA value for the quality metrics to compute. It accepts definitions of SLA that follow the MARTE proposal for expressing required Non-Functional Properties¹. Concretely, the user can define the required value for a property next to the definition of the property to compute. Figure 21 provides an example for the definition of the computation of the response time property together with its required values. In this example, “respT” attribute has two values:

1. the definition of the metric to compute, (expr=\$responseTime, statQ=mean, source=calc), meaning that variable responseTime will store the calculated value for the mean response time.
2. the definition of the limit accepted value for the property, (expr=1.0, unit=s, statQ=mean, source=req), meaning that it is required a mean response time with a limit value of 1 second.

`respT: NFP_Duration [*] = [(expr=$responseTime, statQ=mean, source=calc), (expr=1.0, unit=s, statQ=mean, source=req)]`

Figure 21: Definition of the response time metric to calculate and its SLA of 1 second

¹The *source* attribute of MARTE *NFP_CommonType* allows to express the origin of the NFP specification, being some of its predefined sources *calculated* (‘calc’) –which is used to define the properties to compute– and *required* (‘req’) –which is used to specify that the the NFP is a required value for the stereotyped element.

This enhancement on the specification of SLA that is accepted by the Simulation tool increments the satisfaction level of requirements R3.10.

4 Functionality enhancements to produce simulation results

This section provides the enhancements in the functionality offered by the final version of the *Simulation tool* regarding to the results of the quality of the DIA that are produced during the model-based evaluation. These enhancements affect the functionality of the system in messages `process_raw_results` and delivery of results, highlighted in Figure 22.

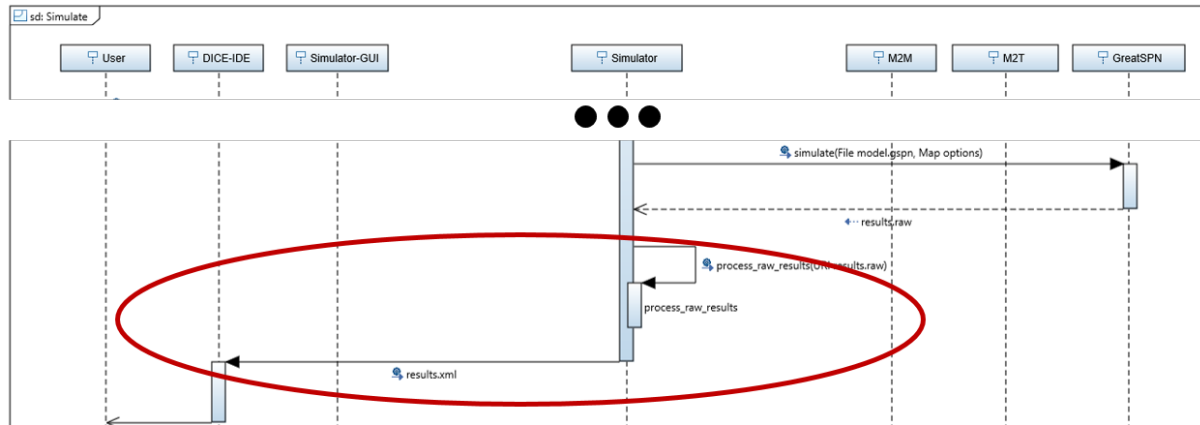


Figure 22: Sequence diagram highlighting the functionality enhancement with respect to the production of new simulation results.

The final version of the tool implements the computation of performance metrics for DIAs based on Spark technology, described in Section 4.1, and the computation of reliability metrics for deployments of DIAs based on Hadoop, Storm and Spark, described in Section 4.2. It is worth noting that performance metrics for DIAs based on Hadoop and Storm technologies were already implemented in the intermediate version of the tool at M24.

4.1 Enhancements on the computation of performance results: Apache Spark

Performance metrics that can be computed by the final version of the *Simulation tool* for Spark scenarios are: throughput of the DIA, response time of the DIA, utilization of its assigned resources. The computation of each quality metric is based on the following *raw_results* of the Petri net evaluation returned by GreatSPN engine:

- Throughput of the DIA is computed from the throughput of transition that represents the final activity of the scenario. Then, the result set of the raw results from the evaluation of the Petri net is explored to find the correct transition and its value translated to the units in which the DIA throughput was requested by the user (in case that no definition of units is provided, they are produced in Hertz by default –the inverse of seconds).
- Response time of the DIA is computed from the throughput of the DIA and the mean number of executions active. The latter value is divided by the former. This means that the calculation of the response time follows Little's Law where the mean response time (R) of a system is equal to the mean number of users (N) divided by the mean throughput (t), $R = N/T$. Therefore, the computation of the response time requires two values among the set of raw results –a throughput of a transition and the mean number of tokens of a place–, operates with them and translates the result to the units in which the DIA response time was requested.
- Utilization of resources used by operations of the DIA are computed from the mean number of tokens in the place that represents that a resource is being used and from the number of existing resources defined in the model. It is computed the proportion between the value of resources used and total existing resources, and later this value is transformed to a percentage. Therefore, the

computation of the utilization requires a value among the set of raw results –a mean number of tokens in a place– and a value in the definition of the Petri net –the number of existing resources, which is the initial marking of the place that represents the resource–.

For further details, Appendix D provides a validation of the computation of performance metrics and transformations of Spark scenarios to analyzable models, by comparing the model-based results with measurements from a running system.

These enhancements on the computation of performance metrics increment the satisfaction level of requirements R3.4.

4.2 Enhancements on the computation of reliability results for Apache Hadoop, Storm, and Spark

Many of the technologies covered for DIAs are designed as fault-tolerant, which means that their failures are internally handled and are not visible to users. Therefore, computed metrics (MTTF, availability and reliability $R(t)$) are calculated from the properties of the resources used by the technologies, rather than from the activities executed by the DIA. The following paragraphs detail how each of the new reliability metrics is computed.

4.2.1 Computation of MTTF

In order to calculate the global MTTF of the DIA, it is required that the user provides information about the expected time to failure of the resources used for computation. This is done by stereotyping with `<<DaComponent>>` the resources, and filling its attribute “failure”. The “failure” attribute is a *DaFailure* complex type and, concretely, the field that is necessary to fill is its MTTF. We will refer to in the following as `DaComponent.failure.mttf`.

In this `<<DaComponent>>` stereotype, the user should also specify the number of existing resources through its attribute “resMult”. We will refer to it in the following as `DaComponent.resmult`.

Since the covered DIA technologies autonomously handle failures of some of its nodes by using the rest of still working nodes, a failure of the application (i.e., when the run of the application is considered as failed and requires human intervention) happens when all of its resources have failed. Therefore, the overall MTTF of the application is the mean time until all elements in a pool of `DaComponent.resmult` independent resources, each of them with specific MTTF of `DaComponent.failure.mttf`, have failed.

Computation of MTTF in Storm applications: For the computation of the MTTF, we consider DIAs based on Apache Storm as special case. The rationale is the following, Storm applications are fault-tolerant and the progress of the computation of elements in the input streams is managed and stored by a differentiated software, Zookeeper [4, 6]. Therefore, for the failure of a DIA based on Storm, the important concept to consider is the failure of the cluster of Zookeeper nodes, rather than the failure of workers and bolts. In this case resources that execute Zookeeper service have to be stereotyped with `<<StormZookeeper>>` to point out their functionality and with `<<DaComponent>>` to specify their multiplicity in “resMult” attribute and the estimated MTTF of the resource in the “failure” attribute.

To compute the MTTF of a Storm DIA, the Simulation tool traverses all Nodes and Devices represented in the Deployment Diagram of the DIA looking for the element stereotyped as `<<StormZookeeper>>`. When it finds this element, it gets from it the information in its `<<DaComponent>>` stereotype regarding the number of resources used for the Zookeeper cluster and the estimated MTTF of each of them. Finally, the computation of the global MTTF considers that the Storm DIA takes place when all the resources dedicated to execute Zookeeper cluster have failed. Figure 23 shows the MTTF results of a Storm platform whose Zookeeper cluster comprises 3 resources, each of them with a MTTF of 10h. The result is a global MTTF of the DIA of 18.3h.

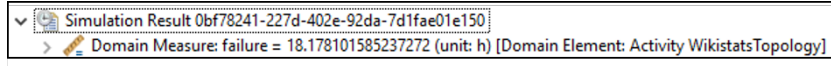


Figure 23: Computed MTTF in a Storm application

4.2.2 Computation of Availability

Availability property of a system is defined as the readiness for correct service [8]. We compute the percentage of time that a system is available, i.e., the percentage of time that the system is ready for executing a correct service for its users. It is defined as the mean time that the system is working correctly between two consecutive failures (i.e., MTTF) with respect to the Mean Time Between two consecutive Failures (called MTBF). In turn, MTBF is defined as the mean time of correct service until a failure plus the Mean Time to Repair (MTTR). Therefore, as traditionally calculated:

$$Availability = \frac{MTTF}{MTTF + MTTR} \cdot 100$$

As described in the previous section, we opted to offer a computation of the system availability when users chose to use preemptable resources, such as Amazon AWS EC2 spot instances. Users need to fill the a) expected amount of time that a preemptable resource will be granted for utilization, and b) the expected amount of time required to choose an alternative affordable set of resources, boot them and set up and configure the technology (i.e., repair time of the application). This information is provided by the stereotypes devoted to mark the nodes of the computing platform, for instance, stereotype <<HadoopComputationNode>> for Hadoop applications, or <<SparkNode>> for Spark applications. These stereotypes have again the attributes “failure” and “repair”. Information a) is filled into field MTTF of “failure” attribute and information b) is filled into field MTTR of “repair” attribute. The computation of the availability is computed from these two values.

An interesting utilization of this availability metric arises in the moment when the user is deciding the value of the bid for spot resources. Higher bids entail less frequent preemption and higher costs to run the application. Lower bids entail more frequent preemption, search for an alternative set of affordable resources, and setting them up; but lower costs to run the application. Having the user in mind the quantity of availability s/he wants for her/his application and knowing the amount of time required to repair/redeploy a failed application, the *Simulation tool* can be used to perform a *what-if* analysis over the MTTF variable. Then, the simulation results can be used to identify the minimum value for the MTTF that will offer the desired availability. Finally, using such identified MTTF, the user can decide for a value of the bid for spot instances that is expected to grant resources for such MTTF.

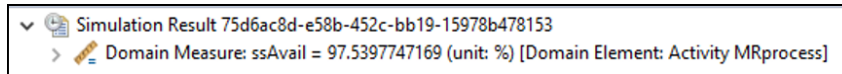


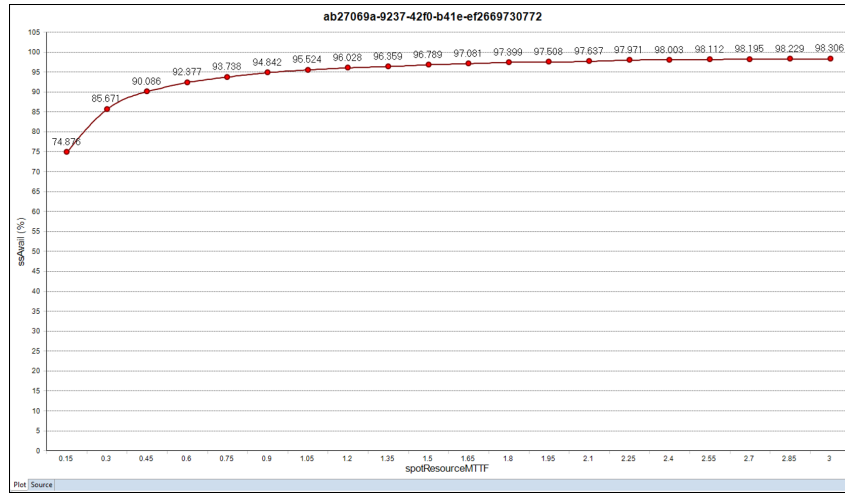
Figure 24: Computed Availability in a Hadoop application

4.2.3 Computation of reliability R(t)

Reliability property is defined as continuity of correct service [8]. Its value is usually computed for a certain time interval of duration t . Then, reliability $R(t)$ is the probability that the system works continuously without any failure for a period of length t . There is a relation between these definitions of $R(t)$ and MTTF, being:

$$MTTF = \int_0^{\infty} R(t) dt$$

When failures in the system are *memoryless* –i.e., there is a constant failure rate which does not depend on the quantity of time that the system has been already working– the reliability $R(t)$ of a single

Figure 25: *what-if* analysis over variable MTTF to obtain a required Availability

element of the system is defined by the exponential distribution $R(t) = P(X > t) = e^{-\frac{t}{MTTF}}$. This is a common assumption in computing systems, that the *Simulation tool* also follows. See also that, mixing the previous two formulas, $MTTF = \int_0^\infty e^{-\frac{t}{MTTF}} dt$ holds.

Having multiple resources to execute the DIA, and having the technologies their internal fault-tolerant mechanisms to handle failures on some resources, again, a failure of the DIA happens only when all resources have failed. The computation of the $R(t)$ result is equivalent to compute the $R(t)$ of a parallel system composed of “resMult” elements. Figures 26, 27 and 28 show three examples of computed reliability $R(t)$ of a Spark platform. Figure 26 shows how the simulation tool presents a single result of the tool when the MTTF of single resources is 3h, the target time t is 4h and there are 3 resources that can execute the Spark operations.

