

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



Iterative quality enhancement tools – Final version

Deliverable 4.6

Deliverable:	D4.6
Title:	Iterative quality enhancement tools – Final version
Editor(s):	Giuliano Casale (IMP), Chen Li (IMP)
Contributor(s):	Giuliano Casale (IMP), Chen Li (IMP), Jose-Ignacio Requeno (ZAR), Marc Gil (PRO)
Reviewers:	Youssef RIDENE (NETF), Simona Bernardi (ZAR)
Type (R/DEM/DEC):	DEM
Version:	1.0
Date:	27-July-2017
Status:	Final version
Dissemination level:	Public
Download page:	http://www.dice-h2020.eu/deliverables/
Copyright:	Copyright © 2017, DICE consortium – All rights reserved

DICE partners

ATC:	Athens Technology Centre
FLEXI:	Flexiant Limited
IEAT:	Institutul e-Austria Timisoara
IMP:	Imperial College of Science, Technology & Medicine
NETF:	Netfective Technology SA
PMI:	Politecnico di Milano
PRO:	Prodevelop SL
XLAB:	XLAB razvoj programske opreme in svetovanje d.o.o.
ZAR:	Universidad de Zaragoza



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 deliverable documents the final work on tools for iterative quality enhancement, developed as part of task T4.3. Therefore, it provides the final versions of the DICE-FG and DICE-APR. This deliverable is an incremental update of the deliverable 4.5 (*Iterative quality enhancement tools – Initial version*) published in M18. The document exclusively focuses on the new contributions of the DICE-APR with respect to the previous deliverable. Mainly this work covers 1) support for transforming UML diagrams annotated with DICE profiles to performance model (i.e., Layered Queueing Network), 2) identify popular anti-patterns of DIAs, 3) provide refactoring decisions for a designer if the selected anti-patterns are found in the performance model, and 4) demonstrate applicability to a specific technology (Apache Storm).

Glossary

APR	Anti-Patterns & Refactoring
APDR	Anti-Patterns Detection and Refactoring
DDSM	DICE Deployment Specific Model
DIAs	Data-intensive applications
DICE	Data-Intensive Cloud Applications with iterative quality enhancements
DMon	DICE Monitoring platform
DPIM	DICE Platform Independent Model
DTSM	DICE Technology Specific Model
FG	Filling-the-Gap
LQN	Layered Queueing Network
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MCR	MATLAB Compiler Runtime
MODAClouds	MOdel-Driven Approach for design and execution of applications on multiple Clouds
M2M	Model-to-Model Transformation
Tulsa	Transformation from UML model to Layered queueing networks for Storm-based Application
UML	Unified Modelling Language
VM	Virtual Machine

Table of contents

Executive summary.....	3
Glossary	4
Table of contents.....	5
List of Figures	7
List of Tables	7
1. Introduction	8
1.1. What is new in Year 3.....	8
1.1.1 DICE Anti-Patterns & Refactoring.....	8
1.1.2 DICE Filling-the-Gap tool.....	9
1.2. Structure of the Document	9
2. Requirements.....	10
2.1 Requirements	10
3. DICE-APR Tool.....	12
3.1 Model Transformation Module.....	12
3.1.1 Transformation design.....	13
3.1.2 Source model	14
3.1.3 Target model.....	15
3.1.4 Mapping rules.....	16
3.2 Anti-Patterns Detection & Refactoring.....	18
3.2.1 Overview of popular Anti-Patterns.....	18
3.2.2 Anti-Patterns selection and specification	18
3.2.3 Anti-Patterns Detection & Refactoring process	20
3.3 Tool Usage	22
3.3.1 Tulsa Configuration.....	23
3.3.2 Running APR	24
3.3.3 APR configuration file	24
3.4 Obtaining the Tool	25
3.5 Tool Validation	25
3.5.1 UML model	26
3.5.2 LQN model and solved LQN model.....	26

3.5.3	Anti-Patterns detection & refactoring.....	27
4.	DICE-FG Tool.....	29
4.1	EST-LE: a new maximum likelihood estimator for hostDemands	29
4.1.1	Validation	29
5.	Conclusions and future plans	31
5.1	Achievements.....	31
5.2	Summary of Progress at M30	31
	References.....	33
	APPENDIX A. Main Elements and Stereotypes of DICE UML Model Supported by DICE-APR.....	35
A.1	Deployment Diagram Model Elements.....	35
A.2	Activity Diagram Model Elements	35
	APPENDIX B. Core Functions for Anti-Patterns Detection & Refactoring	37
B.1	AP1: Infinite Wait	37
B.2	AP2: Excessive Calculation	37

List of Figures

1	Figure 1. Updated Architecture of Enhancement Tool.....	12
2	Figure 2. Transformation Steps	13
3	Figure 3. Model Transformation Process.....	13
4	Figure 4. Example of Deployment Diagram of the Source Model	14
5	Figure 5. Example of Activity Diagram of the Source Model	15
6	Figure 6. Example of LQN model	16
7	Figure 7. AP IW detection and refactoring process	21
8	Figure 8. AP EC detection and refactoring process	22
9	Figure 9. Example of APR configuration file.....	25
10	Figure 10. Activity Diagram of WordCount Example	26
11	Figure 11. Deployment Diagram of WordCount Example	26
12	Figure 12. Excerpt of LQN Model of WordCount Example	27
13	Figure 13. Excerpt of Solved LQN Model of WordCount Example	27
14	Figure 14. Anti-Patterns detection & refactoring results of WordCount Example	28
15	Figure 15. Validation results for the EST-LE.....	29

List of Tables

1	Table 1: Resource consumption breakdown Requirement	10
2	Table 2: Bottleneck Identification Requirement	10
3	Table 3: Semi-automated anti-pattern detection Requirement	10
4	Table 4: Enhancement tools data acquisition	11
5	Table 5: Enhancement tools model access Requirement	11
6	Table 6: Parameterization of simulation and optimization models Requirement.....	11
7	Table 7: Propagation of changes/automatic annotation of UML models Requirement	11
8	Table 8: Model Mapping: from UML+DICE+MARTE to LQN Element	17
9	Table 9: Anti-Patterns for DICE-APR.....	19
10	Table 10: Output parameters of analysis results supported by lqns and LINE	24
11	Table 11. Status of the Enhancement tool at M30. Data brought from Deliverable D4.5 [1].....	32

1. Introduction

The goal of DICE is to offer a novel UML profile and tools that will help software designers reasoning about quality of data-intensive applications, e.g., reliability, safety and efficiency. Furthermore, DICE develops a new methodology that covers quality assessment, architecture enhancement, continuous testing and agile delivery, relying on principles of the emerging DevOps paradigm. In particular, the goal of WP4 is to build tools and techniques to support the iterative improvement of quality characteristics in data-intensive applications obtained through feedback to the developers that will guide architectural design change.

This deliverable presents the final release of the DICE Enhancement tools (i.e., DICE Filling-the-Gap, DICE Anti-Patterns & Refactoring), which are being developed in task T4.3, to provide feedback to DICE developers on the application behaviour at runtime, leveraging the monitoring data from the DICE Monitoring Platform (DMon), in order to help them iteratively enhance the application design.

This deliverable describes the final version of the DICE Enhancement tool at M30 of the project. The initial version of Enhancement tool was reported in deliverable D4.5 [1] at M18. The DICE Enhancement tools include DICE Filling-the-Gap (DICE-FG), a tool focusing on statistical estimation of UML parameters used in simulation and optimization tool, and DICE-APR (Anti-Patterns & Refactoring), a tool for anti-patterns detection and refactoring.

Compared to the initial version of the Enhancement tool, released at M18, the final version has enhanced the functionality and interaction with the user of DICE-FG and developed the DICE-APR to provide the refactoring suggestions if anti-patterns (APs) are detected. Regarding the functionality enhancements and interaction, the DICE-FG has been extended to include a novel estimation algorithm for resource consumption [22].

DICE-APR transforms the UML model annotated with DICE profiles [2] to the Layered Queueing Networks (LQNs) [3] model for performance analysis, and the results will be used for APs detection and generating refactoring decisions. DICE Enhancement tool concerns the quality properties, e.g., performance, of the DIAs and offers possibilities to annotate and analyse the DIA models at DTSM and DDSM level. By using the Epsilon framework [4] and queueing theory [5], the new DICE-APR tool offers possibilities to analyse more quality properties of DIAs (e.g. response time distribution for reliability, utilization for performance, etc), detects if there is an AP (e.g. a server performs all of the work of an application) in the UML model and provides the corresponding advice to designer to refactoring the architecture. The implementation of M2M transformations, from UML to LQN, used by the DICE-APR is a tool called Tulsa which is mainly developed using Java and Epsilon languages (i.e., Epsilon Transformation Language (ETL) and Epsilon Object Language (EOL)). Solvers, e.g., LINE [6], lqns [7] can be used to solve the LQN model which is generated by Tulsa. The APs detection is implemented by using Matlab scripts. The Enhancement tool is developed as standalone tool. We also integrated Enhancement tool in the DICE IDE. It is published as an open source software that can be downloaded from the DICE-FG [8] and DICE-APR [9] GitHub repository.

1.1. What is new in Year 3

Updates in Y3 to the presented Enhancement tools were an outcome of a development process, which aimed at a) addressing any open or partially addressed requirements, and b) to improve general stability and usability of the tools.

1.1.1 DICE Anti-Patterns & Refactoring

Inferring the bad practices in software design (i.e., performance anti-patterns) according to the data, especially performance data, acquired at runtime during testing and operation. In order to achieve the above goal, DICE enhancement tools introduces a new methodology to close the gap between runtime performance measurements and design time model for the anti-patterns detection and refactoring.

New features and properties of the DICE-APR tool include the following:

- Developed Tulsa, a M2M transformation tool, to transform the design time model (i.e, UML model), which is annotated with runtime performance quality characteristics by DICE-FG tool, into performance model (i.e., Layered Queueing Network model); a series of transformation tasks can be specified in an Ant build file; a specific launch configuration can be invoked from the IDE run-configuration panel. The run-configuration in question invokes the APR back-end and performs the model-to-model transformation that parses the diagrams and returns a LQN model for performance anti-pattern detection.
- Specified the selected popular AP of DIAs and formally defined it by using Matlab scripts; implemented the AP detection algorithm and provided refactoring suggestions to the designer.
- Improved support of the existing general applications and support for Big Data technologies (e.g. Storm).

1.1.2 DICE Filling-the-Gap tool

- Integrated a novel estimation algorithm for *hostDemand*, called est-le, that outperforms several state-of-the-art algorithms.

1.2. Structure of the Document

The structure of this deliverable is as follows:

- Chapter 2 presents updates requirements of final version of the architecture of Enhancement tools.
- Chapter 3 presents new DICE-APR tool including M2M transformation, AP detection and tool usage, obtaining the DICE-APR tool and evaluation result.
- Chapter 4 presents updates of the initial version of the DICE-FG with respect to resource demand estimation.
- Chapter 5 summarizes achievements and outlines the future work.

