# MODEL-Based Performance Quality Assessment for IoT Applications

https://doi.org/10.3991/ijim.v15i12.21287

Teeba Ismail Kh. (✉)
Lebanese French University (LFU), Erbil, Iraq
Salahaddin University, Erbil, Iraq
teeba.ismail@lfu.edu.krd

Ibrahim I. Hamarash
University of Kurdistan, Hewler, Iraq
Salahaddin University, Erbil, Iraq

**Abstract**—The number of applications incorporating Internet of Things (IoT) concepts increases extraordinarily. This increase cannot continue without high-quality assurance. There are some difficulties in testing IoT applications; the system heterogeneity, test cost and time are taken to test the system and the precision percentage of test results.

A well-known and possibly the best solution to overcoming these difficulties is to model the system for evaluation purposes, known as model-based testing (MBT). This paper aims to design a model-based testing approach to assess IoT applications performance quality attributes. The ISO 25000 quality model is used as a standard for software quality assurance applications. IoTMaaS is used as a case study to implement the methodological approach. The possible test cases were generated using the ACTS combinatorial test generation tool. The performance metrics of each test case were monitored until the optimum test case was identified, with the shortest response time and the least amount of resources used. The proposed testing method appears to be successful, according to the results.

**Keywords**—Internet of things, IoT, prediction, MBT, quality assurance, performance prediction

## 1 Introduction

Internet of things is a grid of objects, cars, consumer electronics, or other items embedded with electronics, software, and sensors that allow connecting and exchanging data with these objects [1]. IoT is increasing rapidly, with vast applications in diverse fields such as medicine, industry, economy, and military [2].

The global market size of IoT-connected devices is projected to hit 41.6 bn in 2025 [3]. This expansion rate involves ensuring the consistency of IoT applications before marketing them. One of the most important methods to check the quality of the soft-

ware is the Quality Assurance (QA) approach. QA is a standardized means of testing whether a product or service meets defined specifications and standards [4]. Furthermore, QA can be used to achieve an optimal stakeholder satisfaction level. Many essential requirements should be achieved: performance, efficiency, security, maintainability, etc [5]. In contrast, the QA model simulates these requirements to sure the quality of the system. The quality model of software applications has been extensively demonstrated within ISO/IEC 25000, defined as "Software Quality Requirements and Evaluation" (SQUARE) [6]. Quality model ISO/IEC 25000 covered eight characteristics with 31 sub-characteristics to ensure the software products quality.

Despite many analogies between traditional software and IoT software applications. Moreover, many challenges can arise when well-established software testing methods and QA approaches are applied to the IoT application world. For examples, the heterogeneity of the system and unavailability of an implemented existing system as a testbed are challenging [7]. As a consequence, there is a gap in addressing specific methodologies for assessing the test generation model in IoT software applications. Several studies have attempted to solve the challenges of assessing IoT software technology.

Currently, the MBT is one of the promising methods [8]. Originally, MBT was automatically extracted from an abstract system model tested (SUT). The MBT can apply to assess the functional and non-functional characteristics of a software product [9]. Research on using the MBT methodology to ensure the consistency of IoT software applications is early due to the novelty of the IoT systems.

This study aims to assess the IoT system performance metrics by designing a novel MBT approach of an IoTMaaS as a case study.

The paper is illustrated as follows: Section 2 illustrates related works. Section 3 presents software application quality assurance, while Section 4 explains the software testing approach—the case study as described in Section 5. Section 6 describes the potential test cases, and section 7 analyzes the findings. Finally, the conclusion is stated in section 8.

## 2 Background and Related Work

A major effort has been spent over the past decade on IoT research. One of this research's most essential topics is quality assurance. In 2016, IoT QA specifications were outlined in six categories: climate, consumer, service level agreement, organization, protection and data management [10]. They also defined the IoT applications QA by splitting it into four categories: defect prevention, defect analysis, user integration, and organizational. In contrast, in 2017, Banerjee and Sheth categorized quality data for IoT applications into two category requirements and conformance. They also enhanced the consistency of IoT by evaluating two daily life cases requiring high quality. [11]. Several studies have been deals with the testing methodology to ensure QA. Ahmad et al. in 2016, they presented the combining MBT technique and a service-oriented solution for IoT platforms in terms of conformity and interoperability [12]. In 2017 Li and Huang propose a predictive approach of reliability-aware performance

evaluation for IoT services using Generalized Stochastic Petri Net (GSPN) [13]. In the same year other authors designed a MBT for the IoT communication [14]. Also, they carried out experiments in which models of five implementations of MQTT brokers/servers, and they examined these models. In 2018, Wang et al. were evaluated the performance quality aspect of blockchain-based IoT security model [15]. Simultaneously, Hiun et al. introduced a service-based approach to an integrated IoT testing system for alignment, costs and scalability. While in the same year, Ying et al. dealt with the reliability which is one attributes of the software quality model. Moreover, they used a continuous-time Markov chain model to evaluate the reliability of Blockchain Based IoT applications [16]. In 2019, an environment of hybrid testing was designed between the execution test and the model checking for the IoT system [17]. In 2020 Rateb et al. using the MBT system to assess the load and safety aspects of IoT-based vehicle communication [18]. On the other side, Miroslav et al. designed an automated model-based approach to assess the smart TV usability and accessibility [14].

Finally, despite numerous published studies on quality assurance research, there is a gap in addressing particular methodologies for assessing the test generation model particular approach in IoT software applications. Therefore, the paper aims to design a specific system for testing the performance quality attributes of IoT applications.

