# Towards Automated Testing of the Internet of Things: Results Obtained with the TESTAR Tool

Mirella Martínez[2], Anna I. Esparcia-Alcázar[2], Tanja E. J. Vos[1,2(✉)], Pekka Aho[1], and Joan Fons i Cors[2]

[1] Open University, Heerlen, The Netherlands
{tanja.vos,pekka.aho}@ou.nl
[2] Universidad Politecnica de Valencia, Valencia, Spain
{mmartinez,aesparcia,tvos,jjfons}@pros.upv.es
http://www.testar.org

**Abstract.** As the Internet of Things (IoT) becomes a reality, the need of ensuring the security and reliability of massively interconnected devices becomes a pressing necessity. A means of satisfying this need would be automated testing of IoT devices; however, this presents many difficulties, such as the lack of standards, multitude of manufacturers, restricted capabilities (such as power), etc.

In this work we present the first results on using TESTAR tool for automating IoT testing of smart home devices. TESTAR is a tool for automated testing at the Graphical User Interface (GUI) level of an application. The tool uses the Accessibility API the obtain information about the GUI and derive actions that can be executed in test sequences. Many IoT systems use the REST API to access the resources that compose the system. Consequently, this paper looks into IoT system testing as a natural field for extending the TESTAR philosophy from GUI (Accesibility) to IoT (REST) APIs.

The results show the potential of TESTAR in this new environment.

**Keywords:** Automated testing · Internet of Things · IoT

## 1 Introduction

The number of devices connected to the Internet has experienced a tremendous growth in the recent past and it is expected to keep growing. Gartner [15] forecasts that this number will reach 26 thousand million by 2020. We are already being witnesses of this, as connected vehicles, homes, cities and health monitors have made their appearance in the past few years [6].

The popularity of the Internet of Things (IoT) stems from the fact that it has the potential to change the way we work and live. However, its uptake may be hindered by the cost of devices, as consumers see them as a non-essential expense [4]. At the same time, it is necessary to ensure the reliability of the


T. Margaria and B. Steffen (Eds.): ISoLA 2018, LNCS 11246, pp. 375–385, 2018.
https://doi.org/10.1007/978-3-030-03424-5_25

devices connected as a malfunction could have a very negative impact, even endangering lives.

Automated testing is already an important part of the software development cycle, but it becomes crucial in ensuring security and reliability of IoT devices, as well as reducing development costs. However, testing the IoT presents many challenges [8] stemming from the distributed functionalities and services, resource restrictions (such as limitations in memory, processing power, bandwidth and battery life), and the performance of the network communications.

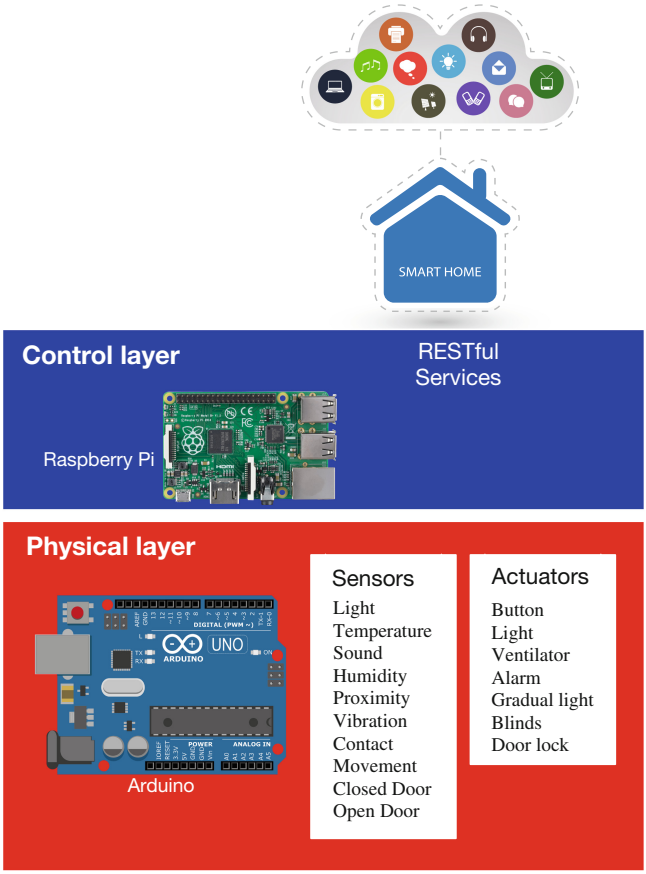


**Fig. 1.** Smart home architecture

In this paper we present the first results obtained when applying the TESTAR[1] tool [16] for automated testing of various IoT devices, in the context of a smart home. TESTAR has already been applied successfully to automate