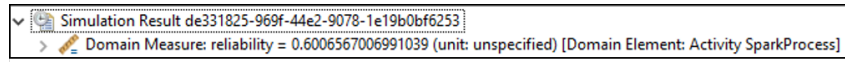
Figure 26: Computed $R(t)$ in a Spark application

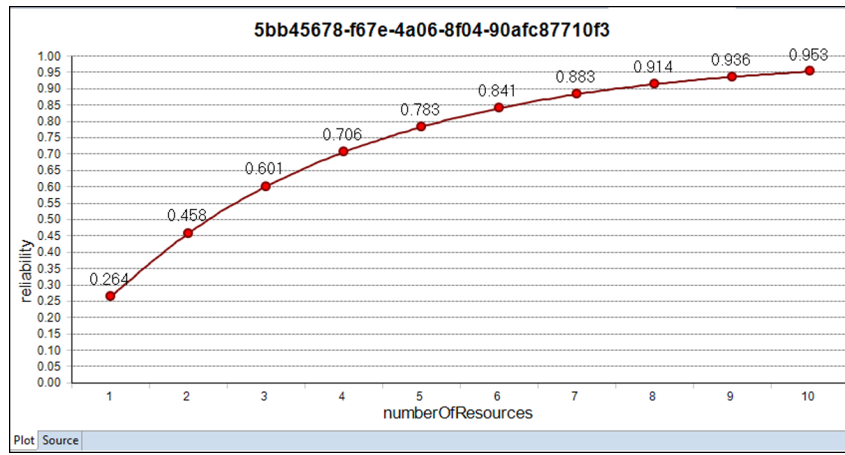
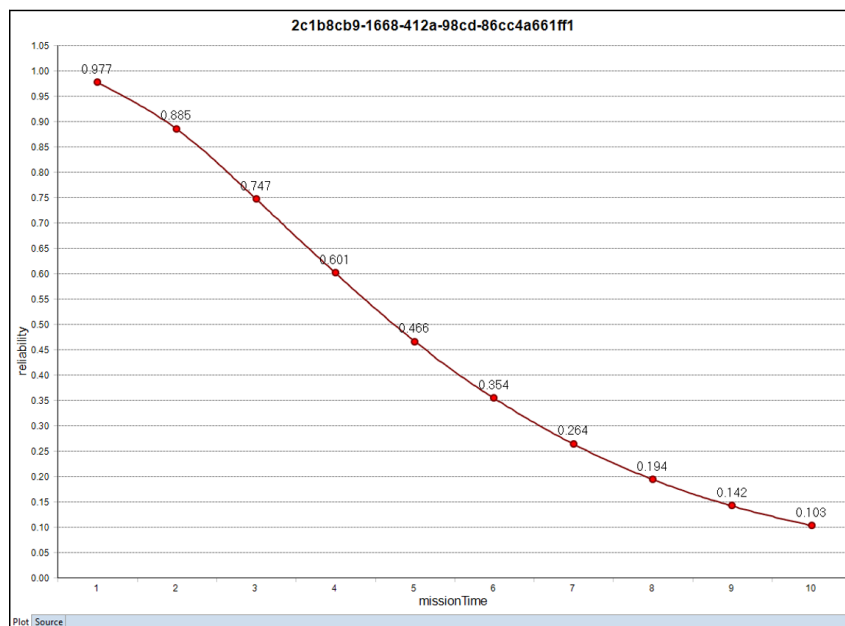
Figure 27 shows the computation results of a *what-if* analysis over the number of resources necessary to obtain a certain $R(t)$, having each resource the same $MTTF=3h$ and being the target time $t = 4h$. This is useful when the user is deciding the number of cloud resources to rent for the DIA deployment.

In turn, Figure 28 shows a different case of interest for the user. In this case the $MTTF=3h$ of each resource and the number of resources is fixed to 3. It is performed a *what-if* study over the mission time of the application. This is useful for the user when s/he can decide the size of the jobs that are accepted by the DIA. Depending on the size of the job, it will require an amount of time to complete; i.e., the mission time of the execution. Thus, the user is informed about the probability of failure of the DIA depending on the expected quantity of time that is required for a concrete execution.

These enhancements on the computation of performance metrics increment the fulfillment level of requirement R3.4.

4.3 Enhancements on the solvers implementation

For some quality properties computed by the Simulation tool exists closed-form analytical solution. For this reason, it was considered convenient to create an additional plugin with calculators of these properties that does not depend on the results of the Petri net simulation carried out by the GreatSPN simulation engine. Two metrics that are computed by this new solver plugin are the availability of

Figure 27: Computed $R(t)$ in a Spark application in function of the number of resourcesFigure 28: Computed $R(t)$ in a Spark application in function of the mission time t

the DIA when using preemptible instances and the reliability $R(t)$. Following paragraphs describe the formulas for computing each of these metrics.

Computation of availability: Availability can be directly computed with the formula presented in Section 4.2.2, as every variable mentioned in that formula is already present in the input model (i.e., its value is not only known after a simulation). Therefore, the availability of a DIA, which uses preemptible resources and has a repair mechanism with known mean duration that gets an alternative set of resources and redeploys the DIA, is calculated as (example given for DIA based on Spark, for other technologies it is sufficient to read the same information from the corresponding stereotype):

$$Availability = \frac{SparkNode.failure.mttf}{SparkNode.failure.mttf + SparkNode.repair.mttr} \cdot 100$$

Computation of reliability $R(t)$ Reliability $R(t)$ can be directly computed using theory of Reliability Block Diagrams (RBD) and reliability of series/parallel systems [30]. The reliability to compute in the

domain of the *Simulation tool* refers to the reliability of a parallel system. Being $R(t)$ the reliability of the system at time t , existing n components in the system and being $R_i(t)$ the reliability of component i ($i \in [1..n]$) of the system at time t , the formula to compute is:

$$R(t) = 1 - \prod_{i=1}^n (1 - R_i(t))$$

Assuming independent and memoryless failures of components, the formula becomes:

$$R(t) = 1 - \prod_{i=1}^n (1 - e^{\frac{-t}{MTTF_i}})$$

Since resources are defined as a cluster of “resmult” with identical characteristics, all $MTTF_i$ have the same value. Therefore, it is possible to transform the previous formula to:

$$R(t) = 1 - (1 - e^{\frac{-t}{MTTF}})^n$$

As every variable in the previous formula is present in the input model, the reliability $R(t)$ of the DIA is calculated using the information in the input model as (the example is given using the DaComponent stereotype that has to be applied in the Deployment Diagram to the resources that execute the DIA):

$$R(DaComponent.missionTime) = 1 - (1 - e^{-\frac{DaComponent.missionTime}{DaComponent.failure.MTTF}})^{DaComponent.resMult}$$

These enhancements on the computation of performance metrics increment the fulfillment level of requirement R3.4.

4.3.1 Solving with JMT

In Task 3.2 some effort has been devoted to the enhancement of JMT for evaluating Petri nets through simulation. JMT was extended to handle Petri net models as it allows to efficiently simulate certain classes of stochastic models such as hybrid models mixing queueing networks and Petri nets. Moreover, with JMT it is possible to run experiments that read traces of monitoring data, allowing a more realistic parameterization of complex models, where needed.

The enhanced functionality of JMT to manage Petri nets uses the output of the M2M transformations developed within the *Simulation tool*. The simulation tool writes the output model of its transformations in XML PNML format. A collaboration to match interfaces has been carried out during the third year of the project. As already planned in Deliverable D3.3 [15], which reported the intermediate version of the simulation tool, JMT has been extended to read these PNML files to increase compatibility. Therefore, a partial integration between outputs of *Simulation tool* transformations and input Petri net models of JMT has been achieved, although currently the invocation requires human intervention since the set of plugins to automatically manage the execution cycle and outputs of JMT as an alternative simulator have not been implemented. Appendix C provides further details on these JMT enhancements.

5 Updates in the Graphical User Interface

This section presents the enhancements in the GUI of the final version of *Simulation tool*. The intermediate version of the tool at M24 already provided a mature graphical interface with the user. Therefore, new elements in the GUI, here reported, refer to new additions to configure and depict the new functional enhancements, rather than to modifications in what it was already graphically depicted in the intermediate version of the tool.

The two main additions in the GUI are the following: graphical representation in a plot of the SLA together with simulation results, which is detailed in Section 5.1, and two windows to configure the solver to use graphically, which is presented in Section 5.2.

5.1 Graphical representation of specified SLA

The graphical representation of SLA affects the final interaction with the user, which is highlighted message `visual_results` in Figure 29.

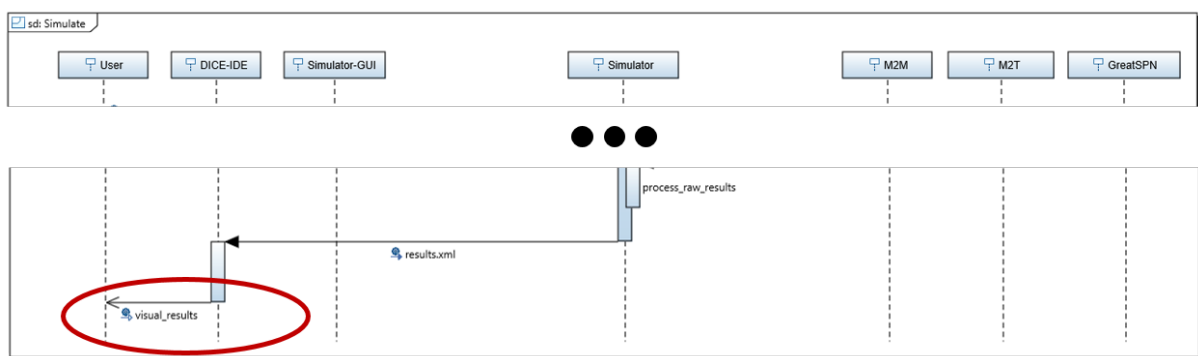


Figure 29: Sequence diagram highlighting the GUI enhancement with respect to the representation of results.

After specifying an SLA value for quality properties as described in Section 3.3, the user can check the compliance of the expected properties evaluated of the system with respect to the SLA limit. In the case of single evaluation of a model, this reduces to a simple comparison of values. What it is more interesting is to get information about the compliance of the system with the SLA limit when the user executes the *Simulation tool* for *what-if* analysis over some variable to decide.

To help the user in such task, the final version of the tool implements the visual representation of the SLA together in the plot that is produced for *what-if* analysis. To illustrate it, let us reuse the previous examples given in Section 4.2.3 and assume that the user is interested in building a DIA that has a probability of 0.85 of finishing its execution without incurring in a failure. This is, the user specifies a reliability $R(t)$ SLA of 0.85.

In this case, according to Section 3.3 the user defines an SLA in the reliability attribute (`expr=0.85,source=req`). The output of the *what-if* analysis is depicted in Figures 30 and 31. Regarding the number of resources necessary when each of them have an MTTF=3h and the mission time is 4h (i.e., plot in Figure 30), the user can easily check how the progression of satisfaction/dissatisfaction of the SLA evolves, and see that using above 6 resources the platform will be executed as expected.

In turn, regarding the study of what kind of executions can be accepted by the DIA if this is composed of 3 resources and each of them with MTTF=3h (i.e., plot in Figure 31), the user can easily check how evolves the dissatisfaction of the SLA for larger mission times, and also check that executions whose mission time is below 2h20m can be launched with acceptable confidence in its reliability.

This enhancement increments the satisfaction level of requirements R3.10 and R3IDE.7.

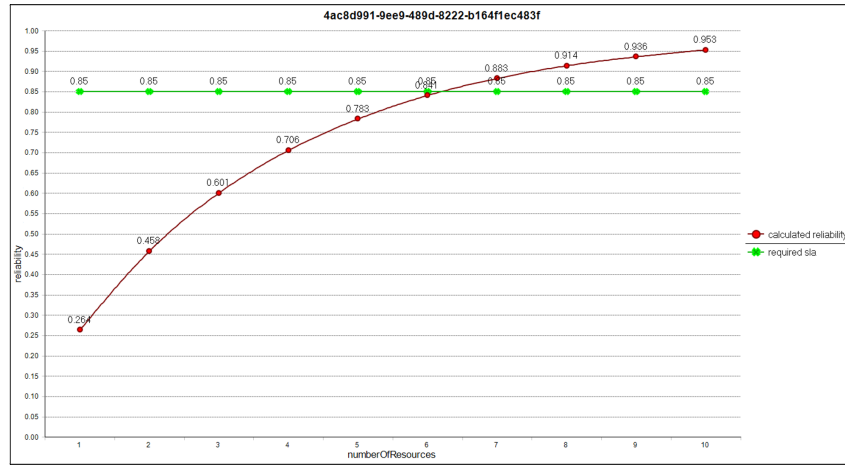


Figure 30: Computed $R(t)$ and limit SLA in a Spark application in function of the number of resources.

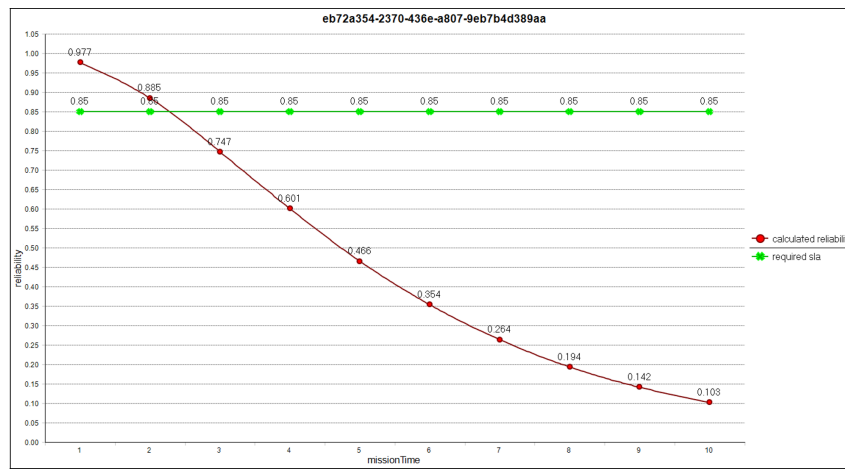


Figure 31: Computed $R(t)$ and limit SLA in a Spark application in function of the mission time t .

5.2 Graphical configuration of quality properties solver

As presented in Section 4.3, the final version of the tool is equipped with a new solver for some quality properties for which exist a direct analytical solution. The solver that is used in a model evaluation is a user choice. To make easier the assignment of a solver, the GUI has been extended in two different places.