## 3 Quality Assurance of Software Applications

Software quality assurance is the process to define how software quality can be achieved and how the producer company knows that the product has reached quality [19]. Figure 1 shows the main activities of the quality assurance process; the main activities are the selecting and definition of the quality model that will be applied to assess the quality of the product. The quality characteristics of the product can be assessed by utilizing the quality model. The design of quality model requires to specify the quality attributes to assess the features of a product software. The quality characteristics were defined based on a set of standards such as ISO/IEC 9126 [6].

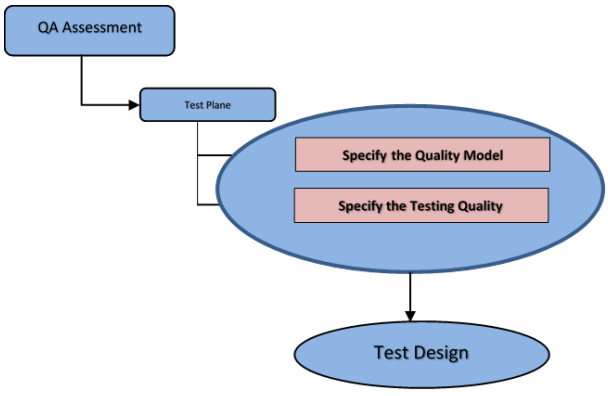


**Fig. 1.** The main activities of the QA

The International Organization for Standardization (ISO) illustrates these standards to discuss the quality aspects of the software as a quality model [20]. For the IoT system, the quality is the degree to which the system satisfies the stated and implied needs of its various users [21]. The requirements of users are performance, security, maintainability, portability, etc. They are precisely what is represented in the quality model, which classes the product quality into main characteristics and sub-characteristics. ISO/IEC 25000 is a quality model of software applications thoroughly outlined in a set of standards known as "Software Quality Requirements and Evaluation" (SQUARE) [6]. The ISO/IEC 25000 standard provides a set of quality attributes for the software: performance, compatibility, usability, reliability, security, maintainability, functionality and portability. These quality attributes are further divided into sub-features such as time-behavior, resources utilization, the fault tolerance, and availability, which can be considered to design the quality model. The paper's scope is the performance aspect of the quality model.

## 4 Software Testing Approaches

There are four approaches that can be used to test the quality of software application; Model-Based Testing (MBT), mathematical model, testbed and prototype [20]. As shown in Figure 2, many different steps are required to test the software application, starting with the test plane, designing a model test, running the test, and analyzing the results. The products will be delivered if the test results reach the required specifications; otherwise, the product will need to be developed and tested again.

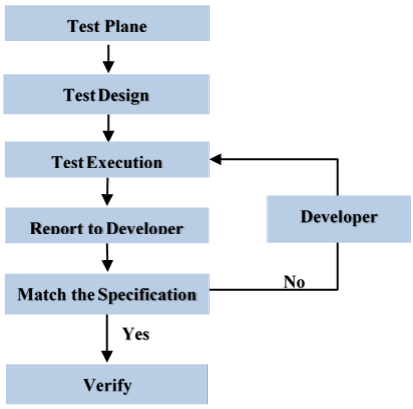


**Fig. 2.** Testing cycle

MBT was used in literature to test a software product's functional and non-functional characteristics [22]. Unlike traditional software systems, research using the MBT approach to ensure the quality of IoT software applications is early due to the uniqueness of the IoT approach itself. However, the past decade has seen many attempts in this direction. Originally, MBT was a research methodology in which the

"test cases and expected results" were automatically extracted from an abstract system model under test (SUT). Specifically, MBT techniques derive abstract test cases from an MBT model, formalizing the behavioral aspects of SUT in its environment and at a given abstraction level. Test cases developed from such models allow validation of functional aspects of the system by comparing back-to-back results observed with those described by the MBT model. MBT is usually performed for black-box inspection. It's a widely used methodology that has gained much attention from academic and industrial domains in recent years. MBT is an industry-wide technology to avoid collisions, improve performance and minimize costs. MBT is a method of creating automated test procedures based on system criteria and behavior models [23]. While this form of testing necessitates more effort in terms of model development, it has a number of advantages over conventional software testing methods, as shown below:

1. Once rules are defined.
2. Lower project maintenance. No need to write new tests for each new function. Once we have a model, creating and regenerating test cases is simpler than hand-coded test cases.
3. The architecture is fluid. When introducing a new function, a new action is introduced to the model in conjunction with existing actions. A simple change will cascade through the entire suite of test cases.
4. Design more, code less.
5. [High coverage]. Tests continue to find bugs, not just regressions due to code path changes or dependencies.
6. Model authoring is independent of the implementation and actual research so that both tasks can be performed simultaneously by separate team members.

In this paper, MBT approach is employed to design the testing model to assess the performance attribute of internet of things software applications.

### 4.1 Model-Based Testing for Performance Attribute (MBTPA)

In Software Engineering, performance testing is necessary. It must be made before marketing the software product. This test will ensure customer satisfaction & protects an investor investment against product failure. The performance of the software component was influenced by the usage [3]. The performance metrics may vary depending on system nature [5]. Generally, the performance metrics can be summarized by the response time, throughput, and resource utilization [24]. For IoT software systems, the performance has the same metrics. Designing a novel MBT to assess the performance quality aspect of IoT software applications is the paper contribution. During the test execution period as illustrated in Figure 3, the performance metrics of the MBT method will be monitored. This test will provide an early indication of the possible performance issues.

