Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements

Design and quality abstractions - Final version

Deliverable 2.2

Deliverable:	D2.2
Title:	Design and quality abstractions - Final version
Editor(s):	José Ignacio Requeno (ZAR)
Contributor(s):	Marcello Maria Bersani (PMI), Michele Guerriero (PMI), José Merseguer (ZAR), Diego Pérez (ZAR), José Ignacio Requeno (ZAR) and Damian A. Tamburri (PMI)
Reviewers:	Matej Artač (XLAB) and Madalina Erascu (IEAT)
Type (R/P/DEC):	Report
Version:	1.0
Date:	31-January-2017
Status:	Final version
Dissemination level:	Public
Download page:	http://www.dice-h2020.eu/deliverables/
Copyright:	Copyright © 2017, DICE consortium – All rights reserved

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

Executive summary

This document presents the final version of the DICE design and quality abstractions. Therefore, it provides the final versions of the DICE Models and the DICE Profile, both artifacts are briefly introduced with an executive summary and extensively described in the Appendix A of this document. This deliverable is an incremental update of the deliverable D2.1 (*Design and quality abstractions - Initial version*) published in M12. The document exclusively focuses on the new contributions of the DICE Models and DICE Profiles with respect to the previous deliverable. Mainly, this work covers 1) support for quality and reliability annotations, 2) the addition of data protection and privacy constraints for privacy-by-design modeling, and 3) the refinement of DICE Profiles for two Big Data technologies (Apache Hadoop MapReduce and Apache Storm). The work presented in this deliverable has been carried out within tasks T2.1 (Data-aware functional models) and T2.2 (Data-aware quality annotations). All the artifacts described in this document are publicly available in the so-called DICE-Models Repository [1] and DICE-Profiles Repository [2].

Glossary

DAM	Dependability Analysis and Modeling
DDSM	DICE Deployment Specific Model
DIA	Data-Intensive Application
DICE	Data-Intensive Cloud Applications with iterative quality enhancements
DPIM	DICE Platform Independent Model
DTSM	DICE Technology Specific Model
DDSM	DICE Deployment Specific Model
DSML	Domain-Specific Modelling Languages
MARTE	Modelling and Analysis of Real-Time and Embedded Systems
MDD	Model-Driven Development
MDE	Model-Driven Engineering
M2M	Model To Model
NBDRA	NIST Big Data Reference Architecture
NIST	National Institute of Standards and Technology
NFP	Non-Functional Properties
QoS	Quality of Service
RBAC	Role Based Access Control
TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modelling Language
VSL	Value Specification Language

Contents

List of Figures

List of Tables

1 Introduction and Objectives

The focus of the DICE project is to define a quality-driven framework for developing data-intensive applications that leverage Big Data technologies hosted in private or public clouds. DICE offers a novel profile and tools for data-aware quality-driven development. This document describes the final version of the DICE Profile and the DICE Metamodels, which update the DICE Profile and DICE Metamodels presented in the DICE *Design and quality abstractions - Initial Version* deliverable [3]. The DICE Metamodels are needed to develop the DICE Profile according to the approach described in Section 3 of this document. The DICE Metamodels and the DICE Profile, developed in the scope of WP2 as Tasks 2.1 (Data-aware functional models) and 2.2 (Data-aware quality annotations), are published in the DICE-Profiles [2] and DICE-Models [1] repositories.

1.1 Objectives of WP2

The goal of WP2 is to provide and evaluate the necessary design abstractions for specifying data-intensive cloud applications. The work captured in this deliverable focuses on Tasks 2.1 and 2.2 (see below), namely, the conceptual definition of the DICE design abstractions. These two tasks include the specification of quality annotations, data protection and privacy constraints and introduce the DICE Platform Independent Model (DPIM), DICE Technology Specific Model (DTSM), and DICE Deployment Specific Model (DDSM) logical layers of the DICE Profile.

1.2 Objectives of Task 2.1 and Task 2.2

Task 2.1 provides DPIM, DTSM and DDSM abstractions of the DICE Metamodel and Profile to describe data-intensive application properties (structural or otherwise), their usage requirements (e.g., functional decomposition of compute nodes w.r.t., visualization nodes) and transformations, among other data-intensive architecting and design concerns [**CicchettiRP06**]. Both Metamodel and Profile levels of abstraction need to be extended with technology-specific elements each time a new supporting technology is incorporated. Such extensions allow a data-intensive application developer to specify operations performed as part of data-intensive computing (e.g., a mapping operation or applying the “reduce" operand) while also qualifying inputs and outputs.

Task 2.2 defines DPIM, DTSM and DDSM abstractions (UML-based languages and profiles) to specify reliability, efficiency, safety and privacy requirements for data-intensive applications along with their application subcomponents. This task uses as baselines DAM and MARTE profiles (described in the DICE *State of the Art Analysis* deliverable [4]) as well as SecureUML [17] for what concerns the privacy part - a key innovative contribution part of this deliverable.

1.3 Objectives of this document

This document presents the final version of the DICE Metamodels and DICE Profiles at DPIM and DTSM levels. The DICE Metamodels and Profiles presented in this deliverable update or extend the Metamodels and Profiles published in the previous deliverable D2.1 (*Design and quality abstractions - Initial version*) published at M12.

The main contributions of this deliverable with respect to the previous one can be summarized in three points:

- The introduction of the DICE Metamodel and Profile for SecureUML;
- The introduction of reliability parameters for models at DPIM level in Metamodel and Profile;
- The extension of the DTSM profile to elaborate technological models for Apache Hadoop MapReduce and Apache Storm technologies. While the corresponding Metamodels can be found in deliverable D2.1, the purpose of this document is to show how new technologies can be incorporated into the DICE approach and be made available to the application designers by means of a familiar UML profile. Other technologies part of the DICE technical space (see Pag. 8 of [6]) will be

considered beyond this final version of the core Metamodels and DICE Profile, prioritising them based on the needs of the DICE case-studies and the scope of their evaluation.

Finally, the DICE Metamodels and Profiles for the DDSM abstraction level are the focus of Task 2.3 and will be presented in their final version in the DICE *Deployment abstractions - Final version* deliverable [5] due at M27.

1.4 Structure of the document

The structure of this deliverable is as follows:

- Section 2 summarizes the requirements that Tasks 2.1 and 2.2 aim to cover.
- Section 3 presents the approach developed for constructing the Metamodels and for developing the Profile.
- Section 4 summarizes the contribution of this deliverable at DPIM level.
- Section 5 summarizes the contribution of this deliverable at DTSM level.
- Section 6 summarizes the goals achieved, and outlines the future work.
- Appendix A details the current version of the DICE Profile at DPIM and DTSM levels.

The tables with the Profile annotations of the SecureUML Metamodel are not presented in a separated Appendix, but they can be found in the corresponding citation reference.

2 Requirements

Deliverable D1.2 [6, 7], released on month 6, presented the requirements analysis for the DICE project. The outcome of the analysis was a consolidated list of requirements and the list of use cases that define the project's goals that guide the DICE technical activities. During the progression of DICE project, the requirements and goals can be changed or adapted dynamically. For that reason, an online version of the requirement document [8] is constantly updated in order to register all the modifications and the current status.

In the following, we summarize the requirements concerning Tasks T2.1 and T2.2. They will be reviewed in the Conclusions so as to evaluate the technical advances in the scope of WP2. Note that Domain Assumptions upon which some of these requirements (e.g., R2.18 or R2.1,2.2) are based, are omitted for the sake of brevity and can be inspected on D1.2 directly [6, 7].

2.1 Requirements

ID	R2.0
Title	Profile Structure
Priority	Must have
Description	Following the basic approaches to formal languages design, the DICE profile will necessarily require a meta-modelling notation to cover for the basic structure and semantics of the language intended behind the DICE profile. Also, the DICE profile will need the implementation of said basic structure and semantics following a commonly usable format as best fit with respect to DICE goals and tenets.

ID	R2.1
Title	Profile Basis
Priority	Must have
Description	The DICE profile MUST follow the default abstraction layers known and supported in Model-Driven Engineering, namely, Platform-Independent Model, Platform-Specific Model and add an additional layer specific to supporting the modelling of Deployment-ready implementations, i.e., a Deployment-Specific Model.

ID	R2.2
Title	Abstraction Layer Origin
Priority	Must have
Description	Every abstraction layer (namely, DPIM, DTSM and DDSM) of the DICE profile MUST stem from UML.

ID	R2.3
Title	Relation with MARTE UML Profile
Priority	Must have
Description	The DICE Profile MUST define required and provided properties of a DIA as well as metrics (estimated, measured, calculated and requirements) to monitor them. Said metrics will be specifed following the MARTE NFP framework.

ID	R2.4
Title	DICE Constraints Definition
Priority	Must have
Description	The DICE Profile MUST allow definition of values of constraints (e.g., maximum cost for the DIA), properties (e.g., outgoing flow from a Storage Node) and stereotype attributes (batch and speed DIA elements) using the MARTE VSL standard.

ID	R2.5
Title	DICE Profile Performance Annotations
Priority	Must have
Description	The DICE Profile shall define annotations for performance based on the MARTE::GQAM framework.

ID	R2.6
Title	DICE Profile Reliability Annotations
Priority	Must have
Description	The DICE Profile shall define annotations for reliability based on the DAM profile.

ID	R2.7
Title	DICE Profile Main DIA Concerns - Structure and Topology
Priority	Must have
Description	The DICE Profile shall define annotations that address structural and topological concerns behind DIAs. Also, the DICE Profile shall separately define storage and computation elements to allow for fine-grained specification.

ID	R2.8
Title	DICE Profile Main DIA Concerns - Flow and Behavior
Priority	Must have
Description	The DICE Profile shall define annotations that address behavioral and flow concerns behind DIAs. Also, the DICE Profile shall define annotations for flow-control across DIAs.

