Noname manuscript No.
(will be inserted by the editor)

A UML Profile for the Design, Quality Assessment and Deployment of Data-intensive Applications

D. Perez-Palacin · J. Merseguer · J.I. Requeno · M. Guerriero · E. Di Nitto · D.A. Tamburri

Received: date / Accepted: date

Abstract Big Data or *Data-Intensive* applications (DIAs) seek to mine, manipulate, extract or otherwise exploit the potential intelligence hidden behind Big Data. However, several practitioner surveys remark that DIAs potential is still untapped because of very difficult and costly design, quality assessment, and continuous refinement. To address the above shortcoming, we propose the use of a UML domain-specific modelling language or *profile* specifically tailored to support the design, assessment, and continuous deployment of DIAs. This article illustrates our DIA-specific profile and outlines its usage in the context of DIA performance engineering and deployment. For DIA performance engineering, we rely on the Apache Hadoop technology, while for DIA deployment, we leverage the TOSCA language. We conclude that the proposed profile offers a powerful language for data-intensive software and systems modelling, quality evaluation and automated deployment of DIAs on private or public clouds.

Keywords UML · Profile · Data Intensive Applications · Software Design · Big Data · Performance Assessment · Model-Driven Deployment · Apache Hadoop · TOSCA language

D. Perez-Palacin
Department of Computer Science, Linnaeus University (Sweden)
E-mail: diego.perez@lnu.se

J. Merseguer and J.I. Requeno
Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza (Spain)
E-mail: {jmerse, nrequeno}@unizar.es

M. Guerriero, D.A. Tamburri · E. Di Nitto
Dipartimento di Elettronica, Informazione e Bioingegnieria, Politecnico di Milano (Italy)
E-mail: {michele.guerriero, elisabetta.dinitto, damianandrew.tamburri}@polimi.it

1 Introduction

The development of data-intensive applications [11] is a branch of data-intensive computing, an emerging computing paradigm connected to data-intensive science [5], which is recognised as a fourth scientific paradigm combining the previous three, namely empirical science, theoretical science, and computational science. Data-intensive applications (DIAs) leverage tools and technologies that handle Big Data. However, a research at Capgemini [15] shows that only 13% of organizations have achieved full-scale production for their DIA implementations on Big Data. The reason for this is mainly related to the technical difficulty of developing effective DIAs, able to actually cope with the volume, velocity and variety of the data to be handled. Hence, there is a need for creating languages, methodologies and tools that help increasing productivity in the development of DIAs by assisting and guiding developers in the adoption and usage of the many frameworks and architectural concepts introduced by this new field.

The aim of this paper is to propose a *domain-specific modeling language* (DSML) to assist DIA developers in various aspects of the design and the assessment of DIAs. Our DSML is defined as a UML profile [38], in agreement with Selic who in [51] states that profiles can be used to define expressive DSMLs.

The objective of the proposed UML profile is three-fold. First, it provides guidance for an initial design of the DIA architecture by identifying its main elements and the roles that they play. Second, the profile enables the quantitative assessment of the DIA, with particular focus on performance and reliability. Third, the profile enables the automated deployment of the DIA in public or private clouds.

Inspired by the OMG Model-Driven Architecture [65], the profile is then conceived at three abstraction levels, that correspond to our view of how the development of the DIA should evolve for addressing our three objectives. The first level is called DPIM (DIA Platform Independent Model) and allows designers to define the main architecture of a DIA thus addressing the aforementioned first objective of the profile. The second level, which we call DTSM (DIA Technology Specific Model), includes the specifics of the technology used to develop the DIA, e.g., Apache Hadoop or Apache Storm. Quality assessment, which is our second objective, is carried out at DTSM level since performance and reliability are greatly impacted by the peculiarities of the underlying platforms. Finally, the third level, which we call DDSM (DIA Deployment Specific Model), allows the automatic deployment of the DIA and the frameworks it exploits in the specific cloud environments that users have selected. Table 1 provides a summary of the mentioned abstraction levels and of their goals.

Our profile enhances the state of the art on DIA development and offers the following benefits.

- *Single language.* Being a UML profile, our DIA DSML encompasses the three abstraction levels using a single language. Hence, the DIA profile can

Acronym	Description	Goal
DPIM	DIA Platform Independent Model	Architecture description
DTSM	DIA Technology Specific Model	Quality assessment
DDSM	DIA Deployment Specific Model	Cloud deployment

Table 1 DIA profile abstraction levels

be initially used by DIA developers to better understand the architecture of the application. Then, in a continuous and iterative manner and following the three abstraction levels, the same software models are subsequently reused, i.e., for design, assessment and deployment.

- *The profile disengages developers from knowing details of quality assessment, i.e., performance and reliability assessment.* The profile pinpoints the UML model-elements where quality-related information needs to be specified, e.g., duration of MapReduce operations or definition of performance metrics. At this regard, it promotes quality assessment of the DIA during the design of its functionalities. As an example, the simulation tool [61] uses the DIA profile and automatically computes quality metrics. Such tool has been used, in Section 8, for validating the profile from the perspective of its ability to support performance and reliability assessment.
- *The profile disengages developers from knowing details on the complex tasks for continuously deploying the DIA.* In fact, the DIA profile defines the essential layer required for simple UML diagrams to be turned into fully deployable cloud infrastructure blueprints. This is achieved by developing model-to-model (M2M) and model-to-text (M2T) transformations, that transform DDSM level diagrams into deployment scripts written in TOSCA[1]. As an example, the DICER [64] tool implements this deployability feature, having as input, models annotated with the DIA profile. The DICER tool has been used, in Section 9, for assessing the ability of the profile to actually support TOSCA-based deployment.

The rest of the paper is organised as follows. Section 2 motivates and defines the approach followed for defining the profile. Sections 3, 4 and 5 address the definition of the profile, one section per abstraction level, DPIM, DTSM and DDSM, respectively. Section 6 summarises the way the profile can be used in practice and experiments that help to validate the profile. Section 7 introduces the goals of the evaluation. Sections 8 and 9 present the actual profile validation from the point of view of performance engineering and automated deployment. Section 10 explores the related work. Section 11 concludes the paper.

[1] TOSCA is a language to specify deployable blueprints in line with the emerging Infrastructure-as-Code (IasC) paradigm [14].

2 Motivations and Approach

2.1 DIAs and DIA frameworks

A DIA is a software application acquiring data from some *data sources*, manipulating such data and producing some output into some kind of *data sink*. The complexity of such applications is in the potentially high number of data sources from which to acquire data, the potentially low quality of such data, the typically high frequency with which they are generated, the complexity of the computation to be executed, the characteristics of data sinks and the presence of time constraints for the production of the manipulated data. Addressing such aspects all together is certainly challenging and requires very experienced teams able to design and develop the system in an appropriate way, assess the quality of the defined solution, which is, typically, highly distributed and parallelized, fine tune it, deploy it on the appropriate resources and continuously monitor and improve it.

Fortunately, thanks to the large interest around DIAs, various frameworks and architectural models have been developed and provide support to the architectural definition of the DIA and to its implementation, see for instance [57, 58, 55, 54]. However, still there are many aspects that require attention and that are not fully addressed by the state of the art (see Section 10 for details).

First of all, there is no modeling language that today specifically supports the design of DIAs. Of course, it is possible to use UML, possibly in combination with a formalized set of stereotypes, such as the ones offered by MARTE[2] [41, 52] and DAM[3] [7, 6]. However, this does not allow to express in a simple and direct way particular DIA concepts, nor it accounts for the specific characteristics of current big data frameworks as execution environments for DIAs.

Second, DIAs must be highly performant, scalable and reliable to ensure they can process potentially very large quantities of data. As such, we see the need for an integrated modeling language and toolset that supports developers in modeling and assessing DIA quality of service by exploring the effects of the adoption of different DIA technologies.

Third, deployment of DIAs based on complex distributed frameworks is complex per se. Also, different deployment solutions have different impacts on the non-functional characteristics featured by the DIA. As such, the combination of a modeling and code generation approach to support users in this phase would help.

2.2 Approach overview

In the context defined above, the aim of our work is to give to DIA development teams a design language and a toolset that allows them to accomplish the

[2] Modeling and Analysis of Real-Time Embedded Systems
[3] Dependability Analysis and Modeling

activities highlighted in Figure 1. More specifically, thanks to the UML profile-based language we propose, our user will be able to define, at the conceptual level we call DPIM, the high level structure of a DIA in terms of data sources and sinks and computational components. He/she will be able at this point to run a preliminary QoS analysis to start identifying potential bottlenecks and critical components for the DIA. After refining and reiterating on these activities as needed, the user move to the next level of abstraction, the DTSM, and will identify the specific technologies to be adopted for the DIA. For instance, he/she will select one of the technologies for data sources, say Apache Spark [57], a system to support parallel execution of computations in batch and streaming mode, as the framework for executing the computations, and will map the high level components to these elements. At this point he/she will be able to perform a technology-specific QoS analysis that will either convince him/her of the advantages of the envisaged solution or solicit a change in the defined components. After having finalized this step, he/she will exploit the features of the next conceptual level, the DDSM, and will be able to create the deployment model for the resulting system and generate automatically the corresponding blueprint ready for deploying all needed technologies and application level components.

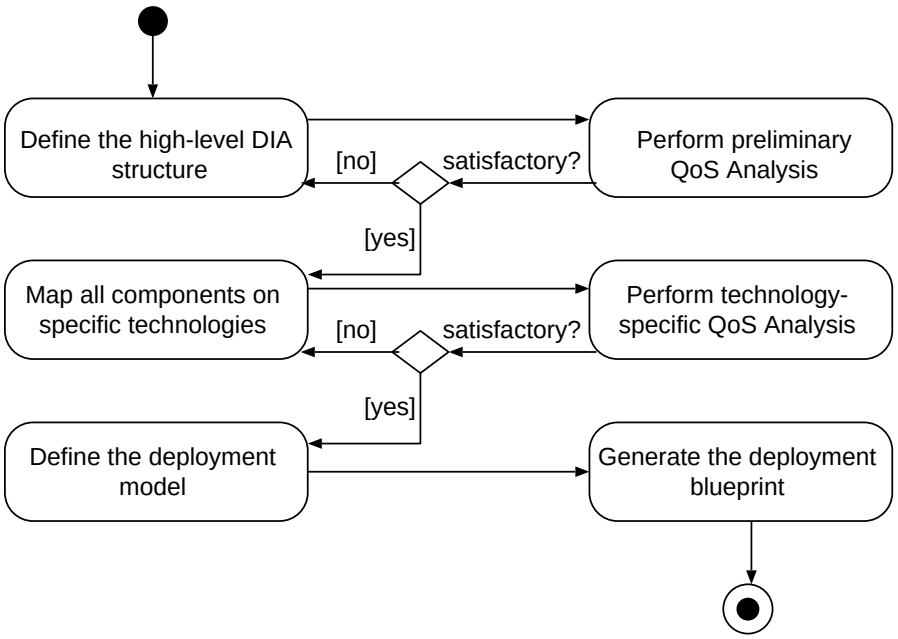

Fig. 1 Overview of the proposed approach

The proposed approach is then based on the definition of a proper DIA profile, which offers to its users all linguistic structures to define the relevant in-

formation associated to a DIA and its underlying technologies, complemented with an analysis tool and a deployment blueprint generator.

This paper develops and significantly extends the work presented in [23], where we described the preliminary version of the profile. In [44] we have modeled, still at the DPIM level, an industrial application focusing on tax fraud detection. In this case, we had to model the main part of the application as a black box as we did not have, at that time, the proper linguistic tools to describe its internals. This experience has allowed us to identify some of the initial requirements for the profile. A fullfledged example of DTSM modeling and related analysis is, instead, presented in [47,29] where we have shown how the profile is suitable for capturing the characteristics of Apache Storm [58] applications. Finally, [22] focuses on DDSM modeling and on the generation of the corresponding blueprint. All in all, main contributions of this paper are the presentation of the complete and consolidated DIA profile that takes into account the main characteristics of DIAs and of the most used technologies adopted for their development, as well as the definition of the methodological steps followed to develop such profile and also methodological step for practitioners to develop models at DTSM and DDSM levels. We also show, through examples, how it can be used within the context of the workflow of Figure 1.

2.3 Approach for the design and validation of the DIA profile

Our goal is to produce a technically correct high-quality UML profile. Hence, we followed design principles and guidelines proposed by Selic [51] and Lagarde [38], which define the state of the art in UML profile construction. We then adopted a systematic and iterative approach applied in three main steps: i) definition of a DIA domain model, ii) design of the DIA profile and iii) DIA profile validation. Figure 2 sketches this workflow.

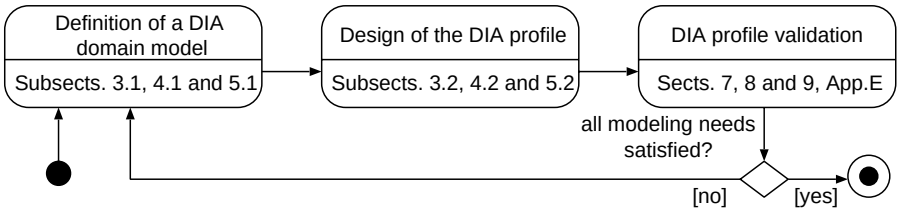

Fig. 2 Workflow for defining the DIA profile

Definition of a DIA domain model. We started studying the concepts behind the three aforementioned abstraction levels summarized in Table 1. The idea was to initially create an independent domain model for each level, that should exactly and only represent those needs for addressing the goal of the corresponding level.

- For the DPIM level, we revised the work in [13] for representing architecture concepts and the works surveyed by [11] for the concepts in DIA. Subsection 3.1 reports our work in this sub-step.
- For the DTSM level, we studied and then conceptualized five of the most well-known DIA technologies: Apache Spark [57], Apache Storm [58], Apache Hadoop [55], Apache Cassandra [54] and Apache Tez [59]. In several refinement stages we isolated the core concepts reused by many of these technologies. Being quality assessment the main target of this level, we envisioned the opportunity of reusing other UML profiles already defined in the literature for this purpose. In particular, we leveraged on MARTE and DAM UML profiles, well established in literature and practice. Subsection 4.1 reports our work in this sub-step.
- For the DDSM level, we relied on our previous experience developing the MODACloudsML [20] language for cloud infrastructure deployment. This language provided us with the fundamental concepts for cloud infrastructure provisioning required in this level of abstraction. In addition to the details offered by MODACloudsML, we studied the particular needs on deployment for each of the aforementioned technologies supported by the DIA profile. Finally, we merged all these concepts to define the DDSM domain model. Subsection 5.1 reports our work in this sub-step.

Iteratively, while we were constructing and reviewing the domain models, we advanced on the understanding of their relationships. Figure 3 shows that we achieved a simple integration result, where DTSM is supported by the definition of concepts in the DPIM, while the DDSM elaborates over the definitions on the two previous ones.

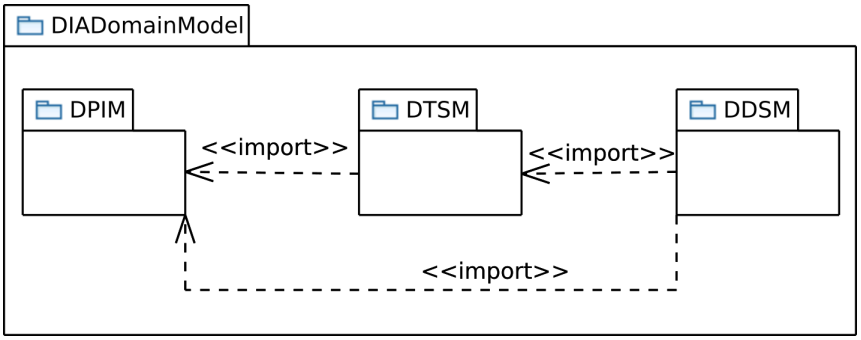

Fig. 3 DIA domain model relationships

Design of the DIA profile. Once a domain model has been defined, then the process of mapping it to a UML profile can be carried out, this step is known as the design of the profile. A UML profile is made of a *library*[4], that defines

[4] The DIA *library* is described in the technical Appendix B.

data types, and a set of UML *extensions*. The UML *extensions* provide the domain expert with a set of stereotypes (and their corresponding tags) to be applied at model specification level. Hence, the usage of the stereotypes will represent the DIA view in a concrete UML model.

For designing each of the three profiles, DPIM, DTSM and DDSM, we have followed suggestions and patterns proposed in [38]. This way we have identified a small yet sufficient set of stereotypes to be actually used in practical modeling situations. This is an iterative process that comprises three steps.

In a first step, each class in the domain model is candidate for becoming a stereotype. Classes are examined, together with its attributes, associations and constraints, and are characterized as abstract, firm, uncertain or parametric. This criterion aims to clearly characterize the role of each concept in the domain model. Abstract classes refer to accessory concepts then they do not map into stereotypes. Firm classes can be either transformed into stereotypes or datatypes. Uncertain classes sort indeterminate concepts, and parametric classes categorize concepts that can change depending on the problem domain. In a second step, for the DPIM and DTSM levels, we also need to decide whether each stereotype should inherit from some MARTE or DAM stereotype or not. Such inheritance is useful to enable the possibility of exploiting and extending the analysis and simulation tools specifically developed for the MARTE and DAM profiles. In a third step, for each stereotype, we need to search in the UML standard for suitable extensions, i.e., the actual UML meta-classes to be extended by the stereotype, as suggested in [51]. For those stereotypes, that in the second step were decided to be inherited from MARTE or DAM, the UML meta-classes that extend the parents also will extend the children.

DIA profile validation. The final step was to conduct an exhaustive validation of each of the profiles. Validation of initial versions of the DPIM, DTSM and DDSM profiles has been performed by modeling the architecture of two industrial case studies as it was explained at the end of Section 2.2. A complete validation is reported in this paper in Sections 8 and 9 with reference to the WikiStats [69] problem example. This is a DIA computing simple statistics on contents published on Wikimedia. Finally, a validation in terms of the usability of the profile is carried out in Appendix E.