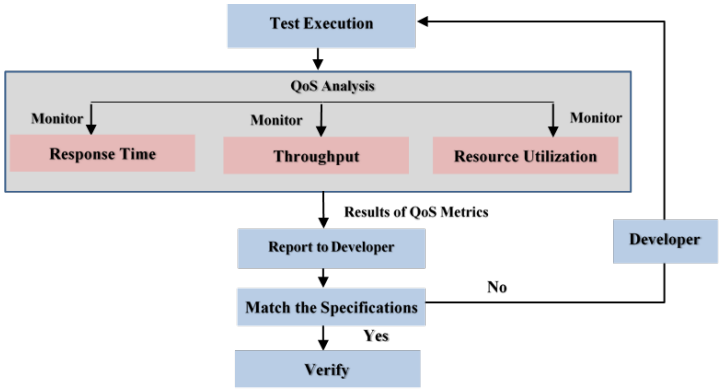


**Fig. 3.** The cycle of MBTPA

## 5 Case Study

To build the model-based quality prediction approach in IoT software applications, a real-world IoTMaaS case study is utilized has been addressed to assess the QA for it. IoTMaaS is already used to overcome the heterogeneity of devices in IoT systems and it is a new class of cloud-based IoT mashup service model [25]. In the next subsection, this case will be illustrated in detail.

### 5.1 IoTMaaS architecture

IoTMaaS characterize as a mashup of software, things, and computation resources as shown in Figure 4. The mashup process includes three components; they are often customized to the user preference, processing software and the number of computation resources to use during the run-time. The results can be used as actuators for user alerts or monitoring items. IoTMaaS aims to alleviate the heterogeneity of devices by following the Model-Driven Architecture (MDA). It brings the system interoperability and mashup service without limiting them on the same platform and protocol. The IoTMaaS Service Planner (SP) may also be an IoTMaaS template-defining technical service designer. The Software Component (SWC) assembly definition customized later. The SP is responsible for IoTMaaS proper, reliable service. IoTMaaS Instantiation Client (IC) is sent to the IoTMaaS Frontend Service (FS). FS sends an IaaS supplier set-request to run a virtual machine on the IoTMaaS Service Deployer (SD). In addition, FS sends an application to SD to retrieve SWCs and Thing Service Drive (TSDs) from the SWC repository and Thing Profile Service (TPS). Thing Identifier Service (TIDS) offers a TPS address recovery mechanism. TPS is run by manufacturers and registered with TIDS. In comparison, SD recovers object addresses via TIDS and Name Service (DNS).

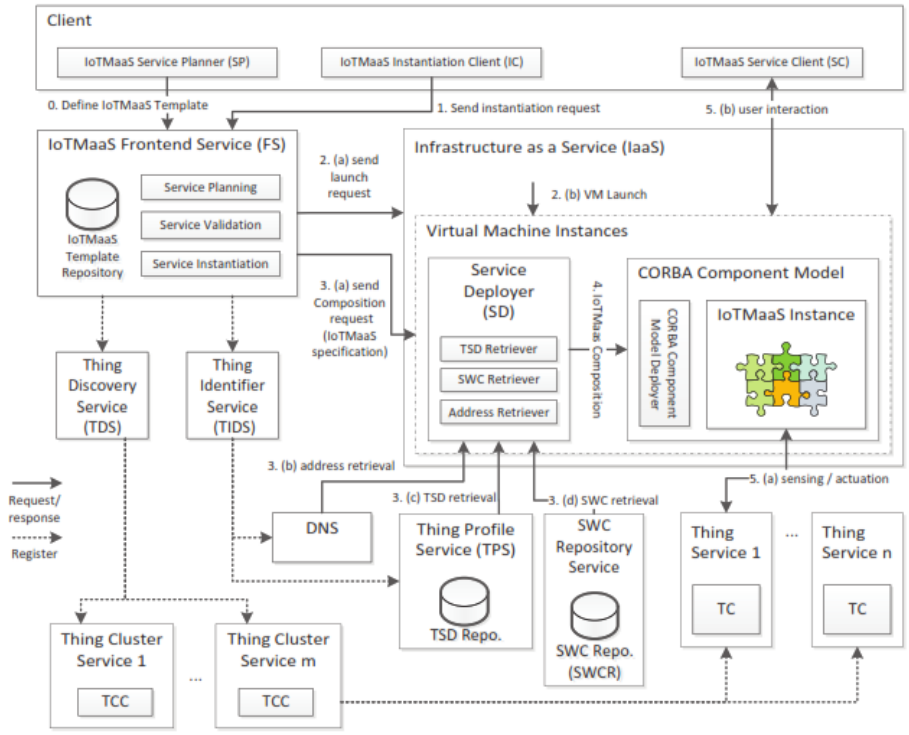


**Fig. 4.** The Case Study (IoTMaaS) Architecture [25]

Finally, SD assembles SWCs and TSDs as defined by IoTMaaS. The assembled service is known as the IoT Service Instance (SI), which gathers and processes the information from items to notify or function on the Service Client (SC). SI can also have a Useful Interface (UI) and User Experience (UX). FS uses Stuff Discovery Service (TDS) to urge things and verify their availability. Additionally, TDS maintains a Stuff Cluster Service (TCS) listing for discovery and availability checks. Thing Cluster Controller offers TCS (TCC). TCC handles stuff hierarchically and provides access control. All are abstracted through Thing Controller (TC), which presents the functionalities of things as a service [25].

## 5.2 MBT for the IoTMaaS

To build an MBT for the IoTMaaS case study, Palladio Component Model (PCM) was used. PCM is a meta-model describing component-based architectures. Software developers used the PCM-Bench to build PCM meta-model instances and effectively predict quality-of-service (QoS) analyzes [24]. Our model was built to predict QoS performance attributes. The main components used to design the MBT will be illustrated.

### 5.3 Component repository model

MBT central idea is to build complex repository software systems by integrating essential system components. Figure 5 shows the Component Repository Model (CRM) of (IoTMaaS) case study. This model contains component and interfaces. The entities which are stored in the repositories will introduce firstly. After that, the Service Effect Specifications (SEFF) which is the model component services abstract behavior and performance properties. The composite structures improve device predictability during early design phases. On the other hand, it is enabling system architects to improve it. The initial goal of the modelling is to predict the level of quality of the IoT applications before realizing them to the user.