ID	R2.9
Title	DICE Profile Pre- and Post-Processing
Priority	Must have
Description	The DICE Profile shall define constructs for pre- and post-processing of Big Data (e.g., for filtering input data or visualising data).

ID	R2.10
Title	DICE Profile Tech-Specific Constraints
Priority	Must have
Description	The DICE Profile MUST define structural and behavioral constraints typical in targeted technologies (e.g., Hadoop, Storm, Spark, etc.).

ID	R2.11
Title	DICE Profile Separation-of-Concerns
Priority	Must have
Description	The DICE Profile MUST use packages to separately tackle the description of targeted technologies in the respective profile abstraction layers (e.g., DTSM and DDSM). Said packages shall be maintained consistently.

ID	R2.12a
Title	DICE Profile Supervision and Control
Priority	Must have
Description	The DICE Profile shall define constructs and annotations for DIA supervision and process control.

ID	R2.12b
Title	DICE Privacy and Security Aspects
Priority	Must have
Description	The DICE Profile shall focus on DIA-specific privacy and/or security restrictions.

ID	R2.13
Title	DICE Profile Data Structure
Priority	Must have
Description	The DICE Profile shall define QoS annotations for data structure and its specification.

ID	R2.14
Title	DICE Profile Data Communication
Priority	Must have
Description	The DICE Profile shall define annotations to elaborate on structural and behavioral details concerning the channeling and marshalling of information across specified DIAs.

ID	R2.15
Title	DICE Profile Sub-Structures
Priority	Must have
Description	The DICE Profile shall provide annotations for specifying node nesting and replication across the structure of DIAs.

ID	R2.18
Title	DICE Deployment Transformation
Priority	Must have
Description	The DICE IDE needs to be provided with a fully automated transformation that is capable of constructing an ad-hoc TOSCA blueprint stemming from the deployment information that can be made available in a DTSM and DDSM model. The usage of deployment knowledge for each technology in the DTSM shall be used by such transformation as a means to determine the deployment structure. Subsequently, a DDSM model proposal shall be built from this automated understanding. Finally, a TOSCA blueprint shall be constructed from such DDSM model using an appropriate mirroring between the DDSM model instance and the TOSCA notation.

ID	R2.20
Title	DICE Architecture Trade-Off Transformation
Priority	Must have
Description	The DICE IDE needs to be rigged with a M2M transformation that provides coherent and comparable aggregates of the elements in the DICE technological library such as to allow for architecture trade-off analysis - in so doing, the DICE IDE shall assume that a DIA architect is compelled to evaluate several equally valuable alternatives for technological composition of its own DIA solution; the architect shall then evaluate the possible combinations of all technologies in a technological library (e.g., such as the one provided by DICE). From this library the architect will need to be able to instantiate the possible compatible compositions of technologies that match its higher-order architectural specification (i.e., his DPIM model).

3 Research and Development Approach

As recalled in the DICE *State of the Art Analysis* deliverable [4], MDE techniques [9] and MDA in particular [10] define the typical abstraction layers for the purpose of engineering software systems using a model-centric perspective. The fundamental axiom behind this engineering paradigm is that any engineering endeavuor shall be guided by at least three compounding and interoperating perspectives, namely: (a) Computational-Independent perspective; (b) a Platform-Independent perspective; (c) a Platform-Specific perspective. Using these three perspectives, one or more models can be specified to properly and systematically specify a system-to-be. In DICE these perspectives take the form of DPIM, DTSM and DDSM, respectively, while the specification language adopted is UML.

UML [11] is a General Purpose Modelling Language. Therefore, it can be used to model a wide range of systems but not all of its modelling capabilities are necessarily useful in all domains or applications. Conversely, Domain-Specific Modelling Languages (DSML) are conceived for addressing the needs of specific application domains. In this regard, UML offers a solution, the so-called UML profiling mechanism [11]. Profiling opens the possibility of creating DSMLs by extending or restricting UML. A Profile is then an adaptation of UML to fit a specific domain. In short, a UML profile is made of a set of *stereotypes*, a set of *tags* and a set of related *constraints*. A stereotype is just a name that will be attached to certain elements of a UML diagram. Stereotypes have tags, we can see them as the attributes added by the stereotype.

UML has been extended with two Profiles of interest for DICE, namely MARTE [12] and DAM [13]. MARTE (Modelling and Analysis of Real-Time and Embedded systems) provides support for the specification, design, quantitative evaluation, and verification & validation of software systems. DAM (Dependability Analysis and Modelling Profile) provides support for the dependability modelling and analysis of software systems. However, neither MARTE nor DAM has a direct support for expressing data location, data properties such as volume or transfer rates or operations that move data. Hence, addressing such lack is the main objective of the DICE Profile.

For constructing a technically correct high-quality UML profile that covers the necessary concepts according to the DIA technologies, several steps need to be followed. First, metamodels for each abstraction level, i.e. DPIM, DTSM and DDSM, that define the concepts are needed. We have carried out this step by carefully reviewing the abstract concepts for modelling DIA, then obtaining the abstractions for the DPIM level, which conform the DICE Metamodel at DPIM level. Later, we have reviewed the Big Data technologies addressed by DICE (e.g., Hadoop or Storm) and we have defined the abstractions of interest, consequently obtaining the DICE Metamodels at DTSM level. The last level, DDSM, will be presented in another deliverable D2.4 (*Deployment abstractions - Final version* deliverable [5]) due at M27.

As a second step, the DICE Profile, at DPIM and DTSM levels, was defined by mapping the concepts from the DICE domain models or DICE Metamodels to UML, MARTE and DAM. While constructing the initial version of the DICE Profile, following WP2 requirements, we introduced a set of stereotypes that can be easily used by the software engineer.

As a result, an initial version of the DICE Metamodels and Profiles has been presented in deliverable D2.1 (*Design and quality abstractions - Initial version* deliverable [3]). The Metamodels and Profiles introduced in that document are now enlarged and updated in this new release. For instance, reliability and quality annotations are now supported in the stereotypes of the DPIM Profile. In addition, the DICE Metamodel and Profile have been extended for supporting SecureUML, a UML-based modelling language for defining privacy and security constraints in software applications.

The DPIM Metamodel and Profile core parts are technology-independent and, consequently, they are available for supporting modeling of any DIA. They can be, however, extended with concepts peculiar of specific Big Data technologies to simplify the work of designers and operators of DIAs. With the purpose of showing examples of technology-specific extensions, we consider in this deliverable the cases of Hadoop and Storm.

The DICE Profile is employed for stereotyping elements of behavioural UML diagrams (e.g., activity nodes in UML activity diagrams; or lifelines in UML sequence diagrams) and components in structural UML diagrams (e.g., computational nodes in UML deployment diagrams). The information contained in

the annotations of the stereotyped UML elements guides the transformation process from UML models into formal models for performance or reliability analysis. Currently, the stereotypes proposed by the DICE Profile are useful for obtaining performance models by applying M2M transformations at DPIM and DTSM level (see Deliverable D3.1 [14]). Besides, reliability models are now obtained by M2M transformation of DPIM-profiled models using the new annotations incorporated in this version of the DICE Profile. Reliability models for DTSM-profiled models will be reported in Deliverable D2.4 titled *Deployment abstractions - Final version*. In the following we summarize the latest advances regarding the DICE Metamodel and DICE Profile.

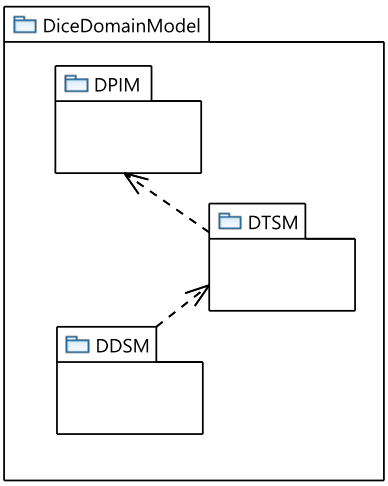

Figure 1: DICE Metamodel - High-level View

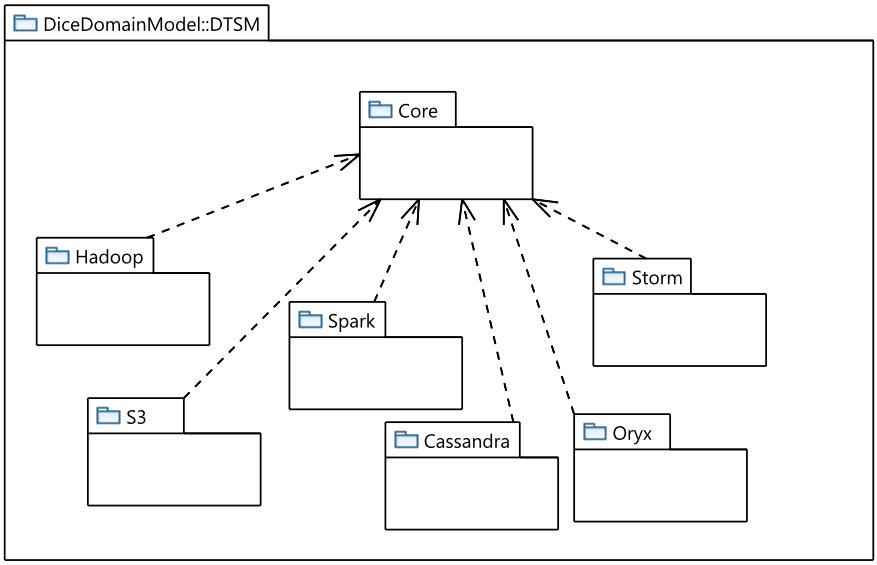

Figure 2: DICE Metamodel - DTSM View

3.1 DICE Metamodel

The DICE metamodel, previously detailed in Appendix A of the deliverable D2.1 [3], is sketched in Figure 1. DICE considers one metamodel per abstraction level: DPIM, DTSM and DDSM. The rationale

behind this organization is that each abstraction level keeps separated, self-contained and mutually incremental. At DPIM level the metamodel provides those abstract concepts needed for DIA modeling. At DTSM level, see Figure 2 the metamodel provides a core-DTSM metamodel and one DTSM metamodel for each technology addressed.