3 DIA profile definition: DPIM

3.1 DPIM domain model

The domain model for this level of abstraction stems from several sources. Firstly, we carefully reviewed the abstract concepts required for modeling, at the highest level of abstraction possible, two widely known middleware for data intensive computing, namely Apache Hadoop [55] and Apache Storm [58].

Secondly, we reviewed and systematically compared our solution with domain models of data-intensive middleware available from related work [10, 50]. Thirdly, we enhanced such model by reviewing literature, surveyed by [11], specifically dedicated to modeling concepts of DIA. Fourthly, we augmented the model and fitted it with necessary architecture properties following the typical architecture description and definitions in [13]. Consequently, we compounded our first solution draft gathering missing constructs and concepts from several sources of the related work and from our experience with particular DIA technologies.

As a result of applying the above process iteratively, we obtained a quasi-final version of the DPIM domain model. Obviously, the validation of the profile has allowed us to discover a few modeling issues that we have fixed in an iterative process, as indicated in Figure 2. Also, our experience with other new technologies, e.g., Apache Spark [57] or Apache Tez [59], has contributed to a small improvement of the model. Figure 4 depicts the final version of the DPIM domain model. The DPIM provides the following constructs:

- *DIA* - this simply represents a generic DIA, which is essentially a set of *DIAElements*. Since the DPIM provides the highest level of abstraction, at this level it is possible to specify the existence of multiple DIAs.
- *DIAElement* - these are generic component nodes which represent the architecture elements of the DIA, to be further specified into simple or composite nodes.
- *ComputationNode* - it is basically responsible for carrying out computational tasks like map, or reduce in MapReduce. An important attribute of ComputationNode is procType. It indicates the processing type, i.e., batch, stream or interactive. Moreover, the ComputationNode is associated to CommunicationChannel to represent communication channels, e.g., of publication and subscription roles in the application;
- *SourceNode* - it provides data for processing, then it is the entry point of data into the application. In other words, the SourceNode represents the source of data which are coming into the application in order to being processed. The attribute sourceType further specifies the characteristics of source. The ultimate goal of a DIA is to process the data that have high volume and velocity;
- *StorageNode* - it is the third key element in the DPIM domain model. As its name may suggest the StorageNode represents the element which is responsible for storing the data, either for long term or not. The concept of StorageNode in DPIM mainly corresponds to the database, in some case it could be filesystem also;
- *CommunicationChannel* - these represent the connectors (e.g., interfaces, specific compute nodes, etc.) or connecting middleware (e.g., specific DIA for communication and brokering such as Apache Kafka [56]) that DIA use for message queuing and transferring; explicit modelling of these elements is needed since they may weigh heavily on the architectural properties and characteristics for the DIA. The CommunicationChannel is then a

representation of governance and data integration, which mainly includes the technologies responsible for transferring the data, like message broker systems;
- *DataSpecification* - these represent the explicit data models (if any) being manipulated across the DIA by any of its components; a data specification is required to express more advanced and data-specific architecture properties such as privacy-by-design [39].

The DPIM domain model supports the modelling and analysis of component-based architecture models for DIAs. To further evaluate the completeness of this model, we exploited Formal Concept Analysis (FCA) [70] on the original Apache Hadoop and Apache Storm middleware. The exhaustive explanation of how FCA works and how it can be used for domain analysis is beyond the scope of this paper but, more specifically, the FCA-based analysis entails systematically isolating and reporting all concepts and relations identified for a domain, using appropriate domain descriptions. Concept analysis takes place by labelling the source files (in our case over 50 pages of Apache Storm and Apache Hadoop technical documentation) and reporting the results of the labelling exercise in a MxN sized table, where M is the set of concepts and N is the set of attributes found. To use the afore-mentioned analysis for evaluation of our domain model we compared the FCA tables resulting from the analysis recapped previously, with our final resulting DPIM domain model. A satisfactory mapping reports a 1 to 1 matching between the two models. Then, as a result of this FCA evaluation process, we established that the model in Fig. 4 contained necessary and sufficient constructs to specify full-fledged, industrial strength DIAs.

Also as part of the DPIM level validation, we realized the need for assessing the QoS properties of the Posidonia Operations case-study [8], with particular reference to the scalability of the system. So, we decided to use the MARTE-DAM profiles for such QoS assessment. However, instead of applying these profiles directly to our UML models, we decided to follow the standard approach of combining the DPIM domain model with the MARTE-DAM domain models. Hence, our DPIM model could be integrated with MARTE-DAM by inheriting the appropriate concepts (i.e., meta-classes) from the latter. Consequently, we revisited our domain model in Figure 4 and searched in the MARTE-DAM domain models for those relevant meta-classes that could be reused at the light of our DIA concepts. Table 2 summarizes our work of fitting the DIA DPIM concepts with those of MARTE and DAM. In the end, all DPIM concepts (left part) inherit some MARTE concept because DAM concepts in turn inherit MARTE concepts. The criteria for reusing was to find in MARTE those concepts offering the semantics we needed for DPIM concepts.

Finally, it is worth to indicate that regarding the initial version of the DPIM domain model published in [23], this final one disregards *QoSRequiredProperty* meta-class, rearranges the associations of *DataSpecification*, revises some inheritances from MARTE and eliminates technology-specific properties.

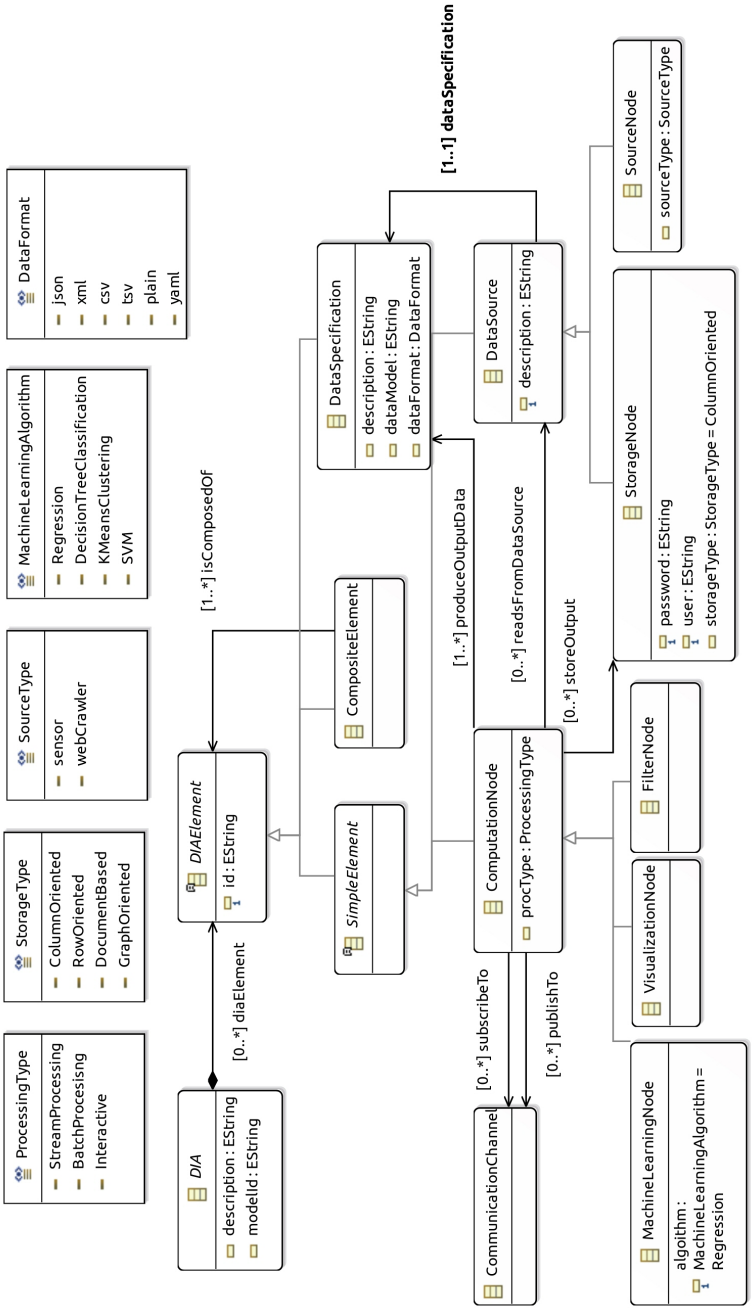

Fig. 4 DPIM domain model

DPIM concept	Parent MARTE-DAM concept
ComputationNode	MARTE::GRM::ResourceCore::ComputingResource
StorageNode	MARTE::GRM::ResourceCore::StorageResource
SourceNode	DAM::System::Core::Component
CommunicationChannel	DAM::System::Core::Connector
DataSpecification	n.a.

Table 2 Inheritance relationships among DIA concepts and MARTE-DAM concepts

3.2 UML DPIM profile

Figure 5 proposes UML extensions, i.e., stereotypes and tags, to conform the DPIM profile. It has been obtained by applying the three steps recalled in Subsection 2.3 (paragraph *Design of the DIA profile*).

For the first step, *StorageNode*, *SourceNode*, *CommunicationChannel* and *ComputationNode* are transformed into `DpimStorageNode`, `DpimSourceNode`, `DpimChannel` and `DpimComputationNode` stereotypes, respectively. The latter is further specialized, so to introduce filtering ratios and for distinguishing non-filtering nodes. *DataSpecification* is transformed into `DiaDataSpecification` datatype[5], which is useful for the `provide` and `respondsTo` tags, that aim to represent the association between *DataSource* and *DataSpecification*. Finally, abstract classes in the DPIM domain model do not map into stereotypes.

For the second step, we identified in MARTE and DAM, at the light of the work in Table 2, the stereotypes to inherit[6]. The three stereotypes (DpimComputationNode, DpimStorageNode, DpimSourceNode) which inherit `MARTE::Resource` allow complete definition, from the performance evaluation point of view, of DIAElements as resources; while the `DaConnector` inheritance also introduces properties such as error propagation to characterise dependability properties of DIA communication channels. Moreover, for enabling scenario-based QoS assessment, as proposed by MARTE, we introduced from scratch the `DpimScenario` as inheritance from `DAM::DaService`, although this stereotype does not introduce additional information, we consider important to distinguish kinds of scenarios for analysis at different abstraction levels. Previous choices introduce capabilities for defining QoS properties, from MARTE and DAM, which is very important since it enables to apply MARTE NFPs and VSL expressions, as explained in Appendix A.

For the third step, we did not need extra work. Due to all new DPIM stereotypes inherit a stereotype of MARTE or DAM, then the UML metaclasses that apply to the latter also apply to the former.

Finally, regarding the initial version published in [23], the profile introduces new stereotypes (`DpimVisualizationNode` and `DpimFilterNode`) and rearranges the inheritance tree corresponding to `DpimComputationNode`.

[5] See Appendix B for details on datatypes.

[6] In Figure 5, stereotypes with dark grey background have been taken from MARTE and the light grey ones from DAM.

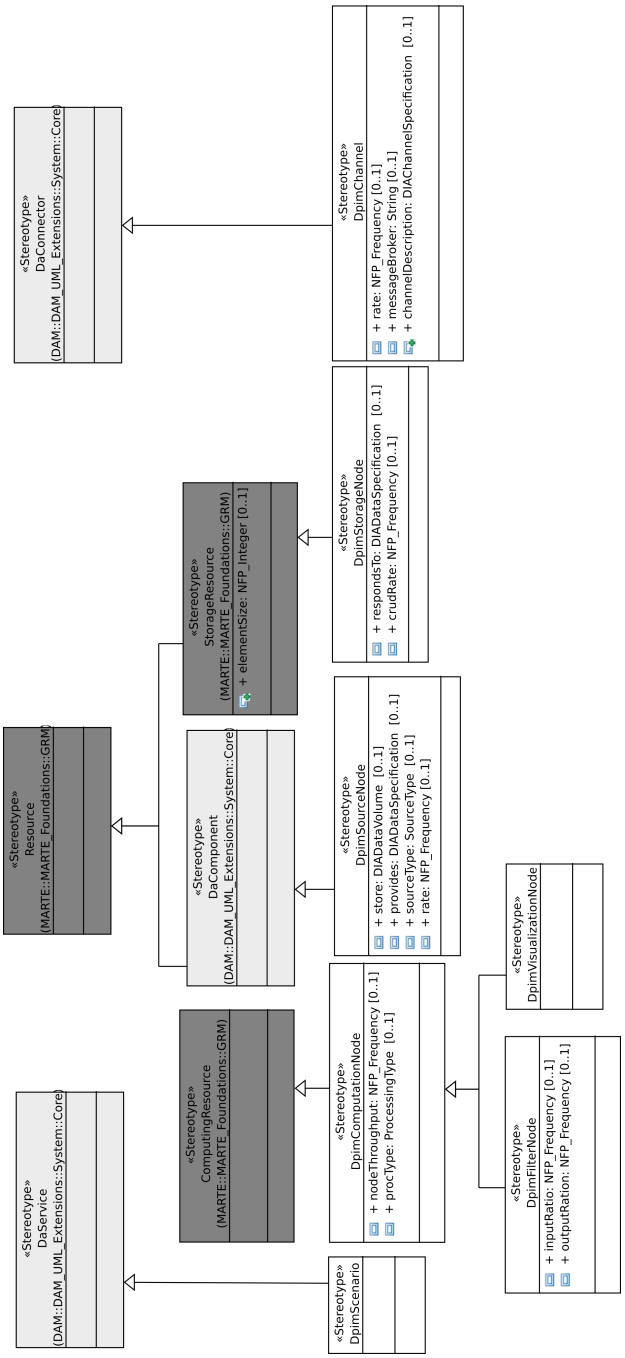

Fig. 5 DPIM UML Profile

4 DIA profile definition: DTSM

As already mentioned, the DTSM domain model gathers those concepts, belonging to concrete DIA technologies, related to the structure, relationships, and parameters of DIA elements for the quality analysis purpose. Therefore, the consequent DTSM profile is focused on the quality assessment of DIA, considering particular technologies.

We have conceived the DTSM level, both the domain model and the profile, into packages. Each package is dedicated to the modeling of each technology, except a *Core* package that is extended by all others. Figure 6 represents this structure of packages. The DIA technologies currently supported by our profile are Apache Spark [57], Apache Storm [58], Apache Hadoop [55], Apache Cassandra [54] and Apache Tez [59]. For space reasons, here we just develop the Apache Hadoop case, as a representative one, and we also offer an overview of the *Core* package. The technical definition of each domain model and consequent profile, for the rest of technologies, can be found in [60] and in [63], respectively.

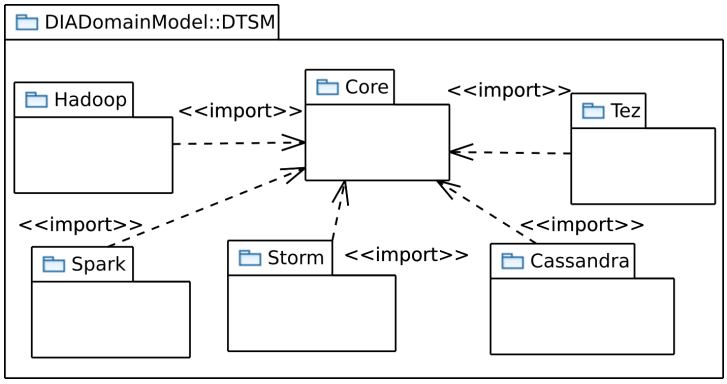

Fig. 6 Package view of the DTSM level

4.1 DTSM domain model

4.1.1 Core package

This package gathers common concepts in the modelling of DIA technologies. In particular, those to represent a *workflow of operations*, the *computational resources* used by the DIA, and the influx and outflux of *data*. During our exploration of DIA technologies, we have found that, in most of the technologies, the operations to execute are typically expressed as a *workflow* in the form of a directed graph, which is actually a directed acyclic graph (DAG). Therefore,

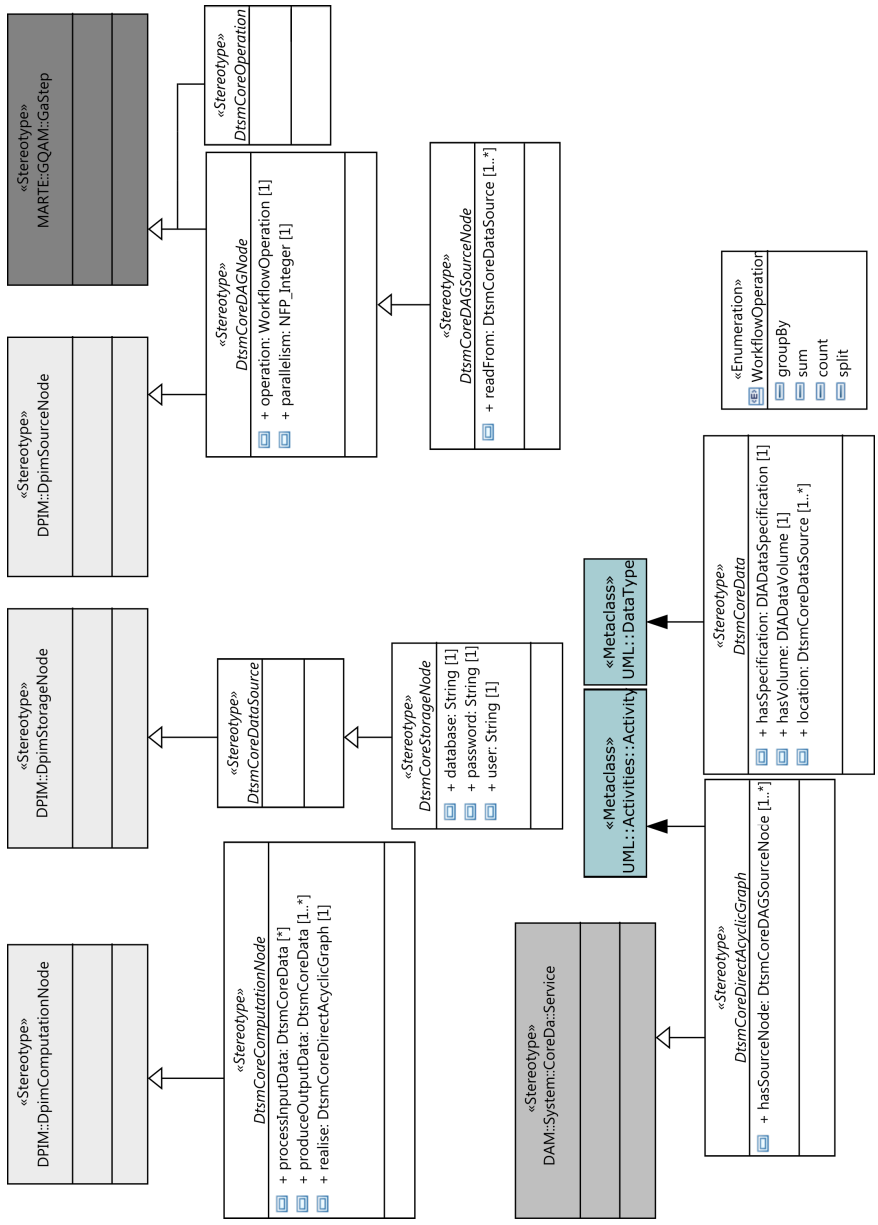

Fig. 7 UML extensions for DTSM Core

the DTSM Core must allow to represent DAG of operations. Each node of the DAG represents an *operation* of the DIA, over the data, at a certain abstraction level. The DAG is executed upon a set of *computational resources*. We can define already at the computational level the type of data that the DIA consumes and the type of data that produces. We can also define the number

of cores and nodes of these computation resources. The concept of *data* covers the specification of the type of data used by the DIA, its arrival to the workflow, and the transformations and eventual storage, of new data produced by the execution of the operations. Since this Core package has been created and refined while finding and extracting common concepts in the DIA technologies, the result is that both, the domain model and profile, represent same concepts, and thus the stereotypes of the profile, which is shown in Figure 7, correspond to classes in the Core domain model.

