A Tool for Assessing Performance Requirements of Data-Intensive Applications

Abel Gómez[1], Christophe Joubert[2], and José Merseguer[1]

[1] Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain
{abel.gomez|jmerse}@unizar.es
[2] Prodevelop SL, Spain
cjoubert@prodevelop.es

Abstract. Big Data is becoming a core asset for present economy and businesses, and as such, Data-Intensive Applications (DIA) that use Big Data technologies are becoming crucial products in the software development market. However, quality assurance of such applications is still an open issue. The H2020 DICE project aims to define a quality-driven framework for developing DIA based on model-driven engineering (MDE) techniques. In this paper we present a key component of the DICE Framework, the *DICE Simulation Tool*. The tool is able to simulate the behavior of a DIA to assess its performance using a Petri net model. To showcase its capabilities we use the *Posidonia Operations* case study, a real-world scenario brought from one of our industrial partners. In addition to this paper, a video demonstrating the tool is available at http://tiny.cc/z1qzay.

1 Introduction

In recent years, the software development world has been witnessing an increasing complexity of systems and data that cloud-based infrastructures have made possible. With the broad availability of distributed clusters and programming models such as MapReduce, stream processing frameworks or NoSQL databases, the software development market expects to grow considerably for data-intensive cloud applications in the next years. Thus, there is now an urgent need for novel, highly productive, software engineering methodologies capable of dealing with software development challenges in such a new environment.

The DICE project [1] aims to define a quality-driven framework for developing data-intensive applications (DIA) that leverage Big Data technologies hosted in private or public clouds. Following the model-driven engineering (MDE) paradigm, applications are described using the Unified Modeling Language (UML), and specific DIA characteristics are annotated using a novel profile, the *DICE profile*. A set of simulation, analysis and optimization tools use DICE-profiled models to obtain high-quality applications. One of these tools is the so-called *Simulation Tool*, which allows evaluating quality properties of DIA, specifically, performance requirements.

Actas de las XXIV Jornadas de Concurrencia y Sistemas Distribuidos (JCSD 2016), pp. 159-169.
15-17 junio 2016, Granada, España. ISBN: 978-84-16478-90-3

The paper is organized as follows: Section 2 presents the *Simulation Tool* architecture with its internal data flows, while Section 3 presents the Posidonia Operations case study. In Section 4 we put the *Simulation Tool* in action: first we use the Posidonia Operations case study to describe the modeling foundations, and second, we show what the tool looks like from the users' point of view. Finally, Section 5 concludes the paper.

2 The DICE Simulation Tool

Figure 1 shows the simplified architecture of the *Simulation Tool* and its internal data flows.

The *DICE-IDE* [2] is an Eclipse-based [3] environment in which the different components are integrated. An *assessment process* starts by defining a set of DICE-Profiled UML models. For this stage, the *Papyrus UML* modeler is used [4]. *Papyrus UML* is one of the open source UML modeling tools that support the MARTE (Modeling and Analysis of Real-time Embedded Systems) profile [5], on which the DICE profile is based. To assess some performance requirement in a model, a user (the *QA Engineer*) should use the *Simulator GUI* to start a simulation. The *Simulator GUI* is an Eclipse component specifically designed to contribute a set of graphical interfaces to the DICE-IDE. These interfaces are tighly integrated within the DICE-IDE providing a transparent way for interacting with the underlying analysis tools. The *Simulation Configuration Component* is in charge of: (i) asking for the model to be simulated; and (ii) asking for any additional data required by the *Simulator*. When the user finishes the configuration of a simulation, the *Configuration Tool* passes two different files to the *Simulator*: the *DICE-profiled UML model* (i.e., the model to be analysed) and the *Configuration model*.