The logical division of the most complex abstraction layers in the DICE Profile, namely, the DICE DTSM and DDSM layers, was arranged using a standard package-like notation. Following a systematic approach tailored from Formal Concept Analysis (FCA) [15], we elicited the core-constructs common to all technologies addressed by DICE and captured said constructs in a core package, to be used by all technological extensions. In addition, specific technological extensions, e.g., Hadoop MapReduce or Storm, were self-contained into separate packages, thus allowing to keep them transparent to less expert users. Nevertheless, DICE will strive to make available these packages as possibly instantiable and modifiable constructs, e.g., to accommodate the needs of more experienced users.

3.2 DICE Profile

For each aforementioned metamodel we propose a mapping for obtaining the DICE Profile. The concepts of the DICE Metamodel are synchronized with the current update of the DICE Profiles. Figure 3 offers a high-level view of the DICE Profile, which basically contains the DICE Library and the DICE Extensions.

The DICE Library, detailed in Figure 4, contains basic and complex DIA types. We have imported the DAM library, which also imports the basic Non-Functional Properties (NFP) types from the MARTE library, for the definition of these types. In particular, the MARTE NFPs sub-profile is applied to the definition of new basic DIA types and the Value Specification Modeling (VSL) sub-profile to the definition of the complex ones.

The DICE Extensions package, detailed in Figure 5, provides the domain expert with a set of stereotypes to be applied at model specification level, i.e., the stereotypes necessary to represent the different system views in concrete UML models.

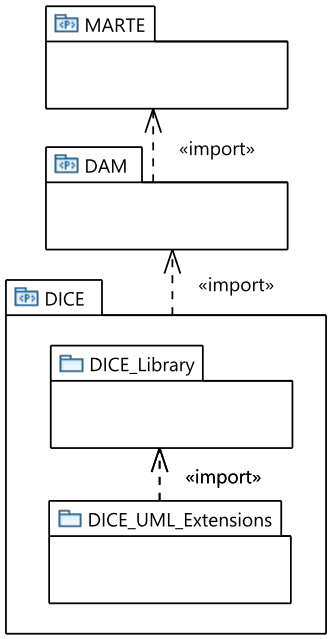

Figure 3: DICE Profile - High-level View

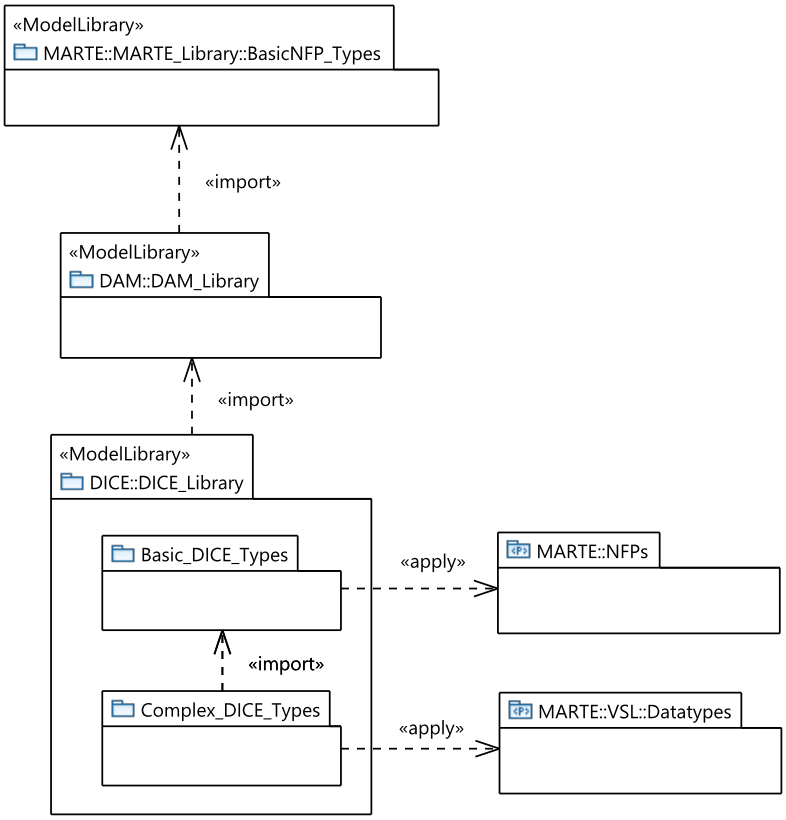

Figure 4: DICE Library

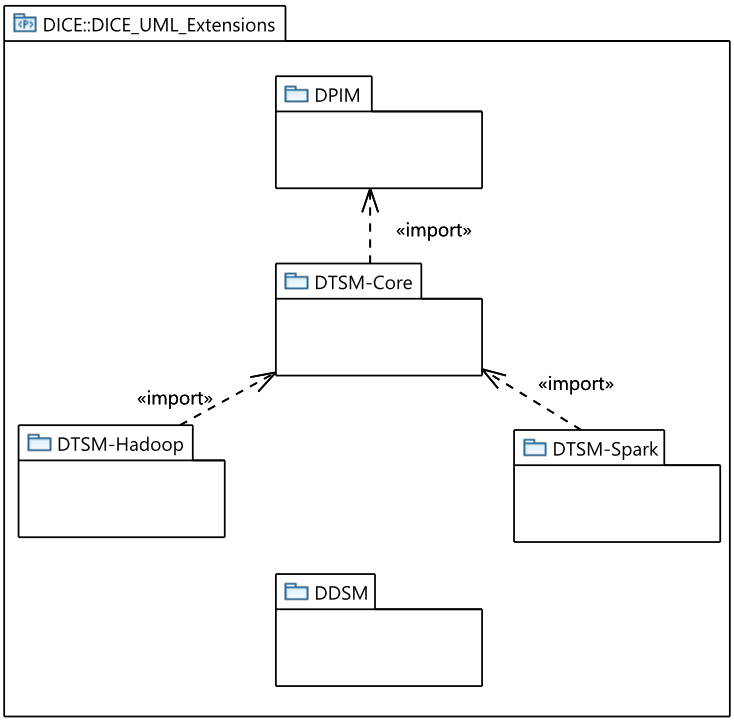

Figure 5: DICE UML Extensions

4 DPIM Logical Layer

The major technical extensions to the DICE platform-independent (DPIM) layer consider the privacy-by-design strategy adopted by DICE, and augments DPIM according to this strategy. Moreover, the DPIM profile introduces new reliability and quality annotations. The next sections address these two aspects focusing on: (a) metamodel - from this perspective we elaborate the privacy-by-design for the architectural level (DPIM); (b) profile - from this perspective we elaborate how the DICE UML profile can be elaborated to use MARTE annotations for the purpose of DICE specific analyses.

4.1 Meta-model

The Data-Intensive Platform Independent meta-model (DPIM) underwent little or no modifications in its final iteration since the main role of this abstraction layer is to allow the component-based elaboration of a data-intensive solution keeping in mind quality and design concerns stated out from customers and/or DICE solution engineers. In this respect, the little modifications that DPIM underwent concern its adaptation to specify and accommodate the use of additional notations and abstractions needed to support DICE-specific concerns and properties [16] along privacy-by-design of DIAs.

4.2 Extending DPIM to support Privacy-By-Design of DIAs

The DICE DPIM meta-model was accommodated to welcome its augmentation by means of the SecureUML notation [17] for the specification and support of privacy-by-design annotations. SecureUML is, quoting from its original specification: “a modelling language that defines a vocabulary for annotating UML-based models with information relevant to access control. It is based on the model for Rule Based Access Control (RBAC) [...], with additional support for specifying authorisation constraints. SecureUML defines a vocabulary for expressing different aspects of access control, like roles, role permissions and user-role assignments. Due to its general access-control model and extensibility, SecureUML is well suited for business analysis as well as design models for different technologies".

As part of this refinement, we defined our own DICE-specific and ad-hoc version of the SecureUML profile originally formulated in [17] to focus on the DICE-required specification of privacy-by-design aspects pertaining to RBAC. Fig. 6 provides an overview of our own re-interpretation of the SecureUML profile using a simple standard UML Profile diagram.

In this respect, DPIM is augmented with aspects that cover:

- Specification of «Role» and «Permission» for certain computing resources, i.e., DICE «ComputeNode», «StorageNode», «SourceNode» and «VisualizationNode» respectively;
- Specification of specific sets of «Action» that «Permission» types are able to restrict with respect to particular «Resource» nodes;
- Specification of deterministic «AuthorizationConstraint» elements that essentially dictate the conditions upon which a certain «Role» can grant «Permission» elements;

The above ternary set of elements is necessary and sufficient to cover our problem of grant-based access control, that is, the primary privacy-by-design concern to be addressed as part of the work in DICE, with a major focus to the context of the NETF fraudster detector case-study (more details in Sec. 5.1.1). Also, inner source properties to the above meta-elements were removed for the sake of simplicity while extending the DPIM meta-model. The rationale behind this decision stems from the assumption that DICE privacy-by-design annotations need to focus on grant- and rule-based access-control and do not need any finer-grained granularity in terms of properties and privacy aspects specification.

Finally, the above elements essentially overlap the DPIM elements we reported in Deliverable D2.1. Therefore, rather than editing previously existing elements, DICE designers focused on adding meta-elements and constraints to the original DPIM meta-model to accommodate the use and effective modelling of SecureUML in action.