4.1.2 Apache Hadoop specific domain model

This section details the concepts we have defined to represent the characteristics of an Apache Hadoop based system that are relevant for quality analysis.

A software platform that offers a MapReduce [19] engine based on Apache Hadoop can be used by multiple *Users*. The allocation and utilization of computing resources of the platform is governed by a single resource manager, *YARN*[7] *resource manager*, and a *YARN Node manager*, for each *Node* in the platform. A *Node* executes containerized *Hadoop Operations*, each operation in a separated container. *Hadoop Operations* can be Map or Reduce operations. The MapReduce application of a user comprises: the operation to execute during the Map phase and the operation to execute during the Reduce phase. The Map operation is fed with an input *Dataset*, and the execution of the Hadoop application produces an output *Dataset*. Figure 8 presents in a domain model these concepts and their relationships, then conceptualising the Apache Hadoop platform structure.

The domain model in Figure 8 also includes those concepts, from the DPIM and MARTE domain models, that we selected to be reused. They are depicted in grey. The whole platform is represented as a *Hadoop Scenario*, which is an extension of the *MARTE::GQAM_Workload::BehaviorScenario* concept. A *Hadoop Scenario* is then composed of *Users*, *YARN resource Manager, Nodes,* and *HadoopOperations*. Each *User* uses the platform with a given workload, which is defined by the *MARTE::GQAM_Workload::WorkloadEvent* concept, through the *pattern* attribute, that allows to define different kinds of workloads. The MapReduce application of a user executes *numMaps* times its *HadoopMap* operation and *numReduces* times its *HadoopReduce* operation, while each *Hadoop operation* has a maximum parallelism (i.e, number of mappers or number of reducers that can execute in parallel). The user has also priority over a subset of resources of the platform, the so called *reservedResources* of a User. *HadoopOperations* are computational steps as defined by *MARTE::GQAM_Workload::Step*, then characterized by a *hostDemand*. A *Node* is a *ComputingResource* as defined in *MARTE::GRM::ResourceTypes*, the attribute *resMult* represents the number of cores of the *Node*. Each *Node* is managed by a *YARN node manager*. In our domain model, it is interesting to represent periodic health checks, in order to asses for how long the platform

[7] Yet Another Resource Negotiator

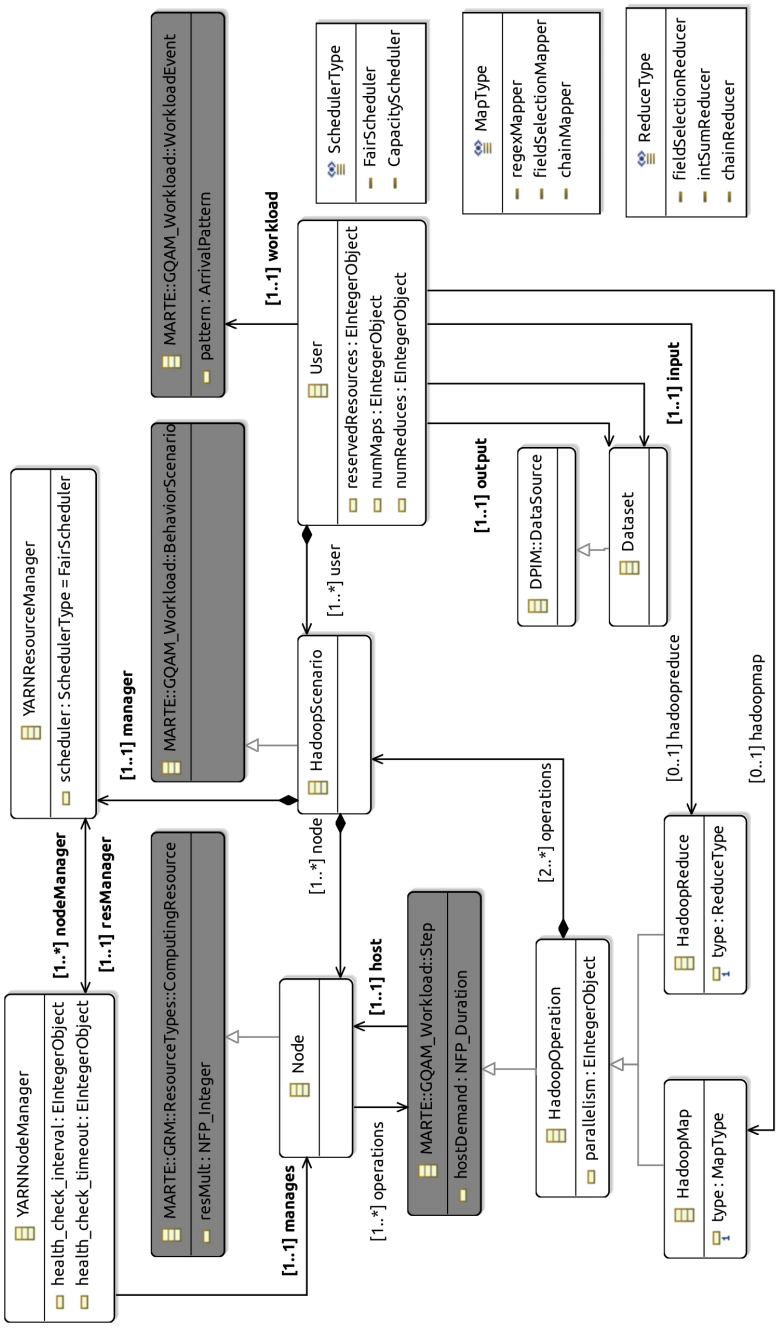

Fig. 8 DTSM domain model for Apache Hadoop

can be "out of full control" since some *Node* has failed but the health check has not been performed yet.

4.2 UML DTSM profile

For designing the DTSM profile, we have applied the steps in Subsection 2.3 (paragraph *Design of the DIA profile*).

The *Core* package defines common technological concepts that each particular technology will assume or refine. Subsection 4.2.1 presents the UML extensions offered by the *Core*. Subsection 4.2.2 presents those proposed for Apache Hadoop specifically. Both extensions together define a complete set of stereotypes to model quality aspects within UML diagrams for Apache Hadoop.

4.2.1 Core package

Figure 7 presents the UML extensions of the *Core* package. They model the technological concepts that we have found in common in the DIA technologies we have explored. Section 4.1.1 described the rational of the decisions to extract common concepts in DIA technologies, and hence to obtain these UML extensions; we have been able to generalize three macro-concepts shared in DIA technologies: the modeling of their workflow of operations, the modeling of the characteristics of the data consumed and produced by the DIA, and the modeling of the computing resources where the DIA is installed. The *Core* defines principles to model:

- **Workflow of operations**: Stereotypes `DtsmCoreDirectAcyclicGraph`, `DtsmCoreDAGNode`, and `DtsmCoreOperation` allow to model these concepts. Typically, DAGS are represented in UML by Activity Diagrams (AD), and thus stereotype `DtsmCoreDirectAcyclicGraph` is applied to UML Activity metaclass. Then, DAG nodes will be the actions and activities of the UML AD, this is congruent with their inheritance from `GaStep` of MARTE, which enables DAG nodes as computational steps.
- **Management of data influx/outflux**: Stereotypes `DtsmCoreDataSource`, `DtsmCoreStorageNode`, `DtsmCoreData` and `DtsmCoreDAGSourceNode` allow to model the presence of data, their storage, and the incoming flow of data to the workflow of operations.
- **Computing resources**: Stereotype `DtsmCoreComputationNode` inherits from `DpimComputationNode` the principles of computation nodes for DIA. Moreover, it adds tags to link with the DAGs that it executes and the definition of data that it supports.

4.2.2 Apache Hadoop specific profile

Figure 9 depicts the stereotypes specific for Hadoop:

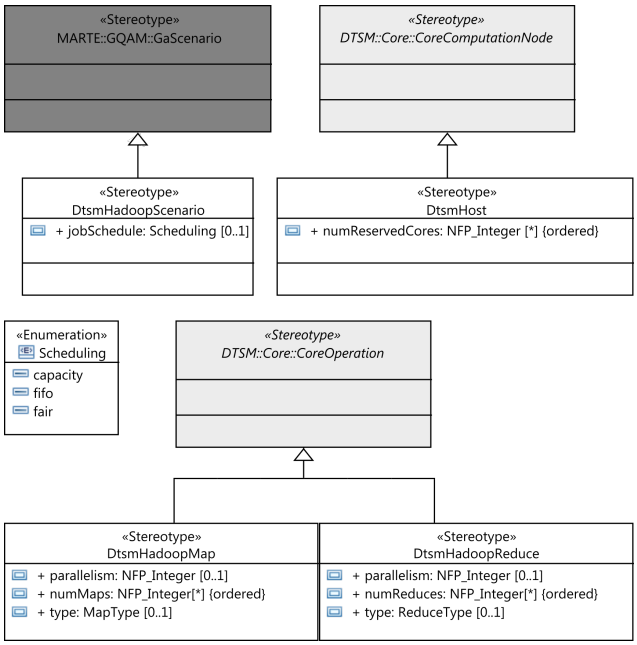

Fig. 9 UML extensions for modeling DIA based on Apache Hadoop

- `DtsmHadoopScenario` is a stereotype that extends `MARTE::GQAM::GaScenario`. So, it can define the job scheduling policy in Hadoop (fifo, fair and capacity schedulers currently) according to the YARN Resource Manager. As for the DPIM, the scenario concept is also needed to carry out scenario-based QoS assessment.
- `Host` applies to computation nodes, and so it extends the `DTSM::Core::DtsmCoreComputationNode`. As new, it introduces the number of reserved cores in the node by each user, `numReservedCores`. Hence, the type of this attribute is a list of `NFP_Integer` elements, being the size of the list equal to the number of users. It will be applied to `UML::Node` elements in Deployment Diagrams representing the resource view to be used in the QoS analysis.
- `DtsmHadoopMap` and `DtsmHadoopReduce` represent the actual Map and Reduce operations, and hence they extend the `Core::DtsmCoreOperation` stereotype. They allow to define the expected number of Map and Reduce operations for each user, `numMaps` and `numReduces`, respectively. The maximum number of Map or Reduce operations that a user can execute concurrently is defined by the `parallelism` attribute. Thus, user i has a number of Map (resp. Reduce) operations equal to `numMaps`[i] (resp. `numReduces`[i]), of which at most `parallelism` operations can execute concurrently. Since `Core::DtsmCoreOperation` also extends `MARTE::GQAM::GaStep`, it thus allows to define the expected time of the Map and Reduce task of each user

through its `hostDemand` attribute. These two operations could be merged into a single `DtsmHadoopOperation` operation and using an attribute to differentiate between Map or Reduce. That solution would also be correct and would have the same modelling power. Therefore, the rationale for proposing an stereotype for each operation is that it helps visualising that the model is logical (e.g., there are not only Reduce operations, or only Map operations); it simplifies the transformation to analyzable models because the transformation patterns can be defined at the stereotype level, rather than looking at the value of an attribute to decide the transformation rule to apply.

The representation of the users workload in Figure 8 can be directly modelled using multiple MARTE `GaWorkloadEvent`, one for each user in the system. However, the modelling power MARTE `GaWorkloadEvent` is higher than the necessities for modelling the users workload, since it allows modelling `periodic, aperiodic, sporadic, burst, irregular, closed,` and `open` arrival patterns, but we only need the `closed`. Therefore, we must add a constraint to allow only the definition of `closed` arrival patterns, being its `population` attribute set to the number of jobs of a user and its `extDelay` set to the think time between the completion of a job and its next execution.

5 DIA profile definition: DDSM

While at the DTSM level the interest focused on those elements of the technology relevant for the quality assessment of DIAs, for instance, the concepts of Map and Reduce in Hadoop, at the DDSM level the interest is in modeling the specific infrastructure components that are to be deployed and configured for the DIA to execute. For instance, Hadoop MapReduce applications run in an Hadoop cluster, which consists of the Hadoop Distributed File System (HDFS), which is the underlying scalable data storage, and a computational resources manager such as YARN (Yet Another Resource Negotiator). It is worth noting that, all technologies of interest are distributed systems running on top of *clusters* of computational resources, e.g. *virtual machines* (VMs) within a Cloud infrastructure.

In the following, as we did for the DPIM and DTSM levels, we present the DDSM domain model (Subsec. 5.1) and the DDSM profile (Subsec. 5.2). Moreover, Section 9 describes an automatic deployment mechanism that leverages the DDSM profile.

5.1 DDSM domain model

From the general observations, given above, and considering that almost every platform adopts either a master/slave or a peer-to-peer architecture, in the DDSM domain model we start by identifying two basic deployment abstractions:

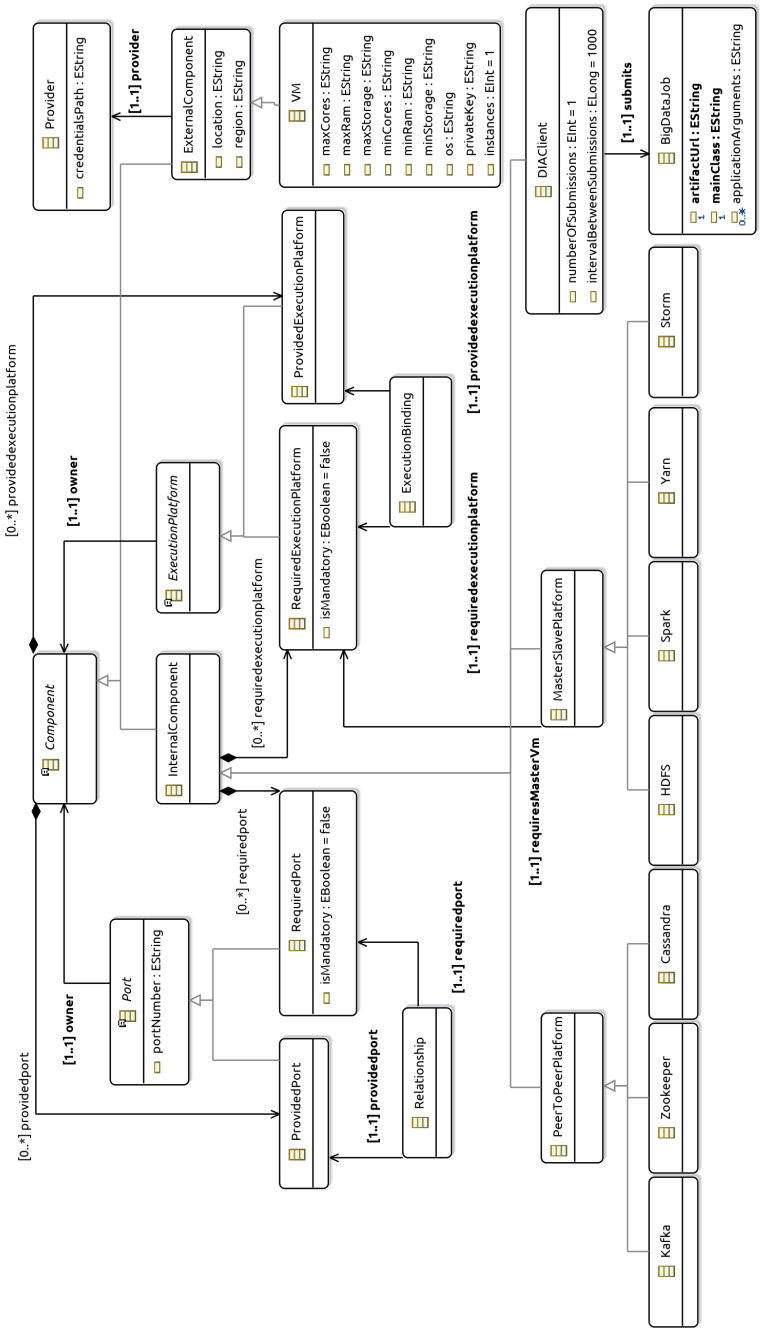

Fig. 10 DDSM domain model

(i) a *PeerToPeerPlatform* represents a distributed component which operates according to the peer-to-peer architecture and consists of a cluster of peer nodes;
(ii) a *MasterSlavePlatform* operates according to the master-slave architectural style and is composed of a master node and a cluster of slave nodes.