Appendix A provides more detail on DICE UML model elements and stereotypes. Appendix B provides core functions for Anti-Patterns Detection and Refactoring.

2. Requirements

The deliverable D1.4 Companion [23], an updated version of the requirement specification of deliverable D1.2 [10], was released at M24. The requirements of the DICE Enhancement tools are basically the same as the previous version.

2.1 Requirements

This section reviews the requirements of the Enhancement tool. The “Must have” requirements of Enhancement tool are list as following. “Should have” and “could have” requirements are available in D1.4 Companion [23] released on the DICE Website¹.

Table 1: Resource consumption breakdown Requirement

ID	R4.11
Title	Resource consumption breakdown
Priority	Must have
Description	The DEVELOPER MUST be able to obtain via the ENHANCEMENT_TOOLS the resource consumption breakdown into its atomic components.

Table 2: Bottleneck Identification Requirement

ID	R4.12
Title	Bottleneck Identification
Priority	Must have
Description	The ENHANCEMENT_TOOLS MUST indicate which classes of requests represent bottlenecks for the application in a given deployment.

Table 3: Semi-automated anti-pattern detection Requirement

ID	R4.13
Title	Semi-automated anti-pattern detection
Priority	Must have
Description	The ENHANCEMENT_TOOLS MUST feature a semi-automated analysis to detect and notify the presence of anti-patterns in the application design.

¹ www.dice-h2020.eu/deliverables/

Table 4: Enhancement tools data acquisition

ID	R4.17
Title	Enhancement tools data acquisition
Priority	Must have
Description	The ENHANCEMENT_TOOLS must perform its operations by retrieving the relevant monitoring data from the MONITORING_TOOLS.

Table 5: Enhancement tools model access Requirement

ID	R4.18
Title	Enhancement tools model access
Priority	Must have
Description	The ENHANCEMENT_TOOLS MUST be able to access the DICE profile model associated to the considered version of the APPLICATION.

Table 6: Parameterization of simulation and optimization models Requirement

ID	R4.19
Title	Parameterization of simulation and optimization models.
Priority	Must have
Description	The ENHANCEMENT_TOOLS MUST extract or infer the input parameters needed by the SIMULATION_TOOLS and OPTIMIZATION_TOOLS to perform the quality analyses.

Table 7: Propagation of changes/automatic annotation of UML models Requirement

ID	R4.27
Title	Propagation of changes/automatic annotation of UML models
Priority	Must have
Description	ENHANCEMENT_TOOLS MUST be capable of automatically updating UML models with analysis results (new values)

3. DICE-APR Tool

In this section, we present the finalized DICE-APR tool. It is integrated with the DICE IDE as a popup menu. It also has a standalone version which includes two sub tools, M2M transformation (i.e., Tulsa) and Anti-Patterns Detection & Refactoring (APDR). Tulsa transforms the design time model (i.e., UML model) to performance model (i.e., LQN model). The LQN solver is needed to solve the LQN model when users use Tulsa as a standalone tool. The APDR, which is implemented in Matlab, invokes AP detection algorithm and provides the refactoring suggestions to designer. In the integration version, DICE-APR will invoke Tulsa and APDR in sequence. Figure 1 shows the updated architecture of Enhancement tool which we defined for task T3.4. It provides more details of functionalities of the DICE-APR.

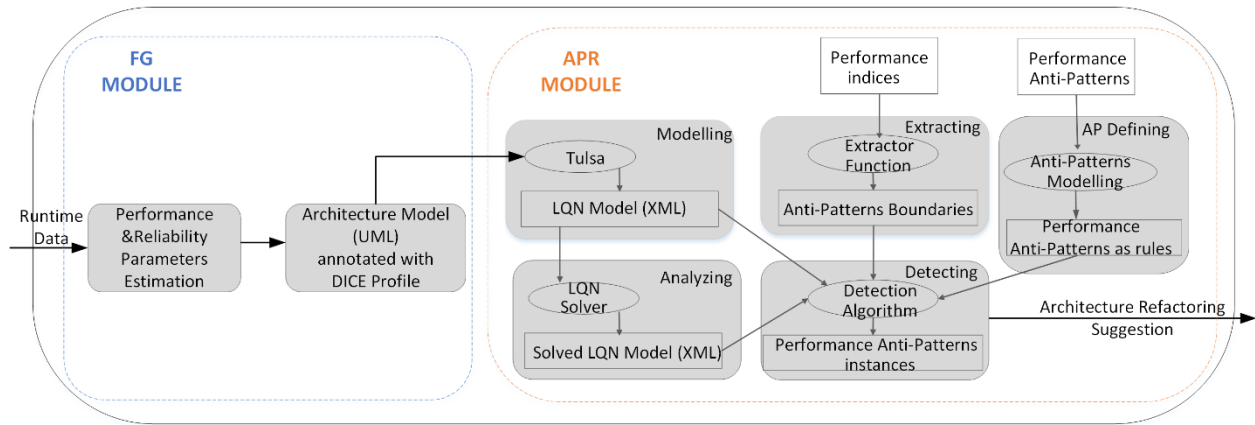


Figure 1. Updated Architecture of Enhancement Tool

The updated DICE-APR is described as follows:

- **Modelling:** this step is focused towards transforming the UML model to LQN model by using Tulsa. Tulsa will generate a XML format LQN model which follows the XML schema of LQN.
- **Analyzing:** the LQN solver (e.g., LINE, lqns) will be used to solve the XML format LQN model and to generate the analysis results, a XML format file as well.
- **Extracting:** extracting the pre-defined performance thresholds indices (e.g., maximum utilization) to set the anti-patterns boundaries.
- **AP Defining:** anti-pattern is defined as rules (i.e., trigger conditions) for AP detection.
- **Detecting:** LQN model, solved model, anti-patterns boundaries and anti-patterns rules will be used for detection algorithms to check if there is AP in the current model. The refactoring suggestions will be provided if AP is detected.

3.1 Model Transformation Module

DICE follows the model-centric perspective to capture different abstraction layers of Big Data applications. In order to support Big Data application modelling, the DICE profile introduces new stereotypes, tags and related constraints to specify the data location and data properties for DIAs. It leverages the UML as its modelling basis, and provides computational independent perspective, platform-independent perspective and platform-specific perspective via DICE Platform Independent Model (DPIM), DICE Technology Specific Model (DTSM) and DICE Deployment Specific Model (DDSM). DICE-APR transformation work mainly focuses on the DTSM and DDSM layer.

To fulfil one of tasks of DICE-APR, we develop a tool, Tulsa, for transforming software architecture models specified through UML into LQNs, which are analytical performance models used to capture contention across multiple software layers. In particular, we generalize an existing transformation based on the Epsilon framework to generate LQNs from UML models annotated with the DICE profile, which

extends UML to modelling DIAs based on technologies such as Apache Storm.

3.1.1 Transformation design

Our transformation follows the transformation principle of [3]. Tulsa takes four steps to implement the model transformation (see Fig 2).

- 1) **Step 1: Refining UML Model.** Identifying invocations within or among Partition(s) by assigning the *inPartition* attribute to *controlflow* in an activity diagram.
- 2) **Step 2: Generating LQN Model.** Performing transformation from UML model to an XML format LQN model (initial version).
- 3) **Step 3: Refining LQN Model.** Modifying the initial LQN model and make it confirm the LQN XML schema. The output of this step is a well formatted LQN model which can be accepted by LQN solver (e.g., lqns, LINE).
- 4) **Step 4: Generating Results.** Showing the results (e.g., utilization, Throughput) which are generated from the LQN solver.

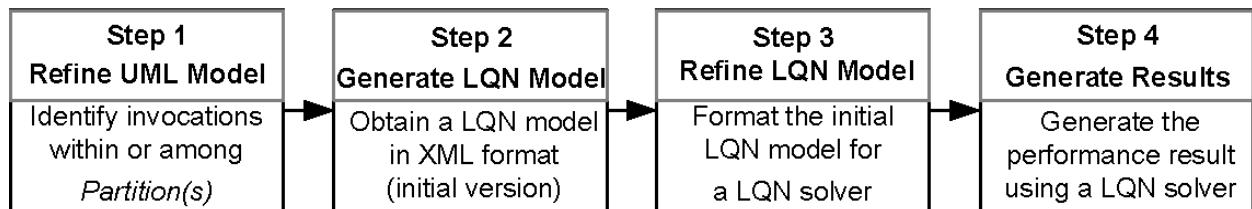


Figure 2. Transformation Steps

Figure. 3 shows the corresponding model transformation process. In figure 3, a UML model is annotated with the runtime parameters of the runtime systems which are obtained by DICE-FG. By using mapping rules, a UML model is then transformed to a LQN model. Then, lqns or LINE is leveraged to solve the LQN model and return the results to the DICE-APR tool. More details of lqns and LINE can be found in section 3.3.1.

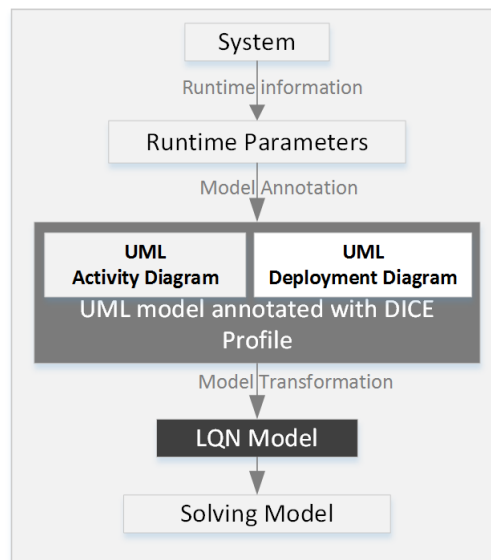


Figure 3. Model Transformation Process

In this section, we first presents the source and target models for Tulsa, which include the required diagrams, model elements and conditions for the transformation. Second, we describe the transformation

rules and operations. Then, we explain the transformation process of Tulsa.

3.1.2 Source model

Tulsa takes the UML model which follows the UML 2.5 standard extended with MARTE [12] and DICE profiles, which provide stereotypes to annotate the UML model for capturing the performance and reliability metrics, as input. In order to obtain a LQN model, the developer not only needs to design the configuration structure for the DIAs, e.g., developing the functional components, assigning key attributes and defining constraints, but also needs to capture the behavior of the design time model for the later analysis. Thus, our transformation mainly considers two UML diagrams: a deployment diagram, to represent the system structure, and an activity diagram, to describe the system behavior. The UML diagrams need to be annotated with core tags of the stereotypes of DTSM and DDSM layers to support performance and reliability analysis. The main elements and stereotypes of the DICE UML model are given in appendix A.

The following figure 4 and figure 5 show the examples of the UML activity diagram on DTSM level and UML deployment diagram on DDSM level respectively.