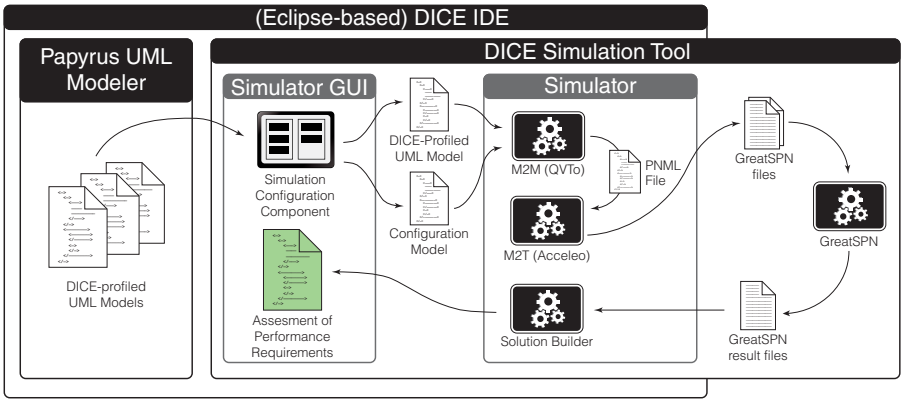

Fig. 1: High-level view of the tool architecture

The *Simulator* is an OSGi [6] component that runs in background. It has been specifically designed to orchestrate the interaction among the different tools that perform the actual analysis. The *Simulator* executes the following steps: (i) it transforms the UML model into a PNML[3] [7] file using a model-to-model (M2M) transformation tool; (ii) it converts the previous PNML file to a GreatSPN [8] specific file format using a model-to-text (M2T) transformation tool; (iii) it analyses the GreatSPN model using the GreatSPN tool; and (iv) it builds a tool-independent solution from the tool-specific file produced by *GreatSPN*.

To execute the M2M transformations we have selected the *Eclipse QVT Operational* [9] transformations engine. QVT [10] is the standard language proposed by the OMG [11] (an international standards consortium that also defined the UML and MARTE standards) to define M2M transformations.

To execute the M2T transformations we have selected *Acceleo* [12]. Starting from Acceleo 3, the language used to define an Acceleo transformation is an implementation of the MOFM2T standard [13], proposed by the OMG too. In this sense, we have selected Acceleo to make all our toolchain compliant to the OMG standards, from the definition of the initial (profiled) UML models to the 3rd party analysis tools (which inevitably use a proprietary format).

The GreatSPN [8] analysis tool is a complete framework for the modeling, analysis and simulation of Petri nets. This tool can leverage those classes of Petri nets needed by our simulation framework, i.e., Generalized Stochastic Petri Nets (GSPN) and their colored version, namely Stochastic Well-formed Nets (SWN).

Finally, the *tool-independent report* produced by the *Simulator* is presented in the *DICE-IDE* using a graphical component of the *Simulator GUI*. This component provides a comprehensive *Assesment of Performance Requirements* report in terms of the concepts defined in the initial profiled UML model.

3 The Posidonia Operations case study

Posidonia Operations [14] is a customizable *Integrated Port Operations Management* System that allows a port to optimize the operational maritime activities related to the vessel flow within the service area of the port, integrating all the involved stakeholders and all the relevant information systems.

The vessel becomes the centre of the system, and all the actions and data are linked to the vessels through an integrated operator console that centralizes all the significant information coming from external sources and systems. Examples of such external systems are radars, Automatic Identification Systems (AIS), vessel traffic services (VTS), meteorology services, communication systems, Port Management Systems (PMS), Port Community Systems (PCS), safety and security systems or cartography services, among others.

Posidonia Operations is designed to cover all the phases of a vessel: request, authorization, port approach, port enter, berthing and unberthing, berth change, anchoring and port leaving. It also fulfils port operations, including

[3] PNML is an ISO to serialize and interchange Petri net specifications. The acronym stands for *Petri Net Markup Language*

berth planning, coordination and register of pilots, tugs and moorers activities, vessel supplies and bunkering, wastes and disposal, incidents, repairs, port inner traffic, etc.

A real time analysis engine based on spatial information can be configured to automatize relevant operational events like anchoring, berthing/unberthing, pilots and tugs operations, bunkering, enter and exit of areas like port service area, waypoints or inner harbour, port exit with pending requested anchoring, etc.

3.1 Architecture

Posidonia Operations is a DIA implemented in Java. It processes streamed data from *AIS receivers* [15, 16]. An AIS receiver is a system that gets vessels position and meta-data in real time. The encoding protocol of an AIS sentence can be found in [17].