We then define a set of technology-specific deployment concepts, one for each technology supported (currently Storm, Hadoop, Cassandra, Zookeeper and Spark). Each technology element extends one of the two abstract concepts presented above and captures technology-specific configurations as well as any dependency on other platforms and components. See these concepts in Figure 10.

As previously stated, Big Data frameworks and applications typically run on top of Cloud infrastructures, composed of execution containers (e.g., Virtual Machines - *VM*), storage and network systems. To support modelling of such aspects we rely on MODACloudsML [3, 20], a modelling language for the design of multi-Cloud applications, and we slightly extended it in our DDSM modeling approach. MODACloudsML provides the concept of *Component* that can be either an *InternalComponent*, which is deployed and managed by application providers, in our case, Big Data frameworks providers and DIAs providers, or an *ExternalComponent*, which is offered by third parties – typically, a cloud *Provider*. A Virtual Machine (*VM*) is a type of ExternalComponent and can be characterized by a set of parameters which specify its size in terms of RAM, cores and disk memory. A VM can mount a specific OS and require a private key in order to be accessed. An *InternalComponent* can rely on the services or execution platforms offered by another *Component* by defining a *RequiredPort*/*RequiredExecutionPlatform* respectively, which can be mandatorily required or not. *Components* can offer a service or an execution environment by exposing a *ProvidedPort*/*ProvidedExecutionPlatform*. The concept of *Relationship* (*ExecutionBinding*) is then used to bind a *RequiredPort* (*RequiredExecutionPlatform*) with a *ProvidedPort* (*ProvidedExecutionPlatform*).

We extended the original concept of *VM* in MODACloudsML by adding the property *instances*, so to be able to model a cluster of homogeneous VMs. This extension is fundamental in order to model distributed systems running on clusters of VMs. A generic InternalComponent that is connected, by means of a triple of the type *(ProvidedExecutionPlatform, ExecutionBinding, RequiredExecutionPlatform)* with a VM, will be actually deployed in as many replicas as the number of available instances of that VM. Hence, this extension provides the right level of deployment abstraction, without having to deal with the inherent complexity of deploying distributed systems. In fact, once we have a simple way to model clusters of VMs hosting Big Data frameworks and the DIAs built on top of them, they are no more than cloud applications organized in various components that run within clusters of VMs. Thus, we define our deployment abstractions as subclasses of the InternalComponent concept. The MasterSlavePlatform concept requires a specialized association

requiresMasterVm to model the deployment of the master node separately from that of the cluster of slave nodes.

The concept of *DIAClient* is defined as a specialized InternalComponent and it submits to a Big Data framework the set of jobs defining the application logic, called *BigDataJob*. The DIAClient can act in different ways while submitting the BigDataJob, and the model allows to define specific scheduling options. For instance the *numberOfSubmission* property indicates how many times a job must be submitted, while the *intervalBetweenSubmission* indicates the delay between job submissions. This can be useful when it is necessary to model a recurring job, like for instance a job that have to be executed every 24 hours for a total of 30 times. Analogously, each BigDataJob is characterized by a number of properties, such as the url from which the executable artifact can be retrieved (*artifactUrl* property), the main class to be considered when launching the job (*mainClass* property), plus additional job's arguments (*applicationArguments* property).

5.2 UML DDSM profile

For designing the DDSM profile, we have applied the steps in Subsection 2.3 (paragraph *Design of the DIA profile*). Here we note that the UML deployment diagram [66] is the natural choice when dealing with the deployment of an application. Hence, the DDSM profile needs to extend the UML deployment diagram only, i.e., the proposed stereotypes will be applied to meta-classes relevant to this diagram.

According to the methodology outlined in Subsection 2.3, here we define, for each class in the domain model, if it maps to an abstract of a concrete (or firm) stereotype, along with the UML meta-classes extended by each such stereotype. Considering the previous grounds, not every class in the DDSM domain model has become a new stereotype since some of the concepts are already supported by UML. In particular, for all the concepts in the domain model that are supposed to model relations and connections between components, such as the abstract concepts of *Port* and *ExecutionPlatform* as well as the concepts of *Relationships* and *ExecutionBinding*, we can identify corresponding concepts and modeling elements already defined by the UML deployment diagram meta-model. More specifically, we identified the following mapping:

- a triple of the type *(ProvidedExecutionPlatform, ExecutionBinding, RequiredExecutionPlatform)* can be modeled in UML deployment diagrams as a *Deployment* between a *DeploymentTarget* and an *ExecutionEnvironment*,
- a triple of the type *(ProvidedPort, Relationship, RequiredPort)* can be modeled in UML deployment diagrams as a *CommunicationPath* between two *DeploymentTarget*s.

Given these premises, the resulting DDSM profile is shown in Figure 11, in which, for the sake of space only the technology-specific stereotypes related

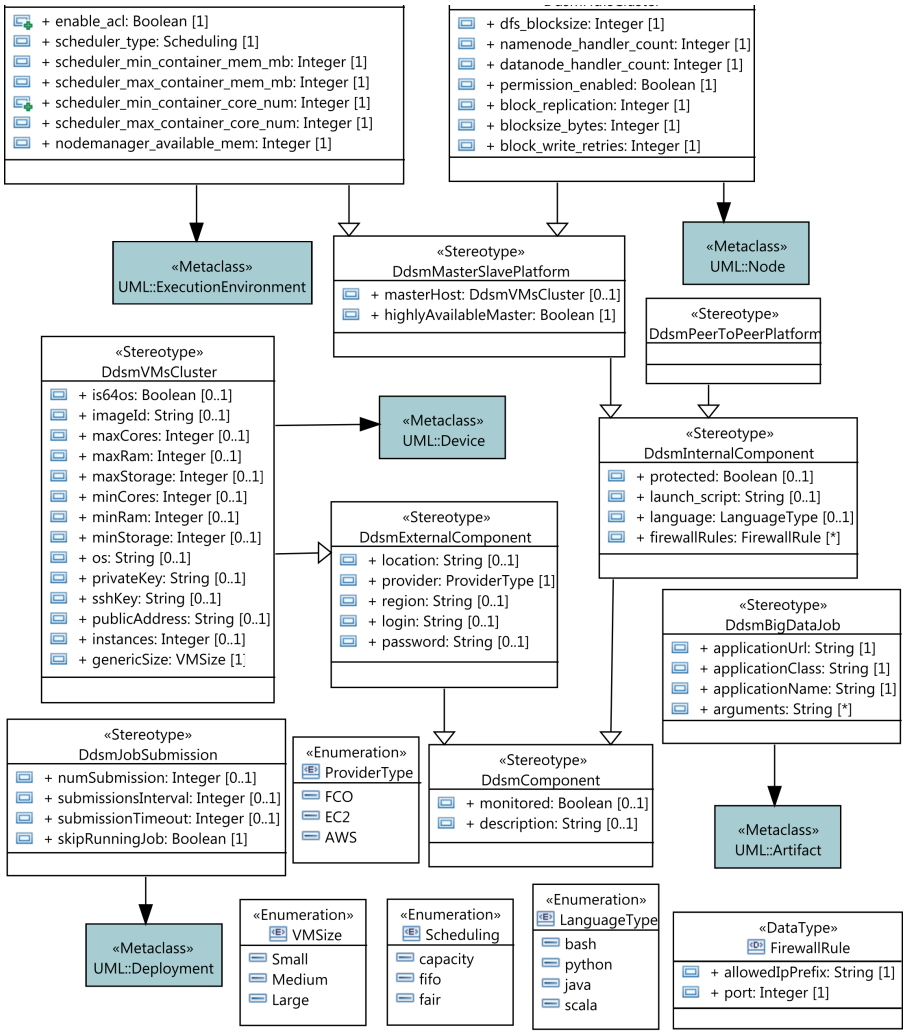

Fig. 11 An extract of the DDSM UML profile

to Apache Hadoop are shown, with all the provided configuration parameters. The meaning of each stereotype it that discussed in Section 5.1 while presenting the domain model. In the following we just elaborate on the decisions taken to design the DDSM profile itself.

First, we decided to define `DdsmComponent`, `DdsmInternalComponent`, `DdsmExternalComponent`, `DdsmPeerToPeerPlatform` and `DdsmMasterSlavePlatform` as abstract stereotypes, since, also in MODACloudsML, these concepts are supposed to be generic and extensible by specific technologies.

`DdsmVMsCluster`, which has been derived from the *VM* concept of the domain model, extends the *Device* UML meta-class. In fact, a cluster of VMs

logically represents a single computational resource with processing capabilities, upon which applications and services may be deployed for execution, which is essentially the definition of the UML Device concept.

DIA execution engines (e.g. Hadoop YARN or Spark) extend the UML *ExecutionEnvironment* meta-class. In fact, similarly to Operating Systems, which provide an execution environment for standard programs and are typically modeled as sub-stereotypes of UML Execution Environment, DIA execution engines provide execution environments for Big Data jobs. All the other technology-specific stereotypes, e.g., those relevant to represent Hadoop HDFS or any other Big Data database/middleware, extend the UML *Node* meta-class. In fact, they are essentially deployment targets, which can be allocated on available Devices and may host Artifacts (for example a `DdsmHdfsCluster` may contains a set of files).

The `DdsmBigDataJob` stereotype extends the UML *Artifact* meta-class, since it corresponds to a DIA executable artifact (e.g. a Hadoop MapReduce or a Spark application). The `DdsmJobSubmission` stereotype, derived from the concept of ClientNode in the domain model, extends the UML *Deployment* meta-class, as it is meant to link deployable Big Data jobs to the available DIA execution engines and to specify additional deployment options for a Big Data job. In this way the same DIA job can be deployed in multiple instances using different deployment options.

The defined stereotypes led to the definition of a small library of UML *Enumerations* and *DataTypes*, consistently with the methodology presented in Subsection 2.3. In particular, the `VMSize` and the `Provider` *Enumerations* are used by the `DdsmVMsCluster` stereotype to specify the size of each VM in the cluster and the Cloud provider to be used in order to provision the cluster respectively (see the `genericSize` and the `provider` Properties of `DdsmVMsCluster` stereotype). The `Language` *Enumeration* is used by the `DdsmInternalComponent` stereotype to specify the programming language using which the component is implemented. The `Scheduling` *Enumeration* is specific for Yarn to specify the scheduling strategy used when allocating resources. Similar ones are defined also for other supported BIg Data technologies, although not shown in Figure 11. Finally the `FirewallRule` *DataType* can be used to set specific firewall rules required by `InternalComponents`, which may, for example, listen for web requests on specific TCP ports.

In the following, we summarize steps for modeling, in a UML deployment diagram, the essential parameters for DIA deployment. Hence, these steps constitute a blueprint for modelers to get UML DDSM-profiled models:

1. Instantiate as many *Devices*, annotated with the `DdsmVMsCluster` stereotype, as the number of distinct homogeneous clusters of VMs you want to deploy.
2. Configure each `DdsmVMsCluster` as desired, using the tags provided by this stereotype, see Figure 11;
3. Allocate within the various `DdsmVMsCluster` as many *Nodes* as the number of Big Data platforms needed by the DIA to deploy. Each *Node* should be

annotated with the corresponding stereotype of the Big Data platform it represents, e.g., `DdsmHdfsCluster` or `DdsmYarnCluster`.
4. Configure each Big Data platform as desired, using the tags of the corresponding stereotype. Execution engines, such as YARN, should be actually *ExecutionEnvironments*, rather than simple *Nodes*.
5. Instantiate as many *Artifacts*, annotated with the `DdsmBigDataJob` stereotype, as the number of DIAs to be deployed.
6. Configure each `DdsmBigDataJob` with information about the actual DIA, using the tags provided by the stereotype. Moreover, each `DdsmBigDataJob` must be connected, by means of a *Deployment* annotated with the `DdsmJobSubmission` stereotype, to the Big Data execution engine that is supposed to execute the DIA. Additionally, deployment specific options can be configured using the `DdsmJobSubmission`'s provided tags. Finally, each `DdsmJobSubmission` should be connected by means of a *Dependency* with each Big Data platform it requires in order to execute.

6 Adopting the DIA profile and the WikiStats example

The methodological steps entailed by the usage of the proposed profile encompass at least the following activities:

1. At the DPIM level, elaborate a component-based representation of the high-level architecture of the DIA (e.g., a DPIM Component Diagram). In the scope of the proposed profile, this is done using the stereotypes and tags necessary to specify the DIA nodes (source node, compute node or storage node);
2. Augment the component-based representation with the properties and non-functional specifications concerning that representation;
3. Refine that very same component-based representation with technological decisions, if any; the decisions themselves represent the choice of which technology shall realise which data-intensive application node. For example, a <<CassandraDataStore>> conceptual stereotype is associated with a <<StorageNode>> in the DPIM architecture view;
4. At the DTSM level, associate several data-intensive technology-specific diagrams representing the technological structure and properties of each of the data-intensive nodes. These diagrams essentially "explode" the technological nodes and contain information specific to those technological nodes. For example, a <<StorageNode>> in the DPIM architecture representation must be necessarily associated to a specific storage technology in the DTSM counterpart; finally, the DTSM layer will feature yet another diagram, more specifically, a data-model for the Cassandra Cluster. These separate technology-specific "images" serve the purpose of allowing data-intensive application analysis and verification;
5. At the DDSM level, elaborate a deployment-specific component deployment diagram where the several technology specific diagrams fall into place

with respect to their infrastructure needs. This diagram contains all necessary abstractions and properties to build a deployable and analysable TOSCA blueprint. Following our <<CassandraDataStore>> example, at this level, the DTSM <<CassandraDataStore>> node (refined from the previous DPIM <<StorageNode>> construct) is finally associated with a DDSM diagram where the configuration of the cluster is fully specified (i.e., VMs type and number, allocation of software components to VMs, etc.);

6. Finally, once the data-intensive deployment-specific component diagram is available, DICE deployment modelling and connected generative technology (DICER) can be used to realise a TOSCA blueprint for that diagram;

In summary, designers exploiting our UML modelling for DIA will be required to produce (at least) one component diagram for their architectural structure view (DPIM), two (or more) diagrams for their technology-specific structure and behavior view (DTSM), and a deployment diagram for their deployment view (DDSM). Our modelling approach does not encourage the proliferation of many diagrams, e.g., for the purpose of re-documentation. Since the focus is on quality-aware design and analysis, then our approach promotes modelling all and only the technological nodes that require quality-awareness. Finally, designers are required to refine their architectural structure view with deployment-specific constructs and connected decisions.

To provide concrete examples of the models designers should aim at, let's consider a concrete example called *WikiStats* [69]. This is inspired by the Traffic Analysis Report software of Wikimedia Foundation[8]. Since 2016, some Traffic Analysis Report of Wikimedia Foundation websites (e.g., Wikipedia) such as browser and operating systems of users[9] and page views by country of users[10] are generated by a Hadoop infrastructure[11] using as input the web access logs. In our case, we are interested on five types of traffic statistics over a reduced set of logs (e.g., the access logs that represent requests to Wikipedia pages in a concrete language), and we target an hourly analysis of the logs. Therefore, the Hadoop platform of our WikiStats will support a population of five jobs that execute periodically, having each of them a think time of 1 hour.

At the DPIM level, we can see the system as a component-based aggregation of three nodes, see Figure 12. A `<<DpimSourceNode>>` is responsible for crawling (see the `sourceType` property of the `Wikimedia` element) the Wikimedia website and fetch the web pages to be processed. A `<<DpimComputationNode>>`, named `WikistatsApplication`, is then responsible for actually processing the retrieved web pages. Since we want the processing to be performed in a streaming fashion, the `procType` property is set to be *streaming*. This component represents the actual DIA that will later run on top of Hadoop and whose internals will be detailed at the DTSM level. The *readsFromDataSource* and

[8] https://www.mediawiki.org/wiki/Analytics/Wikistats/TrafficReports/Future_per_report_B2

[9] https://analytics.wikimedia.org/dashboards/browsers/#all-sites-by-os

[10] https://stats.wikimedia.org/wikimedia/squids/SquidReportPageViewsPerCountryOverview.htm

[11] https://wikitech.wikimedia.org/wiki/Analytics/Systems/Cluster

the *storeOutput* relationships of the `ComputationNode` concept in the DPIM metamodel are modeled re-using the *Directed Association* UML element. Finally the output of the processing needs to be stored into a database. For this purpose, we instantiate a `<<DpimStorageNode>>`, which we may want, for example, to be a column-oriented database (see the `storageType` property of the `InternalDatastore` element). The `<<DpimStorageNode>>` also allows to specify the username and password to be used for accessing the database. No `<<CommunicationChannel>>` is needed since WikiStats does not require any message-queuing. We can notice that, at this abstraction level, it has not been yet decided that the application will execute upon a Hadoop infrastructure.

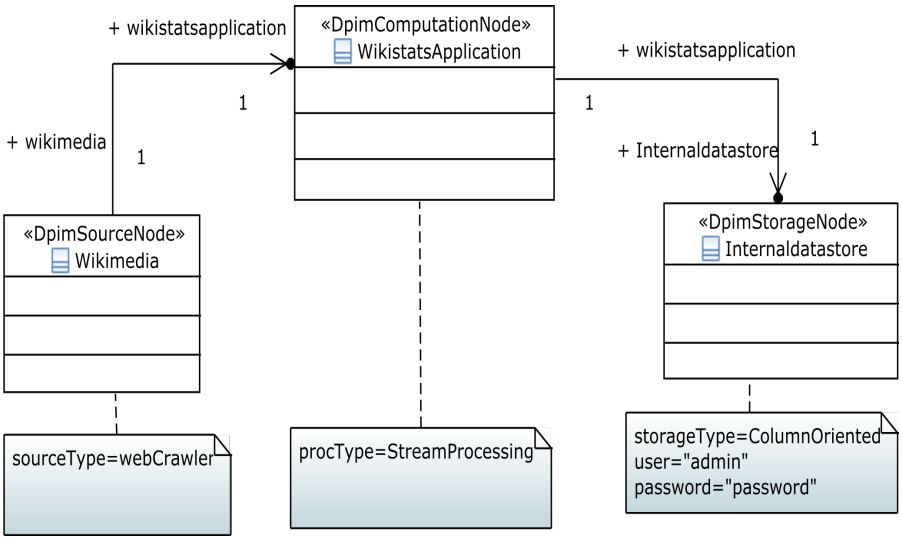

Fig. 12 High-level view of WikiStats

At the DTSM level, the designer decides, in this case, to select Hadoop as the target technological platform for the computation node and defines the models shown in Figures 13 and 14. The first is a behavioural model that identifies two types of components performing map and reduce functions respectively. They can be available in many parallel instances. A detailed description of the main properties of this model is presented in Section 8. The diagram of Figure 14 provides the *resource view* for the DTSM, identifying in this case, the configuration of Hadoop as composed of a single node exploiting four cores. Starting from the behavioral view and the resource view, the designer can run various QoS analyses as discussed in Section 8 and can use their results to further refine the technological choice he/she has made.