The first place is in the *LaunchConfiguration* window shown when a simulation is being configured. Figure 32 highlights the configuration message where this extension is perceived by the user. The new user interaction in this *LaunchConfiguration* window takes place in the *advanced* tab. Figure 33 depicts the interaction highlighting its location and list of choices. The configuration is presented as a list of solvers, which, at present, comprises two possibilities: a simulator based on GreatSPN engine and a local new solver. The user can choose the preferred solver for the current model simulation.

The second place to assign a solver is in the general configuration of the *Simulation tool*, which is located in the general preferences of the DICE IDE under *DICE* → *Simulation Tools* → *Simulation* tab. Figure 34 depicts this extension and highlights its location in the DICE IDE preferences page. The default option selected in the *LaunchConfiguration* window when a new model is simulated corresponds to the choice marked in the this general configuration page of the IDE.

This enhancement increments the fulfillment level of requirement R3.4.

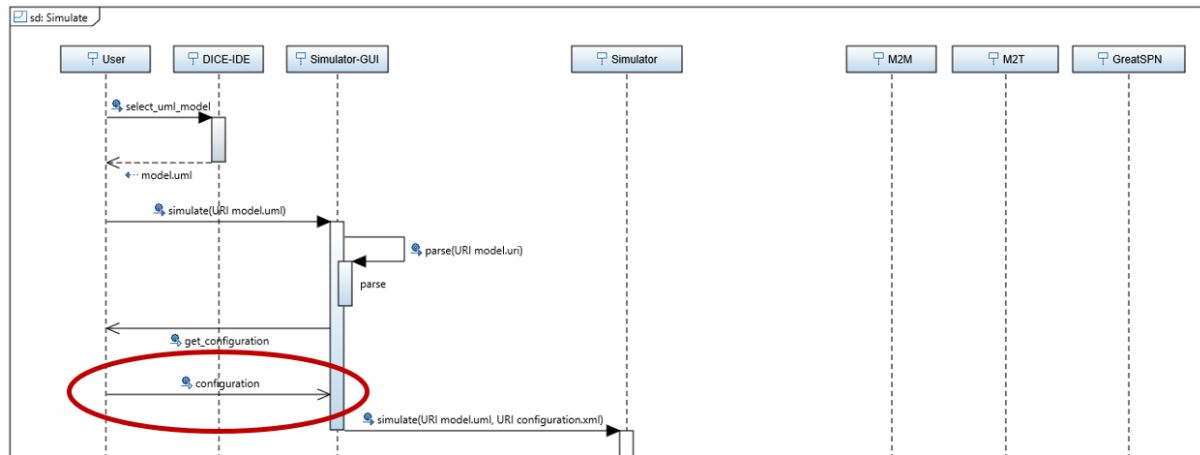


Figure 32: Sequence diagram highlighting the GUI enhancement with respect to the choice of solvers.

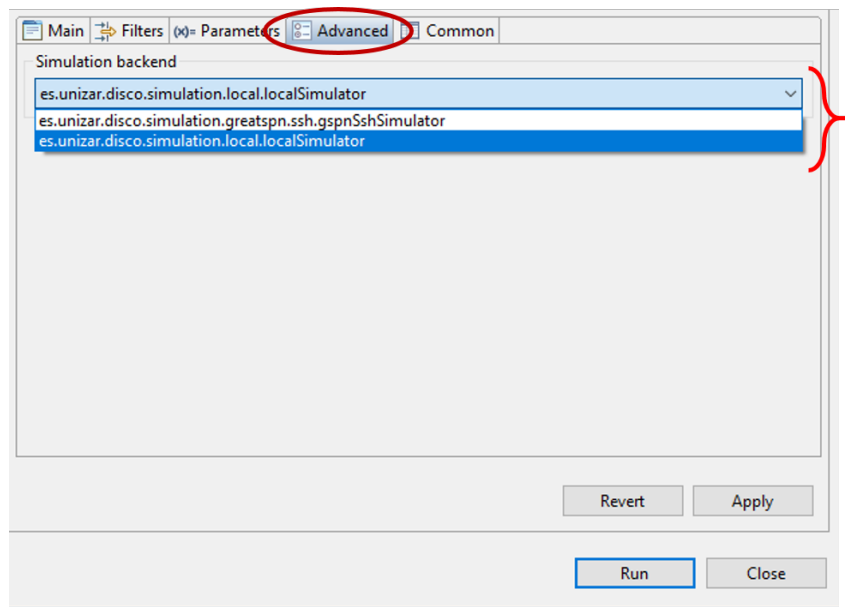


Figure 33: Choice of simulation solver during the simulation configuration in *LaunchConfiguration* window.

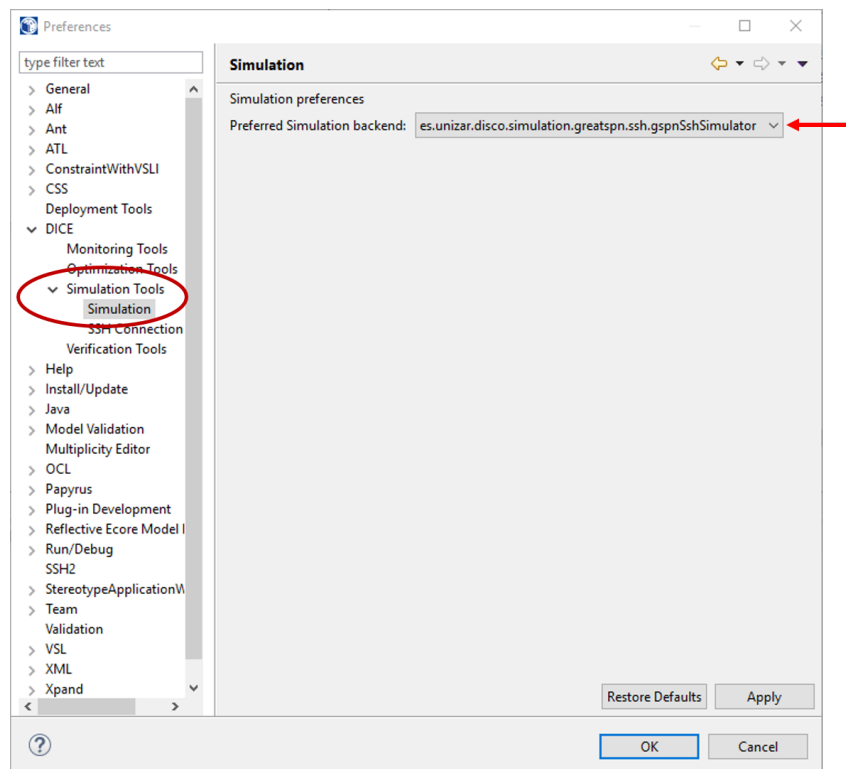


Figure 34: Choice of simulation solver in the general preferences of the DICE IDE.

6 Simulation tool for containerized DIA deployments

This section discusses the utilization of the *Simulation tool* and the accuracy of its quality results when the DIA is to be deployed on a platform based on containers.

The Simulation tool works at Platform-independent (DPIM) and Technology-specific (DTSM) levels. At these levels, the target infrastructure to deploy the DIA may have not been decided yet. Therefore, at first sight, it may seem that the results of the *Simulation tool* are valid regardless the target infrastructure where the DIA will be deployed. However, there are a pair of concerns that should be mentioned and that the users of the tool should keep in mind when the DIA is deployed on containers.

The first concern is the modeling of resources. The simulation tool uses a specification of the number of processing resources available for evaluating quality properties. For instance, each resource may correspond to a future processing core in the infrastructure to deploy the DIA. Containers such as Docker do not have by default a restricted number of resources that can be used. A container could use all resources of a machine, having contention with others. Therefore, this discrepancy between the modeling of available resources and the real availability of resources could create a significant gap between the results of the model and measures from the running system. Nevertheless, there are mechanisms in Docker containers to define the resources that a container is allowed to use. Some examples, taken from Docker [19], are:

- Utilization of option `--cpus` to set how much of the available CPU resources a container can use.
- Utilization of option `cpu-period` and `cpu-quota` to impose a scheduler quota on the container. This guarantees access to the CPU.
- Utilization of `cpuset-cpus` to limit the specific CPUs that a container can use. This restricts the amount of CPU that a container can use.

Unless using container platforms that offer service only as best-effort, application owners will wish to set at deployment time the CPU resources that will be granted to their applications, and will use some of the options to set the quantity of CPU. In this plausible case, the concern regarding the modeling of the available resources for the DIA is avoided because resources specified in the model will match with actual resources in the running platform.

The second concern is the modeling of the demand of operations in the workflow of the DIA. The DIA activities usually have a `hostDemand` attribute that specifies the estimated time that the operation needs to complete. If containers incurred in some computing overload with respect to other techniques such as Virtual Machines, engineers would need to take it into account when setting the `hostDemand` values. Nevertheless, according to some initial comparisons of the accuracy of the Petri net model-based evaluation with respect to the execution of DIAs in containers [17], the model results are accurate. Moreover, *what-if* analysis provided by the *Simulation tool* may be helpful if these differences become significant in the future. Using the *what-if* analysis it is possible to cover a broad set of alternatives when the target deployment platform has not been decided yet.

7 Conclusion

This document has reported the advances achieved in the final version of the *Simulation tool*. This tool is the main outcome of Task T3.2. In its current state, the tool covers all the execution flows both for the platform independent models (i.e., DPIM level) and for the platform specific models (i.e., DTSM level) for Apache Hadoop, Storm and Spark.

The schematic view of their **ID**, **Title** and **Priority** is provided in Table 2. It also shows, in column **Level of fulfillment**, their accomplishment in the intermediate version of the tool. The meaning of the labels used in column **Level of fulfillment** is the following: (i) ✗ (unsupported: the requirement is not fulfilled by the current prototype); (ii) ✓ (partially-low supported: a few of the aspects of the requirement are fulfilled by the prototype); (iii) ✓ (partially-high supported: most of the aspects of the requirement are fulfilled by the prototype); and (iv) ✓ (supported: the requirement is fulfilled by the prototype and a solution for end-users is provided).

Table 2: Level of compliance with requirements of the final version of the *Simulation tool*

Requirement	Title	Priority	Level of fulfillment intermediate	Level of fulfillment final
R3.1	M2M Transformation	MUST	✓	✓
R3.2	Taking into account relevant annotations	MUST	✓	✓
R3.4	Simulation solvers	MUST	✓	✓
R3.6	Transparency of underlying tools	MUST	✓	✓
R3.10	SLA specification and compliance	MUST	✓	✓
R3.13	White/black box transparency	MUST	✓	✓
R3IDE.1	Metric selection	MUST	✓	✓
R3IDE.4	Loading the annotated UML model	MUST	✓	✓
R3.3	Transformation rules	COULD	✓	✓
R3.14	Ranged or extended what if analysis	COULD	✓	✓
R3IDE.2	Timeout specification	SHOULD	✓	✓
R3IDE.3	Usability	COULD	✗	▼
R3IDE.7	Output results of simulation in user-friendly format	COULD	▲✓	✓

The reasons of the declared improvements in the level of fulfillment are the followings:

- R3.1 was partially-high supported and only remained to develop the transformation from Apache Spark to Petri nets, which is now implemented.
- R3.2 was partially-high supported and remained to develop that the simulation tool accepted annotations of Apache Spark and of reliability properties for all technologies at DTSM level (Hadoop, Storm, Spark). All these concerns have been implemented and the simulation tool accepts Spark models where performance and reliability annotations are present, as well as accepting Hadoop and Storm models that now have reliability annotations following DAM profile.
- R3.4 was partially-high supported because the previous version of the tool already had the infrastructure to handle more than one simulation solver if they were implemented as Eclipse plugins, but it lacked the actual solver. Now, the final version has included a solver based on reliability block diagrams and series/parallel systems to compute reliability metrics implemented as an Eclipse plugin. Therefore, the requirement passes to be fully satisfied.
- R3.10 was partially-low supported because it had been studied the utilization of profile annotations to specify the SLA, but nothing had been implemented. The final version of the tool is able to read SLA annotations in the specification of the quality properties to compute and during plotting

activity of *what-if* analysis results, it represents the SLA limit value introduced together with the simulation results to make easily realizable the break-even.

- R3.13 was partially-high supported. This requirement is related to the fulfillment level of R3.1. White/black boxes transparency in the modeling has an effect in DTSM level. At DTSM level, the intermediate version of the tool already took into account the white/black transparency. The rationale for Hadoop and Storm in the intermediate version of the tool was the following: Hadoop and Storm processes were white boxes, meaning that the designer specified the steps that are carried out inside such processes. The maximum level of granularity inside the processes were HadoopMap and HadoopReduce steps for Hadoop technology and StormBolt and StormSpout steps for Storm technology. Therefore, these steps were black boxes and what happened inside them remained as black boxes. The final version of the tool follows the same principles for Apache Spark: the Spark process is a white box and, inside it, can be modeled the sequence of Spark operations, each of them of type either Spark Transformation or Spark Action. This is the maximum level of granularity for Spark, and what happens inside each operation remains hidden. Therefore, all technologies follow the same principles of white/black box transparency, and the requirement is fully satisfied.
- R3.IDE4 was partially-high supported in the intermediate version of the tool. The lacking part to be fully supported was that the impossibility to load execution scenarios of the Apache Spark technology. The final version of the tool is able to load annotated UML models of Spark scenarios, thus the requirement is fully satisfied.
- R3.3. was marked partially-high supported in the intermediate version of the tool because the files that stored the code for transformations in QVT remained separated and in a dedicated folder. After studying different alternatives during Y3 to include QVT files with new transformations from the GUI, this has been discarded because it would mix the developer view and the user view unnecessarily and would increase complexity of the tool for users. The creation of correct transformations requires knowledge of the internal metamodels used by the simulation tool, as PNML models, traces models and variables models. It is not plausible to generate suitable transformations from a pure user perspective. Therefore, to the best of our knowledge, the current trade-off of having dedicated folders for the QVT transformation files is the best for the separation of concerns between developer and user. The intermediate version of the tool should have marked the requirement already as fully satisfied.