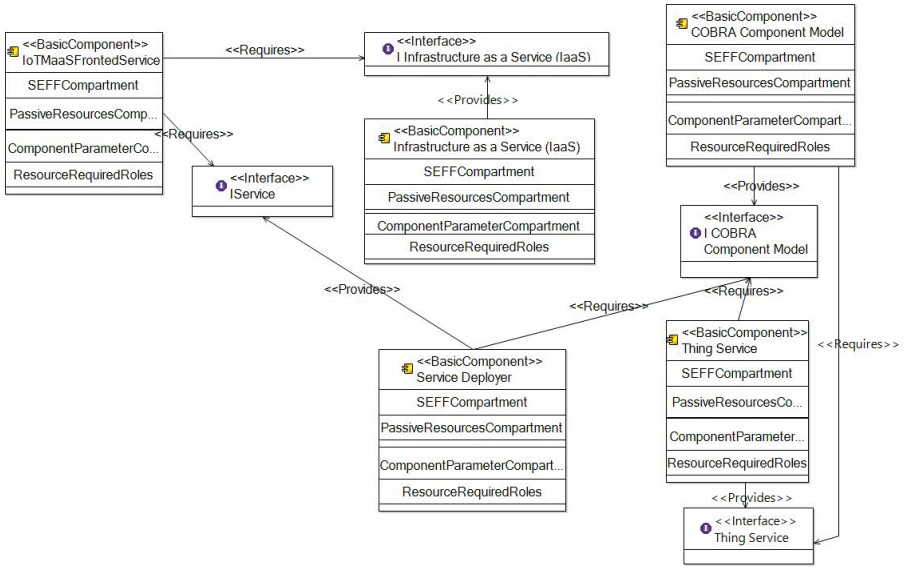


**Fig. 5.** The Component Repository Model of IoTMaaS

### 5.4 Component parameters description

There are five basic components of IoTMaaS. Every component in the CRM of the IoTMaaS has specific resource demands based on its functionality in the system. Table 1 shows the description of these resources parameters.

**Table 1.** The IoTMaaS parameters description

| Parameters | Description | Resources |
|---|---|---|
| P1 | Thing Service to Cobra Component Model Linking Resource | Latency/Throughput |
| P2 | IaaS CPU Basic Component | Number of Replicas/Processing Rate |
| P3 | IoTMaaS Frontend Service Basic Component | Number of Replicas/Processing Rate |
| P4 | Thing Service Delay Basic Component | Number of Replicas/Processing Rate |
| P5 | Thing Service Component – sensing impl. scenario | Impl. Scenario |

Thing Service to Corba Component Model Linking Resource: Cobra Component Model (CCM) computing resources are latency and throughput. Latency is the time takes to pass data in milliseconds between its source and destination. Throughput refers to the amount of requests a system can process per time unit.

IaaS CPU Basic Component: Infrastructure as a service (IaaS) key demand is the CPU, which has two main effects resources, processing rate and number of replicas. The processing rate specifies the frequency at which the resource will execute the abstract units stated in resource specifications. The actual number of processors is categorized by the number of replicas attribute in the Palladio Processing Resource. This attribute is mapped into the number of cores of a virtual IaaS CPU.

IoTMaaS Frontend Service: The IoTMaaS Frontend Service resource container has two main effected resources of the CPU: the processing rate and the number of replicas. The processing rate specifies the frequency at which the resource will execute the resource specifications. The actual number of processors is categorized by the number of replicas attribute in Palladio Processing Resource. This attribute represents the number of cores of a virtual IoTMaaS Frontend Service CPU.

Thing Service Delay: The thing service delay resource container has two main effected resources of the delay function: the processing rate and the number of replicas. The processing rate specifies the frequency at which the resource will execute the resource specifications. The actual number of processors is categorized by the number of replicas attribute in Palladio Processing Resource. This attribute represents the number of cores of a virtual Thing Service delay CPU.

Thing Service component – Sensing Impi. Scenario: Things and their management infrastructure should exist in order to mash up IoTMaaS. There are several ways in which things are managed: Centralized and Distributed. Centralized is that the centralized registry that provides monitoring and management functions should register all things. The distributed way is that things maintain peer-to-peer monitoring and management infrastructure, but large amounts of network traffic should be shared between things. In our model, we implemented two scenarios for sensing (A or B). The Thing Controller (TC) introduced the distributed way scenario in A, so the device specifications of the services would function according to a certain order (see Figure 6).On the other hand, Figure 7 shows, the implementation of scenario B sensing, which uses TC centralized registry. Services are required by the system to perform tasks simultaneously.