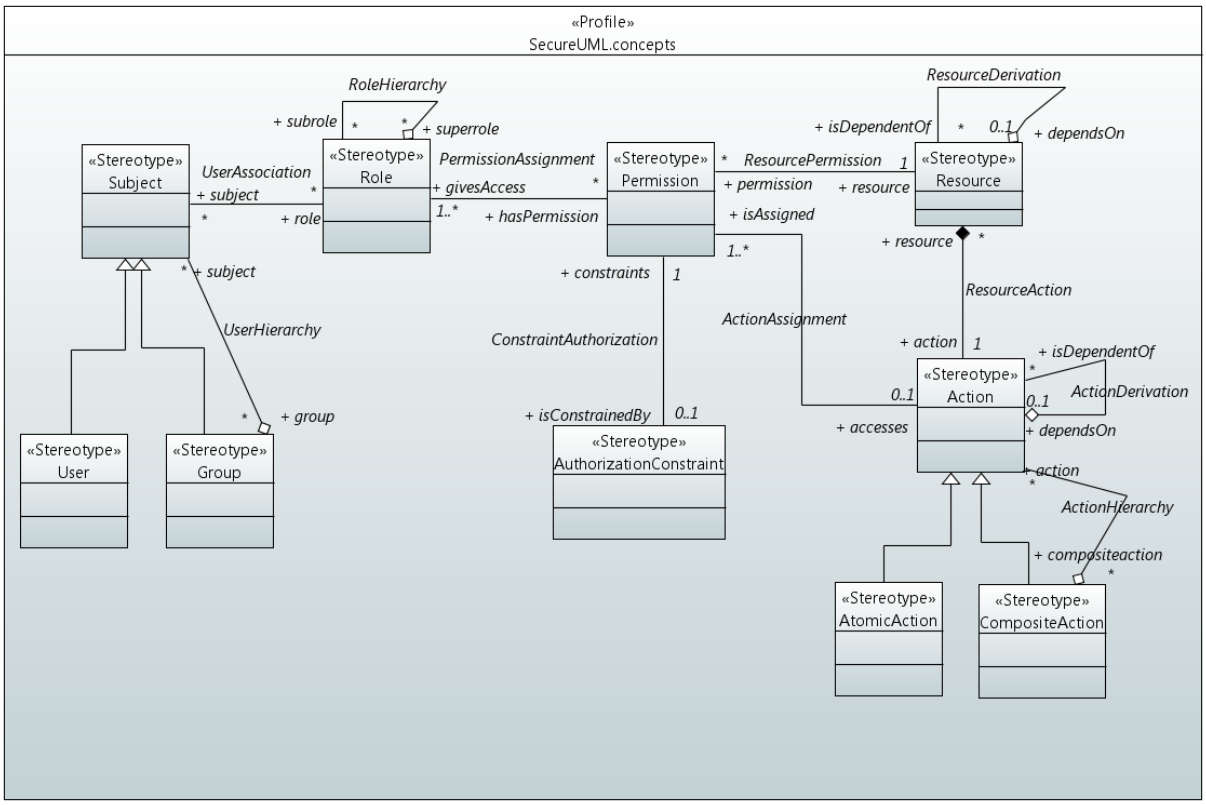

Figure 6: DICE UML Meta-Modelling Extensions: SecureUML

Figure 7 reports the diagram generated from the current version of the DPIM metamodel. The top-level concepts in the abstraction hierarchy, such as DIA and DIAElement, are omitted for the sake of space and clarity. The shown meta-model is first of all an improvement and consolidation of the DPIM meta-model reported in the previous version of this deliverable. Moreover the new concepts derived from the integration of the SecureUML meta-model are now in place.

Table 1 summarizes the current list of meta-elements of the DICE DPIM meta-model, without including the new privacy-specific elements coming from the integration with SecureUML.

Table 1: Meta-Elements at DPIM level

Meta-Element	**Description** (This meta-element is for model elements representing…)
`DIA`	Represents a Data Intensive Application, which might be composed of multiple DIA elements, such as compute nodes, storage nodes, but also users and permissions.
`DIAElement`	An element of a Data Intensive Application. It can be a ComputationNode or a DataSource.
`ComputeNode`	Represents an element of the application whose goal is to perform some computation.
`DataSource`	This entity represents an element of the DIA acting as a data source at the DPIM layer.
`SourceNode`	This entity represents a node of a DIA emitting data, but not providing persistence and storage features.
`StorageNode`	Represents an element of the application whose goal is to store and provide the application data (i.e, a database or a file system). It can be managed by the DIA owner or externally provided.
`Dataset`	Represents the data that an element of the application can take in input and/or produce in output.

SchemaField	Represents a field of a dataset. Can be seen as a column of a table.
FilterNode	Represents a specific type of ComputeNode whose goal is to apply a filter on the input data.
MachineLearningNode	Represents a specific type of ComputeNode whose goal is to perform a machine learning algorithm on the input data.
VisualizationNode	Represents a specific type of ComputeNode whose goal is to properly visualize the input data.

Table 2 reports the list of meta-elements of the SecureUML modelling language proposal implemented within DICE DPIM meta-model.

Table 2: Meta-Element at DPIM level implemented from SecureUML

Meta-Element	**Description** (This meta-element is for model elements representing...)
User	A user of the DIA. A user can own a set of ComputeNode and can have a role over which specific permissions are set.
Role	Represents a role in the context of a DIA. Multiple users can be assigned to the same role. A role can have assigned several permissions.
Permission	According to the RBAC approach, a permission represents basically a relationship between a role and an object of the DIA. In order to be flexible and abstract enough the object of a permission can be any DIAElement, of course with different semantics depending on the case.
ActionType	Represents an action that a permission allows to perform on the referenced object.

In order to obtain the privacy-aware DPIM metamodel, first of all we carefully studied the metamodel behind SecureUML as introduce in [17], in order to understand the right connecting elements. Our analysis shows that there is basically no overlap between the concepts previously existing in the DPIM metamodel and those coming from the SecureUML metamodel - the only connecting element is the object protected by a given permission. According to the original SecureUML metamodel, reported in Figure 8 a protected object can be any modelling element from UML (see again [17]). Following the same principle, in the DICE DPIM metamodel a protected element can be any DIAElement.

Figure 9 shows an example model that was obtained by instantiating the DPIM meta-model. The model refers to a simple DIA developed in the context of DICE called Wikistats. Wikistats main goal is to calculate statistics from the web pages of the popular Wikimedia website. The result of this analysis has to be stored in a managed database. With the aim of exemplifying the usage of the new modelling elements for privacy-aware models, we assume the user Damian to be the owner of the Wikistats application. As an admin user, Damian should be able to query the managed storage system for the result of his analysis. In particular he should be able to read the dataset output of the Wikistats application. Thus a specific privacy permission has been modelled to specify this use case.

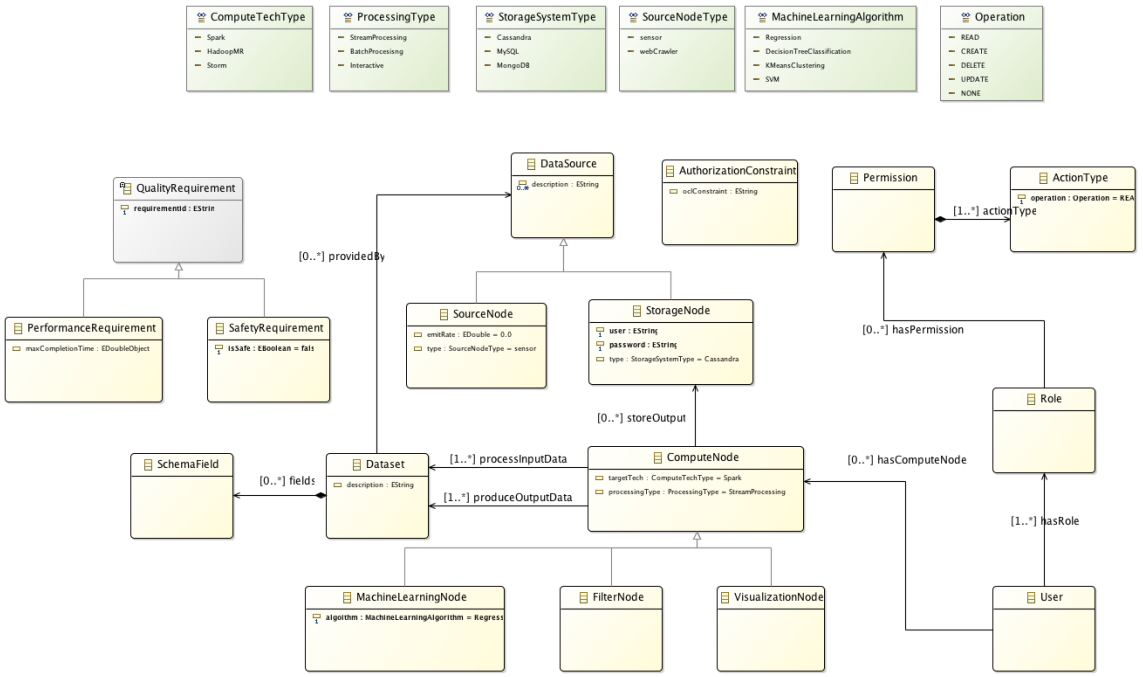

Figure 7: Diagram obtained for the DICE DPIM metamodel.