In summary, all requirements of the *Simulation Tool* have been achieved in its final version. As already expected in the deliverable that reported the intermediate version at M24 [15], the development has not suffered any limitation or delay in the fulfillment of requirements of type MUST. Moreover, now we can state that there have not been delays in the development of requirements of type SHOULD or COULD either.

References

- [1] Apache Hadoop Website. URL: <http://hadoop.apache.org/>.
- [2] Apache Spark Website. URL: <http://spark.apache.org/>.
- [3] Flexiant Cloud Orchestrator Website. URL: <https://www.flexiant.com/>.
- [4] Apache ZooKeeper, Jun., 2017. URL: <https://zookeeper.apache.org/>.
- [5] Amazon AWS . EC2 spot instances, Jun., 2017. URL: <https://aws.amazon.com/ec2/spot>.
- [6] Apache Storm. Setting up a Storm Cluster, Jun., 2017. URL: <http://storm.apache.org/releases/1.0.3/Setting-up-a-Storm-cluster.html>.
- [7] Danilo Ardagna, Simona Bernardi, Eugenio Gianniti, Soroush Karimian Aliabadi, Diego Perez-Palacin, and José Ignacio Requeno. Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets. In *Algorithms and Architectures for Parallel Processing*, pages 599–613. Springer, 2016.
- [8] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.
- [9] F. Bause. Queueing petri nets: a formalism for the combined qualitative and quantitative analysis of systems. In *Proc. of the 5th Int.l Workshop on Petri Nets and Performance Models*, pages 14–23, Toulouse (France), 1993. IEEE Press.
- [10] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets: An Introduction to the Theory*. Vieweg Verlag, 2002.
- [11] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009. URL: jmt.sf.net.
- [12] The DICE Consortium. Requirement Specification. 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/D1.2_Requirement-specification.pdf.
- [13] The DICE Consortium. Requirement Specification - Companion Document. 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/D1.2_Requirement-specification_Companion.pdf.
- [14] The DICE Consortium. DICE simulation tools - Initial version. Technical report, European Union’s Horizon 2020 research and innovation programme, 2016. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/02/D3.2_DICE-simulation-tools-Initial-version.pdf.
- [15] The DICE Consortium. DICE simulation tools - Intermediate version. Technical report, European Union’s Horizon 2020 research and innovation programme, 2016. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/02/D3.3_DICE-simulation-tools-Intermediate-version.pdf.
- [16] The DICE Consortium. DICE transformations to Analysis Models. Technical report, European Union’s Horizon 2020 research and innovation programme, 2016. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/08/D3.1_Transformations-to-analysis-models.pdf.
- [17] The DICE Consortium. D3.8 DICE Optimisation Tools - Final version. Technical report, European Union’s Horizon 2020 research and innovation programme, 2017. To be released at M30.

- [18] 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.
- [19] Docker. Limit a container's resources, Jun., 2017. URL: https://docs.docker.com/engine/admin/resource_constraints/#cpu.
- [20] Daniel J. Dubois and Giuliano Casale. Optispot: minimizing application deployment cost using spot cloud resources. *Cluster Computing*, 19(2):893–909, Jun 2016.
- [21] Eugenio Gianniti, Alessandro Maria Rizzi, Enrico Barbierato, Marco Gribaudo, and Danilo Ardagna. Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications. *ACM SIGMETRICS Performance Evaluation Review*, 44(4):23–36, 2017.
- [22] Abel Gómez, José Merseguer, Elisabetta Di Nitto, and Damian A. Tamburri. Towards a UML Profile for Data Intensive Applications. In *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*, pages 18–23, New York, NY, USA, 2016. ACM.
- [23] ISO. Systems and software engineering – High-level Petri nets – Part 2: Transfer format. ISO/IEC 15909-2:2011, International Organization for Standardization, Geneva, Switzerland, 2011.
- [24] Stuart Kent. Model Driven Engineering. In Michael J. Butler, Luigia Petre, and Kaisa Sere, editors, *Proceedings of the 3rd International Conference of Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298, Turku, Finland, May 2002. Springer.
- [25] Johannes Kroß, Andreas Brunnert, and Helmut Krcmar. Modeling Big Data Systems by Extending the Palladio Component Model. *Softwaretechnik-Trends*, 35(3), 2015.
- [26] Johannes Kroß and Helmut Krcmar. Modeling and Simulating Apache Spark Streaming Applications.
- [27] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.
- [28] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalised Stochastic Petri Nets*. John Wiley and Sons, 1995.
- [29] Marco Ajmone Marsan, G. Balbo, Gianni Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri nets*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.
- [30] P. O'Connor, D. Newton, and R. Bromley. *Practical Reliability Engineering*. Wiley, 2002.
- [31] 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/>.
- [32] OMG. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1, June 2011. URL: <http://www.omg.org/spec/MARTE/1.1/>.
- [33] Rajiv Ranjan. Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks. *IEEE Cloud Computing*, 1(4):14–19, 2014.
- [34] The DICE Consortium. Design and Quality Abstractions - Final Version. Technical report, European Union's Horizon 2020 research and innovation programme, 2017. URL: <https://vm-project-dice.doc.ic.ac.uk/redmine/projects/dice/repository/show/WP2/D2.1/submitted/D2.1.pdf>.
- [35] The DICE Consortium. DICE Profile Repository, Dec., 2015. URL: <https://github.com/dice-project/DICE-Simulation>.

- [36] The DICE Consortium. DICE Simulation Repository, Dec., 2015. URL: <https://github.com/dice-project/DICE-Simulation>.
- [37] The Eclipse Foundation & Obeo. Acceleo, Dec., 2015. URL: <https://eclipse.org/acceleo/>.
- [38] Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for apache spark platform. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conference on Embedded Software and Systems (ICESSE), 2015 IEEE 17th International Conference on*, pages 166–173. IEEE, 2015.

Appendix A. The DICE-Simulation Repository

This appendix describes the *DICE-Simulation repository* [36]. This repository contains the following projects/plugin-ins:

- es.unizar.disco.core** — This project contains the *Core plug-in*. The *Core plug-in* provides some utility classes for I/O, together with the shared logging capabilities.
- es.unizar.disco.core.ui** — This project contains the *Core UI plug-in*. The *Core UI plug-in* provides UI components that are shared across the different plug-ins contained in this repository, such as file selection dialogs.
- es.unizar.disco.pnconfig** — This project contains the implementation of the *Configuration Model* as an EMF plug-in.
- es.unizar.disco.pnml.m2m** — This project implements the M2M transformations from UML to PNML using QVTo.
- es.unizar.disco.pnextensions** — This project provides some utilities to handle some extensions in PNML models. The PNML standard does not provide support for time, probabilities and priorities in transitions. These features are present in the Generalized and Stochastic Petri nets type or the colored Petri nets of type called Stochastic Well-formed nets. Thus, this plug-in provides the utility methods to handle this information by using the *ToolSpecifics* tags provided by the PNML standard.
- es.unizar.disco.pnml.m2t** — This project contains the Acceleo [37] transformation to convert a DICE-annotated PNML file to a set GreatSPN files.
- es.unizar.disco.simulation.greatspn.ssh** — This project contains the OSGi component that controls a remote GreatSPN instance by using SSH commands.
- es.unizar.disco.simulation** — This project contains the core component that executes a simulation by orchestrating the interactions among all the previous components.
- es.unizar.disco.simulation.ui** — This project contains the UI contributions that allow the users to invoke a simulation within the Eclipse GUI and to graphically visualize the simulation results.
- es.unizar.disco.simulation.local** — This project contains the OSGi component that implements a local solver of reliability properties based on RBD.
- es.unizar.disco.ssh** — This project provides a simple extension point contribution to access a remote host by issuing the connection data using a local file.
- com.hierynomus.sshj** — This project contains the *sshj - SSHv2 library for Java* as an OSGi-friendly bundle. This module is required by `es.unizar.disco.simulation.greatspn.ssh` to access a remote *GreatSPN* instance using SSH/SFTP.

Appendix B. Modeling and transformations to Analysis Models of Apache Spark DIAs

This version of the tool has incorporated the quality assessment of the Apache Spark applications. Section B.1 introduces our work on Apache Spark. Section B.2 presents the basics on Apache Spark Core focussing on the parameters that mainly affect the performance of an application. Section B.3 presents our performance modeling approach for Apache Spark applications. Section B.4 introduces the Spark UML profile required for modelling Apache Spark applications. Section B.5 details the transformation that we propose to get an analyzable performance model out of a Spark design. Finally, Section B.5 is devoted to the computation of performance metrics and the validation of our approach. To this end, we compare the results obtained by real Spark applications with the predictions of the automatically constructed performance model.

B.1 Introduction of model-driven performance assessment of Spark applications

In this deliverable, we present a quality-driven approach for developing Spark applications hosted in private or public clouds. In particular, we offer a modeling approach and a novel UML profile for a better performance characterization of a Spark design. We define transformations for these Spark designs into suitable models that are used for performance assessment.

On the previous basis, we have developed a toolchain to guide the early design stages of the Spark applications and also to guide quality evolution once operational data becomes available. In fact, our simulation-based approach, with its corresponding tools and formalisms, is useful for predicting the behaviour of the application for future demands, and the impact of the stress situations in some performance parameters (response time, throughput or utilization).

Several works have been already presented in the literature for the modeling and performance assessment in big data platforms [33]. For instance, a profile for modelling big data applications is already defined for the Palladio Component Model [25]. Mathematical models for predicting the performance of Spark applications are already introduced in [38]. In [26], the authors model and simulate Apache Spark streaming applications in Palladio, but they did not focus on Spark Core (MapReduce) operations.

Generalized stochastic Petri nets (GSPNs), the formalism for performance analysis that we adopt here, have been already used for the performance assessment of Hadoop applications [7]. That work has been already extended for the Spark performance evaluation using fluid Petri nets [21]. Definitely, to the best of our knowledge this is the first work entirely devoted to the Apache Spark Core performance evaluation using a UML profile and GSPNs.

B.2 Spark and Performance

In these paragraphs, we describe the main concepts of the Spark technology, offering then an overview of this framework. In particular, we stress those concepts that directly impact on the performance of the system. They are configuration parameters that, when they are correctly tuned, allow one to improve the performance of the Spark applications. Consequently, these parameters are essential for the performance analysis of the Spark applications.

Apache Spark is a distributed fault-tolerant computation framework for processing large volumes of information in cloud and clusters environments [2]. It relies on Apache Hadoop MapReduce [1], which is based on the MapReduce programming paradigm. Spark solves or mitigates the main limitations that appeared in the previous technology. First of all, programming in Spark is easier and more expressive than in Hadoop MapReduce: while Hadoop MapReduce works with plain maps and reduces, Spark includes more elaborated operations such as filters, joins and groupBy functions. In addition, Spark counts with a rich ecosystem that exports the Spark core operations to other contexts such as structured data (Spark SQL + DataFrames), streaming analytics (Spark Streaming), machine learning (MLlib) or graph computation (GraphX). Finally, and more important, Spark usually gets a better performance than Hadoop MapReduce for equivalent applications. Spark achieves this speedup by keeping in memory the

intermediate results of a MapReduce phase as long as possible, instead of storing them directly in hard disk.

Spark works with resilient distributed datasets (RDD), a fault-tolerant and distributed data structure that allows the partition and storage of the initial dataset (i.e., files, arrays, etc.) in multiple workstations. A Spark application consists of a succession of operations over the portions of those RDDs (i.e., *partitions* in the Spark notation). Both the information and the computations are then distributed among all the computational resources of the cluster. The Spark framework is responsible for the automatic distribution and coordination of all the workflow; which makes it transparent to the end user. Thus, at a top level of abstraction, the execution workflow of a Spark application can be described as a directed acyclic graph (DAG) showing the modifications applied over the RDDs.

Spark operations are grouped in two categories: *transformations* and *actions*. Transformations are similar to Maps in Hadoop MapReduce. They are functions that create a new RDD from the previous one. They are lazy in nature and they are executed only after an action is triggered. A *narrow transformation* converts a partition from a parent RDD into a new one. A *wide transformation* is another kind of transformation where multiple child partitions (i.e., operation results) depend on one partition from the precedent RDD. Examples of narrow transformations are *map* or *filter* functions; while examples of wide transformations are *groupByKey* or *reduceByKey* aggregations. The results of a narrow transformation are shuffled when transmitting them to a wide transformation.

On the other hand, actions are similar to Reduces in Hadoop MapReduce. They are functions that perform a final operation over an existing RDD but they do not create a new one. For instance, examples of actions are *first*, *take*, *collect* or *count*.

Spark organizes a job in multiple stages by breaking the DAG at shuffle boundaries or when a result set (action operation) are requested. Each stage has a sequence of tasks that run *narrow* transformations and finishes with a shuffle operation.

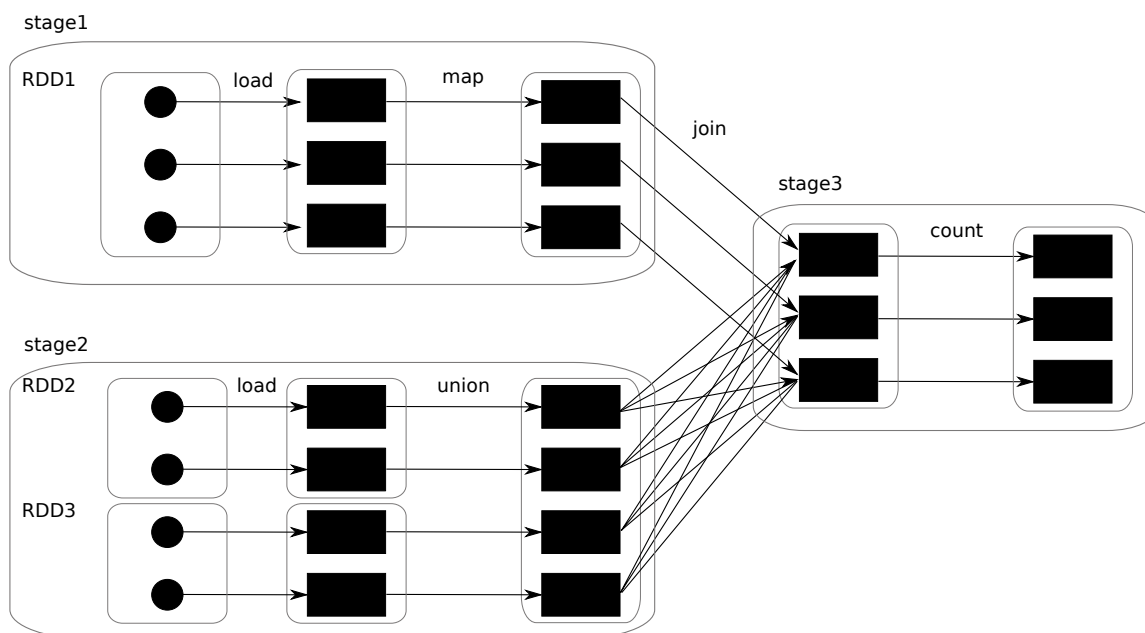


Figure 35: DAG representing a Spark application

Both the transformations and actions are internally divided in multiple parallel subtasks inside the Spark framework. Each task executes the same method defined in the Spark operation but applying it over a smaller chunk of the dataset (i.e., partition of the RDD). By default, the number of tasks run in each Spark operation matches the number of partitions of the RDD that is taken as input.