To get data from an AIS network, a TCP connection to the port AIS receiver is used. Once an AIS stream is parsed, it is published to a message queue for further processing: analysis, complex event processing, data integration, visualization, etc. An AIS message is a binary encoded sentence that can be decoded into key-value objects. Its size is usually under 100 bytes. Velocity and volume of data depends on the number of parallel AIS streams to be processed.

The core components of interest for performance analysis are: (i) a streaming processor, or *AIS parser*, that collects the data from the AIS receiver and parses it; an AIS parser consists of four sub-components: the *Parser*, the *StationManager*, the *StationProcessors* and the *ParsingTask*; (ii) a message queue for subscribing/publishing data such as AIS messages or detected events; and (iii) a complex event processing engine that subscribes to AIS messages and correlates them in time and space to identify events.

The AIS parser's behaviour is modelled by the UML activity diagram of Figure 2, where the stream of messages from the AIS receiver is initially parsed by the *Parser* sub-component. Then, messages are adapted by the *Station Processor* to convert them into business objects (*AIS Sentences*) and are successively post-processed by the *Parsing Task* to be published in the Rabbit message queue. The adaptation and post-processing steps are controlled by the *Station Manager*.

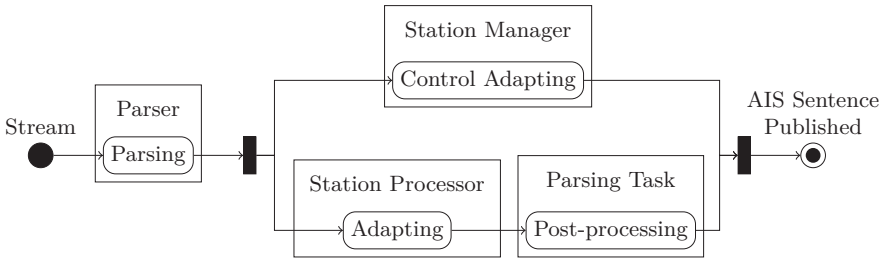

Fig. 2: AIS parser scenario from Posidonia Operations

Posidonia Operations is a commercial product already deployed and operated in several port authorities. Being a product already in production, performance has to be guaranteed under different velocity and volume of data to be processed. For a single area of a port, a velocity of about hundred AIS messages per second with a volume of about five million messages per day can be observed. These numbers may vary and can be multiplied by the number of port areas managed by the product for a given *Port Authority*. For example, several instances of the complex event processing engine would be needed, one for each port area. In this case, one of the challenges is related to the scalability of the product in terms of data processing, storage and analysis.

4 The DICE Simulation Tool in action

This section illustrates the DICE capabilities for modeling DIA. First, we illustrate how we model the DIA using UML and the aforementioned DICE profile. This model will provide the basis to carry out the performance assessment. Second, we describe what the DICE Simulation Tool looks like when performing the analysis of the DIA of interest.

4.1 Modeling background

Performance evaluation is traditionally carried out using scenarios, i.e., typical system paths of usage that specify the system behavior of the DIA. With UML, we can specify a scenario by using behavioral diagrams, such as sequence or activity ones. In particular, the latter are directed graphs that express causal dependencies between computation steps and/or data. As a running example, we will use the *activity diagram* (AD) of the AIS Parser in Figure 2.

In order to get a formal model suitable for performance analysis, we need to enrich the AD with workload characterization and timing specification – such as the durations of computation steps. To that end, we apply the DICE profile, that enables the designer to specify performance characteristics through UML extensions, i.e., stereotypes and tagged values. For quality assessment, the DICE profile indeed relies on two already existing UML profiles, namely the standard MARTE profile (*Modeling and Analysis of Real-time and Embedded Systems*) [5] and the DAM profile (*Dependability Analysis and Modeling*) [18]. The MARTE profile will enable DICE to assess performance, while the DAM profile is its counterpart for enabling dependability assessment.

Figure 3 shows the AD of the AIS Parser enriched with extensions imported by the DICE profile from MARTE. In particular, the initial node *Stream* is stereotyped with *GaWorkloadEvent* to specify the open workload, i.e., the mean arrival rate of messages. Actions are stereotyped using *GaStep* and a mean duration (i.e., the *hostDemand* tagged value) is associated to each one.