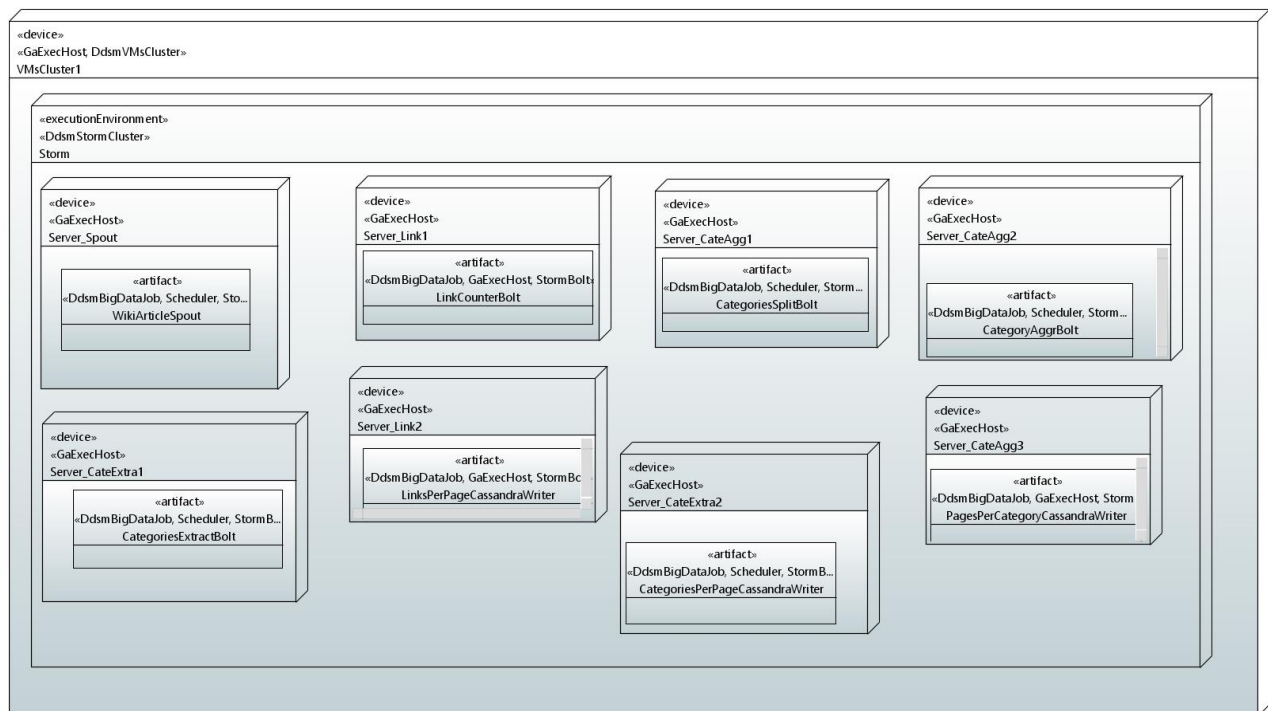


Figure 4. Example of Deployment Diagram of the Source Model

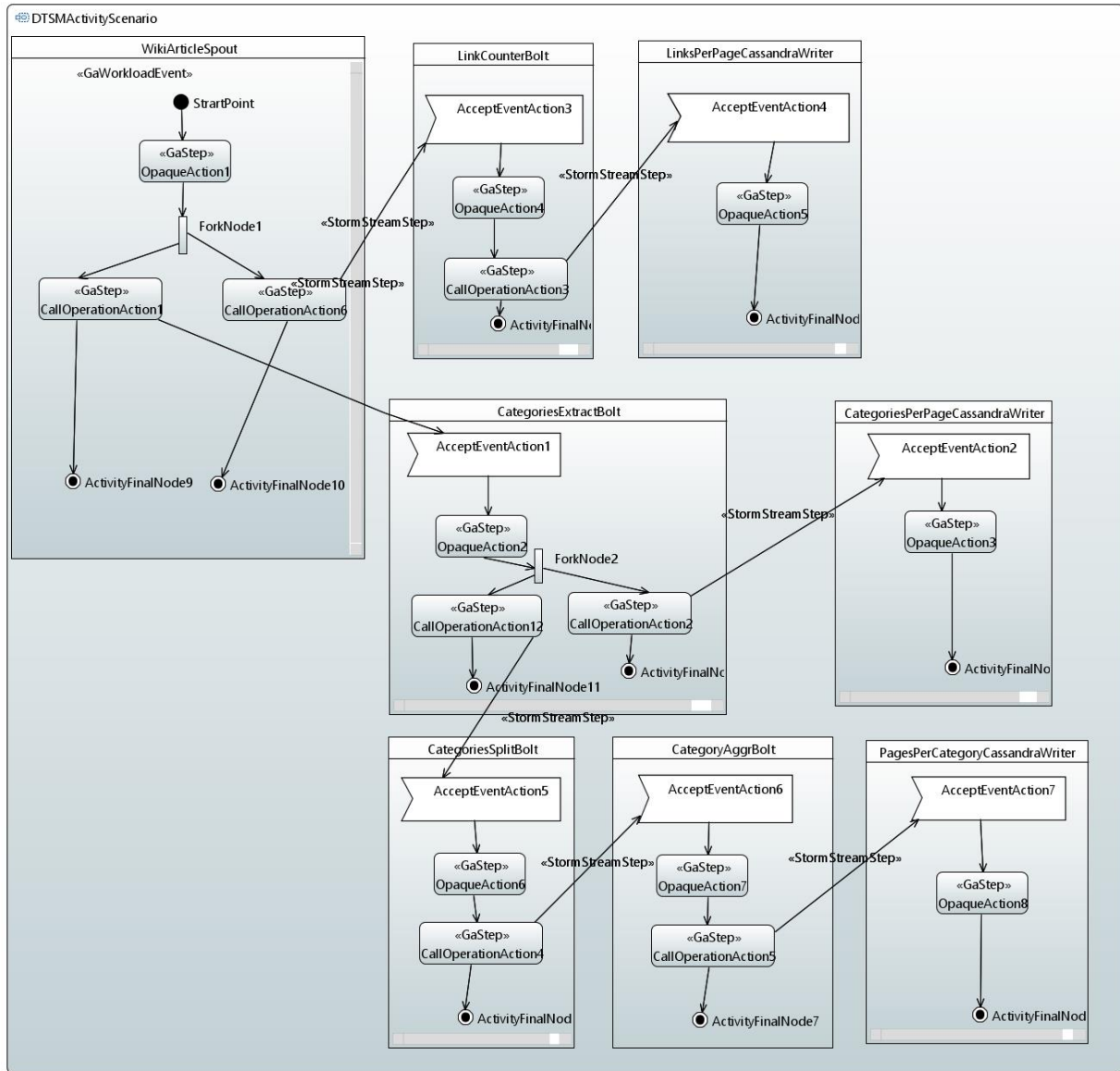


Figure 5. Example of Activity Diagram of the Source Model

3.1.3 Target model

Tulsa chooses the Layered Queueing Network as the target model, which is introduced in this section. There are three reasons for choosing LQNs. First, the core elements of LQN models are semantically similar to the corresponding elements of UML activity and deployment diagrams. Second, a Storm topology may be seen as a network of buffers and processing elements that exchange messages, so it is quite natural to map them into a queueing network model. Third, LQN solvers such as LINE or lqns are available to provide analytical methods to solve the LQN model.

A LQN model can be regarded as a directed graph. It consists of nodes and directed edges. The core model elements are processors, tasks, entries, activities and precedence [15].

Processors: Processors are used by the activities within a performance model to consume time. They model the physical servers that accept requests and execute the operations. They can be actual processors in the system, or may simply be placeholders for tasks representing customers and other logical resources. One of the key properties of the processor is the queueing disciplines, e.g., FIFO, PPR, HOL, PS, CFS, etc.

Tasks: Tasks are used in layered queueing networks to represent resources, e.g., software component, buffers, and hardware devices. Tasks are capable of serving requests and performing the actions defined by the entries. Two key properties of the task are scheduling policy (e.g., FIFO, ref) and multiplicity (i.e., the number of concurrent instances of the task).

Entries: Entries service requests are used to differentiate the service provided by a task. An entry can accept a synchronous call from a closed queueing model and an asynchronous call from the open queueing model.

Activities: Activities are the lowest-level of specification in the performance model. An activity represents the basic computation unit. To represent a non-sequential workflow (e.g., probabilistic choice, fork, join), activities can be organized via *precedence* to form a directed graph.

Precedence: It is used to connect activities within a task to form an activity graph. “Pre” and “Post” are two sub-elements of *Precedence*. The source activity is labelled with “Pre” and the target activity is labelled with “Post”.

```

118 <processor multiplicity="1" name="Server_CateAgg3" scheduling="fcfs">
119   <task activity-graph="YES" multiplicity="1" name="PagesPerCategoryCassandraWriter" scheduling="inf" think-time=
120     <entry name="AC7" type="NONE"/>
121     <service name="MyService"/>
122     <task-activities>
123       <activity host-demand-mean="0.0" name="OP8" bound-to-entry="AC7"/>
124       <activity host-demand-mean="0.0" name="OP88">
125         <synch-call dest="AC7Entry" calls-mean="1.0"/>
126       </activity>
127       <activity host-demand-mean="0.0" name="Stop9"/>
128       <precedence>
129         <pre>
130           <activity name="OP88"/>
131         </pre>
132         <post>
133           <activity name="Stop9"/>
134         </post>
135       </precedence>
136     </task-activities>
  
```

Figure 6. Example of LQN model

The figure 6 shows a screenshot of fragment of a LQN model. The root element is the processor *Server_CateAgg3* with the *fcfs* scheduling policy. It hosts an *inf* task *PagesPerCategoryCassandraWriter*. The task has one entry *AC7* and three activities, *OP8*, *OP88* and *Stop9*. The activity *OP88* also invokes synchronous call *AC7Entry*. The precedence specifies the *controlflow* direction.

3.1.4 Mapping rules

In this paragraph, we explain the mapping rules between the LQN model and the UML model.

1) Mapping from deployment diagram to LQN model

The elements concerned in this transformation are *Device* and *Artifact* of the deployment diagram. A *Device*, annotated with <<GaExecHost>> to represent a server, is mapped to *processor* in LQN model. For Storm-based applications, VM cluster is also represented as *Device* node and annotated with <<GaExecHost>> and <<DdsmVMsCluster>> stereotypes to specify the characteristics of a VM cluster. An *Artifact* can be transformed into a *Task* which stands for the software component in LQN model. To assign the scheduling policy and multiplicity to a *Task* in LQN domain, the stereotypes <<Scheduler>>, <<StormSpout>> and <<StormBolt>> can be added to the *Artifact*.

2) Mapping from Activity diagram to LQN model

The mapping from the Activity diagram to LQN model starts from the *InitialNode* which is the starting point of the activity diagram. It is transformed to an *entry* in LQN model. Stereotype

«GaWorkloadEvent» is applied to describe the workload. *AcceptEventAction*, accepting the event call (i.e., synchronous call or asynchronous call) from *CallOperationAction*, can also be mapped to an *entry* in LQN model. *ControlFlow*, representing the incoming and outgoing invocation paths, is mapped to either a *synch-call* or *asynch-call* in LQN model according to the type of the event call. If it is a synchronous event call, *SendSignalAction* is used to send feedback message to the caller. The *OpaqueAction* represents a specific action in activity diagram. The *CallOperationAction*, *SendSignalAction* and *OpaqueAction*, annotated with the «GaStep», are all mapped to *activity* in LQN model. *OpaqueAction* might also be annotated with Stereotype «StormSpout» and «StormBolt» due to different abstract view of the Storm-based application. Activity diagram leverages control nodes (i.e., *Decision Node*, *Merge Node*, *Join Node* and *Fork Node*) to represent parallelism and choice scenarios. Those nodes are transformed into *precedence* in LQN model.