However, the number of parallel tasks is not constant for all the Spark operations of a Spark workflow, and it may change according to a few factors. The number of parallel tasks can be explicitly defined by the end user; or automatically adjusted by the Spark context according to the RDD size and the cluster configuration. For instance, set transformations such as the union, intersection or the cartesian product

of two RDDs result in a new RDD with a total number of partitions equal to the addition, maximum or product of partitions from the two RDD.

Figure 35 details the DAG of a Spark application. It loads three RDDs. The dataset RDD1 has three partitions, and the datasets RDD2 and RDD3 have two partitions each one. Next, a map function is applied over the partitions of RDD1. RDD2 and RDD3 are united in a new RDD with four partitions. Later, they are joined in a new RDD during the stage 3. At the end, the system counts the number of elements inside each partition of the dataset.

Finally, the Spark *scheduling algorithm* deploys the tasks to the computational resources of the cluster. In Spark, there are two levels of scheduling: an inter-application level, that provides scheduling across all the Spark applications running in the same cluster at the same time; and an intra-application (or task) level, that provides scheduling for all the tasks running within a Spark application.

Spark counts with three predefined schedulers at the inter-application level: YARN, Mesos or the standalone mode. They vary in the management of the cluster resources. They provide either a static partition of the resources among all the applications before launching (YARN, standalone); or a dynamic sharing of CPU cores on runtime (Mesos). Within a Spark application, there are also three possible scheduling policies: *fair*, *fifo* or *none*. A fair scheduling separates all the tasks into a set of pools with the same priority; and applies a fifo scheduling within each queue.

Complex schedulers may take into account the available computational resources and the software requirements (memory and CPU consumption) for defining an optimal distribution of the tasks. In summary, a Spark framework is highly configurable by various parameters that will influence the final performance of the application (see Table 3).

Table 3: Spark concepts that impact in performance

#	Concept	Meaning
1.	<i>RDD</i>	Resilient distributed dataset.
2.	<i>Partition</i>	Chunk of data in a RDD.
3.	<i>Spark Operation</i>	Function that transforms or acts over a RDD.
4.	<i>Transformation</i>	Spark operation that creates a new RDD.
5.	<i>Action</i>	Spark operation that produces non-RDD values.
6.	<i>Parallelism</i>	Number of concurrent tasks per Spark operation.
7.	<i>Stage</i>	Group of transformations before a shuffle.
8.	<i>Scheduling</i>	Deployment of tasks.

B.3 Modelling Spark applications with UML

In this section, we present our approach to model Spark applications for performance evaluation using UML diagrams. At least we need to represent the Spark topology, i.e., the DAG, and the performance parameters already identified in Section B.2. Specifically, we will work with *activity diagrams* complemented with *deployment diagrams*, which are the diagrams that will be transformed to performance analysis models.

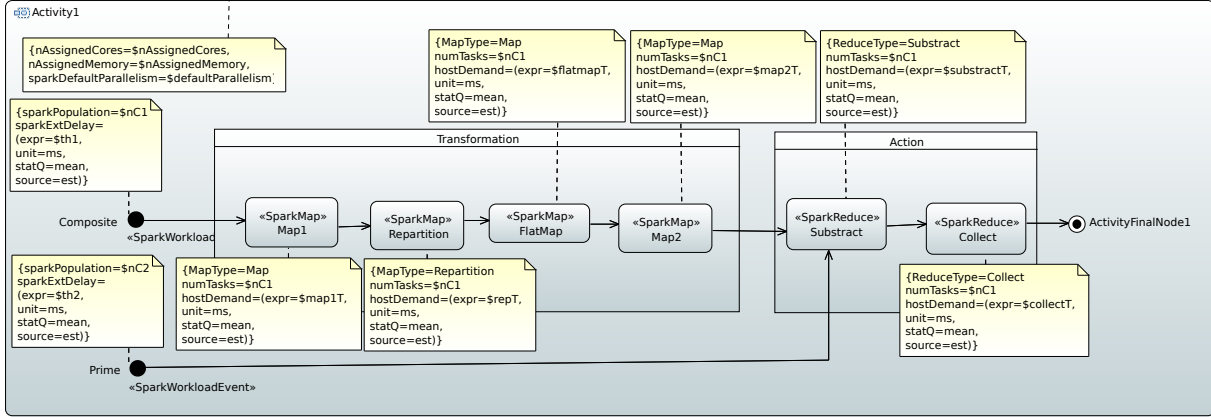


Figure 36: Example of an activity diagram for Spark with profile annotations

In this section, we present a Spark toy example that helps us to introduce the UML models and profile. Figure 36 represents the UML activity diagram of a Spark application that calculates all the prime numbers located in the range $[0, N]$. The idea behind the naive algorithm consists of deleting from the interval all the elements that are not primes (i.e., they are multiple of a in $[0, N/a]$). More in detail, the procedure consists of:

- map1: creating a dataset with all the pairs $(a, \text{list}(2 \text{ to } (N/a)))$, being $\text{list}(2 \text{ to } (N/a))$ the list of integers from 2 to N/a ;
- repartition: rebalancing the lists in the computational nodes so that each node has a portion (i.e., set of lists) of the same size;
- flatmap+map: obtaining new lists by multiplying every element of $\text{list}(2 \text{ to } (N/a))$ by a ;
- subtract: removing all the elements in the integer range $[0, N]$ that are stored in the previous lists and, consequently, they are not primes; and, finally,

collect getting the result set.

Thus, the Spark application is composed of a sequence of transformations (maps) and actions (subtract, collect) over two datasets, named *composite* and *prime*. A dataset corresponds to a RDD in the Spark notation. The procedure consists in the creation of two datasets with integers and operating over them. The first RDD *composite* represents all the non-prime numbers of the interval $[0, N]$. The second RDD *prime* represents all the primes obtained by removing the non-prime numbers (i.e., *composite*) from the interval $[0, N]$.

In our approach, the UML activity diagram is interpreted as the DAG of a Spark application. Our semantic of a UML activity diagram is slightly different from the standard one. The standard UML semantic considers that the UML activity diagram has a single initial node and a single final node. However, in our case we are representing the execution workflow; that is, a successive set of operations over a RDD (Figure 36). A Spark application manages one or more RDDs and, therefore, our UML activity diagram accepts several initial and final nodes.

Each pipeline shows the operations executed for each RDD. Every initial node in the UML diagram corresponds to the initialization of a RDD (i.e., loading of a log file, creation of an array, etc.) in the

corresponding pipeline of the Spark context. There are as many initial nodes as RDDs defined in the application. Multiple final nodes are also allowed; one per RDD pipeline.

Besides, the combination of two RDDs in a single one is shown by the *subtract* set operation, whose activity node receives two input arcs (one per RDD). The response time of the Spark application is considered since the initialization of all the RDDs (initial nodes) until the end of the latest task.

UML fork and join nodes are also supported following the standard UML modelling semantics. The activity nodes (actions, forks and joins) in the UML activity diagram are grouped by UML partitions (e.g., Transformation and Action in Figure 36). Each UML partition is mapped to a computational resource in the UML deployment diagram following the *scheduling* policy defined for the topology. Figure 37 shows the deployment diagram, which complements the previous activity diagram. Each computational resource is stereotyped as *SparkNode* (equivalent to *GaExecHost*) and defines its resource multiplicity, i.e., number of cores. This stereotype is inherited from MARTE GQAM.

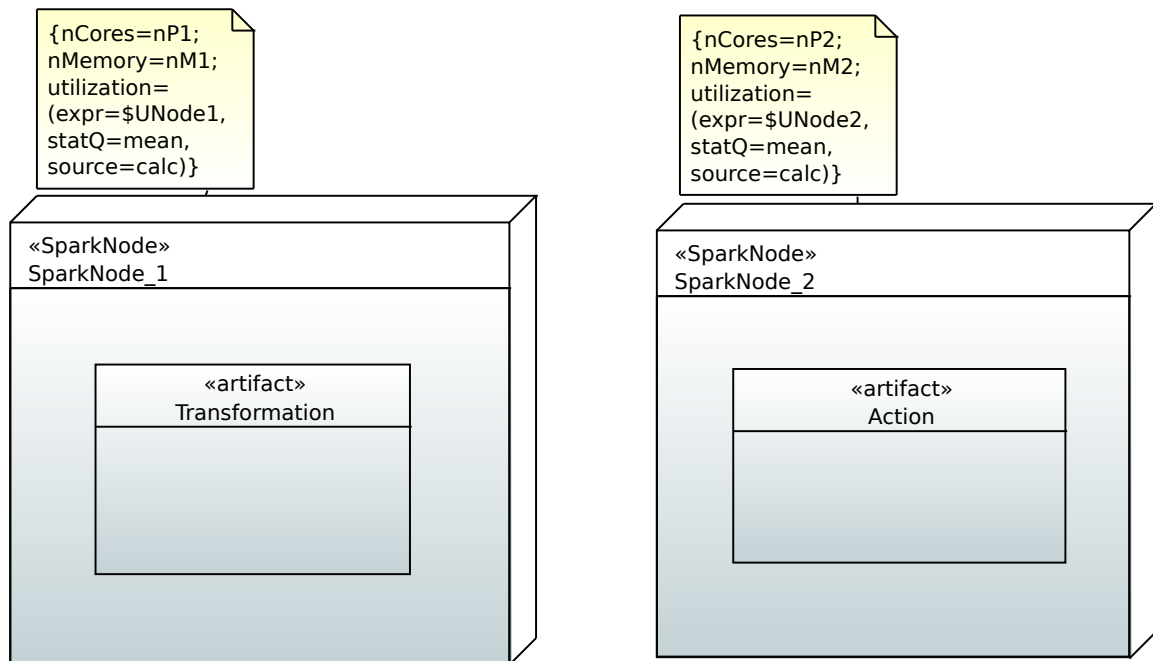


Figure 37: Example of deployment diagram for Spark with profile annotations

B.4 A UML Profile for Spark

Once the Spark topology and operations have already been represented, we still need to address the rest of concepts identified in Table 3. We decided to convert these concepts into stereotypes and tags, which are the extension mechanisms offered by UML. Therefore, we devised a UML profile for Spark. A UML profile is a set of stereotypes that can be applied to UML model elements for extending their semantics [22, 34]. In our case, we are extending UML with the Spark concepts.

The Spark profile heavily relies on the standard MARTE profile [32]. This is because MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis, which is indeed specialized for performance analysis, then perfectly matching to our purposes. Moreover, MARTE offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The latter, VSL sub-profile, provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance, in our particular case.

VSL expressions are used in Spark-profiled models with two main goals: (i) to specify the values of the NFP in the model (i.e., to specify the input parameters) and (ii) to specify the metric/s that will be computed for the current model (i.e., to specify the output results). An example VSL expression for a host demand tagged value of type NFP_Duration is:

```
(expr=$mapT1, unit=ms, statQ=mean, source=est)
(1)           (2)           (3)           (4)
```

This expression specifies that **map1** in Figure 36 demands \$mapT1 (1) *milliseconds* (2) of processing time, whose mean value (3) will be obtained from an estimation in the real system (4). \$mapT1 is a variable that can be set with concrete values during the analysis of the model.

Another VSL interesting example in Figure 37 is the definition of the performance metric to be calculated, the utilization in the example:

```
(expr=$UNode1, statQ=mean, source=calc)
(1)           (2)           (3)
```

This expression specifies that we want to calculate (3) the utilization, as a percentage of time, of the whole system or a specific resource, whose mean value (2) will be obtained in variable \$UNode1 (1). Such value is obviously the result of the simulation of the performance model.

Apart from the MARTE stereotypes, the Spark profile provides genuine stereotypes (see Table 4) for representing those parameters not already addressed, i.e., the Spark concepts 1, 3-6 and 8 in Table 3.

The initialization of a RDD is described by an initial node in the UML activity diagram. The stereotype *SparkWorkloadEvent* is used for labelling the initial node. This stereotype captures the essential details of the creation of the RDD. Mainly, the initialization is represented by two parameters. The first one is the *sparkPopulation* tag. It corresponds to the number of chunks in which the input data is divided when generating the RDD structure. This value will affect to the parallelism of the following operations that are executed over this dataset. That is; it indirectly specifies the number of subtasks that a Spark operation can run. This parameter can be explicitly overridden by the user (see the *numTasks* tag in the *SparkOperation* stereotype). The second parameter of the *SparkWorkloadEvent* stereotype is the *sparkExtDelay*. It shows the time spent in loading and preparing the data (i.e., a log file) into the format of a RDD. By default, all the *sparkPopulation* partitions of the RDD are read and prepared to be scheduled after *sparkExtDelay* units of time.

