Enhancing Big Data Application Design with the DICE Framework

Giuliano Casale, Chen Li *

Abstract. The focus of the DICE project is to define a quality-driven framework for developing Big data applications. DICE offers an Eclipse-based development environment, centered around a novel UML profile, to prototype, deploy, monitor, and test Big data applications. The DICE framework has been natively designed to natively support popular open-source solutions. The framework offers a set of 15 open source tools, which have been validated against industrial case studies in the news and media, port operations, and e-government domains.
Keywords: quality, Big Data, UML, Eclipse, DevOps

1 Overview

DevOps has become mainstream in recent years as a movement that attempts to lower the barrier between IT development and operation teams in order to increase the agility of the application release process. One requirement to implement this successfully is to share a unified set of concepts, models, and tools among developers and operators. Even though DevOps tools are rapidly growing in the industry, few of them are natively designed to support Big data applications, even though these have gained much traction in the industry in recent years. By Big data application, we here mean applications that rely on core Big data processing technologies such as Storm, Cassandra, Hadoop/MapReduce, Spark, MongoDB, and many others.

The main objective of the DICE framework is to deliver a quality-driven DevOps toolchain for Big data applications that natively support these Big data technologies. The DICE architecture is shown in Figure 1 and is centered around the following main components:

– **DICE IDE**: this is an Eclipse-based IDE built upon a new UML profile that abstracts the main characteristics of open source Big data technologies. For example, with the DICE UML profile, a developer can express requirements on the number and topology of spouts and bolts in a Storm-based application, and associate them to specific virtual machine instances in the cloud.

* G. Casale and C.Li are with Imperial College London, UK. Contact: {g.casale,chen.li1}@imperial.ac.uk . This paper has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869. Project full name: DICE - Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements: Feb 2015-2018, website: www.dice-h2020.eu.

Moreover, the Papyrus-based UML diagram visualization allows to iteratively refine the application design using a methodological workflow integrated within the IDE as Eclipse cheat-sheets.
- **Quality analysis plugins**: these tools include discrete-event simulation and verification of the application architecture in the early design stages. These tools allow in particular to predict what latency will be experienced by the users and verify quality guarantees offered by the design. The plugins also include an architecture optimization tool that decides the resources to be assigned upon deployment to the Big data application (e.g., number and type of VMs).
- **Continuous delivery and testing tools**: these tools take care of continuous integration, deployment, configuration, load testing, and fault injection of the application in the runtime environment. The deployment tool, in particular, is TOSCA-compliant and can be activated directly from within the DICE IDE, allowing developers to reduce the time they need to deploy a Big data application prototype to the cloud. Supported cloud environments include AWS, OpenStack, and Flexiant FCO.
- **Monitoring and feedback analysis tools**: these tools collect monitoring data from the running application and analyze it by means of machine learning-based anomaly detection, trace checking, and statistical inference techniques. Altogether, they allow examining whether the application is behaving as expected in the runtime environment. Moreover, these tools supply monitoring information to the development environment, to guide future iterations on the prototypes.

The above capabilities have been validated on industrial demonstrators concerning processing news streams from Twitter, cloud-based software systems for port operations, and Big data applications for tax fraud detection. The outcome of these validations is that an integrated toolchain can substantially accelerate the design and testing of Big data applications, without requiring steep learning curves.

2 The DICE Enhancement tool

In this section, we illustrate a specific result of the DICE project, the *DICE Enhancement tool*. The goal of this tool is to provide feedback to DICE developers on the application behaviour at runtime, leveraging the monitoring data collected from the DICE Monitoring Platform (DMon), in order to help them iteratively enhance the application design. The DICE Enhancement tools include two modules: the DICE Filling-the-Gap (DICE-FG), which helps focusing on statistical estimation of UML parameters used by the quality analysis plugins for simulation and optimization, and DICE-APR (Anti-Patterns and Refactoring), a tool that detects anti-patterns in the UML design and supplies recommendation on how to resolve them.

The methodological workflow of this tool is shown in Figure 2. DICE-FG provides a performance and reliability analysis of big data applications and up-

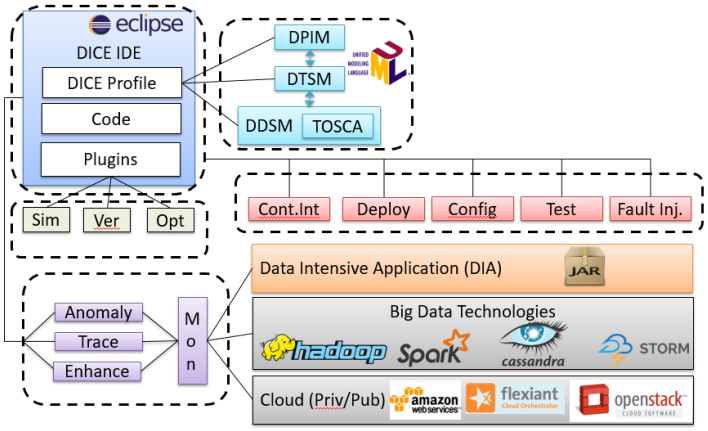

Fig. 1. DICE framework - available at http://github.com/dice-project/.