More details of UML model elements and stereotypes can be found at Appendix A. The table 8 shows the model mapping from UML model to LQN model.

Table 8: Model Mapping: from UML+DICE+MARTE to LQN Element

UML Model Element	DICE + MARTE Stereotype	LQN Model Element
model	None	lqnmodel
Deployment Diagram		
Device	GaExecHost, DdsmVMsCluster	processor
Artifact	Scheduler, DdsmBigDataJob, StormSpout, StormBolt	task
Activity Diagram		
AcceptEventAction	GaStep	entry
InitialNode	GaWorkloadEvent	entry
OpaqueAction	GaStep, StormSpout, StormBolt	activity
CallOperationAction	GaStep	activity
SendSignalAction	GaStep	activity
DecisionNode	None	precedence
MergeNode	None	precedence
JoinNode	None	precedence
ForkNode	None	precedence
ControlFlow	None or StormStreamStep	precedence
ControlFlow	None or StormStreamStep	synch-call
ControlFlow	None or StormStreamStep	asynch-call

3.2 Anti-Patterns Detection & Refactoring

In this section, we review some popular anti-patterns. Then we report the APs we have chosen and specified their conditions. Finally, we discuss the approach we followed in order to detect the APs and provide refactoring suggestions.

3.2.1 Overview of popular Anti-Patterns

In software engineering, anti-patterns are recurrent problems identified by incorrect software decisions at different hierarchical levels (architecture, development, or project management). Software APs are largely studied in the industry. They are catalogued according to the source problem and a generic solution is suggested [18], [19], [20].

According to current formalizations, performance APs in software engineering are categorized in two families: single-value APs, that can be detected by inspecting the mean, max or min values of a performance index; and multiple-values APs, that require the observation of a performance index over the time [16], [21].

On the one hand, examples of popular single-value APs are: *Blob*, *Unbalanced Processing*, *Circuitous Treasure Hunt*, *Empty Semi Trucks*, *Tower of Babel*, *One-Lane Bridge* and *Excessive Dynamic Allocation*. On the other hand, examples of popular multiple-values: *Traffic Jam*, *The Ramp*, and *More is Less*.

More in detail, single-value APs are performance bottlenecks detected by a high utilization of a device, a low response time of the system, etc. For instance, performance APs are caused by an unbalanced utilization of the devices originated in a bad deployment of processes among the hardware resources (*Unbalanced Processing*); a low response time because of an excessive centralization of functionality in a single software package (*Blob*); or a high latency when retrieving data from central sources of information such as databases (*Circuitous Treasure Hunter*). An *Empty Semi Trucks* happens when two processes exchange lots of small messages that lead to a low utilization of the net. In a *Tower of Babel*, the exchange of information requires a constant transformation of formats between files. In a *One-Lane Bridge* APs, the level of concurrency of an application is drastically reduced when the workflow reaches a particular point. Finally, an *Excessive Dynamic Allocation* is detected when a process spends a considerable amount of time in the dynamic allocation/destruction of objects during runtime, while it could be treated more efficiently if the variables were reused or “compiled”.

In opposition, multiple-values APs are performance bottlenecks detected by a variation in the response time of the system during runtime. For instance, the effects of a *Traffic Jam* are a high variability in the response time in the transient behavior; the response time oscillates from low to high. *The Ramp* is revealed by an increasing response time and decreasing throughput over time. Changes in the state of the system (e.g., the amount of data that the system has to manage) affect the execution time of the internal tasks (e.g., searching operations in a database). Finally, *More is Less* happens when the system spends more time thrashing than executing real work.

3.2.2 Anti-Patterns selection and specification

In our case, we investigate three classic APs, *Circuitous Treasure Hunt*, *Blob* and *Extensive Processing* [16] and define two APs (i.e., Infinite Wait and Excessive Calculation). For integration version, APR supports the Excessive Calculation detection in DICE IDE. For standalone version, APR can detect both of the APs. Table 9 lists the corresponding APs we defined.

Table 9: Anti-Patterns for DICE-APR

Anti-Pattern Name	Problem	Solution
Infinite Wait (IW)	Occurs when a component must ask services from several servers to complete the task. If a large amount of time is required for each service, performance will suffer.	Report the component which causes the IW and provide component replication or redesign suggestions to the developer.
Excessive Calculation (EC)	Occurs when a processor performs all of the work of an application or holds all of the application's data. Manifestation results in excessive calculation that can degrade performance.	Report the processor which causes the EC and provide suggestion, adding new processor to migrate tasks, to the developer.

In order to formally specify the above APs (i.e., IW and EC), we interpret them crossing different modelling level, UML model, LQN model and Solved LQN model. We also give the corresponding AP condition.

1) Infinite Wait

UML Model: AP IW mainly concerns the invocations among the software components. A software component is represented as an *ActivityPartition* in the activity diagram. The service calls among the *ActivityPartitions* can be synchronous or asynchronous. In our case, DICE-APR only counts the number of the synchronous calls.

LQN Model: Each software component is regarded as a *task* of a *processor* in LQN model. The service calls (i.e., synchronous calls) among the tasks are specified by the *synch-calls* within *activities* in LQN model.

Solved LQN Model: To check if the software component, which has extensive synchronous calls, requires a large amount of time to fulfil the task, DICE-APR analyses solved LQN model, which is generated by a LQN solver and represented as an XML format file, to see if the response time is greater than the response time threshold.

Condition: given the AP boundaries, maximum number of synchronous calls (*ThMaxCall*) and maximum response time (*ThMaxResT*). The IW can be detected by checking the following conditions:

$$F_{\max \text{SynCall}}(\text{LQN}) \geq \text{ThMaxCall} \quad (1)$$

$$F_{\max \text{ResT}}(\text{Solved LQN}, \text{processor}) \geq \text{ThMaxResT} \quad (2)$$

Where $F_{\max \text{SynCall}}(\text{LQN})$ represents the function for checking if there is any component in LQN model has extensive synchronous calls which are greater than the threshold of synchronous calls, and the $F_{\max \text{ResT}}(\text{Solved LQN}, \text{processor})$ stands for the function for checking if the response time of the corresponding processor, which holds the component, is greater than the threshold of response time.

2) Excessive Calculation (EC)

UML Model: AP EC mainly concerns the server and the software components (or actions) which are deployed on it. A server is represented as a *Device* and a software component is represented as an *Artifact* in the deployment diagram.

LQN Model: Server is regarded as a *processor* in LQN model. The main services of each *Artifact* are represented as the *activities* of *entry* of the hardware *processor* in LQN model. In our case, DICE-APR calculates the number of the *entries* of the hardware processor to see if it is greater than the threshold.

Solved LQN Model: To check if the server, which has extensive *entries*, performs extensive calculation to fulfil the task, DICE-APR analyses solved LQN model, which is generated by a LQN solver and represented as an XML format file, to see if its utilization is greater than the threshold.

Condition: given the AP boundaries, maximum number of entries (*ThMaxEntry*) and maximum utilization (*ThMaxUtil*). The EC can be detected by performing the following conditions:

$$F_{maxEntry}(LQN) \geq ThMaxEntry \quad (1)$$

$$F_{maxUtil}(Solved LQN, processor) \geq ThMaxUtil \quad (2)$$

Where $F_{maxEntry}(LQN)$ represents the function for checking if there is any *processor* in LQN model has extensive entries which are greater than the threshold of entry, and the $F_{maxUtil}(Solved LQN, processor)$ stands for the function for checking if the utilization of the corresponding processor, which holds the entries, is greater than the threshold of utilization.

3.2.3 Anti-Patterns Detection & Refactoring process

Based on the above two APs, this section describes the AP detection and refactoring suggestions generation process.

1) AP IW detection and refactoring suggestion generation

Step1: Calculating the synchronous calls (*numSyn*) from the obtained the LQN model (*preLQN*) to see if there is any task that has the number of synchronous call greater than the threshold (*ThMaxCall*). If the task is found, then goes to Step 2 otherwise exits.

Step2: Calculating the response time of the entry of the found task (*entrRes*) from the obtained solved LQN model (*preSLQN*) to see if corresponding response time is greater than the threshold (*ThMaxResT*). If the answer is *yes*, then goes to Step 3 otherwise exits.

Step3: Reporting AP IW is found and providing two refactoring suggestions, that is redesigning the task (i.e., component) to reduce the synchronous calls or duplicating the task.

2) AP EC detection and refactoring suggestion generation

Step1: Calculating the number of entry (*numEnt*) of hardware processor from the obtained LQN model (*preLQN*) to see if there is any processor that has the number of entry is greater than the threshold (*ThMaxEntry*). If the processor is found, then goes to Step 2 otherwise exits.

Step2: Checking if the utilization (*procUtil*) of the found processor from the obtained the solved LQN model (*preSLQN*) is greater than the threshold (*ThMaxUtil*). If the answer is *yes*, then goes to Step 3 otherwise exits.

Step3: Reporting AP EC is found and providing refactoring suggestion, that is introduce new server to the current cluster and migrate part of components to the new server.

The Figures 7 and 8 show the detection of the two APs and the corresponding refactoring process of the DICE-APR.