The *SparkWorkloadEvent* stereotype inherits from MARTE::GQAM::GaWorkloadEvent stereotype. The inherited attribute *pattern* allows the definition of arrival policies that modify the management of the partitions. For instance, the attribute *pattern* may define the preparation of one partition of the RDD periodically every X milliseconds instead of preparing all the block of partitions in batch. The definition of a periodic workload is compatible with the streaming analytics supported by Spark.

Transformations (*SparkMap*) and *actions* (*SparkReduce*) have independent stereotypes because they are conceptually different, but they inherit from *SparkOperation* stereotype and, indirectly, from MARTE::GQAM::GaStep stereotype since they are computational steps. In fact, they share the *parallelism*,

or number of concurrent tasks per operation, which is specified by the tag `numTasks` of the *SparkOperation* stereotype. Each task has an associated execution time denoted by the tag `hostDemand`, an attribute inherited from *GaStep*. The tag `OpType` specifies the type of Spark operation (i.e., an enumerable `SparkOperation={transformation, action}`) in case of using the *SparkOperation* stereotype when modeling UML diagrams.

More in detail, the *SparkMap* stereotype adds the tag `MapType`, which represents the kind of *transformation* applied over the data (e.g., map, filter or a set operation such as union or intersection). The *SparkReduce* stereotype includes the tag `ReduceType` for specifying the kind of *action* applied over the data (e.g., reduce, count or collect). All these stereotypes are applied to the opaque actions.

The concept of *scheduling* in Spark is captured by the stereotype *SparkScenario*. For simplicity in this first version, we only support a Spark cluster deployed with a static assignation of the resources to Spark jobs (e.g., YARN or standalone modes); and an internal fifo policy (task scheduler). Therefore, the number of CPU cores and memory are statically assigned to the application on launching time. This configuration is reflected in the tags `nAssignedCores`, `nAssignedMemory` and `sparkDefaultParallelism`. They represent respectively the amount of computational cores and memory resources assigned by the scheduler to the current application; and the default parallelism of the cluster configuration. The attribute `sparkDefaultParallelism` specifies the default number of partitions in a RDD when *SparkWorkload-Event*→`sparkPopulation` is not defined. It also determines the number of partitions returned in a RDD by transformations and actions like *count*, *join* or *reduceByKey* when the value of `numTasks` is not explicitly set by the user.

The stereotype *SparkScenario* inherits from `MARTE::GQAM::GaScenario`. It gathers the rest of the contextual information of the application; for instance, the response time or throughput that will be computed by the simulation. The *SparkScenario* stereotype is applied to the activity diagram.

Finally, the *SparkNode* stereotype is applied over the computational devices in the UML deployment diagram. It represents a resource in the Spark cluster where the tasks are run. The main attributes are the `nCores` and `Memory`. The first tag corresponds to the number of available CPUs in the device. The second tag is a boolean that indicates if the size of the partitions in the RDD fit in the memory of the server or they must be stored in a temporal file. The *SparkNode* stereotype inherits from `DICE::DTSM::Core::CoreComputationNode` and it is equivalent to the *GaExecHost* stereotype from MARTE.

The Spark profile has been implemented for the Papyrus Modelling environment for Eclipse and can be downloaded from [35].

Table 4: Spark profile extensions

Spark Concept	Stereotype	Applied to	Tag	Type
<i>Generic Spark Operation</i>	«Spark-Operation»	Action/Activity Node	OpType numTasks hostDemand	SparkOperation NFP_Integer NFP_Duration
<i>Transformation</i>	«SparkMap»	Action/Activity Node	MapType	SparkMap
<i>Action</i>	«SparkReduce»	Action/Activity Node	ReduceType	SparkReduce
<i>Scheduler</i>	«SparkScenario»	Activity Diagram	nAssignedCores nAssignedMemory sparkDefaultParallelism	NFP_Integer NFP_Integer NFP_Integer
<i>RDD Initialization</i>	«Spark-Workload-Event»	Initial Node	sparkPopulation sparkExtDelay	NFP_Integer NFP_Duration
<i>Computational Node</i>	«SparkNode»	Device/Node	nCores Memory	NFP_Integer NFP_Boolean

B.5 Transformation of the UML Design

We need to transform our Spark design into a performance model capable of evaluating the metrics (e.g., response time, throughput or utilization of the devices) already defined. We choose Generalized Stochastic Petri Net (GSPN) [29] as target performance model since they are suitable for modelling Spark applications (See Appendix E). In this section, we propose a set of original *transformation patterns*; each pattern takes as input a part of the Spark design and produces a GSPN subnet. These patterns are ready for implementing a model-to-model transformation (M2M) [24] to automatically generate the GSPN model. The correctness and compositionality of the transformation are validated experimentally in Appendix B.5.

Table 5 presents the patterns for the activity diagram. The first column presents the input of the pattern, i.e., the UML model elements, possibly stereotyped with the Spark profile. The second column proposes the corresponding GSPN subnet. For an easier understanding of the transformation, we depicted in the table: a) text in bold to match input and output elements; b) interfaces with other patterns as dotted grey elements, then they actually do not belong to the pattern.

Pattern *P1* corresponds to the transformation of the UML activity scenario into a Petri subnet. Pattern *P1*, together with pattern *P4*, transforms the activity diagram into a closed Petri net. The subnet consists of two places and two immediate transitions. The place $p_{Activity1}$ is the initial node and the place $p_{Activity2}$ represents the set of assigned resources of the cluster. Our transformation only is compatible with `nAssignedCores` for the moment.

Pattern *P2* shows the transformation of a initial node labelled with the *SparkWorkloadEvent*. Multiple initial nodes are supported and each one initializes a new RDD in concurrency. The subnet consists of two places and a timed transition. It uses two immediate transitions from pattern 1 and pattern 3 as interfaces. After a mean execution time of $1/\$time$ time units (t_1), the RDD is prepared for continuing the workflow. The timed transition (t_1) follows an infinite server semantic; and the execution of tasks is modelled by a exponential distribution. The value `sparkPopulation` is stored temporally and later reused in the following transformation patterns.

Pattern *P3* represents the transformation of a Spark Map and Reduce operation. The subnet consists of four places, three immediate transitions and a timed transition. The first place (p_{A1}) prepares the set of subtasks that will be executed. The number of tasks is specified by `numTasks`, in the case that the value is explicitly defined. The second place (p_{A2}) represents the execution phase. The access to this place is controlled by $p_{Activity2}$ and p_R (see Pattern *P8*). After a mean execution time of $1/\$time$ time units (t_{A3}), the current task has finished and it waits until the rest of tasks are completed (p_{A3}). The timed transition (t_3) follows an infinite server semantic; and the execution of tasks is modelled by a exponential distribution. Finally, the result is broadcast to the next Spark operation via the last place (p_{A4}).

Table 6 presents the special characteristics of the transformation of a Spark Map and Reduce operations. If `numTasks` is not initialized, then the number of tasks is equal to the number of partitions (*SparkWorkloadEvent*→`sparkPopulation`) of the RDD. The number of partitions of a RDD is equal to the default parallelism of the cluster (`sparkDefaultParallelism`) when the user does not explicitly specifies the partitioning (pattern *P8*). In the case of a Spark operation involving two RDDs, then the number of tasks depend on the type of operation (`mapType/reduceType`). For instance, the union, intersection or Cartesian product of two RDDs leads to `numTasks_1 + numTasks_2`, `max(numTasks_1, numTasks_2)` and `numTasks_1 * numTasks_2` respectively (patterns *P9–P11*).

Patterns *P5*, *P6* and *P7* show the concatenation of two Spark operations directly or via fork/join nodes. The element A in *P5* is either a opaque action (`SparkMap` or `SparkReduce`) or a UML fork/join node.

Table 5: Transformation Patterns for Spark-profiled UML Activity Diagrams

UML PATTERN	PETRI NET PATTERN
<p>P1</p> <p>«SparkScenario» $nAssignedCores = \\$nAsC$ $nAssignedMemory = \\$nAsM$ $sparkDefaultparallelism = \\nDf</p>	<p>$M(p_{Activity1}) = 1$ $M(p_{Activity2}) = \\$nAsC$</p>
<p>P2</p> <p>«SparkWorkloadEvent» $sparkExtDelay = (expr = \\$time, unit = s, source = est, statQ = mean)$ $sparkPopulation = \\$nC1$</p>	<p>$r(t_1) = 1/\\$time$</p>
<p>P3</p> <p>«SparkMap» $hostDemand = (expr = \\$time, unit = s, source = est, statQ = mean)$ $numTasks = \\$n0$ $mapType = map$</p>	<p>$W(a_1) = \\$n0$ $r(t_{A3}) = 1/\\$time$ $W(a_2) = \\$n0$</p>
<p>P4</p>	
<p>P5</p>	
<p>P6</p>	

continued ...

...continued

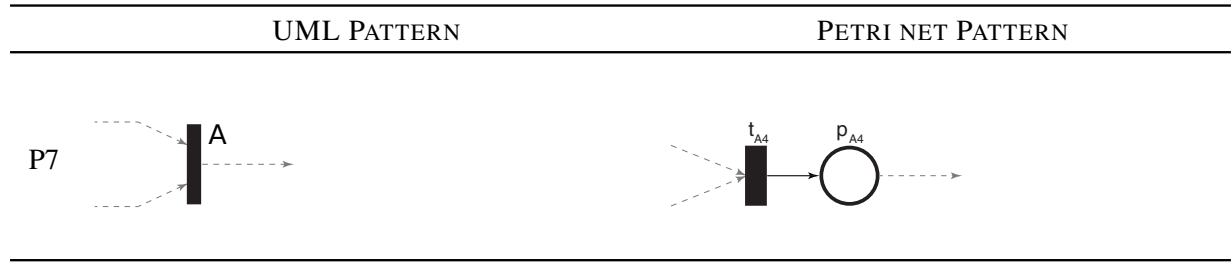


Table 7 illustrates the modifications introduced in the GSPN model by the profile extensions in the deployment diagram. The Spark tasks are first logically grouped into partitions in the activity diagram, later they are deployed as artifacts and mapped to physical execution nodes in the deployment diagram. Pattern *P12* maps the *SparkNode* (*GaExecHost*) to a new place p_R in the GSPN, with an initial marking $nCores$ ($resMult$) that represents the number of computational cores of the node. The addition of such place restricts the logical concurrency, number of tasks of the Spark application, to the number of available cores. In particular, the pattern *P8* corresponds to the acquire/release operations of the cores by the tasks.

Figure 38 presents the final GSPN model for the Spark design in Figures 36 and 37. It has been obtained by applying the patterns and combining the subnets through the interfaces. For readability purposes, the portions of the Petri net are encapsulated in a frame with the name of the UML element that generates it. By default, the number of tasks for each Spark operation is equal to 100. The Spark application has 10 cores assigned from the 16 cores of the cluster.

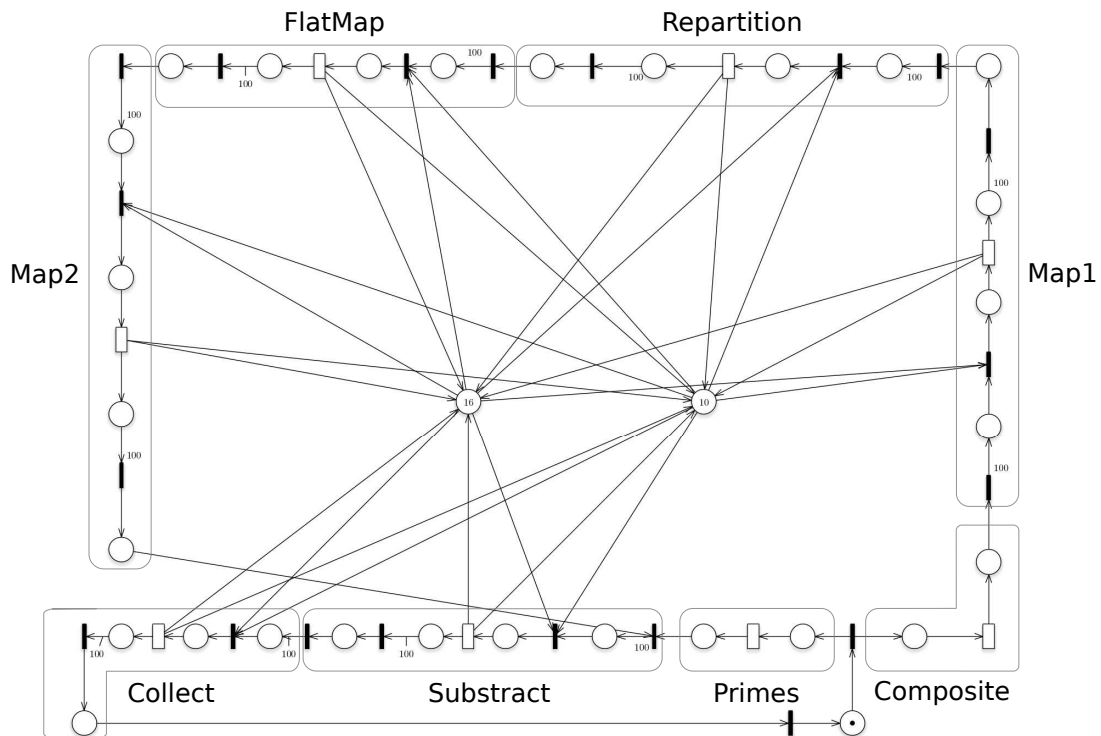


Figure 38: GSPN for the Spark design in Figures 36 and 37

The Spark profile, the transformation patterns, and the evaluation of performance metrics have been implemented for the Papyrus Modelling environment in Eclipse. In particular, they are completely integrated within the DICE Simulation plugin for Eclipse.

The transformation of the UML models to stochastic Petri nets, as well as the evaluation of the performance metrics, are fully automatized and they are transparent to the end user. Firstly, the transformation uses QVT-MOF 2.0 [31] to obtain a Petri Net Markup Language file (PNML) [23], an ISO standard for XML-based interchange format for Petri nets. A trace file is created during the M2M transformation.