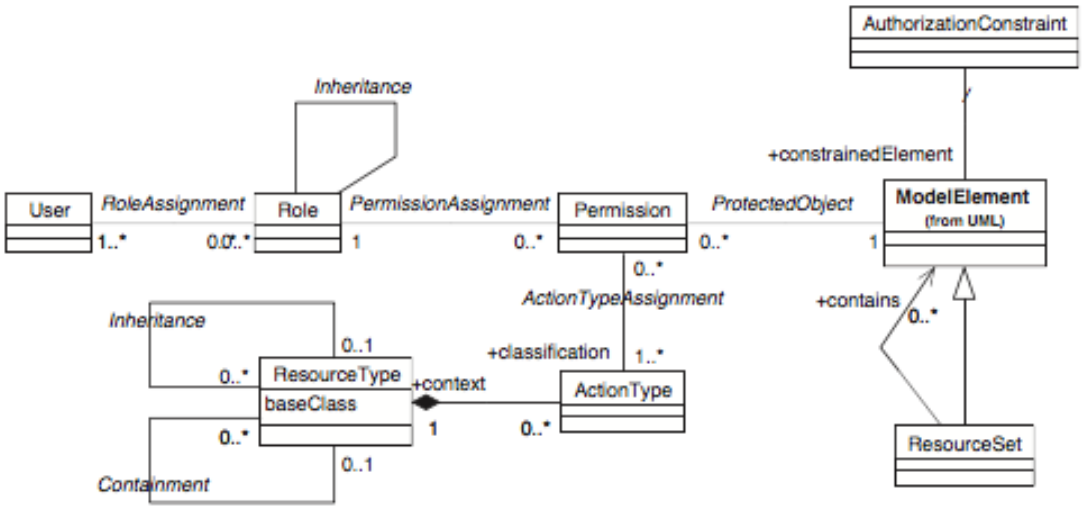

Figure 8: The SecureUML metamodel

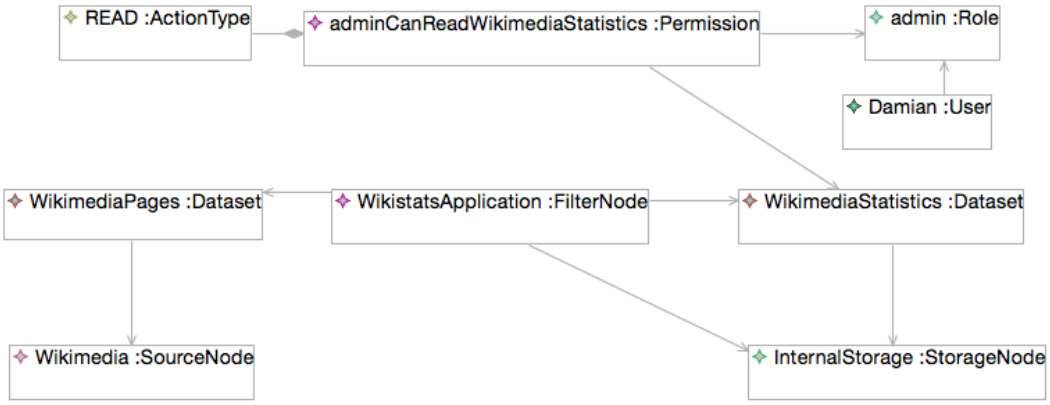

Figure 9: An example of a DPIM model obtained by instantiating the DPIM meta-model, which also includes privacy related aspects.

4.3 Profile

Table 3 summarizes the current list of stereotypes of the DICE Profile for the DPIM level.

Table 3: Stereotypes at DPIM level

Stereotype	**Description** (This stereotype is for model elements representing...)
`DpimComputationNode`	DIA components with computation throughput, type of data processing, and maybe expected target technology.
`DpimFilterNode`	Filter nodes that extend general DpimComputationNode with input and output ratios.
`DpimSourceNode`	DIA components with a given storage volume, type of generated data, and data generation rate.
`DpimStorageNode`	DIA component with resource multiplicity, type of stored data, and speed in terms of maximum operations rate.
`DpimChannel`	Connectors that have a maximum speed and that are subject to failures and propagation of errors.
`DpimScenario`	An execution scenario of the DIA, which defines the quality properties of interest and the scenario quality requirements.

Table 4. depicts the list of stereotypes of the SecureUML modeling language proposal implemented within DICE.

Table 4: Stereotypes at DPIM level implemented from SecureUML

Stereotype	**Description** (This stereotype is for model elements representing...)
`User`	A user of the DIA.
`Group`	A group of users of the DIA.
`Role`	The definition of a role in the interaction with the DIA.
`SubjectAssignment`	The existence of a relation between a `User` or `Group` and a `Role` in the DIA.
`SubjectGroup`	The relation of belonging between a `User` and a `Group`.
`Permission`	The actions that are granted for a certain `Role`.
`AuthorizationConstraint`	A predicate in function of the DIA state that affects the validity of `Permissions` along changes in the DIA state.
`Resource`	A protected resource in the DIA.
`AtomicAction`	An operation that can be executed on a `Resource`.
`CompositeAction`	An aggregation of operations that can be executed on a `Resource`.

5 DTSM Logical Layer

The major technical extensions to the DICE technological layer consider the privacy-by-design strategy adopted by DICE and augments DTSM according to this strategy. Moreover, the DTSM is augmented from a profile point of view to accommodate DICE analyses previously foreseen in the description of work. The next sections address these two aspects focusing on: (a) metamodel - from this perspective we elaborate the DICE strategy in pursuing privacy-by-design from the architectural (DPIM) to the technological level (DTSM); (b) profile - from this perspective we elaborate how the DICE UML profile can be elaborated to use MARTE annotations for the purpose of DICE specific analyses.

As we have already mentioned, the DTSM Profile needs to be extended when support to new technologies is incorporated into DICE. In this document we describe the recent updates of the Apache Hadoop MapReduce and Apache Storm instantiations of the DTSM Profile.

5.1 Metamodel

This section addresses and elaborates on the augmentation of the DTSM metamodel, mostly driven by our process of supporting privacy-by-design.

5.1.1 New Technology-Specific Design Problems: Privacy-By-Design

The design problem reported in this deliverable was formulated following several email and direct interview exchanges with NETF case-study owners with respect to their stringent Security and Privacy Issues. Said conversation is omitted for the sake of brevity but was overseen, analysed and coded by PMI, IeAT and NETF jointly. The technological design problem for the DICE technology-specific layer can be recapped as follows:

Consider the NETF Big Data application which is a tax fraudster detector (see the deliverable D6.1 [19] for a detailed description of the framework). NETF is the system orchestrator and the application provider. The Big Data framework providers include:

- FLEXI: infrastructure provider;
- Cassandra: data platform provider;
- Spark, Kafka, and Akka: processing frameworks providers;

These three layers of Big Data framework providers are fully explained in NIST v1V6 reference architecture for privacy-by-design, section 4.4, page 17 [18]. The data providers are simulated entities such as:

(a) French tax authorities;

(b) French city councils;

(c) French banks.

In the scope of DICE, the above actors are assumed to be replaced by simulators that generate fake but realistic data. The data consumers are viewers, which print tax fraudster lists on end-users' screens.

At this point, we observed that NIST (section 5.1, page 33 of the reference architecture document [18]) provides an extensive outline of standard and necessary measures to implement in order to have secure interactions between the five functional NBDRA components (system orchestrators, data providers, Big Data application providers, data consumers, and Big Data framework providers).

Inspired by this document, we concluded that:

- simulators (which replace tax authorities, city councils, banks, and so on) must encrypt their generated data to prevent FLEXI from reading them. They will be pipelined to some Cassandra databases in their encrypted form;
- the tax fraudster detector, built by NETF, will decrypt them in-memory for analysis. To be sure that these data are not corrupted and come truly from our simulated entities, simulators must digitally sign their generated data. Any other method that guarantee integrity and authenticity is acceptable;
- only NETF must be able to access the streams created by simulators: a protocol, may be based on credentials, must be designed;
- since viewers consume the results of the analytics, a way to prohibit unauthorised viewers must be implemented: encryption, digital signatures, and credentials, again.

Stemming from the above technology-specific design problem, we augmented the DTSM notation to address policy design and structural specification dynamics such that said design problem can be addressed with appropriate design decisions and their documented rationale. Moreover, we enabled DICE transformation technologies (i.e., the DICER tool) to be able to deploy and support said policy specifications and allow for their enforcement at the Deployment specific layer (i.e., DDSM) - this latter part will be addressed by a future deliverable, namely, D2.4 “Deployment abstractions - Final version" to be prepared by PMI in the form of a Public Report within M27.

This not withstanding, having understood and framed our privacy-by-design strategy around the above understanding, we defined an additional requirement for our specification efforts, as follows:

“We restrict the privacy and security policies to be concerned explicitly about the DIA itself rather than the circumstantial technology with which the DIA is developed, operated and evolved. For example, restricting the behaviour of the monitoring platform on top of the privacy-sensitive DIA or reducing monitoring operations in any way due to privacy concerns is out of the scope of the support intended in DICE.”

5.1.2 Privacy-By-Design in DICE: A Technology-Aware and Secure Deployment Process

In summary, an overall DICE privacy & secure deployment process can be defined as follows:

1. Define at the DPIM level a certain specific privacy definition (i.e., a series of privacy policies to be applied across one or more topologies);
2. Refine that definition in the DTSM level applying it to specific technologies with specific monitoring and/or trace-checking needs;
3. Move said privacy definition at DDSM level (e.g., by means of M2M transformation) - at the deployment level privacy and secure deployment become real and need to be reflected by ad-hoc infrastructure specification code (e.g., a Hadoop MR will never access a stream of messages from a certain message broker).

Focusing on the DTSM level, therefore, the previous version was accommodated to model the possible entry-points and all possible interactions across technologies and middleware involved in the DIA itself. As a consequence, an additional topological specification of the entire DIA was needed to make sure the model is consistent with the privacy requirements. To accomodate this addition, the DTSM layer was augmented with the possibility to specify privacy policies on a technology-specific (DTSM) yet component-based (DPIM). In more practical terms, we modified the metamodel and profile to allow the modelling of DTSM technological concepts on a standard UML component diagram, typically used to elaborate a DICE DPIM diagram.