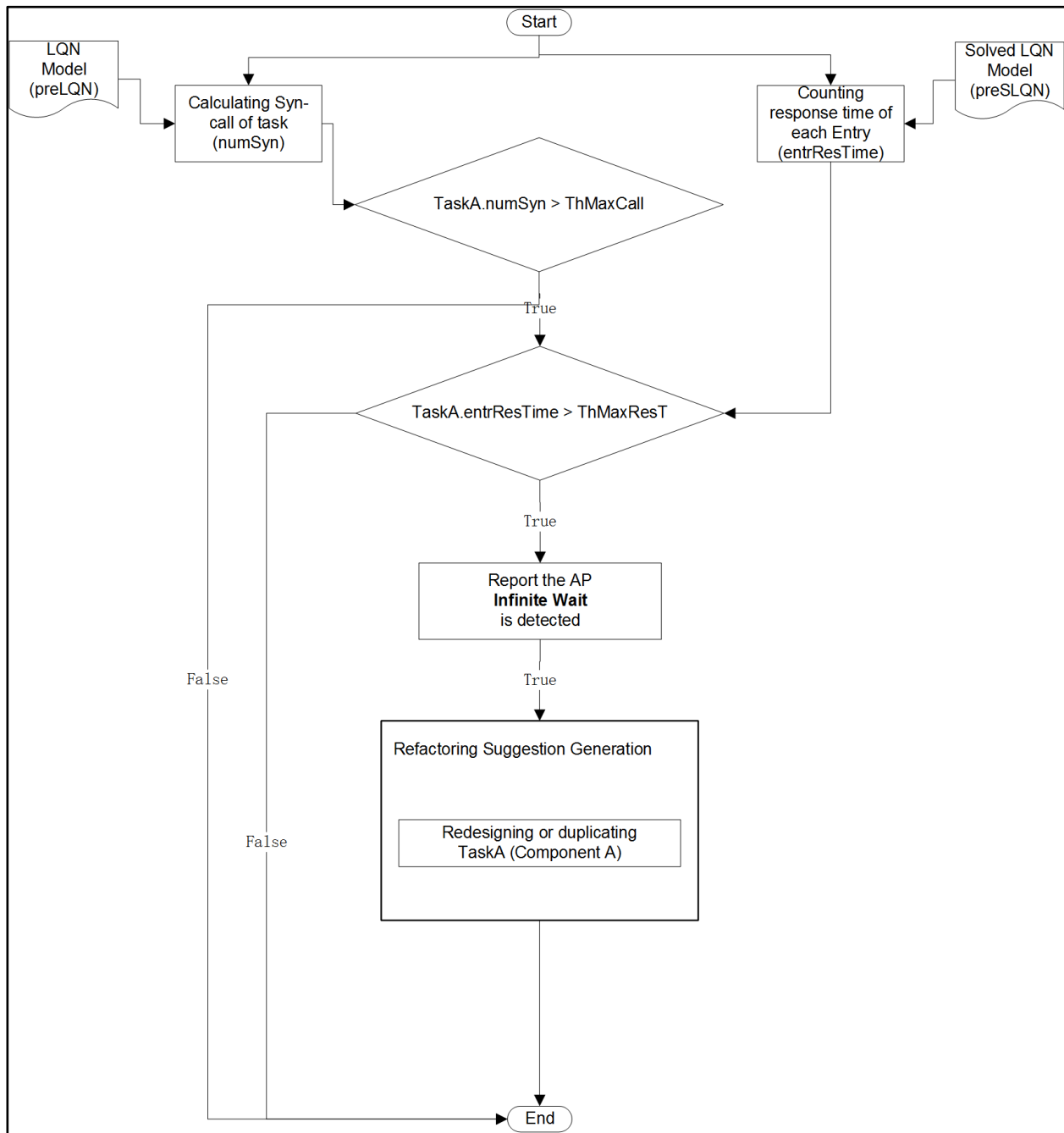


Figure 7. AP IW detection and refactoring process

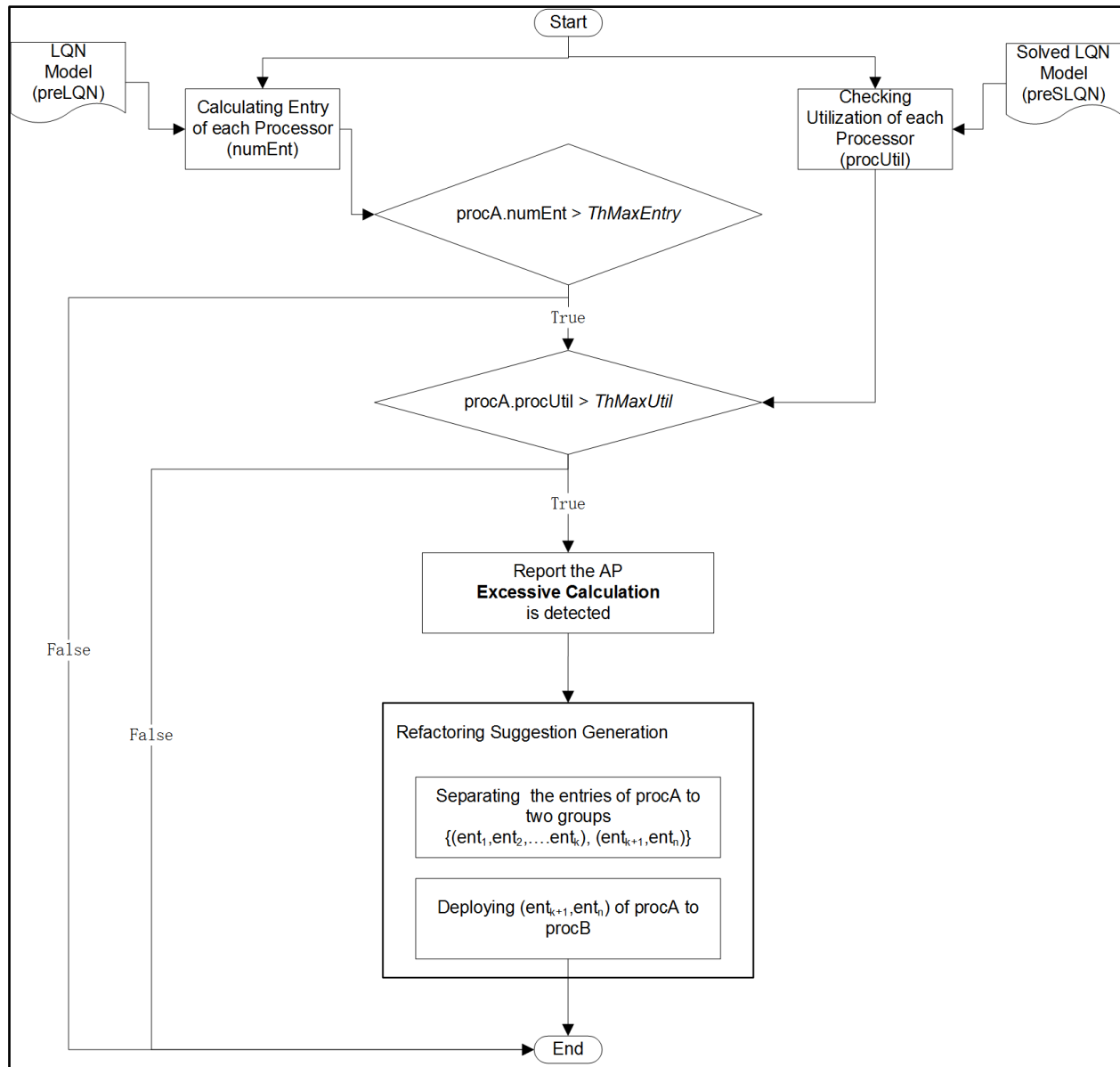


Figure 8. AP EC detection and refactoring process

3.3 Tool Usage

DICE Enhancement tools (i.e., DICE-FG and DICE-APR) were developed as standalone tools. By M30, we integrated DICE Enhancement tools in the DICE IDE as a plug-in. The details of how to use the DICE Enhancement tool within the DICE IDE can be found in Cheat Sheet or the GitHub page: <https://github.com/dice-project/DICE-Enhancement-APR/tree/master/Plugin/doc>. The follow paragraphs show how to configure and run DICE-APR as a standalone tool.

DICE-APR includes two sub tools, Tulsa and APDR. Tulsa is running inside the Eclipse IDE and can be invoked by using the run configuration panel. APDR runs in Matlab. User needs to interact with the Tulsa's ANT build file to configure the source model and target model, and configure the input parameters, LQN model, solved LQN model and APs Boundaries. In this section, we describe how to configure the DICE-APR, input data format for LQN Solver and APDR.

3.3.1 Tulsa Configuration

The pre-requirements of running the Tulsa are to install Eclipse 4.6.1, the DICE Profile and the Epsilon framework [17]. Tulsa is mainly written by Epsilon Transformation Language (ETL), Epsilon Object Language (EOL) and Java. Tulsa takes four steps to implement model transformation (see Figure. 2). Each step has the corresponding script. To run the Tulsa scripts, user has to either interact with configuration panel or build an ANT build file to:

- 1) specify the location of the source model (i.e., UML model) and the target model (i.e., LQN model);
- 2) build link with metamodels for UML, LQN and Trace (i.e., recording links between UML model elements and LQN model elements);
- 3) identify the location of scripts (*.eol and *.etl). The generated LQN model can be viewed as an XML format file.

To obtain the analysis results, we need to solve the LQN model with an existing solver, e.g., *lqns* and *LINE*. Here we present a brief description of the *lqns* and *LINE*.

1) *lqns*

LQNS is an analytic solver for LQN model developed by Carleton University. It accepts the XML format LQN model with the suffix *.lqnx* as input. *LQNS* can be executed with the command line: *lqns Lqnfile.lqnx* and produces two output files with the default name *lqnfile.out* and *lqnfile.lqxo*. The *lqnfile.out* is the summary of the analysis results, e.g., processor identifiers and scheduling algorithms, throughputs and utilizations per phase. The *lqnfile.lqxo* is a parameterized XML format with analysis results. It can be viewed directly or parsed by executing the EOL script to get the summary of the results in console. The LQN model generated by Tulsa can be used as input file of *lqns* directly. However, *lqns* also has some limits of parameter types, e.g., only supporting three scheduling policies, FIFO, HOL and PPR, only accepting phases not greater than 3.

lqns is available from the download page² for Linux and Windows. It has a comprehensive main page describing many options for different solver algorithms, for tracing solutions and for printing more or less details.

2) *LINE*

LINE is an efficient parallel solver for LQN models developed by Imperial College London. It can be integrated with the Palladio Bench suite used for performance analysis of Palladio Component Models (PCM). *LINE* can numerically compute percentiles of response times for service-level agreement (SLA) assessment, describe uncertainty about an operational environment using random environments, and solve models with a parallel solver for multi-core. The LQN model generated by Tulsa can be easily modified to fit *LINE* requirements. *LINE* can be executed in MATLAB environment. It will produce an output file with the default name *lqnfile_line.xml*. Different from *LQNS*, there are two types of the processors in LQN model, software processor and hardware processor. Software processor represents the virtual processor which host the task and hardware processor represents the real computation resource (e.g., server). Comparing with *LQNS*, *LINE* does not has limits of the phases but it does not fully support the parallelism situation (i.e., pre-and, post-and).

LINE is available from [6] and user can also find the documents and installation instructions from it.

The Table 10 shows some core output parameters of the analysis results of the solved LQN model supported by *LINE* and *lqns*.

² <http://www.sce.carleton.ca/rads/lqns/>

Table 10: Output parameters of analysis results supported by Iqns and LINE

Output LQN Model Solved by	Parameter Name	Description
Iqns	utilization (processor)	processor utilization for every entry and activity running on the processor
	utilization (task)	reciprocal of the service time for the task
	service-time	total time a phase or activity uses processing a request
	throughput	the rate at which an entry (or an activity) is executed
LINE	util	processor utilization for every entry and activity running on the processor
	throughput	the rate at which an entry (or an activity) is executed
	responseTime	response time (elapsed time since a user submits a job to the cluster and return of the result)
	responseTimeDistribution	distribution of the response times, the distribution includes information such as the percentiles, i.e., the longest response time x faced by p% of the users is called the p-th percentile.