Table 6: Transformation Patterns for SparkMap stereotype in Spark-profiled UML Activity Diagrams

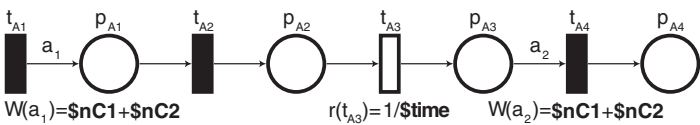
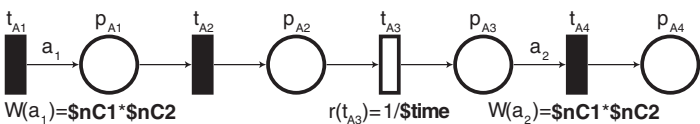
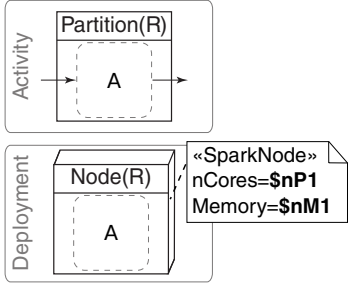
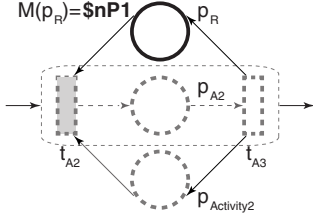
UML PATTERN	PETRI NET PATTERN
<p>P8</p>  <pre> «SparkMap» hostDemand=(expr=\$time, unit=s, source=est, statQ=mean) numTasks=--- mapType=map </pre>	
<p>P9</p>  <pre> «SparkMap» hostDemand=(expr=\$time, unit=s, source=est, statQ=mean) numTasks=--- mapType=union </pre>	
<p>P10</p>  <pre> «SparkMap» hostDemand=(expr=\$time, unit=s, source=est, statQ=mean) numTasks=--- mapType=intersection </pre>	
<p>P11</p>  <pre> «SparkMap» hostDemand=(expr=\$time, unit=s, source=est, statQ=mean) numTasks=--- mapType=cartesian </pre>	

Table 7: Transformation Patterns for Spark-profiled UML Deployment Diagrams

	UML PATTERN	PETRI NET PATTERN
P12		

This file links the elements of the Petri net with the source components of the UML. It helps for the identification of the items in the Petri net that the tool needs to inspect during the performance analysis. Later on, Acceleo [37] has been used to implement a model-to-text (M2T) transformation from PNML into a GSPN tool specific format, concretely for the GreatSPN tool [18].

The UML Profile for Spark is published inside the DICE Profile and can be downloaded from [35]. The transformation of UML profiled models into Petri nets is implemented in the DICE Simulation tool. The code of the transformation and the DICE Simulation tool are publicly available. They can be downloaded from [36].

Appendix C. JMT Petri Net Extension

JMT (Java Modelling Tools²) is an integrated environment for performance evaluation, capacity planning and workload characterisation of computer and communication systems [11]. A number of cutting-edge algorithms are available for exact, approximate and asymptotic analysis of *queueing networks* (QNs), with either product-form or non-product-form solutions. Users can define and solve models through a well-designed graphical interface, or optionally an alphanumeric wizard. Released under GPLv2, JMT benefits a large community of thousands of students, researchers and practitioners, with more than 5,000 downloads per year.

The focus of JMT is particularly on QNs, in which jobs travel through a number of stations demanding a certain amount of service at each station and possibly suffering a queueing delay due to service contention. Although QNs are a prevailing class of formalisms for performance modeling purposes, they are not suitable for describing systems that exhibit logical behaviour, for example synchronization. JMT integrates a discrete-event simulator known as JSIM. Inspired by the DICE project, an extension has been contributed to JSIM for supporting *Petri nets* (PNs), another important class of performance modeling formalisms which are found especially useful for capturing the logical behaviour of a system.

Two novel types of stations, *Place* and *Transition*, have been designed and implemented in JSIM for simulating PN models. With the Place and Transition stations, JSIM well supports almost all the canonical types of PNs including GSPNs (Generalized Stochastic Petri Nets), CPNs (Colored Petri Nets) and QPNs (Queueing Petri Nets) [10]. High compatibility has been achieved between PN and QN stations. Thus, *hybrid models* comprising both PN and QN components can also be simulated with JSIM. In hybrid models, PN components are used to exactly depict logical behaviour while QN components provide succinct representation of queueing behaviour.

Graphical User Interface

JSIM generalizes each station into *input*, *service* and *output* sections, which constitute elementary simulation entities. The Place station consists of *Storage*, *Tunnel* and *Linkage* sections. The Transition station is composed of *Enabling*, *Timing* and *Firing* sections. Users may specify the parameters of Storage, Enabling, Timing and Firing sections. Figure 39 gives an example of a simple PN model with two closed classes. Let us consider this example to see what parameters are provided by these sections.

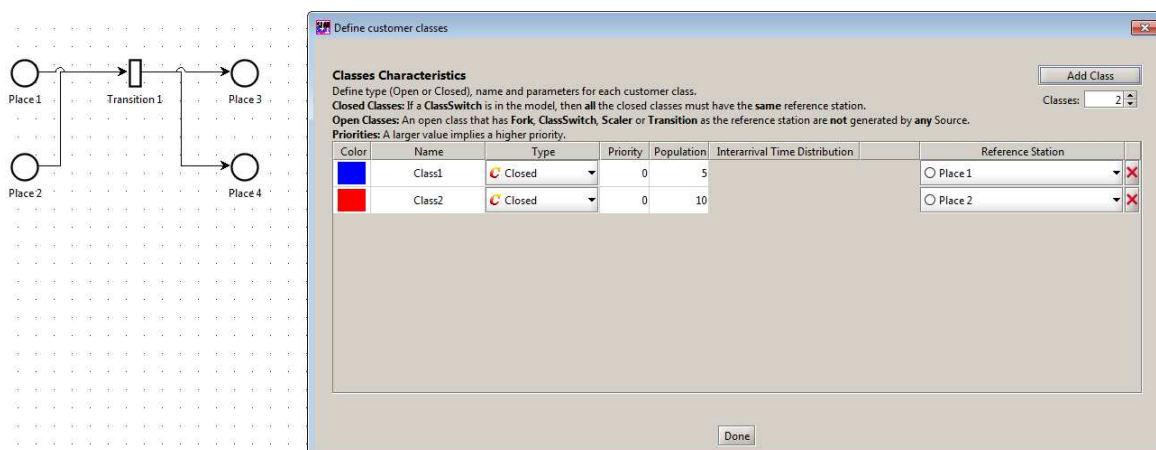


Figure 39: Example of a simple PN model with two closed classes.

The Storage section panel shown in Figure 40 allows users to specify the *storage capacities* and the *storage strategies* of Place 1. The storage capacities can be either *Infinite* or *Finite* for all the classes as well as for each class. The storage strategies include *queue policies* and *drop rules*, and are specified separately for each class. Three queue policies, *FCFS*, *LCFS* and *Random*, are applicable to the Storage

²URL: <http://jmt.sourceforge.net/>

section. The available drop rules are *Drop*, *BAS Blocking* and *Waiting Queue*, which can be selected in case of Finite storage capacities.

Class	Capacity		Queue Policy	Drop Rule
Class1	∞	<input checked="" type="checkbox"/>	FCFS	Infinite Capacity
Class2	∞	<input checked="" type="checkbox"/>	FCFS	Infinite Capacity

Done

Figure 40: Storage section panel of Place 1.

With the Enabling section panel shown in Figure 41, users may specify the *enabling condition* and the *inhibiting condition* of Transition 1 in two tables. Each row of the tables represents an input station of Transition 1, and each column corresponds to a defined job class. Therefore, an entry of the enabling table indicates the number of jobs of a class required at an input station for enabling Transition 1. Similarly, an entry of the inhibiting table indicates the number of jobs of a class required at an input station for inhibiting Transition 1. Please note that for the inhibiting table an input value of 0 means Infinity.

	Class1	Class2
Place 1	1	0
Place 2	0	2

	Class1	Class2
Place 1	10	∞
Place 2	∞	20

Done

Figure 41: Enabling section panel of Transition 1.

As shown in Figure 42, the Timing section panel is used to specify the *number of servers* and the *timing strategy* of Transition 1. The number of servers can be either Infinite or Finite. Two timing strategies, *Timed* and *Immediate*, are available. With the Timed strategy, users may specify the *firing time distribution*, for which all the implemented distributions are applicable. The *firing priority* and the *firing weight* are enabled when the Immediate strategy is selected.

Figure 43 shows the Firing section panel that allows users to specify the *firing outcome* of Transition 1 in a table. Each row of the table represents an output station of Transition 1, and each column corresponds

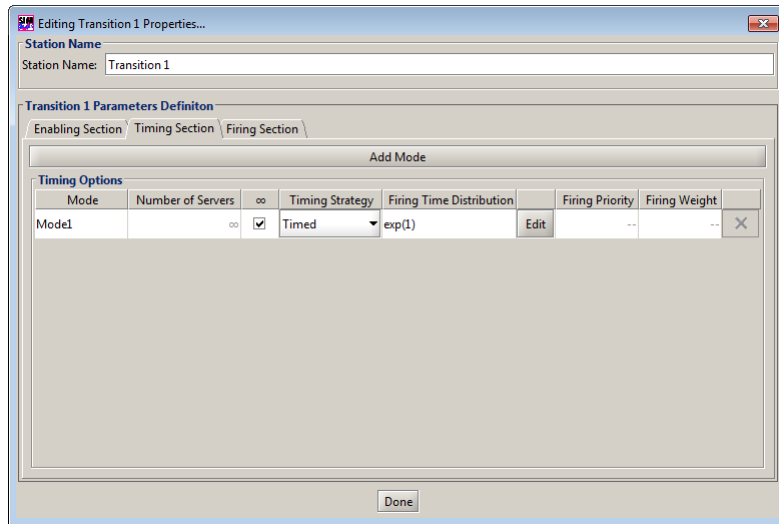


Figure 42: Timing section panel of Transition 1.

to a defined job class. Therefore, an entry of the table indicates the number of jobs of a class released to an output station on firing Transition 1.

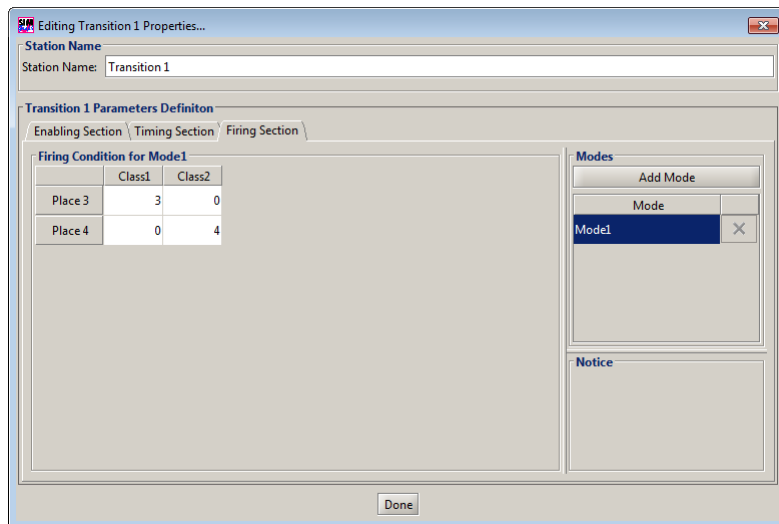


Figure 43: Firing section panel of Transition 1.

The introduction of the Petri net theory is beyond the scope of this document. Please refer to [10] or other books for more details about the basic concepts such as *place*, *timed/immediate transition*, *token*, *input/output/inhibitor arc*, *enabling rule*, *firing rule*, *marking*. The following paragraphs are dedicated to clarifying several advanced issues regarding the design and implementation of the Place and Transition stations. Some examples of PN and hybrid models are also provided in the JMT working folder to help with a better understanding of how to apply these two types of stations.

Multiple Enablings of a Transition

It is possible that a transition is enabled more than once, as multiple enabling sets of tokens may be present in the input places. The number of times that a transition is enabled, is referred to as the *enabling degree*. Formally, transition t has enabling degree k at marking M if and only if

- $\forall p \in \bullet t, M(p) \geq kI(p, t)$

- $\exists p \in \bullet t, M(p) < (k + 1)I(p, t)$

where p denotes a place, $\bullet t$ is the set of input places of transition t , $M(p)$ is the number of tokens in place p at marking M , and $I(p, t)$ is the number of tokens required in place p for enabling transition t [28]. When a Transition station is enabled, its enabling degree is taken into account.

Different meanings are possible when multiple enabling sets of tokens form in the input places of a transition. Thus, special attention must be paid to the *timing semantics* of a transition with enabling degree greater than 1. Similar to a queue, a transition can have three different timing semantics [28]:

- **Single-server Semantics:** There is a single server in the transition. Multiple enabling sets of tokens are processed serially.
- **Multiple-server Semantics:** There are multiple servers in the transition. Multiple enabling sets of tokens can be processed in parallel, but the degree of parallelism is limited by the number of servers.
- **Infinite-server Semantics:** There are infinite number of servers in the transition. Multiple enabling sets of tokens are processed in parallel regardless of the degree of parallelism.

All the three timing semantics are supported by the Transition station. Notably, the single-server and multiple-server semantics are combined together into the finite-server semantics.

Simultaneous Firings of Transitions

JSIM is equipped with a discrete-event simulation engine called JSIMengine. The core component of JSIMengine is an *event calendar*, which works as a message dispatcher between simulation entities. Messages scheduled for the same simulation time are dispatched in the order that they are created on the event calendar. This simple mechanism can result in disruption to the simulation when events completed by sequences of messages are dependent and simultaneous, because messages belonging to different sequences are interleaved arbitrarily so that dependencies between the events are likely to be broken. The firing of a Transition station is an event that must be completed by a sequence of messages. If multiple Transition stations that share some Place stations as their inputs happen to fire at the same simulation time, communication between the Place and Transition stations may probably fail to coordinate the simulation.