At the DDSM level, the designer focuses on the deployment view by modeling in detail all components that have to be deployed for WikiStats (see Figure 21). In this case, besides YARN, which is managing the Hadoop infras-

tructure, a HDFS cluster is used as a storage node. WikiStats is represented in the diagram as an artifact that depends both on HDFS and Hadoop. All infrastructural elements are deployed on a cluster of 5 medium size virtual machines. Details on the transformation from this deployment view into an executable blueprint are discussed in Section 9.

In the scenario we have described above, the designer has decided to select Hadoop as main implementation technology for WikiStats. Other technologies could have been selected as well. In [29] we describe how a different Storm-based case study can be modeled at the DPIM and DTSM level. In [22] we have shown how we can model at the DDSM level a Storm-based and a Spark-based deployment of WikiStats that exploits Cassandra as a storage node. In general, changing technologies means adopting the architectural style typical of that technology, exploiting the corresponding profile stereotype, and personalize, if needed, the properties associated to the technology. Each of these properties has associated a default value that can allow non-expert users to exploit a typical configuration of the technology.

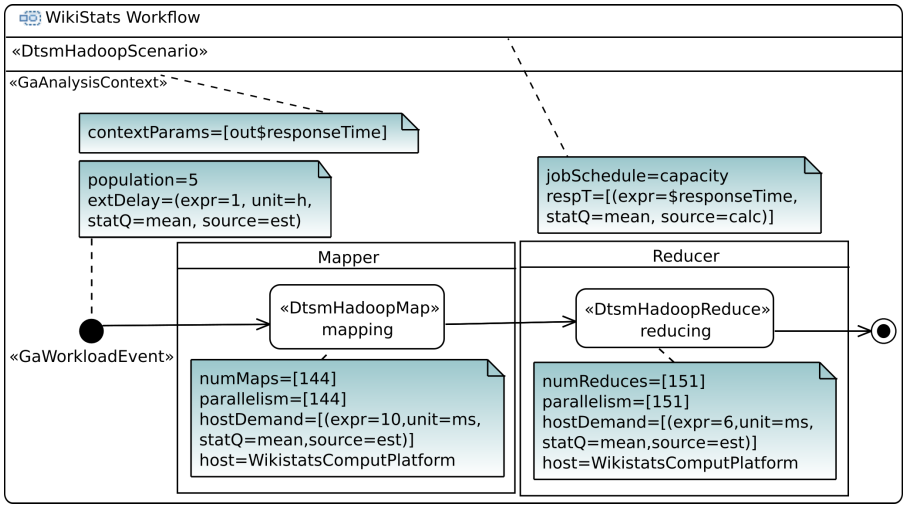

Fig. 13 WikiStats: UML activity diagram

7 Goal of the validation

Once the three levels of the DIA profile have been presented, we want to carry out the profile validation, according to the workflow in Figure 2. The validation of the profile means to demonstrate its adequacy for addressing the objectives we established in the third paragraph of the Introduction:

– DPIM: Guidance for an initial design of the DIA architecture.

– DTSM: Enabling the quantitative assessment of the DIA.
– DDSM: Enabling the automated deployment of the DIA.

Regarding the DPIM level, as stated in Section 2, the profile has been validated through two case studies: the Posidonia Operations and a fraud detection system, reported in [8] and [44] respectively. Each case study is a project developed by a different partner of the DICE project [9], for which the DIA profile has been developed. For both systems, we modelled the high-level architecture, using the DPIM profile to pinpoint the basic architectural constructs regarding the DIA concepts. Moreover, we modelled, also assisted by the DPIM profile, their main workflows using UML activity diagrams and their initial deployments. For both systems, we conducted an assessment of the architecture design, at the DPIM level. These two experiences have shown the suitability of the DPIM profile for modeling the two cases and for enabling the analysis of their QoS characteristics.

The validation of the DTSM and DDSM profiles is carried out in the next sections using the *WikiStats* application example introduced in the previous section.

8 DTSM profile validation: Quality assessment

This section carries out the validation of the DTSM profile. Such validation accomplishes our claim in the Introduction: *The profile disengages developers from knowing details of quality assessment, i.e., performance and reliability assessment.* Hence, we need to demonstrate that the DTSM profile: a) allows to gather all the quantitative information needed for eventually assessing the quality of a DIA; and b) helps to identify those model elements involved in the assessment, e.g., service demands for mappers or reducers. In this validation, we rely on the Hadoop technology since it has been the profile used to illustrate the DTSM level. Another satisfactory experience was carried out with the DTSM profile in [47], which has been extended in [29], but using the Apache Storm technology. Finally, Subsection 8.5 discusses choices of suitable performance languages for assessment.

8.1 Modeling for quality assessment

In the following, we summarize steps for modeling quality input parameters, which will be eventually used in the performance assessment of the DIA. Hence, these steps may constitute a guide for practitioners to create models using the DTSM profile, as those in Figures 13 and 14, for the Apache Hadoop technology:

1. Define the number of users in the cluster. This is represented by the `population` attribute of the `<<GaWorkloadEvent>>` stereotype, see Figure 13;

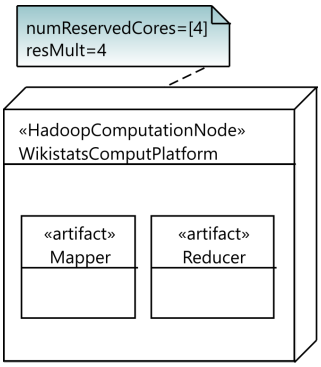

Fig. 14 Wikistats: UML DTSM resource view

2. Estimate the think time between consecutive executions of jobs. This is represented by the `extDelay` attribute of the `<<GaWorkloadEvent>>` stereotype. In this case, we are estimating a delay of one hour. The definition of the values for this attribute are annotated using a MARTE-VSL expression, see Appendix A for details;
3. Define the number of Map (Reduce) operations. This is represented by the `numMaps` (`numReduces`) attribute of the `<<HadoopMap>>` (`<<HadoopReduce>>`) stereotype, see Figure 13;
4. Estimate the mean execution time of each Map (Reduce) operation. This is represented by the `hostDemand` attribute of the `<<HadoopMap>>` (`<<HadoopReduce>>`) stereotype. These stereotypes extend the `<<Core::CoreOperation>>`, which in turn extends `<<MARTE::GaStep>>`, that in fact provides the `hostDemand`, which is needed for defining in the model the service time that a Map (Reduce) operation requires from its computing resources, see Figures 7 and 9. Performance models use this `hostDemand` for setting the frequency with which they can process elements, for instance, `hostDemand` will define the servers service time in case of using Queueing Networks or the average firing delay of transitions in case of using Stochastic Petri Nets;
5. Decide the number and parallelism of the cores in the platform to be used. This is represented by the `numReservedCores` and `resMult` attributes of the `<<HadoopComputationNode>>` stereotype, see Figure 14;
6. Define the metric to be computed using the `<<HadoopScenario>>` stereotype. In this case, we want to compute the response time. Use this stereotype also for defining the Hadoop scheduling policy, in case you want to use the Hadoop scheduling facility. See Figure 13.

At this moment, we need to recall that the UML models are not 'per se' useful to carry out performance prediction [18]. In fact, performance prediction, based on models, needs the use of analyzable models, e.g., Petri nets [1]. Therefore, we have used the DICE Simulation tool [61] to carry out the actual

performance evaluation of our example inspired on WikiStats problem and its Hadoop based solution.

8.2 Simulating DTSM-profiled models

The DICE Simulation tool [61] reads UML models annotated with the DPIM or DTSM profiles, and then it creates the corresponding Petri net, by leveraging techniques already developed in the literature [16, 71]. In particular, for the DTSM Hadoop models, the tool creates Stochastic Well-formed Nets (SWN) [12], a sub-class of Petri nets well-fitted for performance evaluation following the transformation patterns described in Appendix D. Once the SWN has been created, the tool invokes GreatSPN [21], a Petri net solver engine, and it computes the quality metrics specified in the UML model –response time in our case–.

8.2.1 Proof-of-concept simulation

In order to test the potential of the performance simulation, we first built a proof-of-concept Hadoop application. This application receives F text files, each of them with containing a single line with N random values. The application computes the median value of each array in the Map activity, and then the average value of the F medians in the Reduce activity. One of the goals of this proof-of-concept is to experiment the performance prediction with a Map activity that does not scale linearly with the size of the input. Therefore, to calculate each median value, the Map activity first sorts the values using an $O(N^2)$ method (concretely, it requires $\frac{N^2+N}{2}$ operations) and then it returns the tuple <"median",$calculatedMedian$>. In turn, the computation of the average value in the Reduce activity is $O(F)$ (i.e., the Map activity has produced a tuple to reduce for each single-line input file). To keep the proof-of concept simple and to make simpler the execution time measurement, we do not set think time between executions, we set a single resource, and we do not bound the maximum allowed `parallelism` (although this last setting does not matter due to the single resource).

To give values to the stereotype attributes, we executed the Hadoop application, with $F = 30$ and $N = 15000$ elements each. The 30 computation of medians finished in 25 seconds, so we got that an input line containing 15000 elements can be Mapped in around 833ms. Knowing the time to Map an array of 15000 elements and the Map complexity $\frac{N^2+N}{2}$, we can create UML models with several values for the attributes in the profile and then compare the simulation prediction with the actual running time. Table 3 shows the values given to the attributes of stereotypes in this proof-of-concept. The input parameters have the following values:

- F=[10,20,30]
- N ranges from 5000 to 30,000 in steps of 5000.

– timeSingleOp=$\frac{833 \cdot 2}{15000^2+15000}$

Stereotype	Attribute	Value
`<<GaAnalysisContext>>`	contextParams	[out$responseTime, in$F, inN, intimeSingleOp]
`<<DtsmHadoopScenario>>`	jobSchedule	capacity
`<<DtsmHadoopScenario>>`	respT	[(expr=$reponseTime, statQ=mean, source=calc)]
`<<GaWorkloadEvent>>`	pattern	closed(population=1, extDelay=0)
`<<DtsmHadoopMap>>`	numMaps	F
`<<DtsmHadoopMap>>`	hostDemand	[($expr = \frac{N^2+N}{2} \cdot$ timeSingleOp, statQ=mean, source=calc)]
`<<DtsmHadoopReduce>>`	numReduces	1
`<<DtsmHadoopReduce>>`	hostDemand	[(expr=$F \cdot$ timeSingleOp, statQ=mean, source=calc)]
`<<DtsmHadoopComputationNode>>`	resMult	1

Table 3 Attributes defined for the Hadoop simulation proof-of-concept

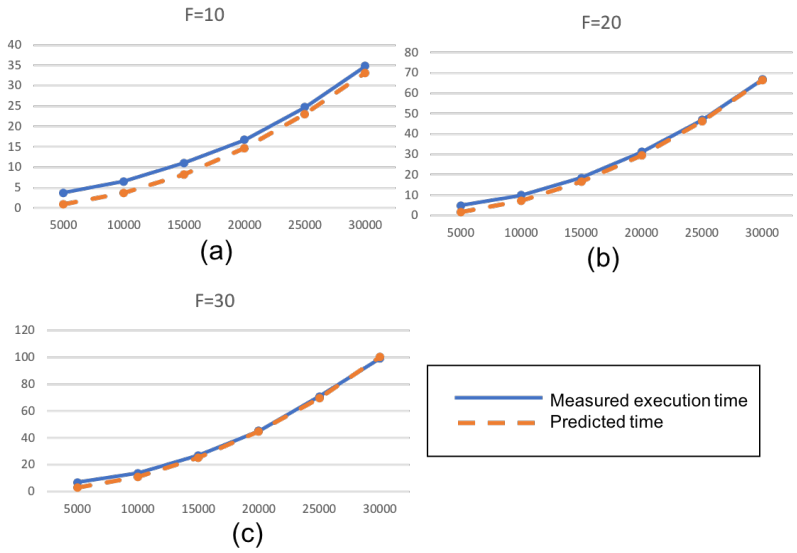

Fig. 15 Proof-of-concept results: Measured Vs Prediced response times in function of the input size. X-axis: N. Y-axis: execution time in seconds.

We have simulated these 18 combinations of F and N. We have also executed all the 18 combinations and measured their running time. Figure 15

shows our results. We can see that, for this initial proof-of-concept, the simulation of Hadoop profiled UML models gives results that are very similar or, in some cases, almost identical to the measured values. For higher N or F values the differences between the measured and predicted execution times are indistinguishable while, we can observe a small gap between the two lines in the plots when N and F have lower values.These proof-of-concept results encourage us to further explore the potential of the simulation-based performance assessment of DTSM-profiled UML models for more complex applications. We remark that we have used the measured values of only one of these 18 combinations to compute the parameters for our profiled UML models. The remaining 17 combinations have been measured after the model parameterization.

8.2.2 WikiStats simulation

Figure 16 depicts the SWN created by the DICE Simulation tool from the profiled UML design. The concepts modeled by the Petri net are those captured by the DTSM annotations: Number of users, number of Map (Reduce) operations per job, number of cores, mean execution time of each Map (Reduce) operation, and think time between jobs. Additional and interesting characteristics of the Petri net are the following: It is a closed net, where jobs pass through maps and reduce phases. The Reducing phase of a job starts right after all the Mappings have finished. Also, when the Mapping phase of a job has been completed, the Mapping phase of the next job can start. Therefore, Mapping activities of different jobs do not compete for resources since they do not execute concurrently. However, the Mapping activity of a job can execute concurrently to the Reducing activity of a precedent job, albeit the Reducing activity has higher priority over resources.

8.3 WikiStats Performance assessment

In the following, we elaborate different performance scenarios for the reader to gain insight on the usefulness of the DTSM profile for guiding the performance assessment process [53].

Increasing the workload in the system We have studied what would happen when increasing the traffic statistics in our WikiStats application. In the annotation of the UML model with our profile, this will increment the population of DIA users, i.e., an increase in the intensity of use of the DIA. In terms of the DTSM profile annotation, we just need to change the `hadoopPopulation` attribute, it will no longer be a constant but a variable. We set the variability interval from the initial 5 up to 15 different statistics that execute in an hourly basis, meaning that we will get insight of the expected performance up to the case when the initial workload triples. Then, we ran the DICE Simulation tool again and obtained the results in Figure 17. It is worth noting that the variability in the population is not represented in the design model

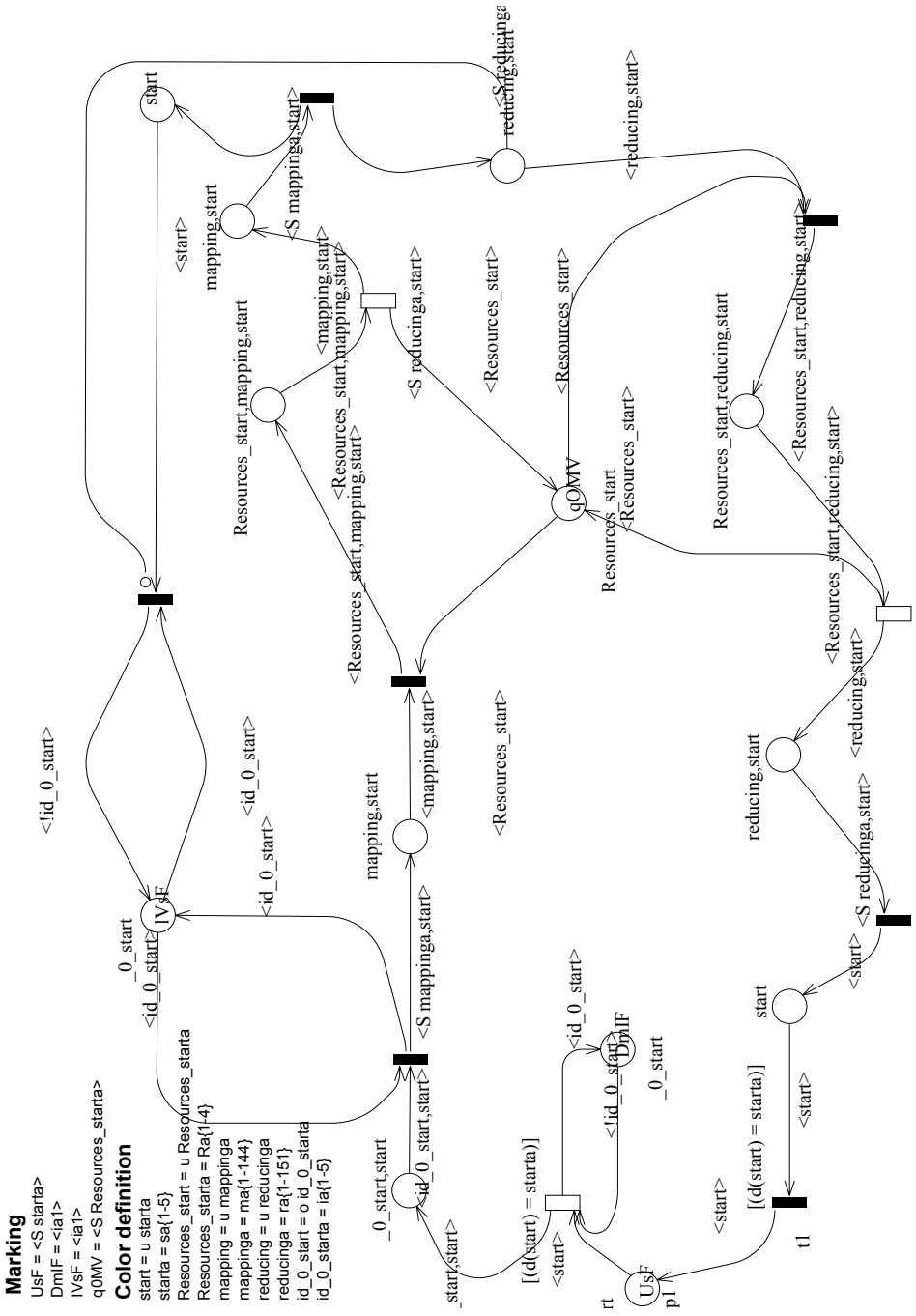

Fig. 16 Petri net automatically obtained

but it is a feature of the DICE Simulation tool [17], which allows defining ranges of values for variables for *what-if* analysis, as exercised in [31]. The results show that the response time remains constant up to the 15 statistics, which means that the DIA can support this workload without compromising performance. To confirm this conclusion, Figure 18 shows the utilization of the computing platform, also from the initial population of 5 up to 15. The utilization grows linearly and it never goes beyond 0.25, which explains that the response time keeps almost constant even for a population of 15 different statistics. For getting the results in Figure 18, we needed to add, in the UML model, the metric *utilization* to `<<HadoopScenario>>` stereotype, obviously.