3.3.2 Running APR

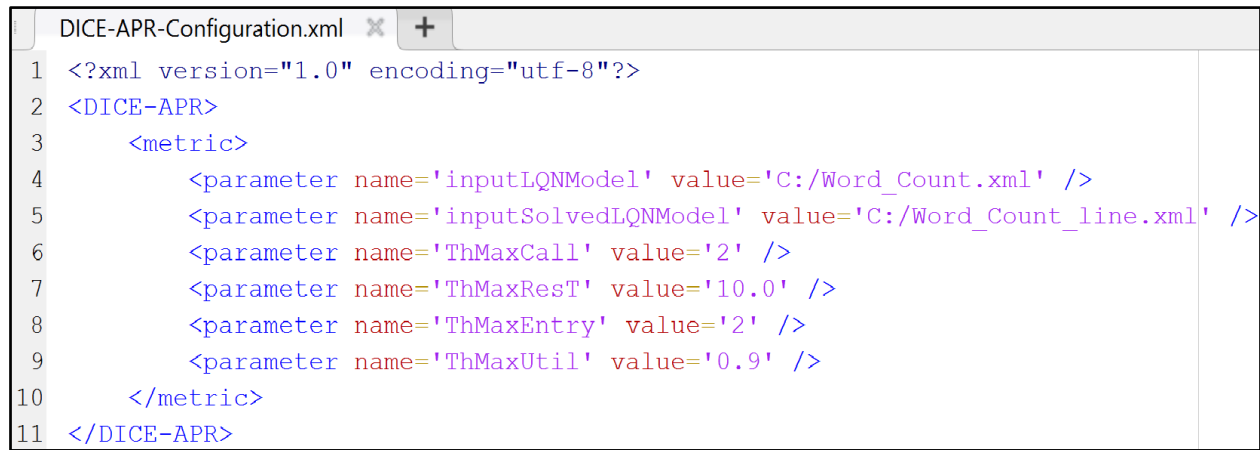
The prerequisites of running the APR is installing the MATLAB. End-users can either install MATLAB (2012a or later) with a valid license or Matlab Compiler Runtime (MCR) R2015a which is a royalty-free runtime that does not require owning a Matlab license. Before we run the APR tool, we need to specify the input data format and the configuration file provided for anti-patterns detection and refactoring. These are the only inputs required to run the tool. It may be invoked from the command line, e.g., the following is an example of running APR on Windows Operating Systems:

```
>diceAPR DICE-APR-Configuration.xml
```

where *diceAPR* is a standalone executable file generated from APR source code in Matlab, and the *DICE-APR-Configuration.xml* is a configuration file which will be explained in the next section.

3.3.3 APR configuration file

In this paragraph, we describe the specification of the input data that is requested to the user in order to use DICE-APR. Like DICE-FG, we also use an XML file to specify the input parameters for DICE-APR. The following figure is an example of *DICE-APR-Configuration.xml* for standalone version. For the integration version, you can find the sample configuration files at GitHub page: <https://github.com/dice-project/DICE-Enhancement-APR/tree/master/Plugin/doc/Configuration%20Files>



```

1 <?xml version="1.0" encoding="utf-8"?>
2 <DICE-APR>
3   <metric>
4     <parameter name='inputLQNModel' value='C:/Word_Count.xml' />
5     <parameter name='inputSolvedLQNModel' value='C:/Word_Count_line.xml' />
6     <parameter name='ThMaxCall' value='2' />
7     <parameter name='ThMaxResT' value='10.0' />
8     <parameter name='ThMaxEntry' value='2' />
9     <parameter name='ThMaxUtil' value='0.9' />
10  </metric>
11 </DICE-APR>

```

Figure 9. Example of APR configuration file

The above configuration file specifies the input parameters of the DICE-APR:

- **inputLQNModel:** the value of the parameter *inputLQNModel* represents the location of the LQN model file.
- **inputSolvedLQNModel:** the value of the parameter *inputSolvedLQNModel* stands for the location of the solved LQN model file.
- **ThMaxCall:** the value of the parameter *ThMaxCall* means the maximum number of the synchronous calls.
- **ThMaxResT:** the value of the parameter *ThMaxResT* defines the acceptable maximum response time.
- **ThMaxEntry:** the value of the parameter *ThMaxEntry* specifies the maximum number of entries of a processor.
- **ThMaxUtil:** the value of the parameter *ThMaxUtil* means the allowed maximum utilization of a processor.

Detailed installation and running instructions of standalone version of Tulsa are available on the DICE-Enhancement-APR wiki at <https://github.com/dice-project/DICE-Enhancement-APR/wiki>.

3.4 Obtaining the Tool

The source code of standalone version of DICE-APR Tool is available at the following GitHub page:

- <https://github.com/dice-project/DICE-Enhancement-APR>

The source code of plug-in version of DICE Enhancement Tool is available at the following GitHub page:

- <https://github.com/dice-project/DICE-Enhancement-APR/tree/master/Plugin>

3.5 Tool Validation

Word Count³ is a well-known Storm-based application. We modify the original version of the Word Count and design a chain-like topology with one spout and two bolts. The spout *SentenceSpout* reads a file one line at a time, and sends each line as a tuple to the bolt *SplitBolt* which splits the sentence to single words. The bolt *WordCountBolt* receives the tuple (i.e., word) from the *SentenceSpout* and increments counters based on distinct input word tuples. We use this as an example to demonstrate the how DICE-APR perform the model transformation, anti-patterns detection and refactoring.

³ <http://storm.apache.org/releases/1.1.0/Tutorial.html>

3.5.1 UML model

Figures 10 and 11 shows, respectively, the deployment and activity diagrams of the WordCount Example.

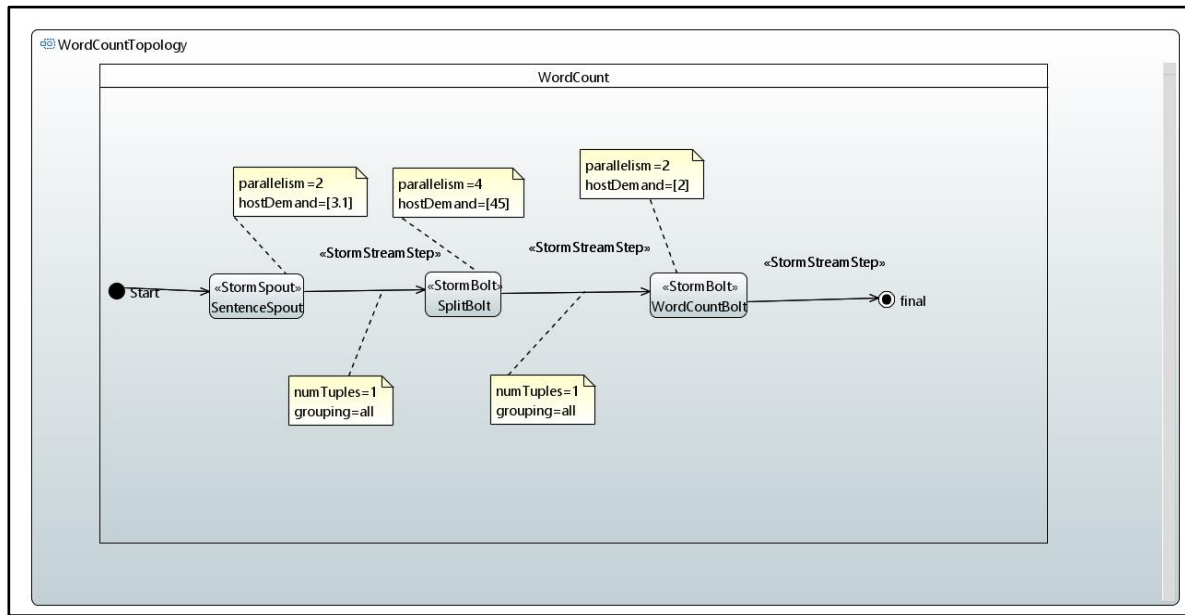


Figure 10. Activity Diagram of WordCount Example

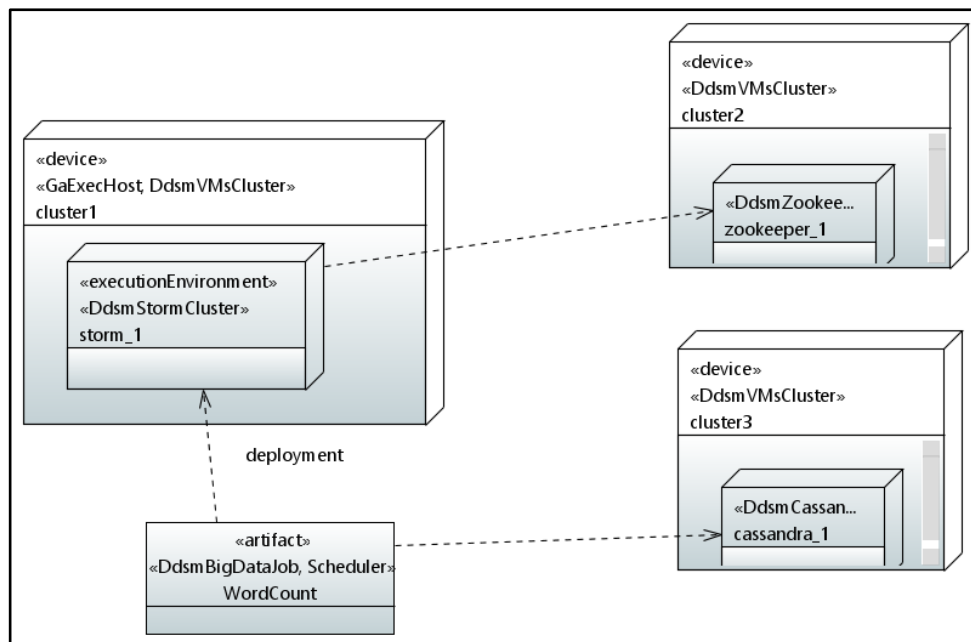


Figure 11. Deployment Diagram of WordCount Example

In DICE UML model, the application can be regarded as an artifact in deployment diagram and the components are represented as OpaqueAction in activity diagram. Thus, the *WordCount* application is designed as an Aritfact which is deployed on platform *storm_1*. It is held in cluster *cluster1* in deployment diagram. The *SentenceSpout*, *SplitBolt* and *WordCountBolt* are represented as three OpaqueActions in activity diagram. DICE and MARTE stereotypes, e.g., *GaExecHost*, *StormSpout*, are used to annotate the hardware resources and behaviours.