The AD is complemented with a *deployment diagram* (DD), that models the mapping of the logical resources onto processing nodes: Figure 4 shows the DD of the running example, where the *AIS Parser* is stereotyped *PaLogicalResource* to

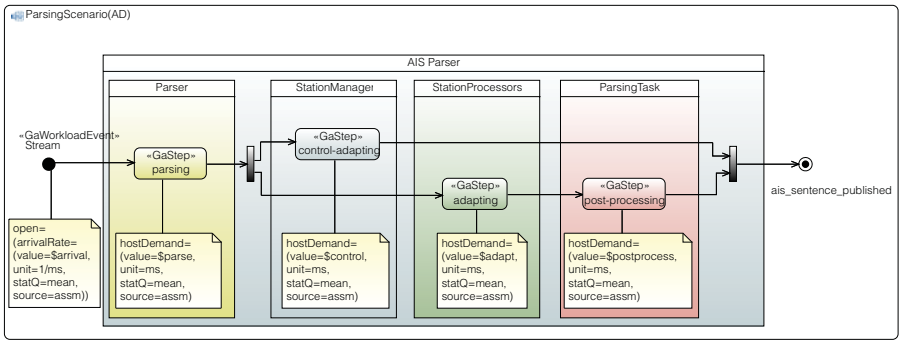

Fig. 3: AIS parser scenario with performance stereotypes

specify the number of concurrent threads (*poolSize* tagged value). All the tagged values are input parameters, i.e., variables prefixed by the dollar symbol. The use of variables is a feature, provided by the MARTE profile, that the DICE profile heavily exploits. As we will see in Section 4.2, such variables can be instantiated at a later stage.

From the AD of Figure 3, a Generalized Stochastic Petri Net (GSPN) model can be obtained by following the transformation approach proposed in previous work [19]. Based on this proposal, the *Simulation Tool* produces the GSPN of Figure 5. The transformation also considers the (logical) resource restriction from the DD of Figure 4. In particular, the initial and final nodes of the AD are mapped to GSPN transitions, i.e., t_1 (timed) and t_9 (immediate) respectively. Each *GaStep* action of the AD corresponds to a timed transition in the GSPN model, while the fork and join nodes are translated to immediate transitions, i.e.,