In principle, with this additional technologically-refined and privacy-augmented architecture layer in place, privacy-by-design can be enforced by checking, once all computation has ended and before releasing the result to DICE users, the execution and monitoring traces by means of trace-checking certifiers to offer formal guarantees that privacy policies were not violated and the datum was not changed

in its data qualities as well as no privacy issue was generated (e.g., no one entered in contact with the datum unless she was allowed to do so). As specified above, this process, as well as the process of enforcing the above-stated DTSM privacy structure and policies at DDSM level are addressed in future deliverables concerning deployment abstractions in their final form.

5.2 Profile

The Data-Intensive Technology Specific profile (DTSM) underwent some modifications in its final iteration. It includes a DTSM::Storm Profile, the update of the DTSM::Hadoop Profile, and the refinement of the DICE::DICE_Library. The Hadoop and Storm Profiles do not fully incorporate quality and reliability aspects yet, but they will be added incrementally in the following releases. Other DTSM profiles, such as the Spark profile, will be included and refined in the near future.

The Hadoop and Storm profiles have been validated by the transformations presented in the deliverable D3.1 [14], and used by the DICE Simulation and Verification Tools. The modeling of qualities using MARTE, i.e. NFP specification, was reported in the Deliverable 1.1 - State of the Art [4] (See sections B3.3.2 Modelling with MARTE, and B3.3.3 Modelling with DAM). Some examples showing the utilization of these profiles can be found in the deliverable D3.1 [14] (See section 3.1.1 for the use of quantitative analysis tags at DPIM level, and Tables 4, 6 and 7 for Hadoop and Storm technologies).

5.3 Updating DTSM to support Apache Hadoop MapReduce

The DTSM Profile for the Apache Hadoop MapReduce technology has been updated in this deliverable. The DTSM Profile includes a list of stereotypes that addresses the main concepts of the Apache Hadoop MapReduce technology identified in Table 6. In particular, we stress those concepts that directly impact on the performance of the system. Consequently, these parameters are essential for the performance analysis of the Hadoop applications and are useful for the DICE Simulation, Verification and Optimization tools.

#	Concept	Meaning
1.	*Map* (task)	Filtering phase
2.	*Reduce* (task)	Composition of results phase
3.	*Workload*	Number of tasks per user in the cluster
4.	*Scheduling*	Policy for assigning cluster resources to the map and reduce phases

Table 6: Storm concepts that impact in performance

The Hadoop framework[1] allows for parallel and distributed computing on large scale clusters of commodity hardware, focusing on batch and, nowadays, also interactive processing of huge datasets. Furthermore, it guarantees beneficial properties of fault-tolerance, scalability, and automatic parallelization and distribution of computation across the cluster, at the expense of a simple and stiff programming paradigm.

MapReduce jobs are composed of two main phases, namely, map and reduce. The former takes as input unstructured data from the filesystem, filtering them and performing a preliminary elaboration, according to the instructions in a user-defined function. The intermediate results are returned as key-value pairs, grouped by key in the shuffle phase and distributed across the network, so as to provide each node taking part in the reduce phase with all the values associated with a set of keys. In the end, every reducer applies a second user-defined function to complete data elaboration and outputs the result. Each map and reduce phase is carried out by a set of concurrent tasks, i.e., mappers and reducers.

More recent versions of Hadoop allocate cluster resources relying on a hierarchical resource management framework. The central resource manager has the role of providing resources for the execution of jobs, based on a configurable scheduling policy and the workload of the cluster. In previous versions,

[1] `http://hadoop.apache.org`

resources, i.e., cpu slots, were allocated statically to the mappers and reducers. Recently, the new version Hadoop allows for the dynamic assignation of resources among ready tasks, leading to a better utilization of the cluster. It is possible to supply a custom scheduler, nonetheless the Hadoop project provides three ready general purpose alternatives: the fifo, Fair, and Capacity schedulers.

For instance, the fifo scheduler is very basic and follows the scheduling policy with a single global queue for all the users served on a fifo basis. More refined schedulers such as Fair and Capacity schedulers split the global queue into several ones (for instance, depending on the number of users or institutions that utilize the cluster), and internally manage the queue of jobs on a fifo basis in order to prevent starvation among users.

The parameters of the Hadoop MapReduce cluster are captured by the updated DICE::DTSM::Hadoop profile annotations presented in Table 13. Table 13 links the Hadoop MapReduce concepts with the DICE::DTSM::Hadoop profile annotations. The DICE::DTSM::Hadoop profile includes five genuine stereotypes that are created for representing the schedulers, the workload, the mappers, the reducers and the cluster. The stereotypes and annotations for Hadoop are based on MARTE [12], DAM [13], the DICE::DPIM and Core profiles. The DICE::DTSM::Hadoop stereotypes inherit or refine information from the mentioned profiles and also add new information. For instance, part of the annotations for the mapper and reducer stereotypes use MARTE-DAM for including temporal information (i.e., host demands) in the UML models.

The *workload* is described by the *«HadoopWorkloadEvent»* stereotype through two tags: *hadoopPopulation* and *hadoopExtDelay*. These annotations define the number of jobs for each class (user) that are initially in the system; and the arrival rate of each class of job to the cluster. The number of jobs of each class is specified by the *hadoopPopulation* tag, i.e., an array of integers. The arrival rate of jobs is specified by the *hadoopExtDelay* tag, i.e., also an array of integers. The element *i* of the arrays represent the number of jobs of the class *i*; and the time between the arrival of a new job of the class *i* to the cluster and the next one.

The *mappers* are functions that divide and preprocess the data during the mapping phase. The composition of the intermediate results is done by reduce functions (i.e., *reducers*) in the reducing phase. They perform Hadoop operations and, for that reason, they belong to the same *«HadoopOperation»* stereotype. That is, the *«HadoopOperation»* stereotype models any operation in the Hadoop cluster. The stereotype includes the tag *nTasks*, i.e., an array of integers for specifying the number map (reduce) tasks in which a class of job is divided. Other tags of the stereotype are inherited from MARTE. For instance, *hostDemand* is used for indicating the execution time of the map (reduce) function.

Next, the stereotype *«HadoopScenario»* includes information of the cluster context. For instance, the *scheduler* of the Hadoop MapReduce cluster is determined by the *jobSchedule* tag. It is an enumerable of the type *Scheduling* that can be any of three common schedulers ({capacity, fifo, fair}).

Finally, the *«HadoopComputationNode»* stereotype is used for annotating the computational nodes of a Hadoop cluster in a UML deployment diagram. It defines the physical assignation of map (reduce) functions to hardware resources during the execution of the system. The stereotype includes an array of integers that associates a number of computational resources to each class of job.

5.4 Updating DTSM to support Apache Storm

The DTSM Profile for the Apache Storm technology is one of the main novelties of this deliverable. The DTSM Profile includes a list of stereotypes that addresses the main concepts of the Apache Storm technology identified in Table 8. In particular, we stress those concepts that directly impact on the performance of the system. Consequently, these parameters are essential for the performance analysis of the Storm applications and are useful for the DICE Simulation, Verification and Optimization tools.

Storm is a distributed real-time computation system for processing large volumes of high-velocity data [20]. A Storm application is usually designed as a directed acyclic graph (DAG) whose *nodes* are the points where the information is generated or processed, and the *edges* define the connections for the transmission of data from one node to another. Two classes of nodes are considered in the topology. On the one hand, *spouts* are sources of information that inject *streams* of data into the topology at a

#	Concept	Meaning
1.	*Spout* (task)	Source of information
2.	*Emission rate*	Number of tuples per unit of time produced by a spout
3.	*Bolt* (task)	Data elaboration and production of results
4.	*Execution time*	Time required for producing an output by a bolt
5.	*Ratio*	Number of tuples required or produced
6.	*Asynchronous policy*	The bolt waits until it receives tuples from any of the incoming streams
7.	*Synchronous policy*	The bolt waits until it receives tuples from all the incoming streams
8.	*Parallelism*	Number of concurrent threads per task
9.	*Grouping*	Tuple propagation policy (shuffle/all/field/global)

Table 8: Storm concepts that impact in performance

certain *emission rate*. On the other hand, *bolts* consume input data and produce results which, in turn, are emitted towards other bolts of the topology.

A bolt represents a generic processing component that requires n tuples for producing m results. This asymmetry is captured by the *ratio* $\frac{m}{n}$. Spouts and bolts take a certain amount of time for processing a single tuple.

Besides, different *synchronization* policies shall be considered. A bolt receiving messages from two or more sources can select to either 1) progress if at least a tuple from any of the sources is available (*asynchronously*), or 2) wait for a message from all the sources (*synchronously*).

A Storm application is also configurable by the parallelism of the nodes and the stream grouping. The *parallelism* specifies the number of concurrent tasks executing the same type of node (spout or bolt). Usually, each task corresponds to one thread of execution. The *stream grouping* determines the way a message is propagated to and handled by the receiving nodes. By default, a message is broadcasted to every successor of the current node. Once the message arrives to a bolt, it is redirected randomly to any of the multiple internal threads (*shuffle*), copied to all of them (*all*) or to a specific subset of threads according to some criteria (i.e., *field*, *global*, etc.).

In summary, a Storm framework is highly configurable by various parameters that will influence the final performance of the application. These concepts are converted into stereotypes and tags, which are the extension mechanisms offered by UML. Therefore, we devised a new UML profile for Storm. In our case, we are extending UML with the Storm concepts. The stereotypes and annotations for Storm are based on MARTE [12], DAM [13], the DICE::DPIM and Core profiles. The DICE::DTSM::Storm profile provides genuine stereotypes (see Table 14).