3.5.2 LQN model and solved LQN model

By using Tulsa, the corresponding LQN model can be generated automatically. The obtained LQN model

can be solved directly by solver lqns. The format of the LQN model for solver LINE is slightly different from LQNS, but it can be easily modified to meet the LINE requirement. Figure 12 and 13 show an excerpt of the LQN model and solved LQN model, respectively, of WordCount example supported by LINE.

```

1 <?xml version="1.0" encoding="us-ascii"?>
2 <lqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/u
3 <processor multiplicity="1" name="Cluster" quantum="0.001" scheduling="ps" speed-factor="1.0">
4   <task activity-graph="YES" multiplicity="1" name="CPU_Task" scheduling="inf" think-time="0.0">
5     <entry name="SentenceSpoutEntry" type="PH1PH2">
6       <entry-phase-activities>
7         <activity host-demand-mean="3.1" name="SentenceSpoutActivity" phase="1"/>
8       </entry-phase-activities>
9     </entry>
10    <entry name="SplitBoltEntry" type="PH1PH2">
11      <entry-phase-activities>
12        <activity host-demand-mean="45.0" name="SplitBoltActivity" phase="1"/>
13      </entry-phase-activities>
14    </entry>
15    <entry name="WordCountBoltEntry" type="PH1PH2">
16      <entry-phase-activities>
17        <activity host-demand-mean="2.0" name="WordCountBoltActivity" phase="1"/>
18      </entry-phase-activities>
19    </entry>
20    <service name="MyService"/>
21  </task>
22 </processor>

```

Figure 12. Excerpt of LQN Model of WordCount Example

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <cqn-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CQNmodel">
3   <processor name="Cluster" util="9.960991344468124e-01"/>
4   <processor name="storm_1" util="3.976443650486277e-03"/>
5   <workload name="storm_1" responseTime="5.009999999999999e+01" throughput="1.988221825243139e-02">
6     <station name="Cluster" responseTime="5.009999999999999e+01"/>
7     <station name="storm_1" responseTime="2.000000000000000e-01"/>
8     <responseTimeDistribution>
9       <percentile level="1.000000000000000e-01" value="9.559206432874753e+00"/>
10      <percentile level="2.000000000000000e-01" value="1.521508447259754e+01"/>
11      <percentile level="3.000000000000000e-01" value="2.129457031168220e+01"/>
12      <percentile level="4.000000000000000e-01" value="2.824329160236550e+01"/>
13      <percentile level="5.000000000000000e-01" value="3.645022047473938e+01"/>
14      <percentile level="6.000000000000000e-01" value="4.649356650829058e+01"/>
15      <percentile level="7.000000000000000e-01" value="5.944799001274072e+01"/>
16      <percentile level="8.000000000000000e-01" value="7.770010876514547e+01"/>
17      <percentile level="9.000000000000000e-01" value="1.089054048281287e+02"/>
18    </responseTimeDistribution>
19  </workload>
20  <Entry name="SplitBolt" responseTime="4.699999999999999e+01" throughput="1.988221825243139e-02">

```

Figure 13. Excerpt of Solved LQN Model of WordCount Example

3.5.3 Anti-Patterns detection & refactoring

Figure 14 shows a screenshot of the Anti-Patterns detection & refactoring results of WordCount example.