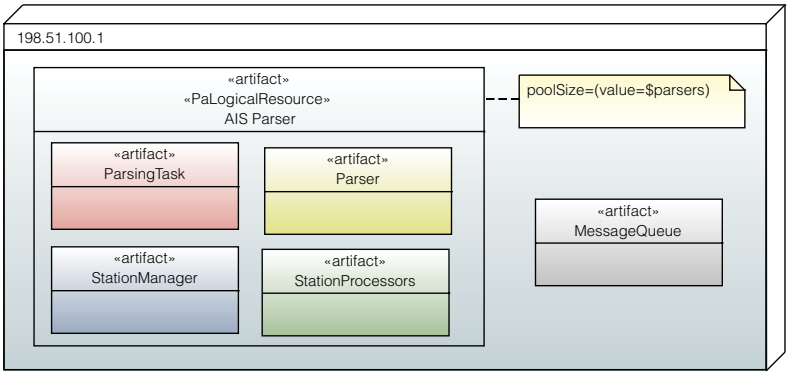

Fig. 4: Stereotyped Posidonia Operations deployment diagram

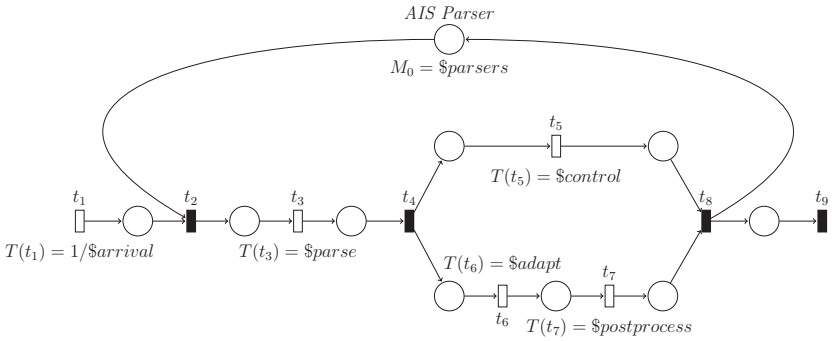

Fig. 5: Generated Petri net

t_4 and t_8 respectively. The firing times of the timed transitions are exponentially distributed random variables, where the mean parameter $T(t_i)$ is derived from the tagged value associated to the mapped AD initial node or action. Finally, the *AIS Parser* in the DD is mapped onto the corresponding GSPN place, where the initial marking (M_0) is set to the *poolSize* tagged value associated to the logical resource.

4.2 Using the DICE Simulation Tool

This section shows what the *Simulation Tool* looks like from the users' point of view. UML models, as already mentioned in Section 2, are defined using the *Papyrus UML* modeling tool. Figure 6, which depicts the scenario for the *AIS parser* activity diagram, shows part of the Papyrus modeling perspective. The *Model Editor*, shown at the top, is used to build models by dragging and dropping UML elements into the editor canvas. The *Properties* view, shown at the bottom, is complementary to the *Model Editor* and is used to apply profiles, stereotypes and to specify tagged values.

As it can be seen in the figure, the selected element in the editor is the initial node, and thus, the *Properties* view shows its properties. In particular, the bottom view shows that the node has the *GaWorkloadEvent* stereotype, and the value of the *pattern* tagged value (see bottom right of Figure 6) is `(open=(arrivalRate=(value=$arrival, unit=1/ms, statQ=mean, source=assm)))`[4]. It is noteworthy the use of the variable `$arrival` in this expression to specify the actual value of *arrivalRate*.

Once the modeling stage is complete and the *QA Engineer* has introduced all the performance information needed, he/she can launch an assessment process using *Simulation Configuration UI* shown in in Fig. 7.

[4] *Tagged values* are specified in Papyrus-MARTE using the so-called *Value Specification Language*. Details on this language can be found in the MARTE Standard [5].

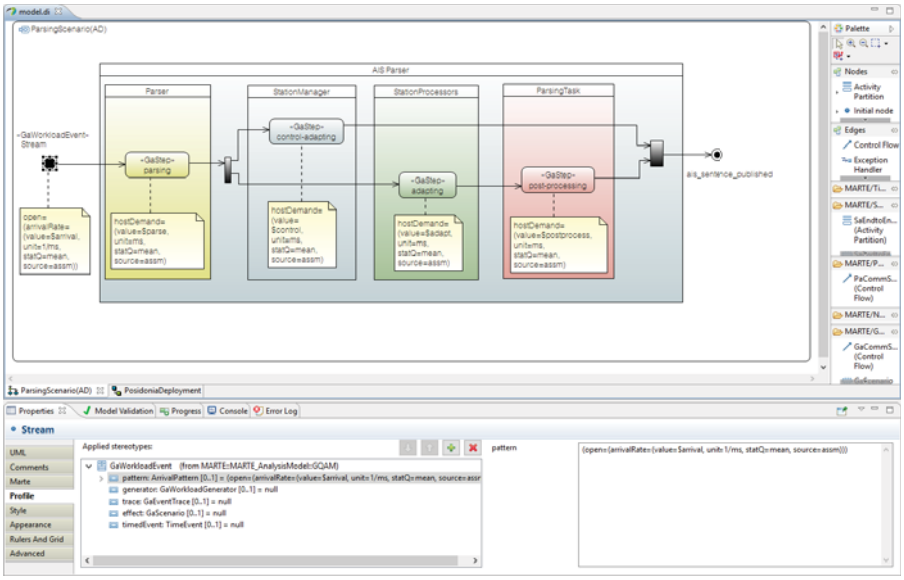

Fig. 6: Papyrus model editor with the *AIS parser* activity diagram

The topmost configurable element visible in this UI is the *Model* to be analysed. It is the basic input information, and when an input model is selected, the *Variables* table is populated with the appropriate entries. This table allows instantiating the values of the variables found in the tagged values of the applied stereotypes. Once the simulation has been fully configured, it can be launched by clicking *Run*. From this point, all the subsequent steps are automatically executed until the results are obtained from the underlying analysis tool (i.e., *GreatSPN*).