[1] www.testar.org.

software testing at the graphical user interface (GUI) level in various organisations [1,3,10,12,16]. This paper will present the results of its application to an IoT system as a natural field for extending the TESTAR philosophy from GUI (Accesibility) to IoT (REST) APIs.

The rest of the paper is structured as follows. Section 2 summarises the challenges and existing work carried out in IoT testing. In Sect. 4 we describe the TESTAR tool and the modifications that were necessary in order to adapt it to support IoT testing. Section 6 describes the smart home case study, the procedure carried out for experimentation and the results obtained. Finally, in Sect. 7 we provide conclusions and outline areas for future work.

## 2 IoT Systems and Testing

Although different definitions for the term IoT can be found, they all have in common the fact that IoT tries to integrate the physical world with the virtual world of the Internet. The physical objects that you can interact with are the so-called "Things" of the IoT, or as defined in [9], the entities of interest. In order to interact with these physical objects, we need devices that can be either attached or embedded into the entities (constituting the so-called intelligent things), or installed into the environment to be monitored. Among these devices we find, for example, RFID readers, sensors, actuators, embedded computers and even mobile devices. These devices usually host resources that provide a link to the entities of interest and, for example, provide information about things (including performance capabilities).

Embedded systems have been around for a long time. However, traditionally these systems were self-contained and could work in isolation. As the IoT becomes popular, the need for connected objects "conversing" with each other to properly function is changing the considerations that need to be taken into account. As stated in [14], developers must consider ways to streamline device-to-device (D2D) and device-to-server (D2S) communication, and take into account the human interaction resulting of our everyday objects being connected to the Internet.

Therefore, testing IoT devices shares the problems encountered when testing both embedded and distributed systems. Moreover, the heterogeneity and the large scale of objects [17] increase the number of challenges when trying to ensure their security and reliability.

According to OWASP IoT Top Ten (www.owasp.org), insecure cloud interfaces need to be considered in order to greatly improve the security of any IoT product. This involves the assessment of API and cloud-based web interfaces. Testing an API requires the use of software to send calls and process the response given by the system. An overview of API testing can be found in [7,13]. In this work, we present a different approach based on automated scriptless testing. Through our approach, given a valid specification of the System Under Test (SUT), the calls made to interact with it will be automatically derived. Thereby unattended testing will be enabled.

## 3 The Smart Home

The smart home we test in this paper has a series of physical devices (things) offered on an IoT [5] platform through a set of RESTful [11] services. An overview of the architecture is illustrated in Fig. 1. The IoT platform has a microcontroller (from Arduino) to which the different electronic devices that offer capacities to perceive external situations and to act on the physical world are connected. The microcontroller is also connected to a reduced low-cost computer (Raspberry Pi) that functions as a connection gateway for the physical devices and hosts the IoT accessibility platform based on RESTful services. These services can be consumed by various customers, including laptops and smartphones, and anything else that is part of the IoT and wants to communicate with the resources available in the smart home.

The smart home tested in this paper, being a system for academic purposes [5], has a basic security mechanism through which only users with valid credentials can modify the status of resources. There are 17 available resources (10 sensors and 7 actuators), and each one of them has its own identifier (ID) and specific functionalities that determine the ways it can be interacted with. The resources and their corresponding functionalities available in the smart home and the interactions that they accept are showed in Table 1.