Spouts and *bolts* have independent stereotypes because they are conceptually different, but *«StormSpout»* and *«StormBolt»* inherit from MARTE::GQAM::GaStep stereotype via the DTSM::Core::CoreDAGNode and CoreDAGSourceNode since they are computational steps. Moreover, they share the *parallelism*, or number of concurrent threads executing the task, which is specified by the tag `parallelism`.

On the other hand, the spouts add the tag `avgEmitRate`, which represents the *emission rate* at which the spout produces tuples. Finally, the bolts use the `hostDemand` tag from GaStep for defining the task execution time. The time needed to process a single tuple can also be expressed through the `alpha` tag in case that the temporal units are not specified. The minimum and the maximum time to recover from a failure is denoted by the *minRebootTime* and *maxRebootTime* tags.

The Storm concept of *stream* is captured by the stereotype *«StormStreamStep»*. This stereotype also inherits from MARTE::GQAM::GaStep stereotype, which enables to apply it to the control flow arcs of the UML activity diagram. This stereotype has two tags, `grouping` and `numTuples` that match the *grouping* and *ratio* concepts in Storm, respectively. The type of `grouping` is StreamPolicy, an enumeration type of the package Basic_DICE_Types that has the values {all, shuffle, field, global} for indicating the message passing policy. The ratio of $\frac{m}{n}$ of a bolt can be expressed either through the attribute `sigma` in the bolt stereotype, or by specifying the incoming and outcoming `numTuples` of a

bolt via the stream stereotype.

Finally, the *«StormScenarioTopology»* stereotype is introduced for defining contextual execution information of a Storm application. That is, the reliability of the application or the buffer size of each task for exchanging messages.

5.5 Example of MARTE annotation

The Apache Storm and Apache Hadoop MapReduce profiles heavily rely on the standard MARTE profile [12]. This is because MARTE offers the GQAM sub-profile, a complete framework for quantitative analysis, which is indeed specialized for performance analysis, then perfectly matching to our purposes. Moreover, MARTE offers the NFPs and VSL sub-profiles. The NFP sub-profile aims to describe the non-functional properties of a system, performance in our case. The latter, VSL sub-profile, provides a concrete textual language for specifying the values of metrics, constraints, properties, and parameters related to performance, in our particular case.

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

```
(expr=$b_1, unit=ms, statQ=mean, source=est)
    (1)        (2)        (3)        (4)
```

This expression specifies that the operation modelled in the UML diagram demands `$b_1` (1) *milliseconds* (2) of processing time, whose mean value (3) will be obtained from an estimation in the real system (4). `$b_1` is a variable that can be set with concrete values during the analysis of the model.

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

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

This expression specifies that we want to calculate (3) the `utilization`, as a percentage of time, of the whole system or a specific resource, whose mean value (2) will be obtained in variable `$use` (1). Such value is obviously the result of the simulation of the performance model. The examples presented here try to guide the use of MARTE expressions during the modelling of Big Data applications with the DICE Profile.

6 Conclusions

In this document we have presented the latest versions of the DICE Metamodels and DICE Profile, which are the main outcomes for Tasks T2.1 and T2.2, respectively. The updated DICE Metamodels and Profiles include:

- a new SecureUML metamodel and profile for the specification and support of privacy-by-design annotations,
- reliability annotations for modules at DPIM level, and
- a stable and updated version of the DICE Profile for the Apache Hadoop MapReduce and Apache Storm technologies.

The DICE Metamodels and Profiles at DDSM level will be discussed in the DICE *Deployment abstractions - Final version* deliverable in M27. This deliverable will also include the reliability annotations for modules at the DTSM level.

Table 9 summarizes the main achievements of this deliverable in terms of compliance to the initial set of requirements presented in Section 2. The labels specifying the *Level of fulfillment* should be understood as follows: (i) ✗ (unsupported: the requirement is not fulfilled by the current version); (ii) ✔ (partially-supported: most of the aspects of the requirement are fulfilled by the current version); and (iii) ✔ (supported: the requirement is fulfilled by the current version).

The notes shown in the last column of the table provide an explanation in the case the level of fulfillment of some requirement is not complete and, with the exception of R2.9 whose full fulfillment is out of the scope of DICE, indicate when the requirement will be fully addressed.

6.1 Further Work and Roadmap

As demonstrated by the status of Table 9, the work in WP2 is completed in its main parts. The points that require some more attention are the following:

- The reliability Profile annotations at the DTSM level. These will be released at M27 as part of Deliverable D2.4.
- The inclusion in the Metamodels and Profile of the set of technologies adopted within the Case Studies. As discussed in this deliverable, the Metamodels and Profile have been designed to be extensible and to accomodate the never-ending need of adding new technologies. The additions that will be developed for the purpose of supporting the DICE case studies will be reported in Deliverable D2.5 *DICE metodology*, due at month 30, and in the final release of the case studies themselves.
- Finally, the Deployment and Architecture Trade-Off Transformations are the main subject of Deliverable D2.4.

Table 9: Level of compliance of the current version with the initial set of requirements

Requirement	Title	Priority	Level of fulfillment	Notes
R2.0	Profile Structure	MUST	✔	
R2.1	Profile Basis	MUST	✔	
R2.2	Abstraction Layer Origin	MUST	✔	
R2.3	Relation with MARTE UML Profile	MUST	✔	
R2.4	DICE Constraints Specification	MUST	✔	
R2.5	DICE Profile Performance Annotations	MUST	✔	
R2.6	DICE Profile Reliability Annotations	MUST	✔	Reliability annotations at DTSM level will be released within D2.4
R2.7	DICE Profile Main DIA Concerns - Structure and Topology	MUST	✔	
R2.8	DICE Profile Main DIA Concerns - Flow and Behavior	MUST	✔	
R2.9	DICE Profile Pre- and Post-Processing	MUST	✔	Each technological sub-profile addresses this requirement.
R2.10	DICE Profile Tech-Specific Constraints	MUST	✔	New technologies will be incorporated based on the needs of DICE case studies. A report on this will be provided as part of D2.5
R2.11	DICE Profile Separation-of-Concerns	MUST	✔	
R2.12a	DICE Profile Supervision and Control	MUST	✔	
R2.12b	DICE Privacy and Secure Aspects	MUST	✔	
R2.13	DICE Profile Data Structure	MUST	✔	
R2.14	DICE Profile Data Communication	MUST	✔	
R2.15	DICE Profile Sub-Structures	MUST	✔	
R2.18	DICE Deployment Transformation	MUST	✔	This work will be finalized within D2.4
R2.20	DICE Architecture Trade-Off Transformation	MUST	✔	Support to this is under development and will be delivered as part of D2.4

References

[1] The DICE Consortium. *DICE Models Repository*. URL: https://github.com/dice-project/DICE-Models. Dec., 2015.

[2] The DICE Consortium. *DICE Profiles Repository*. URL: https://github.com/dice-project/DICE-Profiles. Dec., 2015.

[3] The DICE Consortium. *Design and Quality Abstractions - Initial Version*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/02/D2.1_Design-and-quality-abstractions-Initial-version.pdf. European Union's Horizon 2020 research and innovation programme, 2015.

[4] The DICE Consortium. *State of the Art Analysis*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.1_State-of-the-art-analysis1.pdf. European Union's Horizon 2020 research and innovation programme, 2015.

[5] The DICE Consortium. *Deployment Abstractions - Final Version*. Tech. rep. To Appear. European Union's Horizon 2020 research and innovation programme, 2015.

[6] The DICE Consortium. *Requirement Specification*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.2_Requirement-specification.pdf. European Union's Horizon 2020 research and innovation programme, 2015.

[7] The DICE Consortium. *Requirement Specification - Companion Document*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2015/08/D1.2_Requirement-specification_Companion.pdf. European Union's Horizon 2020 research and innovation programme, 2015.

[8] *DICE Requirement List (Online version)*. 2015. URL: https://docs.google.com/spreadsheets/d/1Wn9OXGsTknrAs5ASUadOp9IpQ9BM_WM4NsyAuXL6_Ug/edit?usp=sharing.

[9] Douglas C. Schmidt. "Model Driven Engineering". In: *IEEE Computer* 39.2 (Feb. 2006), pp. 25–31.

[10] The Object Management Group (OMG). *Model-Driven Architecture Specification and Standardisation*. Tech. rep. URL: http://www.omg.org/mda/.

[11] The Object Management Group (OMG). *Unified Modelling Language: Infrastructure, 2011. Version 2.4.1. OMG document: formal/2011-08-05*. Tech. rep. URL: http://., 2011.

[12] OMG. *UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, Version 1.1*. URL: http://www.omg.org/spec/MARTE/1.1/. Object Management Group, June 2011. URL: http://www.omg.org/spec/MARTE/1.1/.

[13] S. Bernardi, J. Merseguer, and D. C. Petriu. "A dependability profile within MARTE." In: *Software and Systems Modeling* 10.3 (2011), pp. 313–336.

[14] The DICE Consortium. *Transformations to Analysis Models*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/08/D3.1_Transformations-to-analysis-models.pdf. European Union's Horizon 2020 research and innovation programme, 2016.

[15] *Formal Concept Analysis, Foundations and Applications*. Vol. 3626. Springer, 2005.

[16] G. Casale et al. "DICE: Quality-Driven Development of Data-Intensive Cloud Applications". In: *Proceedings of the 7th International Workshop on Modelling in Software Engineering (MiSE)*. May 2015, pages.

[17] T. Lodderstedt, D. Basin, and J. Doser. *SecureUML: A UML-based modeling language for model-driven security*. 2002. URL: citeseer.ist.psu.edu/lodderstedt02secureuml.html.