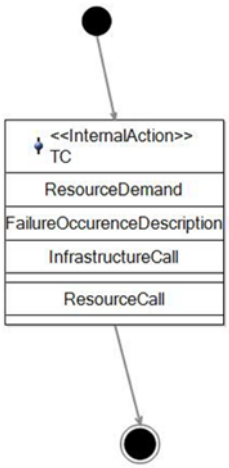


**Fig. 6.** Distributed way of sensing scenario "A" in TC component

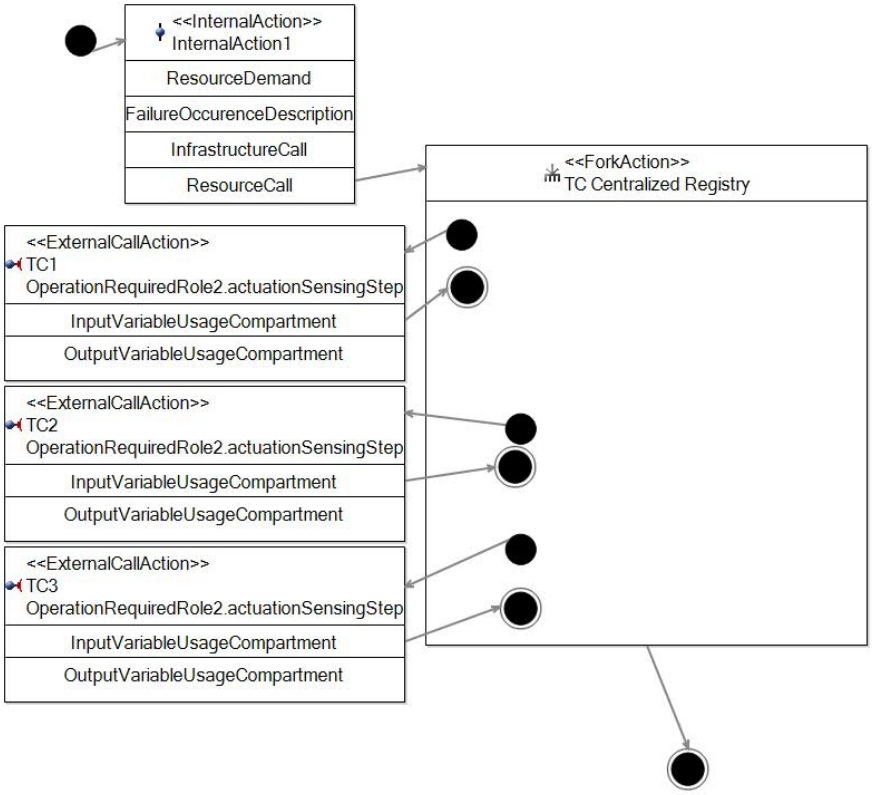


**Fig. 7.** Centralized registry sensing scenario "B" in TC component

## 6 Test Cases

A combinatorial test generation method called ACTS was used to produce the best test cases from the random values of the P1, P2, P3, and P4 [26][27]. The generation of t-way test sets with t ranging from 1 to 6 was supported by ACTS. Combinatorial testing is useful for detecting defects that occur from unexpected interactions among contributing factors. ACTS has provided both command line and graphical user interfaces. The main test generation algorithm used in ACTS is IPOG. IPOG normally achieves a good balance between the size of the test and the time it takes to generate it. Additional features of the IPOG algorithm include mixed strength test generation and constraint management. As a result of the ACTS tool, we got the five best test cases after running the random values of the P1, P2, P3, and P4. In both sensing scenarios, the best test cases were implemented (A and B). As a result, the best test cases in the end were ten. Table 3 presents the configurations of the best test cases. Figure 8 depicts the test case generation statistics process.

**Table 2.** The implementation values of IoTMaaS parameters

| Parameters | The implementation values |
|---|---|
| P1 | 0.2/400<br>0.6/800<br>1/1000 |
| P2 | 2/4000<br>4/2000<br>8/1000 |
| P3 | 1/4000<br>2/2000<br>4/1000 |
| P4 | 1/1000<br>1/2000<br>1/4000 |

**Table 3.** The best test cases generated by ACTS tool

| # Degree of interaction coverage: 2 | | | | | |
|---|---|---|---|---|---|
| # Number of parameters: 5 | | | | | |
| # Maximum number of values per parameter: 3 | | | | | |
| # Number of configurations: 10 | | | | | |
| **Test Case No.** | **P1** | **P2** | **P3** | **P4** | **P5** |
| 1 | 0.2/400 | 4/2000 | 1/4000 | 1/4000 | A |
| 2 | 0.6/800 | 8/1000 | 1/4000 | 1/1000 | A |
| 3 | 0.6/800 | 2/4000 | 2/2000 | 1/4000 | A |
| 4 | 1/1000 | 4/2000 | 2/2000 | 1/1000 | A |
| 5 | 0.2/400 | 4/2000 | 4/1000 | 1/2000 | A |
| 6 | 0.2/400 | 8/1000 | 2/2000 | 1/2000 | B |
| 7 | 0.2/400 | 2/4000 | 4/1000 | 1/1000 | B |
| 8 | 0.6/800 | 4/2000 | 4/1000 | 1/2000 | B |
| 9 | 1/1000 | 8/1000 | 4/1000 | 1/4000 | B |
| 10 | 1/1000 | 2/4000 | 1/4000 | 1/2000 | B |

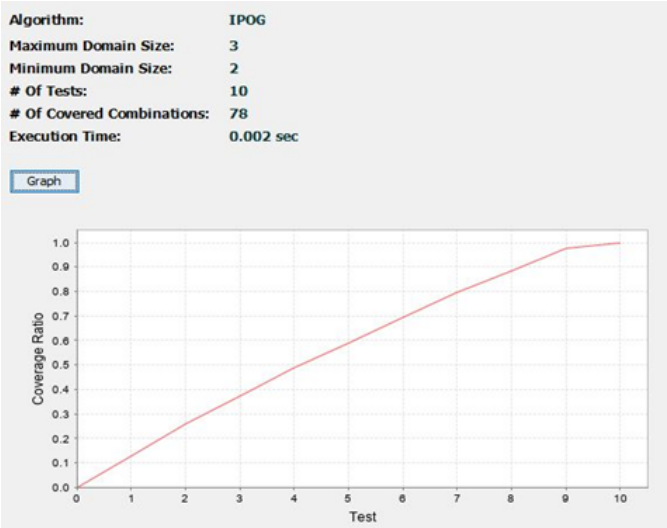


**Fig. 8.** Test generation process by ACTS tool

The ten best test cases were run by our PCM-Bench model to assess its performance metrics and analyze the results. The duration of the simulation time was 119.0520151 seconds which is a reasonable time. Figure 9 shows the simulation process.