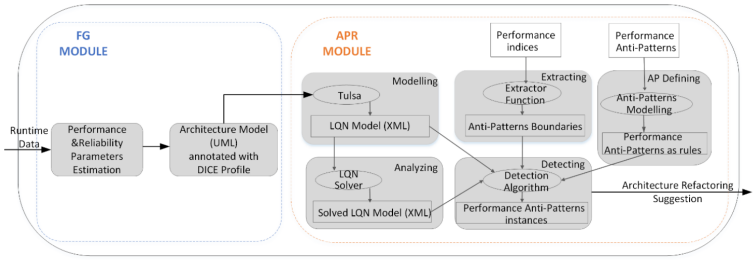

Fig. 2. DICE enhancement tool workflow.

dates UML models with analysis results. DICE-APR transforms the DICE UML model annotated with DICE profiles into a Layered Queueing Network (LQN), which is a queueing model that can be used for performance prediction. For example, an LQN can predict the latency expected for the application under a given workload. The results are predicted CPU utilizations and latencies at the resources, which will be used for APs detection and for recommending refactoring decisions.

DICE-FG is designed to achieve the following objectives:

- Provide statistical estimation algorithms to infer resource consumption of an application.
- Provide fitting algorithms to match monitoring data to parametric statistics distributions.
- Use the above algorithms to parameterize UML models annotated with the DICE profile.
- Acquire data via JSON and the DICE Monitoring platform (DMon).

The main logical components of the DICE-FG tool are the Analyzer and the Actuator.

- **DICE-FG Analyzer**: The DICE-FG Analyzer executes the statistical methods necessary to obtain the estimates of the performance models parameters, relying on the monitoring information available on the input files.
- **DICE-FG Actuator**: The DICE-FG Actuator updates the parameters in the UML models, e.g., resource demands, which are obtained from the DICE-FG Analyzer.

The DICE-APR module is designed to achieve the following objectives:

- Transforming UML diagrams annotated with DICE profiles to performance model (i.e., Layered Queueing Networks) for performance analysis.
- Specifying the selected popular AP of DIAs in a formal way (e.g., a logic formula which is suitable for model checking, executable codes).
- Detecting the potential AP from the performance model.
- Generating refactoring decisions to update the architecture model to fix the design flaws according to the AP solution.

The components of the DICE-APR module are Model-to-Model Transformation, Anti-patterns Detection and Architecture Refactoring as detailed below.

- **Model-to-Model Transformation**: It provides the transformation of annotated UML model with DICE Profile into quality analysis model. The target performance models is Layered Queueing Networks.
- **Anti-patterns Detection**: The Anti-patterns detection component relies on the analysis results of the Model-to-Model Transformation component. The selected anti-patterns are formally specified for identifying if there are any anti-patterns issues in the model.
- **Architecture Refactoring**: According to the solution of discovered anti-patterns, refactoring decisions will be proposed, e.g., component reassignment, to solve them. The architecture model will be shared back to the DICE IDE for presentation, to the user in order to decide if the proposed modification should be applied or not.

The implementation of the model-to-model transformations, from DICE UML to LQN, used by DICE-APR rely on a tool called Tulsa [2], which is based on the Epsilon language (i.e., Epsilon Transformation Language (ETL) and Epsilon Object Language (EOL)). The LINE solver[1] is then used compute performance metrics from the LQN model generated by Tulsa. The APs detection is implemented using Matlab scripts that iteratively call the LINE API to determine if a specific refactoring action will yield a better latency or resolve CPU bottlenecks.

We have applied the DICE APR tool to case studies of the Wikistats open source application, which is based on Apache Storm, finding that it can effectively detect and resolve two common antipatterns: Infinite Wait (IW) and Excessive Calculation (EC):

[1] LINE website: http://line-solver.sf.net.

– IW occurs when a component must call several servers to complete the task. If a large amount of time is required for each service, performance will suffer. The solution is to report the component that causes the IW and provide component replication or redesign suggestions to the developer.
– EC occurs when a processor performs all of the work of an application or holds all of the application data. This anti-pattern manifests itself as an excessive amount of CPU calculations that degrade performance. The solution is to report to the developer the processor that causes the EC and suggest to add a new processor to migrate tasks to the developer.

We point to [3] for details and experimental results of the DICE APR tool.

A pre-requisite for the APR tool to generate meaningful predictions is to know how many resources each software components requires to carry out its operation, under a given workload demand. The DICE FG tool delivers this capability, based on a number of statistical estimators that transform monitoring metrics such as throughputs and utilizations into CPU requirements for the application. DICE-FG uses a collection of algorithms, ranging from regression methods to statistical inference techniques over queueing networks. We have recently reported on the relative accuracy of these methods in estimating resource demands [4], finding that no single method dominates the others, thus a module like FG that offers multiple estimation algorithms allows to adapt the situation to the different situations that arise in practice.

3 Conclusion

The DICE IDE and updates on the project are available at http://www.dice-h2020.eu, together with a project vision document [1]. In the future we plan to release specialized "thinned" product versions of the IDE dedicated to specific technologies, such as Apache Storm, in order to supply to end users an environment targeting specific application classes (e.g., stream-based applications).

References

1. G. Casale and *et al.* DICE: Quality-driven development of data-intensive cloud applications. In *Proc. of MiSE workshop*, 2015.
2. C. Li, et al. Tulsa: A tool for transforming UML to layered queueing networks for performance analysis of data intensive applications. *Proc. of QEST*, 295–299, 2017.
3. C. Li and G. Casale. Performance-aware refactoring of cloud-based big data applications. *Under review, pre-print available upon demand.*, 2017.
4. S. Spinner, G. Casale, F. Brosig, and S. Kounev. Evaluating approaches to resource demand estimation. *Perform. Eval.*, 92:51–71, 2015.