Figure 8 shows the *Simulation properties* dialog within the *Debug* perspective. The *Debug* perspective allows controlling the simulation process (e.g., tracking its state or killing the process), while the *Simulation properties* window shows simulation-related information such as identifier, execution time, and – once the simulation has finished – the analysis results, e.g., the throughputs of the computation steps. These results are the basic information used to build the *Assesment of Performance Requirements* report in terms of the concepts defined in the initial UML model.

5 Conclusions

In this document, we have presented the *Simulation Tool* of the *DICE* Framework. The tool is currently able to provide an initial assessment of performance requirements of a DIA from an initial UML model. Using the POSIDONIA Op-

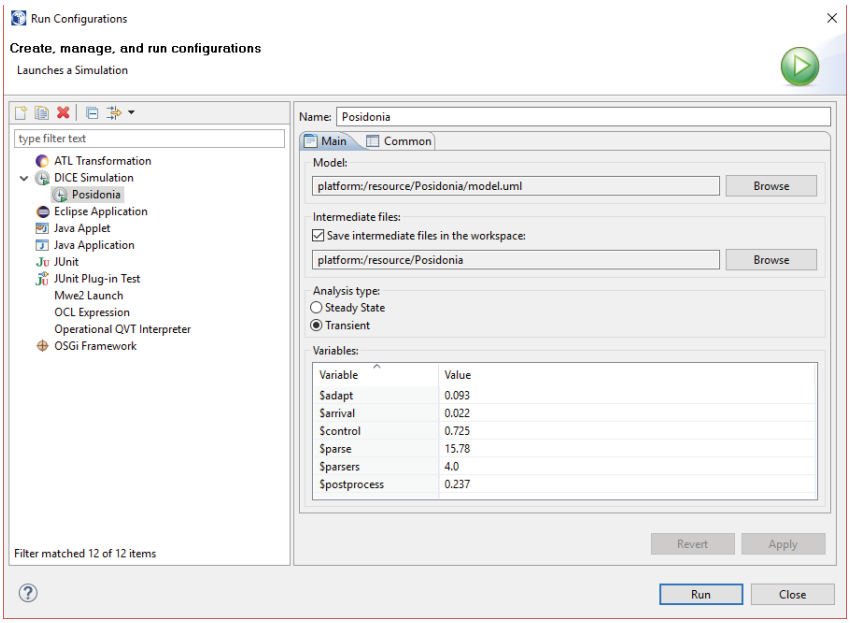

Fig. 7: Simulation Configuration UI from the *DICE Simulation tool*

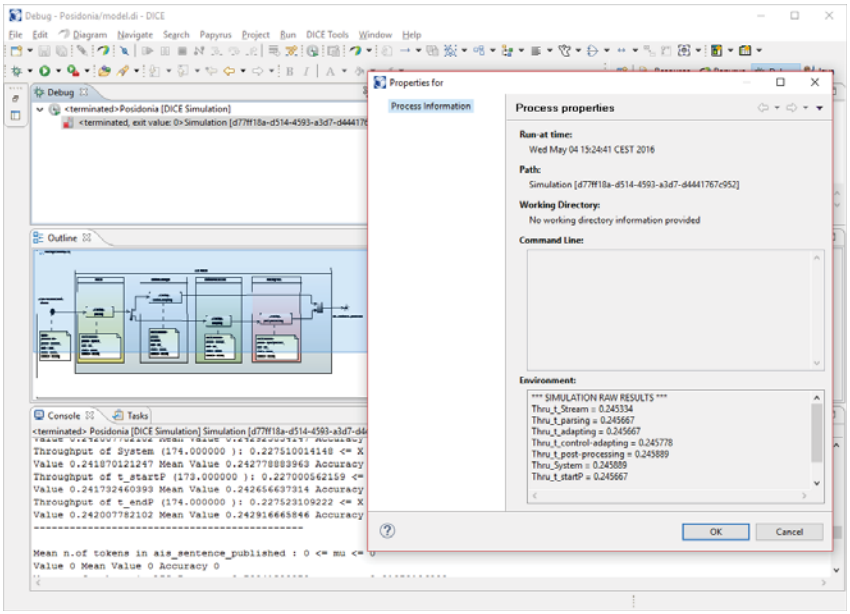

Fig. 8: Results UI from the *DICE Simulation tool*

erations case study – a real-world scenario brought from one of our industrial partners – we have illustrated the Simulation Tool capabilities. At its current state, the prototype provides a user-friendly interface and covers all the steps of the DICE simulation workflow with full integration within the *DICE-IDE*. The *DICE Simulation Tool* has been released as an open source tool in the project website [20], and can be seen in action in `http://tiny.cc/z1qzay`.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation framework programme under grant agreement No. 644869 (DICE), the Spanish Government (*Ministerio de Economía y Competitividad*) under project No. TIN2013-46238-C4-1-R and the Aragonese Goverment Ref. T27 – DIStributed COmputation (DISCO) research group. We also thank Marc Gil for his valuable contribution in the integration of the *DICE Simulation Tool* within the DICE-IDE.

References

1. Casale, G., et al.: DICE: Quality-driven Development of Data-intensive Cloud Applications. In: Proceedings of the Seventh International Workshop on Modeling in Software Engineering. pp. 78–83. IEEE Press, NJ, USA (2015)
2. The DICE Consortium: DICE Platform (2016), URL: `https://github.com/dice-project/DICE-Platform`