**Table 1.** Resources of the smart home with their interactions and functionalities

| Resources | Functionality | Interactions |
|---|---|---|
| Light sensor, Temperature sensor<br>Sound sensor, Humidity sensor<br>Proximity sensor | Numeric | set, read |
| Vibration sensor, Contact sensor<br>Movement sensor, Closed door sensor<br>Door lock, Push button<br>Fan, Light, Alarm | Bistate | on, off<br>toggle, read<br>pulseOn, pulseOff |
| Blinds | Movement | open, close<br>stop, stepOpen<br>stepClose, movePosition<br>moveNamePosition, up<br>down, stepUp<br>stepDown, read |
| Gradual light | Dimmer | set%, set0x<br>setAng, read |

The REST API [11] that allows us to access the resources of the Smart Home, explicitly and consistently use HTTP methods. The following association is established between create, read, update and delete (CRUD) and the corresponding HTTP methods:

- POST is used to create a resource on the server
- GET allows you to obtain an available resource

– PUT is used to carry out updates of a resource
– DELETE allows you to delete an available resource

RESTful resources perform stateless communications. An application or client of a REST web service includes, between the header and the body of the HTTP request, all the necessary data to generate a response, so that each request works in isolation. This allows to improve the performance of the service and, since there is no need to synchronize session data with external applications, the design and implementation of server-side components is simpler. RESTful services are resource oriented and each has a uniform resource identifier (URI). The different URIs serve both as the name and address of a resource. In addition, they must be simple, predictable and easy to understand in a way that favors the fact that developers can predict what is being targeted and even derive related resources. The coding format of the data exchanged between an application and the service must be standard such as the extensible markup language (XML) or JavaScript Object Notation (JSON).

Since the transfer protocol is HTTP, the error codes and exceptions of HTTP must be used, we have listed them below in Table 2.

**Table 2.** HTTP response codes

| CODE | Description |
|---|---|
| 1xx | Informative response |
| 2xx | Correct request |
| 3xx | Redirections |
| 4xx | Client errors |
| 5xx | Server errors |

## 4 The TESTAR tool

TESTAR is a tool for automated testing at the Graphical User Interface (GUI) level [2]. We will first show the philosophy of this tool to make clear how it has been adapted to test the RESTful implementation of the Smart Home. To explain the high level logical flow of TESTAR tool, it performs the following steps (also shown in Fig. 2) to test a given System Under Test (SUT) at the GUI level:

1. Start the SUT;
2. Obtain the current *State* (when testing GUIs the *State* is represented by a widget tree that is obtained through the Operating System's Accessibility API, which has the capability to detect and expose a GUI's widgets, and their corresponding properties, such as: display position, widget size, ancestor widgets, etc.);
3. Derive a set of available actions that a user could execute in a specific state of the SUT (for example: clicks, text inputs, mouse gestures);

4. Select one of the available actions (randomly or using some search-based or other type of optimization criteria);
5. Execute the selected action;
6. Apply the available online state oracles to check (in)validness of the new GUI state. If a fault is found, stop the SUT (7) and save a re-playable sequence of the test that found the fault. If not, keep on testing if more actions are desired within the test sequence.

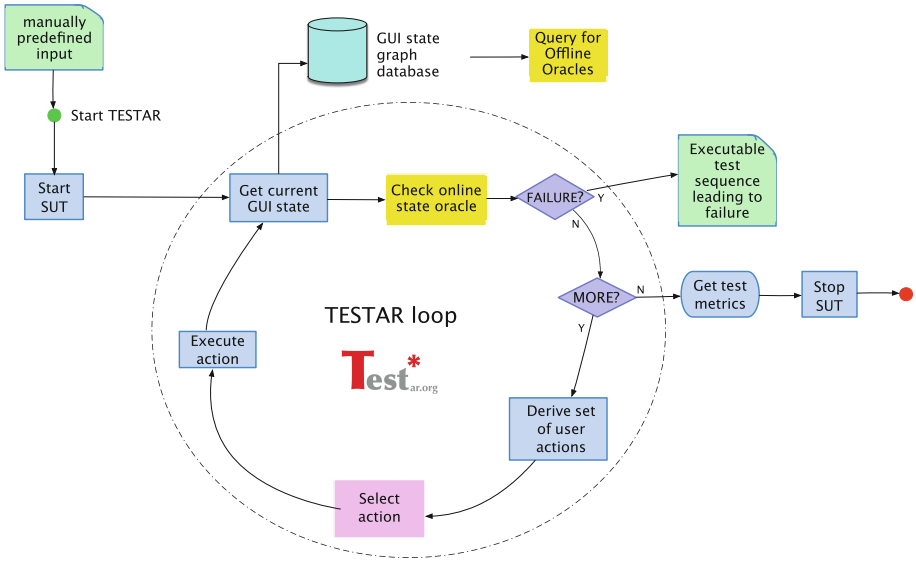


**Fig. 2.** TESTAR testing flow

The default behaviour of TESTAR includes random selection of actions and implicit oracles for the detection of the violation of general-purpose system requirements:

- the SUT should not crash,
- the SUT should not find itself in an unresponsive state (freeze), and
- the UI state should not contain any widget with suspicious titles like *error*, *problem*, *exception*, etc. The suspicious titles can be configured in TESTAR through its settings Dialog and the Suspicious titles functionality.