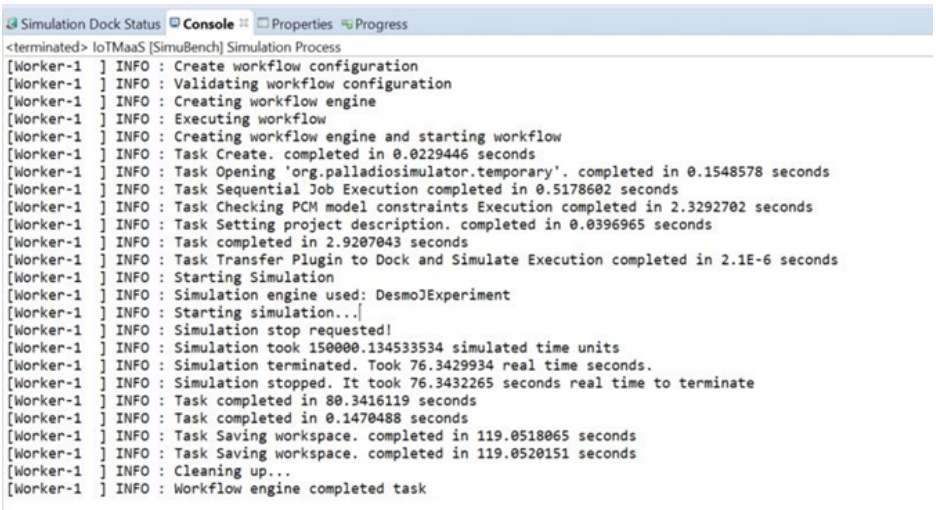


**Fig. 9.** IoTMaaS [SimuBench] simulation process

## 7 Analysis of Results

In this section, the simulation results will be analyzed in terms of response time and resource utilization, both of these are performance metrics. The response time is the most significant factor in the output quality attribute; Figure 10 shows the Cumulative Distribution Function (CDF) of the response time for the ten test cases. The test cases (2, 4 and 7 which are depicted in light blue, yellow, and light brown, respectively) have the best response time actions and should be noted.

The "B" sensing implementation scenario was implemented for test cases 2, 4, and 7, which have the best response time; therefore, the "B" sensing implementation scenario must be implemented to improve the response time.

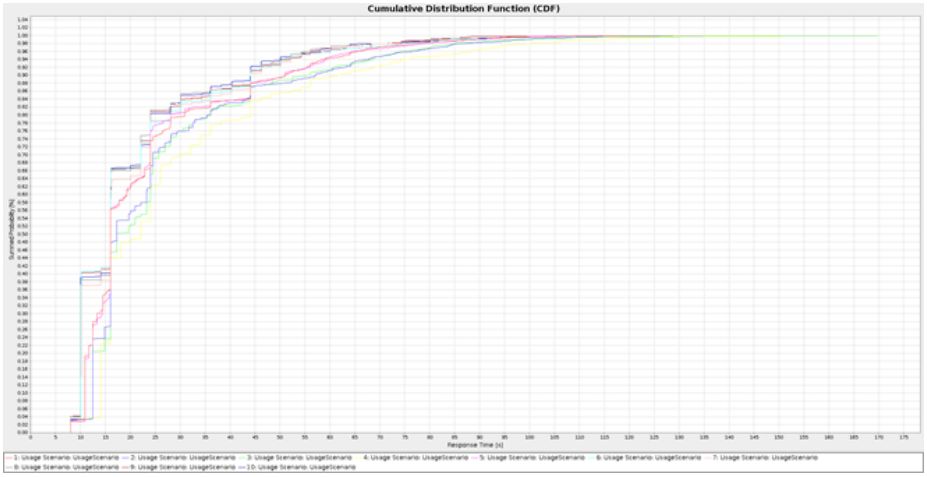


**Fig. 10.**The CDF of the response time for the ten test cases

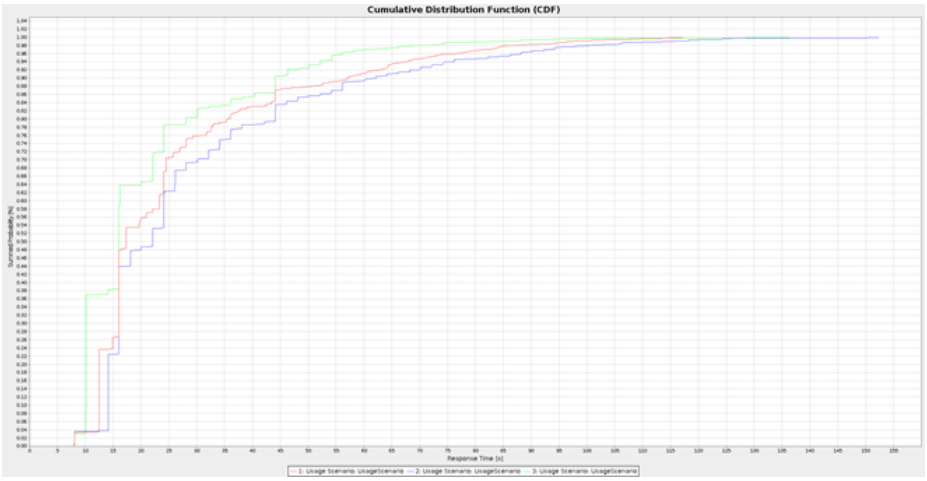


**Fig. 11.**The CDF of the response time for the 2, 4, 7 test cases

The test cases (2, 4, and 7) were rerun in order to select the optimal test case for enhancing the system performance. Figure 11 shows three different patterns, each representing a behavior of response time—the first, shown in light-green, is the second Test Case (TC2) in table 3, with P1=0.6/800. The second, which was represented in blue, was the (TC4 in table 3) P1 = 1/1000, which had a longer delay in the process. Finally, when P1 = 0.2/400, the third one (TC7 in table 3) was presented in green color. To achieve a good response time, a lower latency value (TC7) must be implemented because in this scenario, the system has a 64% chance of responding to a call in less than 15 seconds, which is considered a reasonable response time. On the other hand, our system throughput tends to be sufficient.