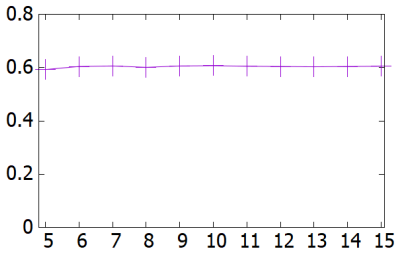

Fig. 17 Response time

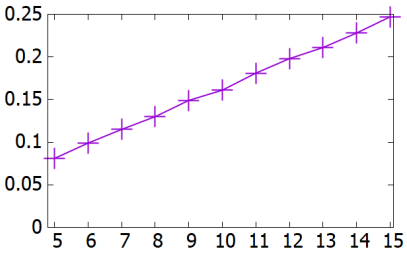

Fig. 18 Resource utilization

Stressing the Map operation Suppose that developers want to know the effects of re-programming the *Mapping* operation, so to get a more functional but demanding one. Then, we changed the value of the `hostDemand` attribute. Assuming that the complexity of a study can grow more easily than the number of different studies to compute, rather than tripling the initial value of 10ms, we now multiply the upper limit by 9. Thus, the value of the `hostDemand` will vary from the initial 10ms up to 90ms. Figure 19(a) shows the variation in the expected response time of the DIA, while Figure 19(b) the variation in the utilization of the computing platform. It can be assessed that both metrics will increase linearly with the increment in the demand of the *Mapping* operation.

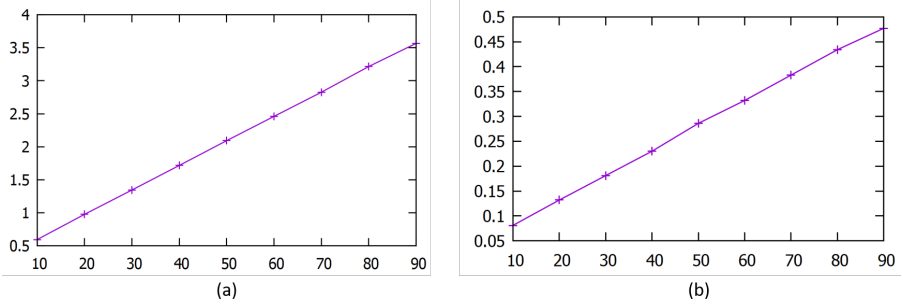

Fig. 19 (a) Response time and (b) utilization, when the demand of the *Mapping* increases

Increasing the workload and stressing the Map operation Figure 20 shows the results when both previous situations occur in combination. The response time, part (a), is sensible to the increments of the demand of *Mapping* operation, while the utilization, in part (b), is sensible to both variables and it increases linearly with them. These results show that the computing platform is powerful enough to support more intensive utilization and still offer good performance to users. For instance, other users running a Hadoop application different from Wikistats can also install it on the same Apache Hadoop computing platform.

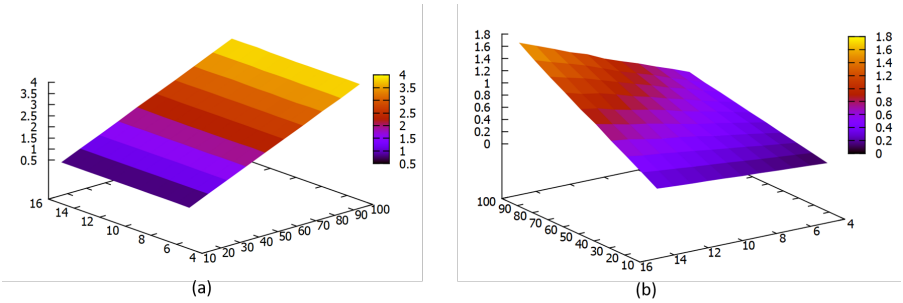

Fig. 20 (a) Response time and (b) utilization, in both scenarios

8.4 Validity of the results

Work in [2] presented a SWN modeling approach, for Apache Hadoop applications, that achieved below 20% of prediction error with respect to system real executions. The transformations implemented by the DICE Simulation tool come from the modeling proposal in [2]. Since the SWN in Figure 16 was automatically produced by the DICE Simulation tool, then we assume that the results obtained in the previous performance assessment present such accuracy levels. Although 20% of divergence is a significant error, we consider this prediction error still an acceptable value. One of the main reasons is that the models are not used to predict the application performance within the same interval as their parameters are calibrated. On the contrary, their parameters are calibrated using knowledge of the system behavior in a certain working interval (small values of inputs) and later their predictions are requested over a different working interval (large values of inputs, in order to foresee the performance of the production system using its full input). Providing accurate prediction results in these circumstances is a hard problem. Nevertheless, model-based evaluations allow carrying out several studies with different values of the parameters, and these results give insights of the tendency in the system performance. Therefore, even though the concrete figures from model-based evaluations under these constraints may hold significant errors, they

are valuable. We also acknowledge that this prediction error reported is not a hard-limit of the minimum accuracy of the approach but it would be possible to find a Hadoop application where this prediction error is trespassed.

8.5 Discussion on alternative performance languages

The quality assessment presented before used SWN models. However, we do not restrict the utilization of the profile for quality evaluation to SWN models. In this subsection, we discuss a number of representative performance assessment languages, selected from [26], belonging to different modeling languages families.

Queueing Networks (QN)[27]. While the most basic QN model does not allow to represent synchronisation of jobs, some extensions in the QN family do. Fork-Join QN class provides the primitive for the synchronisation of jobs, then allowing to model both the number of tasks in the Map and Reduce phases of a job, and the fact that its Reduce phase starts after all Map tasks have completed their execution. Extended QN class and their *passive nodes* can be used to restrict the concurrent access of new jobs to their Map phase before precedent jobs have obtained their resources for their Reduce phase. The concept of Finite Capacity Region allows to model that a resource is shared by more than one stations; in our case, this happens when Map and Reduce phases share the the same computing cores.

Petri nets. While the most basic place-transition Petri net allows to represent the concepts of concurrency and synchronisation, it does not allow to represent the concept of time. Extensions to Petri nets such as Stochastic Petri nets (SPN), Generalized SPN (GSPN) [1], or the used SWN are options that provide this concept. We have used SWN for this work, which is a rather elaborated member of the Petri nets family, although other Petri net models could have been used. SPNs can model the concept of time in their transitions by assigning them a probability distribution function that allow them to represent non-deterministic firing times. GSPNs generalise SPNs with immediate transitions, which make the DTSM simpler to transform –especially the modelling of the job scheduling logic as well as acquisition and release of resources– and more easily understandable the resulting performance model. SWNs extend GSPNs with some concepts such as *color* in tokens, arc functions and color conditions in transitions (Appendix D describes SWN in more detail). A SWN model can represent in a concise and compact model cases cases where a GSPN model shows symmetries. Therefore, the transformation to GSPNs of the Hadoop DTSM is possible and it would offer the same results (the simulation of GSPNs could be slower than the SWN, specially if the engine takes advantage of the *symbolic markings* in SWN), but the resulting model would become much larger and less visual, in particular when modelling that several jobs exist in the Hadoop platform.

Markov Chains. Using Continuous Time Markov chains (CTMC) it would be possible to represent the behavior and concepts represented in the DTSM of Hadoop. The CTMC would have a finite number of states since the Hadoop profile allows to model only closed workloads. However, the chain would need to use at least[12] one state for representing each possible state in the system (or a state for each possible marking in the Petri net model). Due to the concurrency and synchronisation present in the SWN model, the number of states in the CTMC would be exponential in the size of the Petri net. That is the main reason why we discarded the option of modelling with Markov chains.

Other formalisms that may be suitable for the performance evaluation of these Hadoop systems belong to the *Process algebras* family –for instance, PEPA nets [32] which is a formalism in the family of process algebras for the performance evaluation of systems– or to Stochastic Automata Networks [33]; but a deeper study should be carried out to give concrete recommendations for their utilization.

9 DDSM profile validation: Automatic deployment

This section carries out the validation of the DDSM profile. Such validation accomplishes our claim in the Introduction: *The profile disengages developers from knowing details on the complex tasks for continuously deploying the DIA.* Hence, we need to validate that the modeling capabilities of the DDSM profile allow to gather the information needed for achieving the deployment of DIA.

One of the manners to validate this claim is by showing that the information contained in a DDSM profiled UML diagram is enough for generating an Infrastructure-as-Code [42] (IaC) blueprint, and that such blueprint is accepted by a deployment executor, called *orchestrator*, that automates the deployment of the application based on the IaC blueprint. IaC is the practice of specifying the deployment infrastructure using human-readable notations. The IaC paradigm features: (a) domain-specific modeling languages (DSMLs) for Cloud application specification, such as the TOSCA[13] (*T*opology and *O*rchestration *S*pecification for *C*loud *A*pplications) standard, to program the way a Cloud application should be deployed; (b) *orchestrators*, that consume IasC blueprints and automate the deployment based on those IasC blueprints.

The DICER tool [64] is able to perform both tasks: the creation of IaC blueprints and the creation of the actual running platform from a blueprint. Therefore, we have used DICER to validate the DDSM profile through the example in Section 6. Actually, DICER tool allows to generate an IaC blueprint in TOSCA language from a UML deployment diagram annotated with the

[12] We say "at least" because we use Erlang-k distributions for the firing times, which are possible to be represented in CTMC, although increasing even further the number of states in function of the number of Erlang-k transitions and the value of k.

[13] TOSCA is a language to specify deployable blueprints in line with the IaC paradigm [43]. See Appendix C for TOSCA details.

Listing 1 DICER transformation pseudo-code.

```
1 for(UML::Device dev in UML::Model) {
2   if (dev.hasStereotype('DdsmVMsCluster')) {
3     generateHostNodeTemplate(dev);
4     for(UML::Node node in dev.nestedNodes) {
5       stereotypes := node.getAppliedStereotypes();
6       for (UML::Stereotype ster in stereotypes) {
7         if(ster.isSubStereotypeOf('InternalComponent') and ster.
              ↪ getProperty('protected')) {
8         generateFirewall(dev, node);
9       }}}
```

DDSM UML profile presented in Section 5. Regarding the actual deployment of the application, DICER comes with a customised version of the Cloudify[14] orchestration engine, which supports many of the most popular DIA technologies and is able to automatically deploy the generated IasC TOSCA blueprints.

Let us consider the UML deployment diagram shown in Figure 21, which uses the DDSM profile to model the deployment of the Hadoop Wikistats application. Given this UML-profiled model, the execution of the DICER tool is able to automatically generate the deployable TOSCA blueprint reported in Listing 2. The DDSM profile allows to properly model the deployment of the Hadoop cluster, allowing the designer to dimension the cluster, setting firewalls and configure the Big Data platforms in all their major configuration parameters.

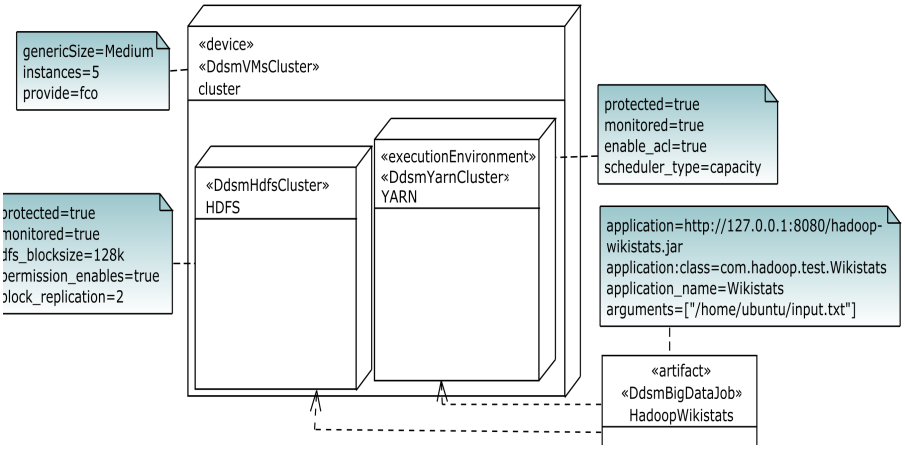

Fig. 21 The DDSM model for the deployment the Hadoop Wikistats sample application.

Listing 1 reports in pseudo-code an excerpt of the DICER transformation workflow. It transforms each `DdsmVMsCluster` into a TOSCA node template

[14] http://cloudify.co/

like the `cluster` node template (see Listing 2, line 4), which is generated starting from the *cluster* element with `DdsmVmsCluster` stereotype of Figure 21. The type of such node template (line 5) depends on the *genericSize* property of the `DdsmVMsCluster` stereotype. The number of VM instances (line 6) and the selected Cloud provider (line 13) correspond to the properties `instances` and `provider` respectively of the `DdsmVmsCluster` stereotype (see again Figure 21). For each *protected* `Node` contained into a `DdsmVMsCluster Device` (see the `protected` property of the `DdsmInternalComponent` stereotype), the transformation generates a set of node templates (which depends on the actual Big Data platform being deployed) from the package `dice.firewall_rules.*` (Listing 2, lines 30-33 and 56-59) and binds these with the generated `dice.host.*` node template using a `dice.relationships.ProtectedBy` relation (Listing 2, lines 8 and 9). Similarly, the TOSCA library provides specific relationships to configure dependencies among Big Data technologies. For instance the Hadoop package provides the dice.relationships.hadoop.ConnectedToNameNode relationship, which allows to specify the connection of a HDFS DataNode to the corresponding NameNode (see line 27), or
the `dice.relationships.hadoop.ConnectedToResourceManager`, which allows to connect a YARN NodeManager to the corresponding ResourceManager (see line 53). The transformation implemented in DICER ensures that all the elements of a model are properly instantiated into the corresponding IsaC.

By parsing the generated blueprint, the customised version of Cloudify provided by DICER gets a global view of the application and automatically derives and executes, based on dependencies between nodes, a suitable execution plan. As a result, no node is deployed until all the nodes it depends on have been deployed first. Moreover Cloudify configures each node according to what has been specified in the model through the DDSM profile. In conclusion, our experience in the development of the WikiStats example shows that the adoption of the DDSM profile enables a good level of disengagement of users from the details of infrastructural code frameworks. Indeed, while the profile allows users to fine tune the configuration parameters of the needed Big Data frameworks, at the same time it also provides a default configuration for each of them, which can be directly used without any adjustment. The provided abstractions, i.e. MasterSlavePlatform and PeerToPeerPlatform, do also contribute in achieving such disengagement. Without these abstractions the user should explicitly model all the various virtual machines and their corresponding configuration (e.g. size, amount of physical resources, etc.) one by one, which is repetitive and error-prone. Moreover this is further complicated by the fact that each platform has specific deployment aspects that the profile abstracts from, while still letting the user to customize them when needed. Just to make an example, Apache Cassandra [15], a popular NoSQL database working in a peer-to-peer fashion, requires a special *seed* node to be deployed, which supports the bootstrapping of all the peer nodes. The DDSM profile user is not required to know anything about this technology specific aspect,

[15] http://cassandra.apache.org/

neither she is required to specify anything about the *seed* node, for which a default deployment configuration is provided. Still, the part of the profile which concerns Cassandra allows to fine tune the deployment of the *seed* node is desired, for instance by specifying a dedicated VM for it. Another example of how the DDSM profile disengages users from IaC details are firewalls configurations, for which the profile provides ready to use defaults, which can still be tuned by the user. Finally it is worth to remind that the DDSM profile comes, thanks to UML, as a general language for describing DIA deployments and as an abstraction of top of the various IaC languages and implementations. In DICER, the profile has been used to enable the generation of TOSCA code working for a customised version of the Cloudify orchestrator. Nothing prevents the future development of other code generators that, starting from the DDSM profile, target different TOSCA implementations or even different infrastructural languages.

The main advantage provided by the DDSM profile in the context of the DICER tool resides in the integration it enables between the typical design-level activities and automated deployment, thus leading to the adoption of a DevOps flavor. In fact, by using simple models to describe the deployment of complex infrastructures and by automating the deployment starting from such models, we have experienced considerable gains both in terms of reasoning on and documenting our architecture, as well as in terms of time saved when deploying and re-deploying it.