During testing, TESTAR stores all the information about the visited states in a Graph database. Consequently, after testing the tester can define offline oracles that query the database foe failure patterns.

TESTAR adopts the hypothesis that the majority of GUIs are conceptually very similar. The only thing that varies is the underlying technology and the look and feel. But if sufficient *State* information is available, such as: the types,

positions and properties of all widgets on the screen, then testing an iPhone App is not much different from testing a Windows desktop application or a website.

TESTAR's abstraction layer and the extensible plugin architecture (see Fig. 3) makes it highly technology agnostic. The plugins deal with the process of fetching the state information and executing actions for different platforms. The abstraction layer has a uniform interface that allows to access the UI state information in a standardised way. It allows to simulate end user input in the form of clicks, drag and drop operations, swipes, pinches, audio input, etc, in order to operate the UI. The abstraction layer abstracts from different technologies, shields other components from technological details and allows testers to concentrate on strategic parts of sequence and test suite generation.

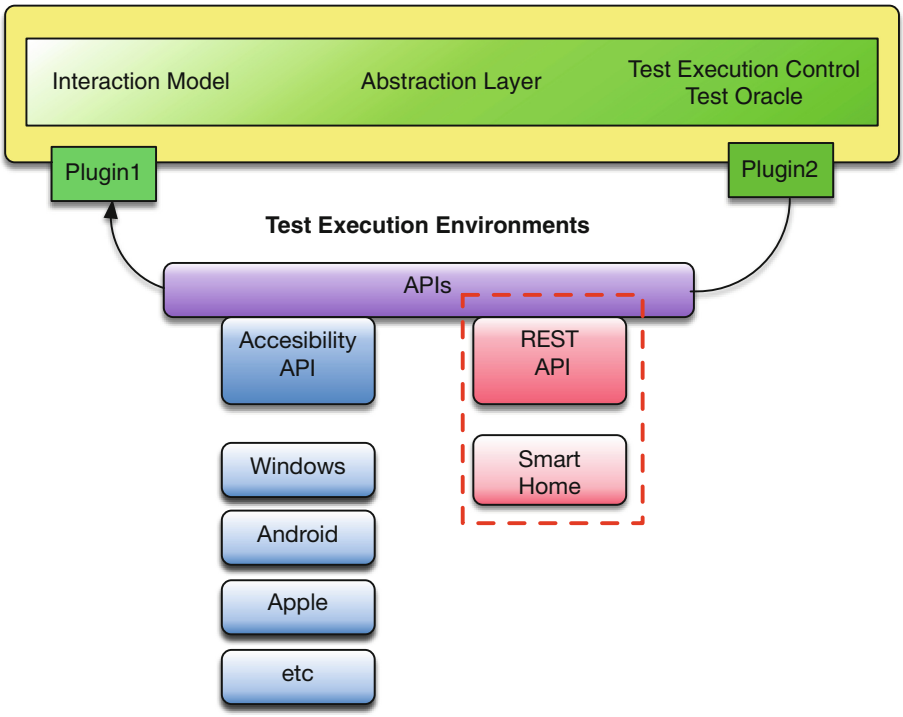


**Fig. 3.** Abstraction layer and the extensible plugin architecture of TESTAR

To implement the necessary plugins and obtain the *State* information, TESTAR needs to know which resources are available in order to interact with them. For testing at the GUI level, the Accessibility API of the Operating System provides enough information. However, when switching to IoT systems, the Accessibility API is no longer useful. In this case, we were able to use the REST API that accesses the resources in the Smart Home. Consequently, we had to develop the plugins that contained the specification how to test the interaction

with the resources of a smart home to give TESTAR the information it needs for action selection, text execution and test oracles. To illustrate the plugin architecture and the work done in this paper, the new components we implemented for TESTAR are highlighted with the dotted square in Fig. 3.