Figure 12 depicts the test cases from the perspective of the resource usage metric. As shown in Figure 12, the lowest-demanding test cases are 3, 7, and 10, which appear in light green, brown, and dark red, respectively. In this scenario, the main CPU had been used at 76 percent for the previous 0.4 milliseconds. This means that the hardware resources consumed per time unit are limited to a minimal level. While in test cases 1, 4, 5, and 8, shown in light red, yellow, pink, and black have middle resources usage. On the other hand, cases 2, 6, and 9 have the largest demand activities; they also appeared in light blue, turquoise, and dark red. As a result, P2 has the significant effects on IaaS CPU demand. Since they had the same value of P2, which was 2/4000, the TC3, TC7, and TC10 outputted the same pattern. The TC2, TC6, and TC9 had the same value of P2, which was 8/1000, and they outputted the same pattern. Finally, using four replicas and a processing rate of 2000 in P2, in TC1, TC4, TC5, and TC8 all output the same pattern in the CDF of CPU demand on IaaS. To improve system resource utilization, set the value of IaaS CPU resources to p2, which has two virtual IaaS CPU cores, and 4000 is the frequency at which the resource will execute the abstract units specified in resource specifications.

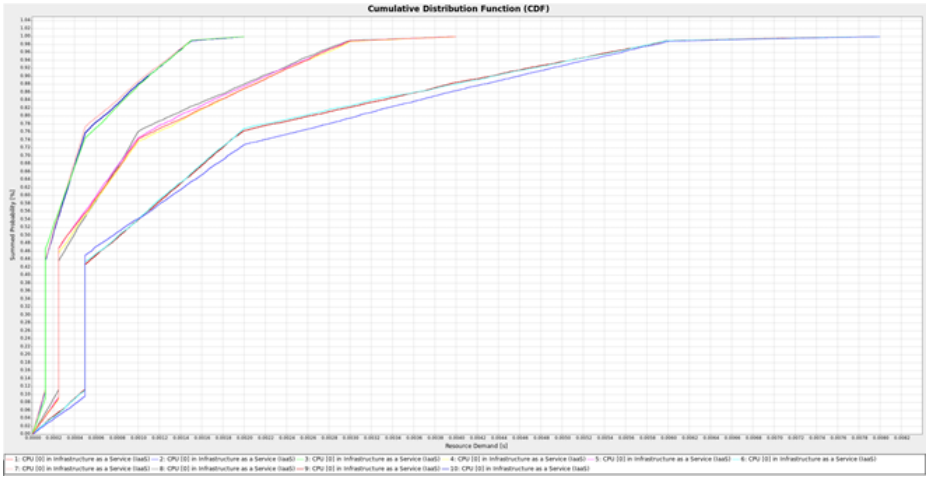


**Fig. 12.**The CDF of CPU demand on IaaS for ten test cases

## 8 Conclusion

The proposed MBTPA overcame system heterogeneity, which is one of the most common IoT testing challenges. As a case study, MBTPA was used to test IoTMaaS. The test cases were run on the PCM-Bench in 119.0520151 seconds, which is a reasonable simulation time. The output of the simulation was assessed from two perspectives: response time and resource utilization. As a consequence, (TC7), which has a lower latency value, is the best test case. From the resource utilization perspective, the Infrastructure as a service (IaaS) (P2) has the significant effects on system resource utilization. The optimum value of p2 is (2/4000) which is set in TC7. As a result, TC7 must be introduced in order to improve IoTMaaS system response time and resource usage, thus improving system performance.

The testers can use MBTPA to assess the performance of IoT software applications prior to launching the implementation stage, saving designers time and resources.

It may be suggested in the future to develop and incorporate MBTRA as a Model-Based Testing for Reliability Attribute.

## 9 References

[1] M. H. Qasem and W. AlMobaideen, "Heterogeneity in IoT-based Smart Cities Designs," *Int. J. Interact. Mob. Technol.*, vol. 13, no. 12, 2019. https://doi.org/10.3991/ijim.v13i12.9763

[2] S. Lam and W. Chung, "Uses of Internet and Mobile Technology in Health Systems for the Elderly.," *Int. J. Interact. Mob. Technol.*, vol. 4, no. 2, 2010. https://doi.org/10.3991/ijim.v4i2.1175

[3] K. Rose, S. Eldridge, and L. Chapin, "The internet of things: an overview, Internet Society." 2015.

[4] D. Nirmala and T. LathaMaheswari, "Automated testcase generation for software quality assurance," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, 2016, pp. 1–6. https://doi.org/10.1109/isco.2016.7727003

[5] I. Areni, A. Waridi, I. Amirullah, C. Yohannes, A. Lawi, and A. Bustamin, "IoT-Based of Automatic Electrical Appliance for Smart Home," *Int. J. Interact. Mob. Technol.*, vol. 14, no. 18, 2020. https://doi.org/10.3991/ijim.v14i18.15649

[6] I. O. f. Standardization, "The International Organization for Standardization (ISO)," *ISO*, 2019. https://www.iso.org/home.html.

[7] C. Adjih *et al.*, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 459–464. https://doi.org/10.1109/wf-iot.2015.7389098

[8] R. M. Gomes and M. Baunach, "A model-based concept for rtos portability," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–6. https://doi.org/10.1109/aiccsa.2018.8612862

[9] B. Legeard and M. Utting, *Practical model-based testing: A tools approach*. Morgan Kaufmann Publishers, 2006.