Listing 2 An excerpt of the DICER generated TOSCA blueprint for an Apache Hadoop cluster.

```
# Imports, inputs and outputs sections omitted.
tosca_definitions_version: cloudify_dsl_1_3
node_templates:
  cluster:
    type: dice.hosts.Medium
    instances: {deploy: 5}
    relationships:
    - {type: dice.relationships.ProtectedBy, target: HDFS_firewall}
    - {type: dice.relationships.ProtectedBy, target: YARN_firewall}
    - {type: dice.relationships.IPAvailableFrom, target: cluster_ip}
    properties:
      monitoring: {enabled: false}
      provider: openstack
  cluster_ip:
    type: dice.VirtualIP
  HDFS_master:
    type: dice.components.hadoop.NameNode
    relationships:
    - {type: dice.relationships.ContainedIn, target: HDFS_master_vm}
    properties:
      monitoring: {enabled: true}
      configuration: {'dfs.blocksize': '128k', 'dfs.permissions.enabled':
          ↪ 'true', 'dfs.replication': '2'}
  HDFS:
    type: dice.components.hadoop.DataNode
    relationships:
    - {type: dice.relationships.ContainedIn, target: cluster}
    - {type: dice.relationships.hadoop.ConnectedToNameNode, target:
        ↪ HDFS_master}
    properties:
      monitoring: {enabled: true}
  HDFS_master_firewall:
```

```
31      type: dice.firewall_rules.hadoop.NameNode
32    HDFS_firewall:
33      type: dice.firewall_rules.hadoop.DataNode
34    HDFS_master_vm:
35      type: dice.hosts.Medium
36      instances: {deploy: 1}
37      relationships:
38      - {type: dice.relationships.ProtectedBy, target: HDFS_master_firewall
            ↪ }
39      - {type: dice.relationships.IPAvailableFrom, target:
            ↪ HDFS_master_vm_ip}
40    HDFS_master_vm_ip:
41      type: dice.VirtualIP
42    YARN_master:
43      type: dice.components.hadoop.ResourceManager
44      relationships:
45      - {type: dice.relationships.ContainedIn, target: YARN_master_vm}
46      properties:
47        monitoring: {enabled: true}
48        configuration: {'yarn.acl.enable': 'true', 'yarn.resourcemanager.
              ↪ scheduler.class': 'org.apache.hadoop.yarn.server.
              ↪ resourcemanager.scheduler.capacity.CapacityScheduler'}
49    YARN:
50      type: dice.components.hadoop.NodeManager
51      relationships:
52      - {type: dice.relationships.ContainedIn, target: cluster}
53      - {type: dice.relationships.hadoop.ConnectedToResourceManager, target
            ↪ : YARN_master}
54      properties:
55        monitoring: {enabled: true}
56    YARN_master_firewall:
57      type: dice.firewall_rules.hadoop.ResourceManager
58    YARN_firewall:
59      type: dice.firewall_rules.hadoop.NodeManager
60    YARN_master_vm:
61      type: dice.hosts.Medium
62      instances: {deploy: 1}
63      relationships:
64      - {type: dice.relationships.ProtectedBy, target: YARN_master_firewall
            ↪ }
65      - {type: dice.relationships.IPAvailableFrom, target:
            ↪ YARN_master_vm_ip}
66    YARN_master_vm_ip:
67      type: dice.VirtualIP
68    hadoopApplication:
69      type: dice.components.yarn.Topology
70      relationships:
71      - {type: dice.relationships.Needs, target: YARN}
72      - {type: dice.relationships.yarn.SubmittedBy, target: YARN_master}
73      - {type: dice.relationships.Needs, target: HDFS}
74      properties:
75        arguments:
76        - get_attribute: [HDFS, ip]
77        - '/home/ubuntu/input.txt'
78        application: http://127.0.0.1:8080/hadoop-wikistats.jar
79        topology_class: com.hadoop.test.Wikistats
80        topology_name: Wikistats
```

10 Related work

Model-Driven Engineering is nowadays a well-established discipline to support conception, design, assessment, and development of software in various application domains. The variety of domains is witnessed by initiatives like

AUTOSAR [48] that focuses specifically on automotive. In the DIA context, authors of [45] focus on modelling for the purpose of application code generation in the Hadoop framework. Similar support is offered by Stormgen [49], a DSL for Storm-based *topologies*. Both approaches focus on a single technology and do not address issues such as the analysis and the deployment of DIAs. Juniper [35] focuses on both, application code generation and deployment aspects, but assumes that DIAs run on specifically developed real-time Java VMs and exploit MongoDB and PostgreSQL as databases. Compared to this approach, our DIA profile aims at supporting a number of technologies enabling parallel computation and allows these technologies to be combined to build complex DIAs.

As for the quantitative assessment aspect, our approach relies on the ongoing effort on MARTE and DAM and extends them to DIAs, which confers a standardized framework not present in other works already presented in the literature. For example, a generic profile for the modelling of big data applications is given in the context of the Palladio Component Model [36], while in [37], the authors model and simulate Apache Spark streaming applications also for Palladio, although they did not focus on batch operations as our Apache Spark DTSM profile. Mathematical models for predicting the performance of Apache Spark applications are introduced in [67]. The work in [46] discusses the role of modelling and performance assessment in big data platforms.

For what concerns deployment, our work builds on top of Infrastructure as Code approaches [42] that are based on the idea that the deployment configuration of a complex system can be coded and executed by a proper orchestrator. Essentially, pieces of code like Chef recipes[16] or TOSCA blueprints [68][40] define a model of the system to be deployed. Compared to these, our DIA profile and our DICER tool enable those in charge of defining the deployment configuration code to work at a higher level of abstraction in a way that is fully integrated with a typical UML-based design context. With this main difference in mind, the work that appears to be closest to our deployment approach is Ubuntu Juju[17]. It offers a framework for orchestrating the deployment of complex systems in a cloud context. Moreover, it offers a graphical Web user interface for building models of cloud applications and a repository, called Charms Store, containing buy-per-use building blocks called Charms for a variety of use cases, including Big Data. Juju supports describing applications in YAML documents called bundles, which can be used in IasC. However, Juju's bundles are particular to Juju's own orchestrator, while in our approach we rely on a portable and orchestration-neutral TOSCA language. Also, Juju's GUI is aimed at Ops staff, while DICER is embedded into the Eclipse IDE to provide for integrated Dev functionalities at the same time.

[16] https://www.chef.io/

[17] `https://jujucharms.com/`

11 Conclusion and future work

This work has presented, to the best of our knowledge, the first domain specific modeling language (DSML) for developing data-intensive applications (DIAs). Our DSML has been designed as a UML profile so to leverage standard languages and practices. Inspired by the OMG-MDA initiative, the DIA profile encompasses the three common abstraction levels in mature MDE approaches, in our case called DPIM, DTSM and DDSM. The DPIM level to assist developers in the early architecture design, by identify the key concepts in DIA development. The DTSM level for evaluating the quality of the architecture design, while taming the jungle of Big Data frameworks and the jungle of concepts behind each one of them. The DDSM level to automate the intricate but key task of deploying the application in the cloud.

The painstaking technical work of designing a profile, with the characteristics of ours, flourishes at the level of the technology. We have addressed four of the most relevant Big Data frameworks today, i.e., Apache HadoopMR, Apache Storm, Apache Spark and Apache Tez. So, we needed to conceptualise from stream to batch processing, also from in-memory to realtime processing. Obviously, for space reasons, we could not show up all the material we produced, but all the technical work is available at [60,62]. While, the reader can find published the Apache Storm profile, not here reported, in [29].

The work in this manuscript has been useful to comprehensively present, for first time, the DIA profile. We think that this work helps to understand the foundations of the profile as well as the design decisions we needed to take. Regarding the important aspect of the profile validation, we have tried to manage the space to showcase one technology at the DTSM level and the whole DDSM level.

Much work can be built around the DIA-profiled UML models and their links to deployment artifacts and performance models. Some of the research paths we believe that should be explored are the automatic code generation from the profiled UML models, a smoother transition for the engineer between DPIM, DTSM and DDSM models by automatic generation of skeletons, the utilization of design and performance models at runtime to detect changes in the operational profile of the DIA, report the detected changes to engineers, and also self-adapt the computing infrastructure to the most convenient number of resources, to name a few.

Acknowledgements This work is supported by the European Commission grant no. 644869 (H2020, Call 1), DICE. D. Perez-Palacin, J. Merseguer and J.I. Requeno have been supported by the project CyCriSec [TIN2014-58457-R] and Aragon Government Ref. T27-DISCO research group.

A MARTE and DAM profiles

MARTE [41] is a standard profile that extends UML for the performance and schedulability analysis of a system. MARTE consists of three main parts: *MARTE Foundations*, *MARTE*

Design Model and *MARTE Analysis Model*. The *Analysis Model* is of our interest since it enables the QoS assessment by allowing the definition of QoS metrics and properties. The Analysis Model consists of a *Generic Quantitative Analysis and Modelling* (GQAM) profile and its specialization, the *Performance Analysis and Modelling* (PAM) profile. In addition to this, two other features are also important for our DIA profile.

The first one is that MARTE enables the specification of quantitative non-functional properties (NFP) in UML models through its Value Specification Language (VSL). The VSL is useful for specifying the values of constraints, properties, and stereotype attributes, particularly related to NFPs. Moreover, VSL allows to express basic types, data types, values (such as time and composite values), as well as variables, constants and expressions. This means that, using VSL we can define complex metrics and requirements to express for example response times, utilizations or throughputs. MARTE also defines a library of primitive data types, a set of predefined NFP types and units of measures. Hence, our DIA profile inherits the VSL altogether.

For understanding the VSL expressions that appear in this paper, it is of interest to briefly recall its syntax. An example of VSL expression for a `host demand` tagged value of type `NFP_Duration` is:

```
expr=6 unit=ms, statQ=mean, source=est
 (1)       (2)        (3)        (4)
```

This expression specifies that the **Reducing** activity in Figure 13, demands `6` (1) *milliseconds* (2) of processing time, whose mean value (3) is obtained from an estimation in the real system (4). We could replace, for example, the value `6` for a variable `$`*host_dem* to parameterise the analysis of the model with different values for this host demand.

The second feature is that the DAM [6] profile specializes MARTE-GQAM for dependability analysis (i.e., availability, reliability, safety and maintainability). Consequently, the DAM profile also inherits the VSL. As MARTE, DAM consists of a library and a set of extensions to be applied at model specification level. Our DIA profile inherits DAM with the purpose of addressing reliability analysis for DIA.

B DIA Profile Library

In this Appendix we present the DIA library. The library defines the data types, basic and complex, used in the attributes of the stereotypes proposed for the three abstraction levels, DPIM, DTSM and DDSM. Basic types appear in Figure 22, while complex ones in Figure 23. From DAM we have imported the *DAM Library* [6], which also imports the *MARTE Library* [41].

C TOSCA

TOSCA provides a flexible and highly extensible DSL for modelling resources and software components. TOSCA *blueprints* are executable IasC composed of *node templates* and *relationships*, defining the *topology* of a hardware/software systems. Node templates and relationships are instances of *node types* and *relationship types*, that are either normative (i.e., defined in the standard), provided by the specific engine that executes a blueprint (the orchestrator), or an extension of one of the above, such as in our case, with DIA-specific node and relationship types. Node types are essentially used to describe hardware or virtual resources (machines or VMs) and software components. Relationship types predicate on the association between node types. For instance, a TOSCA node type representing Wordpress CMS must be associated to a node type presenting VMs through the relationship *hosted_On*. Each node type and relationship type also enables specifying *interfaces*, which are composed of operations that have to be carried out at specific stages of the deployment orchestration. Typical examples of interface operations include installing, configuring or starting of components, and may take form of Python/bash scripts, or pointers to Chef recipes. Node

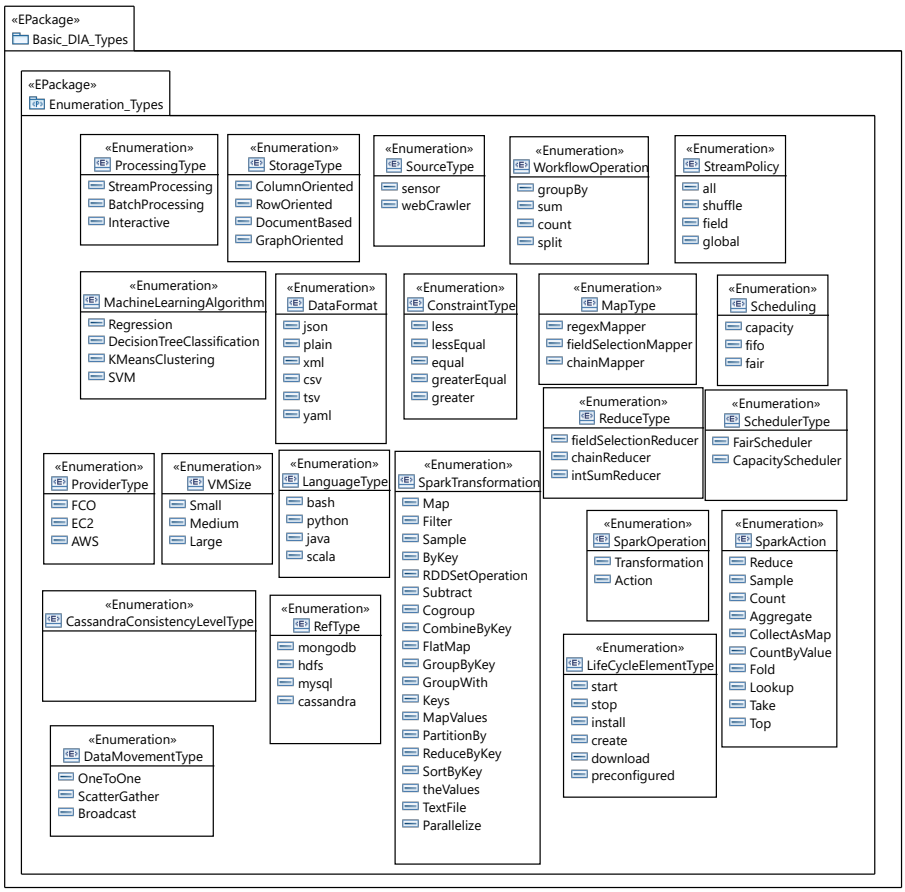

Fig. 22 DIA basic data types

and relationship templates are free to provide their own interface operations, extending or overriding behaviour defined in the corresponding types. TOSCA is being supported by a number of orchestrators that, given a TOSCA blueprint and all node and relationship types used there, are able to execute it deploying the corresponding system and managing its life-cycle. Examples of such orchestrators are Cloudify[18], ARIA TOSCA[19], Indigo[20], Apache Brooklyn[21] or ECoWare [4].

D Transformation of a DTSM design to a performance model

Stochastic Well-formed Nets (SWN) [12] are a modeling formalism suitable for performance analysis purposes. A SWN model is a bipartite graph formed by places and transitions. Places are graphically depicted as circles and may contain tokens. A token distribution in the places

[18] `http://getcloudify.org`
[19] `http://ariatosca.org/`
[20] `https://www.indigo-datacloud.eu/`
[21] `https://brooklyn.apache.org/learnmore/`

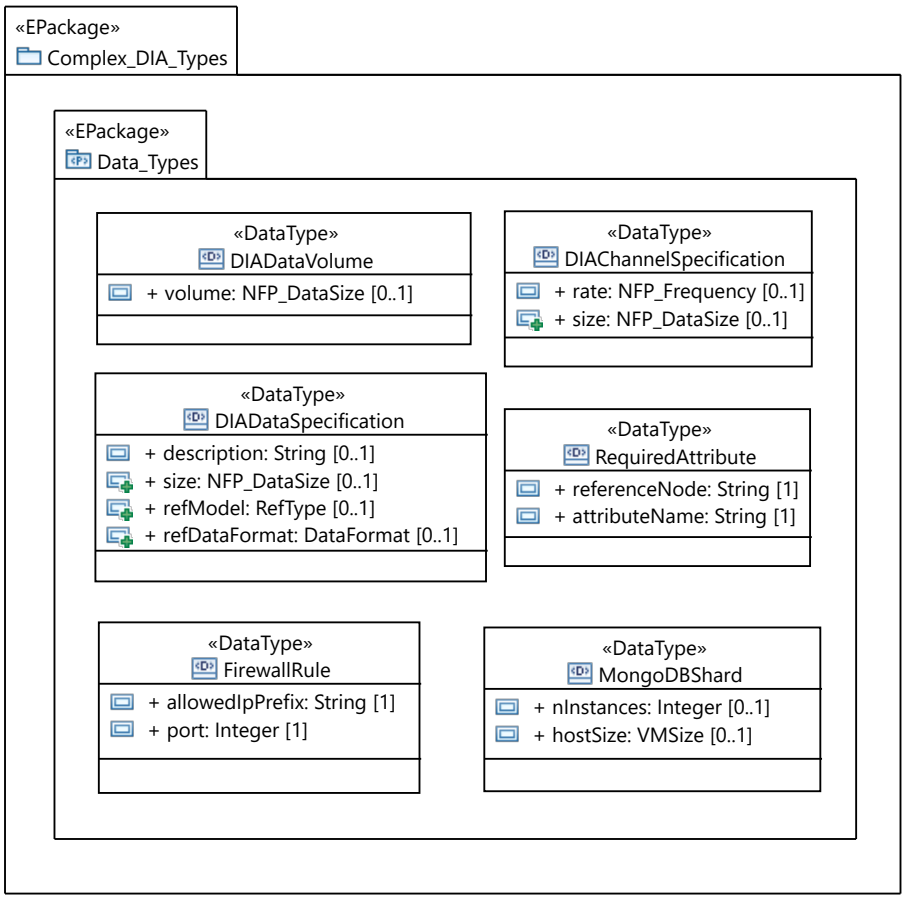

Fig. 23 DIA complex data types

of a SWN, namely a marking, represents a state of the modeled 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 random variable with a given probability distribution function. Immediate transitions are graphically depicted as black thin bars while timed ones are depicted as white thick bars. Tokens may also have an associated *color*, i.e., a *data type*, which enriches the expressiveness of the net and restricts the movement of tokens to compatible places and transitions.