## 5 TESTAR for IoT

### 5.1 Actions

TESTAR for IoT is able to execute the four HTTP methods, discussed in Sect. 1, that are associated with CRUD operations. This means, plugin1 (from Fig. 3) consist of an Action interface for TESTAR that is implemented by Java classes GET, PUT, POST and DELETE.

### 5.2 Derive Actions for Test Execution and Oracle Verdicts

With the aim to improve the user experience and the range of possibilities of this new approach, for plugin2 we have implemented three different modes of execution together with their respective oracles (in the form of regular expressions) for each one of them. These modes have been designed taking into account the intrinsic behaviour of web services.

**Positive Tests.** For positive tests (or happy path tests), all feasible actions will be derived. An authentication token will be provided when this is required for correct execution. The default oracle for these tests are based on TESTAR Suspicious Titles functionality, that checks if the received response has a HTTP response code that corresponds to an error (4xx and 5xx, as explained in Sect. 3). If there is a match, TESTAR will consider that the current sequence found is a failure.

**Negative Tests.** For negative tests, we will derive actions that try to update resources (PUT's from the REST API) in an invalid way. By doing so, we are able to check whether invalid actions will cause any danger or incorrect behaviour. For example, in a Smart House you are not supposed to open the window when the air-conditioning is on. There is a field in the Dialog of TESTAR to indicate the expected response when those kind of actions occurred. If the server response does not matches the expected expression (expected HTTP error code), TESTAR will consider that the current sequence found a failure.

**Unauthorized Tests.** Unauthorized Tests contain actions that need authentication parameters to be correctly executed. However, they will be executed without passing the required authentication token. This allows us to check whether at some point it is possible to interact with the API without the required identification. As in the Negative Tests mode, there is a field in the Dialog of TESTAR

to specify the expected response when those kind of actions are executed. If the server response does not match the given expression (expected HTTP error code), TESTAR will consider that the current sequence found a failure.

## 6 Experiments and Results

### 6.1 Procedure

After having the plugins for the smart home developed, TESTAR had to be configured according to the needs of the System Under Test (SUT). First of all, we indicated which actions require authentication parameters. For the smart home, only the PUT method needs authentication so we indicated a valid token in order to give TESTAR the required credentials to execute those PUT actions.

After that, we filled the simple oracles that came with TESTAR taking into account the following information provided by domain experts:

- When executing actions without the required token, the appropriate response of the server must be 401 Unauthorized.
- For those actions derived in the Negative Tests mode we can expect either 400 Bad Request or 405 Method Not Allowed.
- For all other cases, we used the oracle that comes by default for the Positive Tests mode.

Since the Smart Home does not allow Delete nor Options actions, we needed to modify the protocol of TESTAR in order to let it know that we expect a 405 Method not Allowed as the response for those kind of actions. Another approach will be to tell the tool that it should not derive those actions, but we decided to test them in order to know if something unexpected happens when trying to execute them.

After all the configuration parameters were established, we proceeded to run TESTAR using the three different testing modes.

### 6.2 Results

After our firsts runs we already discovered a fault in which when trying to interact with the Gradual Light with a feasible action, the server response was Method Not Allowed instead of indicating that the resource had been successfully updated. This error appeared each time a run was executed and the selected action implied the Gradual Light Resource. After communicating this to the team in charge of implementing the smart home, they discovered that an error was made when declaring the supposed functionality for that resource. Afterwards, that fault was correctly solved.

Once more test were executed, we discovered a different kind of fault in which some resources were no longer available and a 404 Not Found was received when trying to interact with them.

Moreover, we noticed that after executing several actions and receiving a 404 error when trying to access a resource that was not available, the server stopped responding and a time out fault was found by TESTAR. However, the server seemed to establish itself when some time passed.

On the other hand, we noticed that in the Negative Tests mode, when trying to update the Gradual Light (which is a dimmer) with a toggle (an interaction characteristic of bistate resources), the given response was not as expected. Instead of a Method Not Allowed or a Bad Request Message we received an Internal Server Error.

Finally, we tried the Unauthorized Tests mode with two different setups: 5 sequences of 100 actions and 5 sequences of 300 actions. Therefore, a total of 2000 actions without the required authorization parameters were executed and the smart home behaved as expected through all of them.