In order to solve the above problem, a reordering mechanism has been implemented for dispatching the messages. Particularly, this mechanism gives two properties to the firing event:

- **Atomicity:** The sequence of messages completing each firing event is dispatched as if it is atomic.
- **Certainty:** Any firing event is processed only when no other types of events are left at the same simulation time.

Besides, the reordering mechanism also determines the processing order of simultaneous firing events according to their priorities and weights.

Multiple Colors of Tokens and Multiple Modes of a Transition

Multiple classes of jobs can be defined for PN models. The classes of jobs in QNs conceptually correspond to the *colors* of tokens in CPNs. However, critical distinctions between these two concepts must be pointed out. Jobs waiting in a queue may be sorted by the priorities of their classes, while tokens stored in a place are usually treated equally notwithstanding their colors. QNs allow servers in a queue to process jobs separately or simultaneously, and the service time distributions are associated with the classes of the jobs. By contrast, CPNs allow a transition to work in different *modes*, and each mode determines an enabling condition, an inhibiting condition, a timing strategy and a firing outcome.

To meet these distinctions, only the *Non-preemptive* discipline is applicable to the input section of the Place station, i.e., the Storage section. Further, the Transition station is designed as a multi-mode transition that can process multiple colors of tokens. Therefore, any GSPN model is compressible into a CPN model with a single pair of Place and Transition stations by applying the following correspondence:

- Tokens at different Place stations in the GSPN model are associated with tokens of different colors at a Place station in the CPN model.
- Different Transition stations in the GSPN model are associated with different modes of a Transition station in the CPN model.

Compatibility between PN and QN Stations

QPNs are a hybrid formalism that reconciles PNs with QNs by introducing the *queueing place* [9]. As illustrated in Figure 44, a queueing place is composed of two parts: a queue and a depository. Any token released to the queueing place by the input transitions, enters the queue. Tokens in the queue are invisible to the output transitions. After completion of its service, a token immediately moves to the depository, where it becomes available to the output transitions.

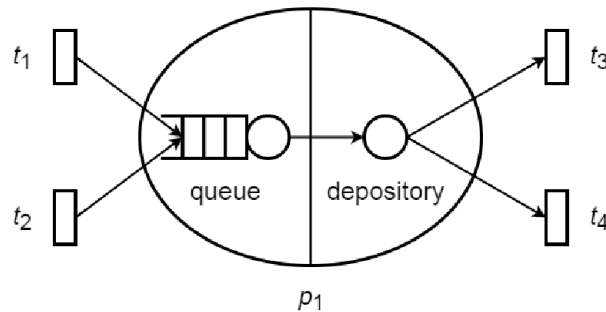


Figure 44: Structure of a queueing place.

With compatibility between PN and QN stations, a queueing place is equivalent to a Queue station connecting to a Place station. Other hybrid structures can also be easily obtained by combining different PN and QN stations together. However, the following constraints are imposed on the connections of PN stations:

- A Place station can only be connected to Transition stations.
- A Transition station can only be connected from Place stations.

These constraints imply that any station is connectable to a Place station or from a Transition station.

Full compatibility between PN and QN stations has not been achieved in terms of certain strategies and performance indices. Since *BAS blocking* can easily cause deadlocks in PN models, PN stations are unblockable in the current implementation. The *utilisation* of PN stations is also not available at present, as an agreement on its definition has not been reached yet.

Import/Export of PNML models

PNML (Petri Net Markup Language³) is a proposal of an XML-based interchange format for Petri nets [23, 23]. ISO/IEC 15909 (Systems and software engineering—High-level Petri nets) is dedicated to standardising the PNML format. This standard consists of two parts: ISO/IEC 15909-1:2004+A1:2010 Part 1 formulates the concepts, definitions and graphical notation of Petri nets; ISO/IEC 15909-2:2011 Part 2 elaborates the specification of the transfer format on the basis of Part 1.

Four types of PNML models are defined by the standard, namely *core models*, *Place/Transition nets*, *symmetric nets* and *high-level Petri nets*. In the current implementation, JSIM only supports import/export of core models and Place/Transition nets. The former are used to convey the topologies of PN models, while the latter have been extended to represent GSPN models. Users can import/export a PNML model by simply opening/saving the model from/into a file suffixed with '.pnml'. The type of PNML model to be exported depends on the number of closed classes in the model, i.e., 0 for core models and

³URL: <http://www.pnml.org/>

1 for Place/Transition nets. However, exporting a PNML model with open classes or PN stations is not allowed.

As illustrated in Listings 1–5, tool specific parameters are applicable to the *Net*, *Place*, *Transition* and *Arc* elements of a PNML model. These parameters enable the model to cover the extra PN features provided by JSIM. An exported PNML model will contain a complete set of tool specific parameters for every element. As for an imported PNML model, this is not necessary and a default value will be set for each absent parameter.

Listing 1: Tool specific parameters for the Net element.

```
<toolspecific tool="JavaModellingTools-JSIM" version="1.0.1">
  <tokens>
    <!-- Data Type: String,
         Valid Values: all the valid strings,
         Default Value: "Token" -->
    <className>Token</className>
    <!-- Data Type: String,
         Valid Values: all the node names,
         Default Value: N/A -->
    <referenceNode>Place 1</referenceNode>
    <graphics>
      <!-- Data Type: Long(hex),
           Valid Values: [#00000000,#FFFFFFF],
           Default Value: #FF0000FF -->
      <color>#FF0000FF</color>
    </graphics>
  </tokens>
</toolspecific>
```

Listing 2: Tool specific parameters for the Place element.

```
<toolspecific tool="JavaModellingTools-JSIM" version="1.0.1">
  <!-- Data Type: Integer,
       Valid Values: {-1}U[1,Integer.MAX_VALUE](-1=infinity),
       Default Value: -1 -->
  <capacity>-1</capacity>
  <graphics>
    <!-- Data Type: Boolean,
         Valid Values: {false,true},
         Default Value: false -->
    <rotate>>false</rotate>
  </graphics>
</toolspecific>
```

Listing 3: Tool specific parameters for the Timed Transition element.

```
<toolspecific tool="JavaModellingTools-JSIM" version="1.0.1">
  <!-- Data Type: Integer,
       Valid Values: {-1}U[1,Integer.MAX_VALUE](-1=infinity),
       Default Value: -1 -->
  <numberOfServers>-1</numberOfServers>
  <!-- Data Type: String,
       Valid Values: {"Timed","Immediate"},
       Default Value: "Timed" -->
  <timingStrategy>Timed</timingStrategy>
  <!-- Data Type: Distribution,
       Valid Values: all the available distributions,
       Default Value: Exponential(1.0) -->
  <firingTimeDistribution>
    <distribution>
      <type>Exponential</type>
      <parameter>
        <name>lambda</name>
        <value>1.0</value>
      </parameter>
    </distribution>
  </firingTimeDistribution>
  <graphics>
    <!-- Data Type: Boolean,
         Valid Values: {false,true},
         Default Value: false -->
```

```

        <rotate>false</rotate>
    </graphics>
</toolspecific>

```

Listing 4: Tool specific parameters for the Immediate Transition element.

```

<toolspecific tool="JavaModellingTools-JSIM" version="1.0.1">
  <!-- Data Type: Integer,
        Valid Values: {-1}U[1,Integer.MAX_VALUE](-1=infinity),
        Default Value: -1 -->
  <numberOfServers>-1</numberOfServers>
  <!-- Data Type: String,
        Valid Values: {"Timed","Immediate"},
        Default Value: "Timed" -->
  <timingStrategy>Immediate</timingStrategy>
  <!-- Data Type: Integer,
        Valid Values: [0,Integer.MAX_VALUE],
        Default Value: 0 -->
  <firingPriority>0</firingPriority>
  <!-- Data Type: Double,
        Valid Values: (0.0,Double.MAX_VALUE],
        Default Value: 1.0 -->
  <firingWeight>1.0</firingWeight>
  <graphics>
    <!-- Data Type: Boolean,
          Valid Values: {false,true},
          Default Value: false -->
    <rotate>false</rotate>
  </graphics>
</toolspecific>

```

Listing 5: Tool specific parameters for the Arc element.

```

<toolspecific tool="JavaModellingTools-JSIM" version="1.0.1">
  <!-- Data Type: String,
        Valid Values: {"Normal","Inhibitor"},
        Default Value: "Normal" -->
  <type>Normal</type>
</toolspecific>

```

Appendix D. Validation of Apache Spark

The objective of this appendix is a) to validate the transformation patterns we have proposed and implemented using QVT, and b) to show the performance metrics that can be computed for analyzing the quality of the Spark applications. To this aim, we firstly execute the tool to transform the UML Spark-profiled model and automatically obtain the GSPN. Then, the simulation tool automatically analyses the GSPN and get results. Finally, we compare such estimated results to the results measured by deploying, in our cloud testbed, the corresponding Spark application.

In order to validate our approach, we have worked with the SparkPrimes example presented in this deliverable. The test suite is run with different job configurations. Mainly, we have varied a) the number of assigned cores to each job and b) the number of tasks per operation. The execution times associated to the initialization of a RDD (*SparkWorkloadEvent*→*sparkExtDelay*) and a Spark operation (*SparkOperation*→*hostDemand*) are required for the calculation of the rest of performance metrics. These values are estimated through the reading of Log4j files and the Spark monitoring platform.

The execution environment where the real Spark applications are launched consists of a Spark Master (coordinator) and a set of six workstations for running the computations. The workstations are virtual machines deployed on the Flexiant cloud [3]. All the workstations were characterized by virtual AMD CPUs (4x2.60GHz) with 4GBytes of RAM, a Gigabit ethernet and Ubuntu Linux (version 14.04) OS.

Concerning the performance analysis of the GSPN model, we have used the event-driven simulator of the GreatSPN tool [18] (confidence of 99% and accuracy of 3%). The performance metrics that can be analysed are: a) the response time of the Spark application; b) the throughput and c) the utilization of the hardware resources.

In the GSPN model, the response time (see pattern *P1*, Table 5) is the mean throughput of transition $t_{Activity2}$ divided by the initial marking of place $p_{Activity1}$. The throughput of the Spark application is equal to the mean throughput of transition $t_{Activity2}$.

The utilization of the device (see pattern *P12*, Table 7) is the mean number of tokens in the place p_{A2} divided by the initial marking of the place p_R . In particular, the utilization of the computational cores assigned to the Spark application is the mean number of tokens in the place p_{A2} divided by the initial marking of the place $p_{Activity2}$. The utilization of the computational cores assigned to the Spark application is proportional to the utilization of the whole device.

Table 8 compares the performance metrics predicted by the simulation of the GSPN and real performance metrics obtained by running the Spark applications in our Flexiant cluster. From the experiments, we get the following insight. The simulation of the Petri net with GreatSPN returns a good prediction of the response time when the modeled Spark application has a high number of tasks and assigned cores (e.g., more than 12 cores and 200 tasks). For the rest of configurations, the relative error is greater than 15%; although relative errors fewer than 30% are still in line with the expected accuracy in the performance prediction field [27].

Particular cases such as (6,200), (12,100), (18,200) and (24,100) have excessively high errors, ranging between 32.2% and 41.19%. These errors seem to be caused by the selected statistical distribution that simulates the mean execution time of the map/reduce tasks in the timed transition t_{A3} of the transformation *P3*. When the number of map/reduce tasks is high, the errors introduced by an incorrect statistical distribution are faded out. Otherwise, the errors become relevant when the number of map/reduce tasks is small.

By default, the temporal transitions of the Petri net follow an infinite server semantics; and the execution of tasks is modeled by an exponential distribution. The use of alternative probability distributions such as a Phase-type or Erlang may provide a better accuracy for the cases where the simulations give big deviations. The selection of the right probability distribution requires a deep and exhaustive analysis of the behavior of the tasks in the Spark cluster (i.e., the probability of arriving multiple tasks at time t , the mean execution and arrival times, and variances). The UML-to-Petri net transformation is currently being extended in order to support Erlang distributions and deterministic transitions.

In any case, the current version of our UML-to-Petri net transformation offers a good prediction of the application response time. An iterative refinement of the transformation process will provide even an increased prediction accuracy of the Petri net.

Table 8: Results of the SparkPrimes experiment

Number of Cores (\$n_{AssignedCores}\$)	Number of Tasks (\$n_{C1}\$)	Response Time (real, s)	Response Time (simulation, s)	Relative error (%)
6	100	49,82	59,96	20,35
6	200	85,61	54,27	36,60
6	400	78,16	75,38	3,55
6	800	141,14	139,12	1,44
12	100	27,99	39,52	41,19
12	200	43,27	51,58	19,21
12	400	55,50	57,40	3,42
12	800	104,63	105,70	1,02
18	100	36,51	36,02	1,35
18	200	45,22	59,79	32,20
18	400	56,17	59,98	6,79
18	800	99,97	102,90	2,94
24	100	30,21	40,96	35,56
24	200	46,69	43,35	7,14
24	400	58,21	63,77	9,54
24	800	106,07	113,09	6,62

Appendix E. GSPN

A GSPN is a Petri net with a stochastic time interpretation, therefore it is a modelling formalism suitable for performance analysis purposes. A GSPN model is a bipartite graph, consisting of two types of vertices: places and transitions.

Places are graphically depicted as circles and may contain tokens. A token distribution in the places of a GSPN, namely a marking, represents a state of the modelled system. The dynamic of the system is governed by the transition enabling and firing rules, where places represent pre- and post-conditions for transitions. In particular, the firing of a transition removes (adds) as many tokens from its input (output) places as the weights of the corresponding input (output) arcs. Transitions can be immediate, those that fire in zero time; or timed, those that fire after a delay which is sampled from a (negative) exponentially distributed random variable. Immediate transitions are graphically depicted as black thin bars while timed ones are depicted as white thick bars.

In our Spark performance model, places represent the intermediate steps of the processing. Transitions represent the execution of Spark tasks and fire when certain conditions are met (e.g., a number of tasks are created or accomplished) or the associated time delay has elapsed (e.g., the execution of a task by the cluster has finished). Besides, tokens represent either the Spark application workflow or the tasks executed by a Spark operation.