```

-----
APR module - 1.0.0 - copyright 2012-2017 (c) - Imperial College London.
-----
Anti-Pattern Detection Start.....
=====
AP1: Infinite Wait.
AP1 (IW): Syncall Call Checking - file: C:\Users\lcroy\Documents\MATLAB\Word_Count.xml
Number of Syn-calls is smaller than thresholds.
-----
AP1 (IW) is not found in LQN model.
-----
AP2: Excessive Calculation.
AP2 (EC): Entry Checking - file: C:\Users\lcroy\Documents\MATLAB\Word_Count.xml
The following processor has too many entries.
processors: Cluster; number of components: 3
AP2 (EC): Utilization Checking - file: C:\Users\lcroy\Documents\MATLAB\Word_Count_line.xml
-----
AP2 (EC) is found in LQN model.
-----
Utilization of the following processor is higher than thresholds.
processors: Cluster, utilization: 0.996099134446812
-----
Refactoring suggestions for AP2 (EC)
-----
Migrating 2 components from Cluster to a new server.
=====
fx Anti-Pattern Detection End.....

```

Figure 14. Anti-Patterns detection & refactoring results of WordCount Example

The results show that the current application does not have AP Infinite Wait. However, utilization of the cluster which holds all components is greater than the utilization threshold, the AP Excessive Calculation is detected and the corresponding refactoring suggestion is provided. The developer might introduce a new server and migrate some components to the new server.

4. DICE-FG Tool

Since the earlier release at M18, the DICE-FG codebase has remained relatively stable, undergoing minor bug fixes and an extension to a novel demand estimation method published in [22]. In the following, we provide an overview of the new introduced method.

We also performed a validation of DICE-FG against Apache Spark data generated in the Fraud detection case study by Netfective. The results of this activity are reported in deliverable *D6.3 - Consolidated evaluation and implementation*.

4.1 EST-LE: a new maximum likelihood estimator for hostDemands

In year 3 we developed a novel estimator for *hostDemands*, which is able to efficiently account for all the state data monitored for a Big data system. Recall that the *hostDemands* of a Big data application may be seen as the time that a request of type c spends at resource k . For example, the execution time of a Cassandra query of type c at node r of a Cassandra cluster.

A new demand estimation method called **est-le** (logistic expansion) has been included in the DICE-FG distribution. Compared to the earlier version of DICE-FG, this method enables to use a probabilistic maximum-likelihood estimator for obtaining the *hostDemands*. Such approach is more expressive than the previous **est-qmle** method in that it includes information about the response time of the requests, in addition to the state samples obtained through monitoring. An obstacle that was overcome in order to offer this method is that the resulting maximum-likelihood method is computationally difficult to deal with, resulting in very slow execution times for the computation of the likelihood function. In [22] we developed an asymptotic approximation that allows to efficiently compute the likelihood even in complex models with several resources, requests types, and high parallelism level.

4.1.1 Validation

To illustrate the difficulties of existing methods for the computation of the likelihood function, we developed in [22] a systematic comparison against alternative methods for *hostDemand* estimation that use the same input data, which is illustrated in the Figure 15. The figure has been obtained by running the following methods against a set of 1000 random *hostDemand* estimation problems, where:

- CUB1 is a method for computing likelihood based on cubature rules.
- LE corresponds to the est-le method we added to DICE-FG.
- LE-A is a variant of the LE method that allows to include known values of mean performance indices.
- MCI3 is a method for computing likelihood based on Monte Carlo sampling.
- NOG is a crude approximation that removes from the likelihood expression the term that is most computationally demanding to compute, i.e., the normalizing constant.
- RAY is an asymptotic approximation based on a method used in optics.

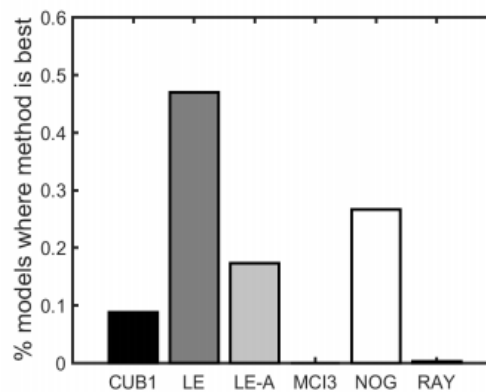


Figure 15. Validation results for the EST-LE

As we see from the figure, the LE method and its variant LE-A are in about 60% of the cases the best *hostDemand* estimation method overall. Their execution time is considerably faster than that of other methods such as CUB1 and MCI3 and generally in the order of a few seconds. Storage requirements are negligible.

5. Conclusions and future plans

DICE Enhancement Tools are an effort of 3-years development aimed at closing the gap between the design time model and runtime model. DICE Enhancement tools bring the Model-Driven Development methodology to DevOps to achieve iterative quality enhancement of DIAs. DICE Enhancement tools are integrated with the DICE IDE and also has standalone versions. DICE-FG aims at estimating and fitting application parameter related to memory and execution times and annotate DICE UML models. DICE-APR enables the model transformation which help to predict the performance of the DIAs, and provides the refactoring suggestions if the APs are detected. DICE Enhancement tools leverage the third part plug and tools, e.g., Epsilon framework, LINE, lqns, MCR, which are easily obtained and installed for end-users. We have evaluated the scalability and performance of the DICE Enhancement tool via general distributed system and Storm-based applications (e.g., WordCount).

5.1 Achievements

In conclusion of this deliverable we summarize the key achievements of this final release of the Enhancement Tools:

- DICE-APR has been developed to achieve the anti-patterns detection and refactoring:
 - Transform the UML model annotated with DICE profile to LQN model.
 - Define and specify two APs and the corresponding AP boundaries for DIAs.
 - Detect the above APs from the models and provide the refactoring suggestions to guide the developer to update the architecture.
- DICE-FG has been consolidated and extended to include a new *hostDemand* estimation method called *est-ld*, that is more powerful than existing *hostDemand* methods released at M18.

The DICE Enhancement tool is available online on DICE's Github repository. The following are the main links:

- Standalone version:
 - DICE-FG Source Code: <https://github.com/dice-project/DICE-Enhancement-FG>
 - DICE-FG Documentation: <https://github.com/dice-project/DICE-Enhancement-FG/wiki>
 - DICE-APR Source Code: <https://github.com/dice-project/DICE-Enhancement-APR>
 - DICE-APR Documentation: <https://github.com/dice-project/DICE-Enhancement-APR/wiki>
- Plug-in version:
 - DICE Enhancement Tool Source Code: <https://github.com/dice-project/DICE-Enhancement-APR/tree/master/Plugin>
 - DICE Enhancement Tool Document: <https://github.com/dice-project/DICE-Enhancement-APR/tree/master/Plugin/doc>

5.2 Summary of Progress at M30

The following table summarizes the status of requirements implementation at the end of reporting period (M30). The meaning of the labels used in column Level of fulfillment is the following: (i) ✗ (not implemented); (ii) ✓ (partial accomplishment); and (iii) ✓ (implemented at M30).

Table 11. Status of the Enhancement tool at M30. Data brought from Deliverable D4.5 [1]

Requirement	Status at M30
R4.11: Resource consumption breakdown	✓: the DICE-FG module is capable of extracting resource consumption data (memory, CPU time) for individual tasks at arbitrary nodes. The estimated data breaks down the usage of individual resources through job types that visit the resource.
R4.12: Bottleneck identification	✓: by estimating the true execution times of requests, sanitized from contention overheads, the DICE-FG makes it trivial to identify bottlenecks. That is, the node with the largest mean execution time will be the bottleneck resource for a job type. Such feature is going to be completed by adding bottleneck identification capabilities in the APR module.
R4.13: Semi-automated anti-pattern detection	✓: An initial architecture and high-level approach defined, and initial proof-of-concept defined.
R4.17: Enhancement tools data acquisition	✓: We have interfaced DICE-FG module with the DMon platform. APR module will not need direct access to the DMon. More metrics will be accessed in the feature to extend the breadth of the automatic UML parameterization.
R4.18: Enhancement tools model access	✓: This feature is an integration feature to be developed in the next period. Currently integration is operated manually, in the future it will be automated.
R4.19: Parameterization of simulation and optimization models	✓: We have conducted validation studies on Hadoop/MapReduce (c.f. D3.8, Section 6), Cassandra, and SAP HANA that illustrate the ability of the DICE-FG module to provide good estimates of parameters.
R4.27: Propagation of changes/automatic annotation of UML models	✓: DICE-FG can successfully modify UML models by annotating parameters. The APR module is planned to introduce changes in the UML models.

References

- [1] DICE Consortium, Iterative quality enhancement tools – Initial version (Deliverable 4.5), 2016, <http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/08/D4.5-Iterative-quality-enhancement-tools-Initial-version.pdf>
- [2] DICE Consortium, Design and quality abstractions - Initial version (Deliverable 2.1), 2016, http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/02/D2.1_Design-and-quality-abstractions-Initial-version.pdf
- [3] Altamimi, T., Zargari, M.H., Petriu, D., Performance analysis roundtrip: automatic generation of performance models and results feedback using cross-model trace links, In: CASCON'16, Toronto, Canada, ACM Press (2016)
- [4] D. Kolovos, L. Rose, A. García-Domínguez, R. Paige, The Epsilon Book, www.eclipse.org/epsilon/doc/book/, last updated July 2015.
- [5] Dubois, D.J., et al., Model-driven application refactoring to minimize deployment costs in preemptible cloud resources, In: CLOUD'16, IEEE Press, USA (2016)
- [6] LINE, <http://line-solver.sourceforge.net/>
- [7] LQNS, <https://github.com/layeredqueuing/V5/tree/master/lqns>
- [8] DICE-FG, <https://github.com/dice-project/DICE-Enhancement-FG/>
- [9] DICE-APR, <https://github.com/dice-project/DICE-Enhancement-APR>
- [10] DICE Consortium, Requirements specifications (Deliverable 1.2), 2015, http://wp.doc.ic.ac.uk/diceh2020/wp-content/uploads/sites/75/2015/08/D1.2_Requirement-specification.pdf
- [11] DICE Consortium, Architecture definition and integration plan - Final version (DICE deliverable 1.4), January 2017, http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/02/D1.4_Architecture-definition-and-integration-plan-Final-version.pdf
- [12] UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.1, Object Management Group (2011).
- [13] DICE Consortium, Design and Quality Abstractions - Final Version (Deliverable 2.2), 2017, http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/02/D2.2_Design-and-quality-abstractions-Final-version.pdf
- [14] DICE Consortium, Transformations to Analysis Models (DICE deliverable 3.1), 2016, http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2016/08/D3.1_Transformations-to-analysis-models.pdf
- [15] Franks, R.G., Maly, P., Woodside, C.M., Petriu, D.C., Hubbard, A., Mroz, M., "Layered Queueing Network Solver and Simulator User Manual", Department of Systems and Computer Engineering, Carleton University, 2015.
- [16] Cortellessa, Vittorio, Antinisca Di Marco, and Catia Trubiani. "An approach for modeling and detecting software performance antipatterns based on first-order logics." Software and Systems Modeling (2014): 1-42.
- [17] Epsilon, <http://www.eclipse.org/epsilon/>

- [18] Smith, C. U. and Williams, L. G. (2000). Software performance antipatterns. In Workshop on Software and Performance, volume 17, pages 127-136.
- [19] Smith, C. U. and Williams, L. G. (2002). New software performance antipatterns: More ways to shoot yourself in the foot. In Int. CMG Conference, pages 667-674.
- [20] Smith, C. U. and Williams, L. G. (2003). More new software performance antipatterns: Even more ways to shoot yourself in the foot. In Computer Measurement Group Conference, pages 717-725.
- [21] C Trubiani, A Koziolk, V Cortellessa, R Reussner. Guilt-based handling of software performance antipatterns in palladio architectural models, Journal of Systems and Software 95, 141-165, 2014.
- [22] G. Casale. Accelerating performance inference over closed systems by asymptotic methods. ACM SIGMETRICS 2017, 25 pages.
- [23] DICE Consortium, Architecture definition and integration plan - Final version (DICE deliverable 1.4 Companion), January 2017, http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/02/D1.4_Companion-2_requirements.pdf

APPENDIX A. Main Elements and Stereotypes of DICE UML Model Supported by DICE-APR

The core elements of the UML model (i.e., activity and deployment diagrams) and the corresponding stereotypes which related to the Tulsa transformation are described as follows. More details can be found in D2.2[13] and D3.1[14].

A.1 Deployment Diagram Model Elements

Device: A node annotated with the stereotype *Device* usually represents a physical computational resource, e.g., server, processor. DICE UML model uses *Device* to stand for a VM cluster or a single server.

ExecutionEnvironment: A node annotated with the stereotype *ExecutionEnvironment*, is used to represent the execution environment for the application, i.e., the platform where the application is deployed. In DICE UML model, *ExecutionEnvironment* is embedded in the *Device* node (i.e., VM Cluster) to provide running environment for the DIAs.

Artifact: A node annotated with the stereotype *Artifact* is used or produced by a software development process or deployment and operation of a system, e.g., software component. DICE UML model uses *Artifact* to stand for a software application or component which is deployed on the *Device* node or *ExecutionEnvironment* node.

«GaExecHost»: This stereotype is defined by MARTE. It provides core tags for specifying characteristics of the execution host, e.g., a server. In DICE deployment diagram, *GaExecHost* is used to annotate the UML node with *Device* stereotype. Two tags of the *GaExecHost*, *schedPolicy* and *resMult*, are used to describe the host's scheduling policy (e.g. FIFO, RoundRobin) and the number of parallel processors available.

«DdsmVMsCluster»: This stereotype is defined for DICE DDSM layer. It provides core tags for specifying characteristics of the VM clusters. In DICE deployment diagram, *DdsmVMsCluster* is used to annotate the UML node with *Device* stereotype. One tag of the *DdsmVMsCluster*, *instances*, is used by Tulsa to obtain the number of single server in the VM cluster.

«Scheduler»: This stereotype is defined by MARTE. It provides core tags for specifying characteristics of resources which have a scheduling policy. In DICE deployment diagram, *Scheduler* is used to annotate the UML node with *Artifact* stereotype. Two tags of the *Scheduler*, *otherSchedPolicy* and *resMult*, are used to describe the tasks scheduling policy (e.g. ref) and the number of concurrent instances of the task available at runtime (i.e., thread pool size).

A.2 Activity Diagram Model Elements

Activity Partition: An *ActivityPartition* is also called swimlane. Each *ActivityPartition* logically organizes the related activities. In Tulsa, each *Artifact* in the deployment diagram is equally mapped to an *ActivityPartition* of the activity diagram, and the actions (e.g., function call) executed in the *Artifact* are specified by Activity element of the *ActivityPartition*.

Initial Node: It is the start point of the activity diagram. There is no activity execution happening before it. It has one or more outgoing control flow(s) which indicate(s) the following activities. Tulsa accepts one *InitialNode* for each activity diagram.

Activity Final Node: It represents the completion of an activity. All the executions will stop when they reach the *ActivityFinalNode*. There can be more than one *ActivityFinalNode* in the activity diagram and Tulsa also supports that.

Opaque Action: An *OpaqueAction* element describes a basic function within an *ActivityPartition*. It is an

ExecutableNode and represents the interaction behaviour of the component.

Call Operation Action: It is used for invoking an action which belongs to the other *ActivityPartition*. The invocation can be synchronous or asynchronous.

Accept Event Action: It is used to define the acceptance or receipt of a request from an action, e.g., *CallOperationAction*. It can be also the start point of the *ActivityPartition*.

Send Signal Action: It is used to generate a signal which is sent to the target object. If an *AcceptEventAction* receives a synchronized event call, the *SendSignalAction* will be used to transmit the signal back to the caller after the job completion.

Control Node: Tulsa considers four types of the control nodes, the Fork Node, the Join Node, the Decision Node and the Merge Node. The ForkNode splits the control flow into a set of concurrent control flows. The Join Node joins (i.e., synchronizes) the flow of a set of concurrent control flows. The Merge Node combines a set of optional control flow paths, and it has multiple incoming control flows and a single outgoing control flow. The Decision Node represents a point of conditional flow, and it has a single incoming control flow and multiple outgoing control flows.

Control Flow: It is used to connect two nodes in an Activity diagram by directing the flow to the target node once the source node's activity is completed. Tulsa checks the property *inPartition* of *ControlFlow* to identify if it is within an *ActivityPartition* or between *ActivityPartitions*.

«GaStep»: This stereotype is defined by MARTE. It provides core tags for the action or message to specifying characteristics of behavior steps. The core tags Tulsa considered are ***blockT*** (specifying customer think time for initial node), ***rep*** (specifying the repetition times of forwarding requests), ***prob*** (specifying the probability of forwarding requests), ***hostDemand*** (specifying the processing time of the task).

One of the key contribution of Tulsa is it also supports the Storm-based applications. Two important concepts of Storm are ***Spout*** and ***Bolt***. DICE profile provides corresponding stereotypes **«StormSpout»** and **«StormBolt»** for them respectively. **«StormSpout»** and **«StormBolt»** also provide tags, e.g., ***blockT***, ***rep***, ***prob***, ***hostDemand***, and ***parallelism*** (representing the number of threads executing the same component), for the Spout and Bolt.

APPENDIX B. Core Functions for Anti-Patterns Detection & Refactoring

Four functions are provided with APR to perform the anti-patterns detection and refactoring. The input data format is XML, **LQN model file** and **solved LQN model**, with the **anti-pattern boundaries**. The following are the functions defined in MATLAB:

B.1 AP1: Infinite Wait

[IWProcessor, IWTask, IWNumSynCall] = IWMaxCallCheck (inputLQNFileName, ThMaxCall);

Where ‘inputLQNFileName’ is location of the generated LQN model file and ‘ThMaxCall’ is the anti-pattern boundary for synchronous calls (i.e., the number of maximum synchronous calls). This function helps to check if there exists any component has synch-calls are greater than threshold *ThMaxCall*.

[AP1Processor, AP1ResTime] = IWMaxResTCheck (outputLQNSolvedFileName, IWProcessor, IWTask, ThMaxResT);

Where ‘outputLQNSolvedFileName’ is location of the solved LQN model file, ‘IWProcessor’ is the software processor which holds the ‘IWTask’ (i.e., component which has synch-calls are greater than threshold *ThMaxCall*) and the ‘ThMaxResT’ is the anti-pattern boundary for the response time (i.e., the maximum response time). This function helps to check if the components detected by function *IWMaxCallCheck* increase the response time.

B.2 AP2: Excessive Calculation

[ECProcessor, ECEntry, ECNumEntry] = ECMaxEntryCheck (inputLQNFileName, ThMaxEntry);

Where ‘inputLQNFileName’ is location of the generated LQN model file and ‘ThMaxEntry’ is the anti-pattern boundary for components (i.e., the number of maximum entry). This function helps to check if there exists any processor has components are greater than threshold *ThMaxEntry*.

[AP2Processor, AP2Util] = ECMaxUtilCheck (outputLQNSolvedFileName, ECProcessor, ECNumEntry, ThMaxUtil);

Where ‘outputLQNSolvedFileName’ is location of the solved LQN model file, ‘ECProcessor’ is the hardware processor which holds the components are greater than threshold *ThMaxEntry* and the ‘ThMaxUtil’ is the anti-pattern boundary for the processor utilization (i.e., the maximum utilization). This function helps to check if the utilization of processor detected by function *ECMaxEntryCheck* is higher than threshold.