## 7 Conclusions and Further Work

Testing IoT is an important an challenging task. In this work, we have presented the first approach of TESTAR for testing IoT devices. The tool has proved to be successful testing the communication with the resources provided by a smart home through a REST API. Once the plugin that specifies what kind of actions can be executed for a determine SUT was implemented, TESTAR offered an automated way of generating and executing tests.

To summarize the results, we have found a total of 4 reproducible faults among the firsts executions of the tool. One of them (The Gradual Light not accepting feasible actions) has already been successfully corrected after being communicated to the team in charge of the smart home. Meanwhile the other three will need further investigation.

Future work will involve different IoT devices accessed through a REST API in order to ensure that the same procedure can be applied. Moreover, we will try to extend TESTAR in order to support other communication protocols beyond REST API.

**Acknowledgement.** The authors would like to thank Vicente Pelechano, Urko Rueda and Francisco Almenar for their invaluable help that has made this work possible. This work was partly funded by the ITEA3 TESTOMAT project (16032).

## References

1. Bauersfeld, S., de Rojas, A., Vos, T.E.J.: Evaluating rogue user testing in industry: an experience report. In: IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), pp. 1–10, May 2014
2. Bauersfeld, S., Vos, T.: A reinforcement learning approach to automated GUI robustness testing. In: Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE 2012), pp. 7–12. IEEE (2012)

3. Bauersfeld, S., Vos, T.E.J., Condori-Fernández, N., Bagnato, A., Brosse, E.: Evaluating the TESTAR tool in an industrial case study. In: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2014, Torino, Italy, 18–19 September 2014, p. 4 (2014)
4. Business. The Internet of Things: where the smart is. Economist **419**(8993), 52–53 (2016)
5. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Using feature models for developing self-configuring smart homes. In: Fifth International Conference on Autonomic and Autonomous Systems, pp. 179–188 (2009)
6. Frizell, S.: The smarter home. Special report. Time Mag. **184**(1) (2014)
7. Grnbk, I.: Architecture for the internet of things (IoT): API and interconnect. In: Second International Conference on Sensor Technologies and Applications (SENSORCOMM 2008), pp. 802–807, August 2008
8. Hagar, J.: Experience report: a guide through the challenges and risks of testing the IoT. LogiGEAR Mag. **VII**(3) (2014)
9. Haller, S., Karnouskos, S., Schroth, C.: The Internet of Things in an enterprise context. In: Domingue, J., Fensel, D., Traverso, P. (eds.) FIS 2008. LNCS, vol. 5468, pp. 14–28. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00985-3_2
10. Almenar, F., Esparcia-Alcázar, A.I., Martínez, M., Rueda, U.: Automated testing of web applications with TESTAR. In: Sarro, F., Deb, K. (eds.) SSBSE 2016. LNCS, vol. 9962, pp. 218–223. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47106-8_15
11. Richardson, L., Ruby, S.: RESTful Web Services (2008)
12. Rueda, U., Vos, T.E.J., Almenar, F., Martínez, M.O., Esparcia-Alcázar, A.I.: TESTAR: from academic prototype towards an industry-ready tool for automated testing at the user interface level. In: Canos, J.H., Gonzalez Harbour, M. (eds.) Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015), pp. 236–245 (2015)
13. Tnjes, R., Reetz, E.S., Moessner, K., Barnaghi, P.M.: A test-driven approach for life cycle management of internet of things enabled services. In: Future Network Mobile Summit (FutureNetw 2012), pp. 1–8 (2012)
14. van der Mersch, V.: Automated testing for the Internet of Things. Nordic APIs Blog, May 2016
15. van der Meulen, R.: Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. Press release, November 2015
16. Vos, T.E.J., Kruse, P.M., Condori-Fernández, N., Bauersfeld, S., Wegener, J.: TESTAR: tool support for test automation at the user interface level. IJISMD **6**(3), 46–83 (2015)
17. Zhang, Z.K., Cho, M.C.Y., Wang, C.W., Hsu, C.W., Chen, C.K., Shieh, S.: IoT security: ongoing challenges and research opportunities. In: IEEE 7th International Conference on Service-Oriented Computing and Applications, pp. 230–234, November 2014