3. The Eclipse Foundation: Website (2016), URL: `http://www.eclipse.org/`
4. The Eclipse Foundation: A slide-ware tutorial on Papyrus usage for starters (2010), URL: `https://eclipse.org/papyrus/usersTutorials/resources/TutorialOnPapyrusUSE_d20101001.pdf`
5. OMG: UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1 (Juny 2011)
6. McAffer, J., VanderLei, P., Archer, S.: OSGi and Equinox: Creating Highly Modular Java Systems. Eclipse series, Addison-Wesley (2009), `http://books.google.com/books?id=RjX8PQAACAAJ`
7. ISO: Systems and software engineering – High-level Petri nets – Part 2: Transfer format. ISO/IEC 15909-2:2011, Geneva, Switzerland (2008)
8. Dipartimento di informatica, Università di Torino: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets (2016), URL: `http://www.di.unito.it/~greatspn/index.html`
9. The Eclipse Foundation: Eclipse QVT Operational (2016), URL: `http://www.eclipse.org/mmt/qvto`
10. 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/`
11. OMG: Object Management Group website (2016), URL: `http://www.omg.org/`
12. The Eclipse Foundation & Obeo: Acceleo (2016), URL: `https://eclipse.org/acceleo/`
13. OMG: MOF Model to Text Transformation Language (MOFM2T), 1.0 (Jan 2008), URL: `http://www.omg.org/spec/MOFM2T/1.0/`

14. Joubert, C., Montesinos, M., Sanz, J.: A comprehensive port operations management system. ERCIM News 2014(97) (2014), http://ercim-news.ercim.eu/en97/special/a-comprehensive-port-operations-management-system
15. Automatic Identification System - Encoding Guide (2012), http://www.uscg.mil/hq/cg5/TVNCOE/Documents/links/AIS.EncodingGuide.pdf, accessed 04/29/2016.
16. Installation and Quick Reference Guide - SRL-200/G AIS Receiver, http://www.comarsystems.com/brochures/Installation%20SLR200G%20Guide%20.pdf, accessed 04/29/2016.
17. AIVDM/AIVDO protocol decoding (2015), http://catb.org/gpsd/AIVDM.html, accessed 04/29/2016.
18. Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with uml. ACM Comput. Surv. 45(1), 1–48 (Dec 2012)
19. López-Grao, J.P., Merseguer, J., Campos, J.: From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. SIGSOFT Softw. Eng. Notes 29(1), 25–36 (Jan 2004)
20. The DICE Consortium: DICE Simulation Repository (2016), URL: https://github.com/dice-project/DICE-Simulation