Figure 24 depicts a schema of how Apache Hadoop stereotypes in UML-profiled models (left) are transformed into an analyzable model such as a SWN (right). Each stereotype is transformed into a subnet by taking into account the information contained in the tags. For each transformation pattern in the Figure, the part of the Petri net inside the blue box corresponds to the part that the transformation creates. The part of the Petri net outside the blue box corresponds to referenced parts, which are in turn created by other stereotypes. Figure 24 depicts only the specific non-functional annotations for Apache Hadoop, the functional part of the UML diagram is transformed according to the works in [28, 71]. Eventually, all the subnets are composed into a single closed Petri net such as in Figure 16.

A Hadoop cluster accepts several categories of users, whose jobs are probably sub-divided into a different number of map-reduce tasks or have assigned a different number of hardware resources. Every user $< i >$ has $\$nC_i$ jobs waiting in the scheduler queue. Hadoop scheduler launches periodically a new job at a given $\$rate$ following a scheduling policy defined by the scenario (e.g., a shared common FIFO queue for all users). By default, our transformation assumes an independent FIFO queue for each user; and always guarantees to take a job of each user.

Jobs are labelled with the user they belong to (loop $< i >$-$< i + 1 >$ in the net, where $< i >$ represents each user). The scheduler waits for the assignment of resources to all tasks in the reduce phase of the precedent job $< i >$ before launching the next job $< i + 1 >$ (inhibitor arc section). This scheduling allows both concurrency among jobs and giving priority over resources to precedent jobs. Job $< i >$ is divided in $\$m_i$ map tasks and $\$r_i$ reduce tasks, that run simultaneously in up to $\$p_i$ cores ($\sum_{i=1}^{n} \$p_i \geq \$host$, being $\$host$ the total number of cores in the cluster). We use the notation $< i >$ for expressing the color of a token. For instance, each user is represented by a different color in the SWN. Notation $\$m_i$ is used for expressing numerical values; for instance, the number of map tasks in which a job of type $< i >$ is divided.

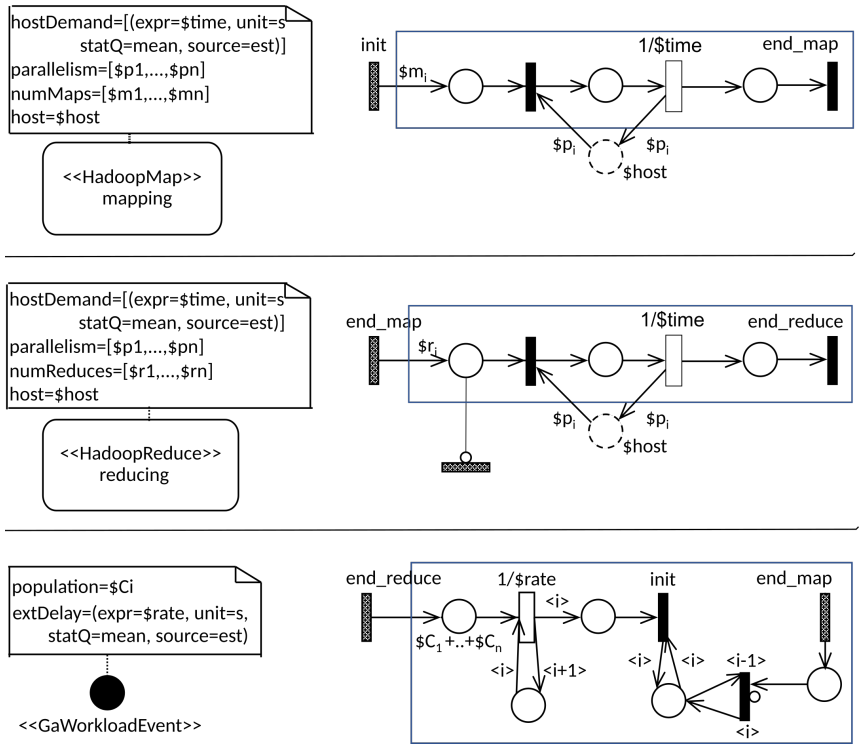

Fig. 24 Transformation of Apache Hadoop specific stereotypes to Petri nets

E Usability of the profile

The validation of the DIA profile has been carried out so far from the point of view of its adequacy to solve the QoS assessment and the deployment. However, we consider also important to learn about the usability of the profile, in terms of easiness of use for engineers. It uses to happen that tools, although offering the required functionalities, do not reach their expectations until a degree of maturity is accepted at this regard.

The DIA profile has been used by engineers in four organizations: Prodevelop[30], ATC [24], BluAge [25] and XLAB R&D [34]. We have prepared eight questions, see Table E, for a total of eight engineers, who have extensively used the DIA profile in the context of the DICE project to carry out industrial applications. From the answers, we see that the profile has been useful for the engineers, specially for the automatic deployment. However, the main lack refers to the Papyrus implementation (see question #6) that also constraints the profile implementation. In fact, the advice of the engineers (see question #8) referred to improve the Papyrus implementation of the profile.

#	Question	Choice	Answers
1	Had you previous experience with UML profiles before using the DIA profile?	Yes, No	4 Yes 4 No
2	Did the DIA profile help you to better understand the architecture of your Data Intensive Application?	Yes, No, Neutral	8 Yes
3	The DIA profile intended to cover the architectural modeling needs of your DIA. Please rate the profile at this regard.	1 to 5 (5 is top)	4 (mean)
4	The DIA profile intended to cover the modeling needs of your DIA for QoS assessment. Please rate the profile at this regard.	1 to 5	4 (mean)
5	The DIA profile intended to cover the deployment needs of your DIA. Rate the profile at this regard.	1 to 5	4.5 (mean)
6	Rate the Papyrus implementation of the UML standard on top of the Eclipse platform in terms of usability (friendly of use)?	1 to 5	3.5 (mean)
7	Rate the DIA profile implementation on top of Papyrus in terms of usability (friendly of use)?	1 to 5	4 (mean)
8	Please, if possible, offer some short advice on usability of the DIA profile.	Text	

Table 4 Research questions on profile usability

References

1. M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1994.
2. 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*, pages 599–613. Springer International Publishing, Cham, 2016. URL: `http://dx.doi.org/10.1007/978-3-319-49583-5_47`, `doi:10.1007/978-3-319-49583-5_47`.
3. Danilo Ardagna, Elisabetta Di Nitto, Giuliano Casale, Dana Petcu, Parastoo Mohagheghi, Sébastien Mosser, Peter Matthews, Anke Gericke, Cyril Ballagny, Francesco D'Andria, Cosmin-Septimiu Nechifor, and Craig Sheridan. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds.

In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, MiSE '12, pages 50–56, Piscataway, NJ, USA, 2012. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=2664431.2664439.

4. L. Baresi, S. Guinea, G. Quattrocchi, and D. A. Tamburri. Microcloud: A container-based solution for efficient resource management in the cloud. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 218–223, Nov 2016. doi:10.1109/SmartCloud.2016.42.
5. Gordon Bell, Tony Hey, and Alex Szalay. Beyond the data deluge. *Science*, 323(5919):1297–1298, 2009.
6. S. Bernardi, J. Merseguer, and D.C. Petriu. A dependability profile within MARTE. *Software and Systems Modeling*, 10(3):313–336, 2011.
7. S. Bernardi, J. Merseguer, and D.C. Petriu. *Model-driven Dependability Assessment of Software Systems*. Springer, 2013.
8. Simona Bernardi, José Ignacio Requeno, Christophe Joubert, and Alberto Romeu. A systematic approach for performance evaluation using process mining: The posidonia operations case study. In *Proceedings of the 2Nd International Workshop on Quality-Aware DevOps*, QUDOS 2016, pages 24–29, New York, NY, USA, 2016. ACM. URL: http://doi.acm.org/10.1145/2945408.2945413, doi:10.1145/2945408.2945413.
9. G. Casale et al. DICE: Quality-driven Development of Data-intensive Cloud Applications. In *Proceedings of the Seventh International Workshop on Modeling in Software Engineering*, pages 78–83, NJ, USA, 2015. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=2820489.2820507.
10. K. Chandrasekaran, Siddharth Santurkar, and Abhishek Arora. Stormgen - a domain specific language to create ad-hoc storm topologies. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *FedCSIS*, pages 1621–1628, 2014. URL: http://dblp.uni-trier.de/db/conf/fedcsis/fedcsis2014.html#ChandrasekaranSA14.
11. C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314 – 347, 2014.
12. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Trans. Comput.*, 42(11):1343–1360, November 1993. URL: http://dx.doi.org/10.1109/12.247838, doi:10.1109/12.247838.
13. Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2001.
14. Constantine Aaron Cois, Joseph Yankel, and Anne Connell. Modern devops: Optimizing software development through effective system interactions. In *IPCC*, pages 1–7. IEEE, 2014. URL: http://dblp.uni-trier.de/db/conf/ipcc/ipcc2014.html#CoisYC14.
15. Mathieu Colas, Ingo Finck, Jerome Buvat, Roopa Nambiar, and Rishi Raj Singh. Cracking the data conundrum: How successful companies make big data operational. Technical report, Capgemini consulting, 2015. URL: https://www.capgemini-consulting.com/cracking-the-data-conundrum.
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. DICE simulation tools. Technical report, European Union's Horizon 2020 research and innovation programme, 2017. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/08/D3.4_DICE-simulation-tools-Final-version.pdf.
18. Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011.
19. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: A flexible data processing tool. *Commun. ACM*, 53(1):72–77, January 2010.
20. Elisabetta Di Nitto, Peter Mattew, Dana Petcu, and Arnor Solberg, editors. *Model-Driven Development and Operation of Multi-Cloud Applications*. PoliMI SpringerBriefs. Springer International Publishing, 2017.

21. Dipartamento 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.
22. Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, Diego Perez-Palacin and Damian Andrew Tamburri. Infrastructure-as-code for data-intensive architectures: A model-driven development approach. In *IEEE International Conference on Software Architecture, ICSA 2018, Seattle, WA, USA, April 30 - May 4, 2018*, pages 156–165. IEEE Computer Society, 2018. URL: https://doi.org/10.1109/ICSA.2018.00025, doi:10.1109/ICSA.2018.00025.
23. 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*, QUDOS 2016, pages 18–23, New York, NY, USA, 2016. ACM. URL: http://doi.acm.org/10.1145/2945408.2945412, doi:10.1145/2945408.2945412.
24. ATC. Athens Technology Center Website, 2018. URL: https://www.atc.gr/default.aspx?page=home.
25. Blu Age. Blu Age, Make IT Digital, 2018. URL: https://www.bluage.com.
26. D.C. Petriu, M. Alhaj, R. Tawhid. *Software Performance Modeling*, volume 7320 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2012.
27. E.D. Lazowska, J. Zahorjan, G. Scott Graham, and C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network models*. Prentice-Hall, 1984.
28. J. P. López-Grao, J. Merseguer and J. Campos. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In *Proceedings of the 4th International Workshop on Software and Performance*, WOSP '04, pages 25–36, New York, NY, USA, 2004. ACM. URL: http://doi.acm.org/10.1145/974044.974048, doi:10.1145/974044.974048.
29. José I. Requeno, José Merseguer, Simona Bernardi, Diego Perez-Palacin, Giorgos Giotis and Vasilis Papanikolaou. Quantitative Analysis of Apache Storm Applications: The NewsAsset Case Study. *Information Systems Frontiers*, 2018. Accepted for publication. doi:10.1007/s10796-018-9851-x.
30. Prodevelop. Prodevelop- Integrating Tech, 2018. URL: https://www.prodevelop.es/en.
31. S. Bernardi, J.L. Dominguez, A. Gómez, C. Joubert, José Merseguer, D. Perez-Palacin, J.I. Requeno and A. Romeu. A systematic approach for performance assessment using process mining. *Empirical Software Engineering*, 2018. Accepted for publication. doi:10.1007/s10664-018-9606-9.
32. Stephen Gilmore, Jane Hillston, Lela Kloul and Marina Ribaudo. Pepa nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79 – 104, 2003. doi:https://doi.org/10.1016/S0166-5316(03)00069-5.
33. W.H. Sanders, J.F. Meyer. *Stochastic Activity Networks: Formal Definitions and Concepts*, volume 2090 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2001.
34. XLAB. XLAB, R&D, 2018. URL: https://www.xlab.si.
35. Juniper Project. Experimental: Models for big data stream processing, 2015. Juniper Project Tutorial. URL: http://forge.modelio.org/projects/juniper/wiki/Tutorial_on_Models_for_Big_Data_stream_processing.
36. Johannes Kroß, Andreas Brunnert, and Helmut Krcmar. Modeling Big Data Systems by Extending the Palladio Component Model. *Softwaretechnik-Trends*, 35(3), 2015.
37. Johannes Kroß and Helmut Krcmar. Modeling and Simulating Apache Spark Streaming Applications. *Softwaretechnik-Trends*, 36(4), 2016.
38. François Lagarde, Huáscar Espinoza, François Terrier, and Sébastien Gérard. Improving UML profile design practices by leveraging conceptual domain models. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), Atlanta (USA)*, pages 445–448. ACM, November 2007.
39. M. Langheinrich. Privacy by design. In G.D. Abowd, B. Brumitt, and A. Shafer, editors, *UBICOMP 2001*, pages 273–291. Springer, 2001.
40. Paul Lipton, Derek Palma, Matt Rutkowski, and Damian A. Tamburri. TOSCA solves big problems in the cloud and beyond. *IEEE Cloud, To appear*, 21(11):31–39, 2016.

41. UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, June 2011. Version 1.1, OMG document: formal/2011-06-02.
42. K. Morris. *Infrastructure As Code: Managing Servers in the Cloud*. Oreilly & Associates Incorporated, 2016.
43. Derek Palma, Matt Rutkowski, and Thomas Spatzier. Tosca simple profile in yaml version 1.0. Technical report, OASIS Committee Specification, http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/cs01/TOSCA-Simple-Profile-YAML-v1.0-cs01.html, 2016.
44. Diego Perez-Palacin, Youssef Ridene, and José Merseguer. Quality assessment in devops: Automated analysis of a tax fraud detection system. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, pages 133–138, New York, NY, USA, 2017. ACM.
45. A. Rajbhoj, V. Kulkarni, and N. Bellarykar. Early experience with model-driven development of mapreduce based big data application. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, volume 1, pages 94–97, Dec 2014. doi:10.1109/APSEC.2014.23.
46. Rajiv Ranjan. Modeling and Simulation in Performance Optimization of Big Data Processing Frameworks. *IEEE Cloud Computing*, 1(4):14–19, 2014.
47. Jose-Ignacio Requeno, José Merseguer, and Simona Bernardi. Performance Analysis of Apache Storm Applications Using Stochastic Petri Nets. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 411–418, 2017. URL: http://ieeexplore.ieee.org/document/8102965/, doi:10.1109/IRI.2017.64.
48. Guido Sandmann and Richard Thompson. Development of autosar software components within model-based design. SAE Technical Paper, 04 2008. doi:10.4271/2008-01-0383.
49. S. Santurkar, A. Arora, and K. Chandrasekaran. Stormgen - a domain specific language to create ad-hoc storm topologies. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1621–1628, Sept 2014. doi:10.15439/2014F278.
50. Markus Scheidgen and Anatolij Zubow. Map/reduce on emf models. In *MDH-PCL@MoDELS*, page 7. ACM, 2012. URL: http://dblp.uni-trier.de/db/conf/models/mdhpcl2012.html#ScheidgenZ12.
51. Bran Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007), 7-9 May 2007, Santorini Island, Greece*, pages 2–9. IEEE Computer Society, 2007.
52. Bran Selic and Sebastien Gerard, editors. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE*. Morgan Kaufmann, Boston, 2014.
53. Connie U. Smith and Lloyd G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.
54. The Apache Software Foundation. Apache Cassandra. URL: http://cassandra.apache.org/.
55. The Apache Software Foundation. Apache Hadoop. URL: http://hadoop.apache.org/.
56. The Apache Software Foundation. Apache Kafka. URL: http://kafka.apache.org/.
57. The Apache Software Foundation. Apache Spark. URL: http://spark.apache.org/.
58. The Apache Software Foundation. Apache Storm. URL: http://storm.apache.org/.
59. The Apache Software Foundation. Apache Tez. URL: http://tez.apache.org/.
60. The DICE Consortium. DICE Models Repository, Jan., 2017. URL: https://github.com/dice-project/DICE-Models.
61. The DICE Consortium. DICE Simulation tool, Oct., 2017. https://github.com/dice-project/DICE-Simulation.
62. The DICE Consortium. DICE Profiles Repository, Sep., 2017. URL: https://github.com/dice-project/DICE-Profiles.
63. The DICE Consortium. DICE Profiles, Sept., 2017. https://github.com/dice-project/DICE-Profiles.
64. The DICE Consortium. DICE-Rollout, Sept., 2017. https://github.com/dice-project/DICER.
65. The Object Management Group (OMG). Model-Driven Architecture Specification and Standardisation. Technical report, , 2018. URL: http://www.omg.org/mda/.

66. Unified Modeling Language: Infrastructure, 2017. Version 2.5.1, OMG document: formal/2017-12-05.
67. Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for Apache Apark platform. In *2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS)*, pages 166–173. IEEE, 2015.
68. J. Wettinger, U. Breitenbücher, and F. Leymann. Standards-based devops automation and integration using tosca. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 59–68, Dec 2014. doi:10.1109/UCC.2014.14.
69. WikiMedia project. Wikistats, Dec., 2016. https://www.mediawiki.org/wiki/Analytics/Wikistats.
70. Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*, pages 1–33, 2005.
71. C. Murray Woodside, Dorina C. Petriu, José Merseguer, Dorin Bogdan Petriu, and Mohammad Alhaj. Transformation challenges: from software models to performance models. *Software and System Modeling*, 13(4):1529–1552, 2014. URL: http://dx.doi.org/10.1007/s10270-013-0385-x, doi:10.1007/s10270-013-0385-x.