[10] H. Foidl and M. Felderer, "Data science challenges to improve quality assurance of Internet of Things applications," in *International Symposium on Leveraging Applications of Formal Methods*, 2016, pp. 707–726. https://doi.org/10.1007/978-3-319-47169-3_54

[11] T. Banerjee and A. Sheth, "Iot quality control for data and application needs," *IEEE Intell. Syst.*, vol. 32, no. 2, pp. 68–73, 2017. https://doi.org/10.1109/mis.2017.35

[12] A. Ahmad, F. Bouquet, E. Fourneret, F. Le Gall, and B. Legeard, "Model-based testing as a service for iot platforms," in *International Symposium on Leveraging Applications of Formal Methods*, 2016, pp. 727–742. https://doi.org/10.1007/978-3-319-47169-3_55

[13] S. Li and J. Huang, "GSPN-based reliability-aware performance evaluation of IoT services," in *2017 IEEE International Conference on Services Computing (SCC)*, 2017, pp. 483–486. https://doi.org/10.1109/scc.2017.70

[14] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-based testing IoT communication via active automata learning," in *2017 IEEE International conference on software testing, verification and validation (ICST)*, 2017, pp. 276–287. https://doi.org/10.1109/icst.2017.32

[15] Z. Wang, X. Dong, Y. Li, L. Fang, and P. Chen, "IoT security model and performance evaluation: a blockchain approach," in *2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, 2018, pp. 260–264. https://doi.org/10.1109/icnidc.2018.8525716

[16] Y. Liu, K. Zheng, P. Craig, Y. Li, Y. Luo, and X. Huang, "Evaluating the Reliability of Blockchain Based Internet of Things Applications," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, 2018, pp. 230–231. https://doi.org/10.1109/hoticn.2018.8606026

[17] T. Kuroiwa, Y. Aoyama, and N. Kushiro, "A hybrid testing environment between execution test and model checking for IoT system," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–2. https://doi.org/10.1109/icce.2019.8661998

[18] R. Jabbar, M. Krichen, M. Kharbeche, N. Fetais, and K. Barkaoui, "A Model-Based Testing Framework for Validating an IoT Solution for Blockchain-Based Vehicles Communication," 2020. https://doi.org/10.36227/techrxiv.12030765.v1

[19] B. R. Maxim and M. Kessentini, "An introduction to modern software quality assurance," in *Software Quality Assurance*, Elsevier, 2016, pp. 19–46. https://doi.org/10.1016/b978-0-12-802301-3.00002-8

[20] I. I. Hamarash, "Model-Based Quality Assessment of Internet of Things Software Applications: A Systematic Mapping Study.," *Int. J. Interact. Mob. Technol.*, vol. 14, no. 9, 2020. https://doi.org/10.3991/ijim.v14i09.13431

[21] M. Fahmideh and D. Zowghi, "An exploration of IoT platform development," *Inf. Syst.*, vol. 87, p. 101409, 2020. https://doi.org/10.1016/j.is.2019.06.005

[22] A. Patel and T. A. Champaneria, "Fuzzy logic based algorithm for Context Awareness in IoT for Smart home environment," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 1057–1060. https://doi.org/10.1109/tencon.2016.7848168

[23] V. Rechtberger, M. Bures, and B. S. Ahmed, "Alternative Effort-optimal Model-based Strategy for State Machine Testing of IoT Systems," in *Proceedings of the 2020 The 2nd World Symposium on Software Engineering*, 2020, pp. 141–145. https://doi.org/10.1145/3425329.3425330

[24] M. K. and K. K. Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, *Modeling and Simulating Software Architectures The Palladio Approach*. London, England, 2016. https://doi.org/10.1145/3183440.3183474

[25] J. Im, S. Kim, and D. Kim, "IoT mashup as a service: cloud-based mashup service for the Internet of things," in *2013 IEEE international conference on services computing*, 2013, pp. 462–469. https://doi.org/10.1109/scc.2013.68

[26] N. Ramli, R. R. Othman, R. Hendradi, and I. Iszaidy, "T-way Test Suite Generation Strategy based on Ant Colony Algorithm to Support T-way Variable Strength," in *Journal*

*of Physics: Conference Series*, 2021, vol. 1755, no. 1, p. 12034. https://doi.org/10.1088/1742-6596/1755/1/012034

[27] L. Yu, Y. Lei, M. Nourozborazjany, R. N. Kacker, and D. R. Kuhn, "An efficient algorithm for constraint handling in combinatorial test generation," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 242–251. https://doi.org/10.1109/icst.2013.35

## 10 Authors

**Teeba Ismail Kh**. B.S. got. Degree in optical electronics from Al-Nahrain University, Baghdad, Iraq, 2004, and M.S. Degree in Laser and optoelectronics from Al-Nahrain University, Baghdad, Iraq, 2007. She is currently studying Electrical Engineering at Salahaldine University, Erbil, Iraq. She is an assistant lecturer at Erbil, Iraq, Lebanese French University Faculty of Engineering. Her research involves model-based testing and IoT software systems quality assurance. E-mail teeba.ismail@su.edu.krd

**Ibrahim I. Hamarash.** Is an engineering professor at Salahaddin University since 1993. He graduated from Salahaddin University (1988), his MSc, in Electronics and Communication Engineering. Mosul University's Control Systems Engineering (1993), Ph.D. In Control Systems Engineering (1999), his ICT project management certificate from UCSD, USA (2005). His research interests include robotics and manipulators, data processing, modelling, simulation and control of complex systems. Email: - ibrahim.hamad@ukh.edu.krd

Article submitted 2021-01-18. Resubmitted 2021-04-08. Final acceptance 2021-04-09. Final version published as submitted by the authors