Artifact Abstract: An Urban-driven Service Request Management Model

Christian Cabrera, Andrei Palade, Gary White, and Siobhán Clarke
Distributed Systems Group, SCSS, Trinity College Dublin, Dublin, Ireland
Email: {cabrerac,paladea,whiteg5,Siobhan.Clarke}@scss.tcd.ie

This document introduces the artifacts that implement the request manager proposed in the paper "An Urban-driven Service Request Management Model". The artifacts can be found in the public TCD GitLab project percom2020-srmm[1].

Three folders compose this project. The folder **Experiments** contains the file experiments-automation-percom.jar that automates the execution of the experiments in the city simulator (i.e., Simonstrator[2]). The folder **Simonstrator/simonstrator-simrunner** contains the next artifacts:

- config/service-discovery: It contains the configuration files for the evaluated approaches.
- osmDublin: This folder contains the Open Street Maps data used by Simonstrator to simulate the smart city environment.
- simrunner-percom2020.jar: This file packages the implementation of the evaluated approaches on top of Simonstrator.

The folder **SmartCitySD/Data** contains the data used by the evaluated approaches and the simulations, it has the next artifacts:

- ontologies: It contains the knowledge models used by the evaluated approaches to describe, organise and compare services. The owls-tc4 folder has the ontologies that describe services inputs and outputs to compare them. The folder surfOntology-1-0-0 has the ontology that describes the smart city domain including concepts such as city service, city place, etc.
- requests-dataset: This folder contains the service requests used in the experiments in the sub-folder all-requests. The sub-folder all-responses contains the responses of each request, which are used to calculate search accuracy. Requests and expected responses are grouped according to the number of services involved (i.e., from 1 to 8).
- services-dataset: This folder contains the services that are registered in each experiment. The sub-folder iot-services contains the services descriptions that are are requested. The sub-folder dummy-services contains services descriptions that are used to create environments with different number of registered services.

The execution of the experiments requires Java-8[3] as Simonstrator is a Java-based simulator. The next command should be executed in the **Experiments** folder:

java -jar ./experiments-automation-percom.jar <approach> <mode> <rounds>

- approach: This parameter defines the evaluated approach. The possible values are: proximity, interactions, apf, biosocial, and urban.
- mode: This parameter defines the mobility of the gateways in the environment. The possible values are: 1 for static environment, 2 for semi-mobile and 3 for fully mobile.
- rounds: This parameter defines the number of repetitions of each experiment (e.g., 10 rounds).

For example, the next command executes the experiments to evaluate the urban-driven model in a fully mobile environment, where each experiment is repeated 10 times: *java -jar ./experiments-automation-percom.jar urban 3 10*

The experiments-automation-percom.jar file evaluates the specified approach, in the given mobility mode, with different number of registered services descriptions (i.e., 20, 40, 60, 80, and 100 thousand) and number gateways (i.e., 100, 300, and 500). Each experiment execution (i.e., combination of approach, mobility mode, registered services and number of gateways) prompts in console the steps of the simulation, which are: gateways initialisation, gateways maintenance, gateways advertising, citizen initialisation, services registration, citizen requests, and results processing and writing.

Each experiment writes a results Excel file where the simulation results are stored for each round. This file is saved in the folder **SmartCitySD/Results**. The published paper reports the results in the sheets $messages$, $solved$, and $requests$. The sheet $messages$ presents the number of exchanged messages by the evaluated approach in the different experiment steps. The sheet $solved$ shows the number of solved and non-solved requests in each experiment, which are used to calculate the rate of solved requests. The sheet $requests$ presents the number of hops, search precision, and response time means. Experiments have a long running time mainly because of the combination of number of gateways (i.e., 100, 300, 500) and number of services (i.e., 20, 40, 60, 80, and 100 thousand) in the setup. Experiments time varies according to the parameters values for evaluated approach, mobility mode, and number of rounds. Table I shows the estimated time for running one round of experiments for each approach and mobility mode. These experiments run on the Kelvin system, a high performance

[1]percom2020-srmm - https://gitlab.scss.tcd.ie/cabrerac/percom2020-srmm
[2]Simonstrator - https://dev.kom.e-technik.tu-darmstadt.de/simonstrator/
[3]Java 8 - https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

978-1-7281-4657-7/20/$31.00 ©2020 IEEE

TABLE I: Estimated Experiments Time

Approach	Mobility Mode	Estimated Time of 1 Round
proximity	1	1 hour approx
	2	2 hours approx
	3	3 hours approx
interactions	1	1 hour approx
	2	2 hours approx
	3	4 hours approx
apf	1	15 hours approx
	2	15 hours approx
	3	6 hours approx
biosocial	1	15 hours approx
	2	15 hours approx
	3	6 hours approx
urban	1	2 hours approx
	2	12 hours approx
	3	15 hours approx

compute cluster managed by the Trinity Centre for High Performance Computing (TCHPC). Each node in the cluster has a Linux OS, 12 2.66GHz Intel processors, and 24GB of RAM[4]. We recommend to run the experiments in parallel to speed up their execution.

[4]Kelvin Details - https://www.tchpc.tcd.ie/resources/clusters/kelvin