[18] Eric Simmon and Robert B. Bohn. “An Overview of the NIST Cloud Computing Program and Reference Architecture.” In: *ISPE CE*. Ed. by Josip Stjepandic, Georg Rock, and Cees Bil. Springer, 2012, pp. 1119–1129. ISBN: 978-1-4471-4426-7. URL: %5Curl%7Bhttp://dblp.uni-trier.de/db/conf/ispe/ispe2012.html#SimmonB12%7D.

[19] The DICE Consortium. *Demonstrators Implementation Plan*. Tech. rep. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/06/D6.1-Demonstrators-implementation-plan.pdf. European Union’s Horizon 2020 research and innovation programme, 2015.

[20] *Apache Storm Website*. URL: http://storm.apache.org/.

Appendix A. Profile mappings

Next we detail the DICE profile. The engineer only needs to use those DICE tags that are useful for him/her to describe the UML model element at hand.

A.1 DICE Profile: The DICE::DICE_UML_Extensions::DPIM package

Table 11: The DICE::DICE_UML_Extensions::DPIM package

DICE DTSM::Core Stereotype Element	**Inheritance**	**DICE Tags**
DpimScenario	«DpimScenario» inherits from «DAM::DAM_UML_Extensions::System::Core::DaService»	
DpimFilterNode	«DpimFilterNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::DpimComputationNode»	New tags: • inputRatio: NFP_Frequency • outputRatio: NFP_Frequency
DpimVisualizationNode	«DpimVisualizationNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::DpimComputationNode»	
DpimSourceNode	«DpimSourceNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::DpimComputationNode»	New tags: • store: DiceDataVolume • provides: DiceDataSpecification • sourceType: SourceType • rate: NFP_Frequency
DpimComputationNode	«DpimComputationNode» inherits from «DAM::DAM_UML_Extensions::System::Core::DaComponent»	New tags: • nodeThroughput: NFP_Frequency • type: ComputationType • targetTech: TechType • procType: ProcessingType
DpimStorageNode	«DpimStorageNode» inherits from «MARTE::MARTE_Foundations::GRM::StorageResource»	New tags: • respondsTo: DiceDataSpecification • crudRate: NFP_Frequency
DpimChannel	«DpimChannel» inherits from «DAM::DAM_UML_Extensions::System::Core::DaConnector»	New tags: • rate: NFP_Frequency • messageBroker: String • channelDescription: DiceChannelSpecification

A.2 DICE Profile: The DICE::DICE_UML_Extensions::DTSM::Core package

Table 12: The DICE::DICE_UML_Extensions::DTSM::Core package

DICE DTSM::Core Stereotype Element	**Inheritance**	**DICE Tags**
CoreComputationNode	«CoreComputationNode» inherits from «DICE::DICE_UML_Extensions::DPIM::DpimComputationNode»	New tags: • hasSuccessor: CoreComputationNode[*] • processInputData: CoreData[*] • produceOutputData: CoreData • realise: CoreDirectAcyclicGraph
CoreStorageNode	«CoreStorageNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreDataSource»	New tags: • database: String • password: String • user: String
CoreDAGSourceNode	«CoreDAGSourceNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreDAGNode»	New tags: • readFrom: CoreDataSource
CoreDirectAcyclicGraph	«CoreDirectAcyclicGraph» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaScenario» and «Metaclass(UML)Class»	New tags: • hasSourceNode: CoreDAGSourceNode
CoreData	«CoreData» inherits from «Metaclass(UML)Classifier»	New tags: • hasSpecification: DiceDataSpecification • hasVolume: DiceDataVolume • location: CoreDataSource
CoreDataSource	«CoreDataSource» inherits from «MARTE::MARTE_Foundations::GRM::StorageResource»	
CoreDAGNode	«CoreDAGNode» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaStep»	New tags: • hasSuccessor: CoreDAGNode[*] • operation: WorkflowOperation • parallelism: NFP_Integer

A.3 DICE Profile: The DICE::DICE_UML_Extensions::DTSM::Hadoop package

Table 13: The DICE::DICE_UML_Extensions::DTSM::Hadoop package

DICE DTSM::Hadoop Stereotype Element	**Inheritance**	**DICE Tags**
HadoopScenario	«HadoopScenario» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaScenario»	New tags: • jobSchedule: Scheduling
HadoopWorkloadEvent	«HadoopWorkloadEvent» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaWorkloadEvent»	New tags: • pattern: ArrivalPattern • hadoopPopulation: NFP_Integer[*] • hadoopExtDelay: NFP_Duration[*]
HadoopOperation	«HadoopOperation» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaStep»	New tags: • parallelism: NFP_Integer • nTasks: NFP_Integer[*]
HadoopMap	«HadoopMap» inherits from «DICE::DICE_UML_Extensions::DTSM::Hadoop::HadoopOperation»	New tags: • type: MapType
HadoopReduce	«HadooReduce» inherits from «DICE::DICE_UML_Extensions::DTSM::Hadoop::HadoopOperation»	New tags: • type: ReduceType
HadoopComputationNode	«HadoopComputationNode» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreComputationNode»	New tags: • nCores: NFP_Integer[*]
HadoopMapReduceJob	«HadoopMapReduceJob» inherits from «Metaclass(UML)Classifier»	New tags: • mapResucePhases: HadoopMapReducePhase
HadoopMapReducePhase	«HadoopMapReducePhase» inherits from «Metaclass(UML)Classifier»	New tags: • hasMap: HadoopMap • hasReduce: HadoopReduce • output: CoreData

A.4 DICE Profile: The DICE::DICE_UML_Extensions::DTSM::Storm package

Table 14: The DICE::DICE_UML_Extensions::DTSM::Hadoop package

DICE DTSM::Storm Stereotype Element	**Inheritance**	**DICE Tags**
StormScenarioTopology	«StormScenarioTopology» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreDirectAcyclicGraph»	New tags: • queueThreshold: Integer • maxTaskParallelism: Integer • maxSpoutPending: Integer • isReliable: Boolean
StormBolt	«StormBolt» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreDAGNode»	New tags: • failure: ArrivalPattern • d: NFP_Real • alpha: NFP_Real • sigma: NFP_Real • minRebootTime: NFP_Real • maxRebootTime: NFP_Real
StormSpout	«StormSpout» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreDAGSourceNode»	New tags: • avgEmitRate: NFP_Real
StormStreamStep	«StormStreamStep» inherits from «MARTE::MARTE_AnalysisModel::GQAM::GaStep»	New tags: • numTuples: NFP_Integer • grouping: StreamPolicy
StormApplication	«StormApplication» inherits from «DICE::DICE_UML_Extensions::DTSM::Core::CoreComputationNode»	New tags: • hasMasterNode: StormNimbus • hasSlaveNode: StormSupervisor • dependsOnZookeeper: StormZookeeper
StormNimbus	«StormNimbus» inherits from «Metaclass(UML)Classifier»	New tags: • taskTimeout: Integer • supervisorTimeout: Integer • monitorFrequency: Integer

StormSupervisor	«StormSupervisor» inherits from «Metaclass(UML)Classifier»	New tags: • monitorFrequency: Integer • workerSatrtTimeout: Integer • workerTimeout: Integer • heatbrackFrequency: Integer • memoryCapacity: Integer • cpuCapacity: Integer
StormZookeeper	«StormZookeeper» inherits from «Metaclass(UML)Classifier»	New tags: • sessionTimeout: Integer • connectionTimeout: Integer • retryTime: Integer • retryInterval: Integer • user: String • password: String

A.5 DICE model library

The DICE model library contains basic and complex types that are used by the DICE UML extensions.

A.5.1 The DICE::DICE_Library::Basic_DICE_Types package

Table 15: The DICE::DICE_Library::Basic_DICE_Types package

Basic_DICE_Types Type Name	Kind	Values
ComputationType	Enumeration	distributed, parallel, distributedParallel, microBench, sorting, grep, wordCount, collabFiltering, naiveBayes, bfs, pageRank, kMeans, connectedComponents, relQuery
TechType	Enumeration	hadoop, spark, storm, oryx2
ProcessingType	Enumeration	stream, batch
SourceType	Enumeration	dataStream, webapi
RefType	Enumeration	mongodb, hdfs, mysql, cassandra
RefDataFormatType	Enumeration	JSON, plain, xml, avro, parquet, yaml
ConstraintsType	Enumeration	less, lessEqual, equal, greaterEqual, greater
MapType	Enumeration	regexMapper, fieldSelectionMapper, chainMapper
ReduceType	Enumeration	intSumReducer, fieldSelectionReducer, chainReducer
Scheduling	Enumeration	capacity, fifo, fair
WorkFlowOperation	Enumeration	groupBy, sum, count, split
StreamPolicy	Enumeration	all, shuffle, global, field
VMSize	Enumeration	small, medium, large

ProviderType	Enumeration	flexiant, openstack
LifeCycleElementTy	Enumeration	start, stop, install, create, download, preconfigured
DDSMcomponentType	Enumeration	MasterSlavePlatform, PeerToPeerPlatform, PeerNode, PeerQuorum, CassandraSeed, MasterNode, SlaveNode
NFP_Privacy	Data Type	• expr: VLS_Expression • source: SourceKind
ScriptType	Data Type	• scriptId: EString • scriptUri: EString • scriptKind: LifeCycleElementType

A.5.2 The DICE::DICE_Library::Complex_DICE_Types package

Table 16: The DICE::DICE_Library::Complex_DICE_Types package

Complex_DICE_Types Type Name	Attributes
DiceDataVolume	• volume: NFP_DataSize
DiceDataSpecification	• description: String • size: NFP_DataSize • refModel: RefType • refDataFormat: RefDFType
DiceChannelSpecification	• rate: NFP_Frequency • size: NFP_DataSize