

Using Policy Refinement to Assist IoT Device Management in Smart Hospitals

Jessica M. de Castro(✉), Luiz Fernando F. P. de Lima(✉), Iury Araújo(✉), Eudisley G. dos Anjos(✉), and Fernando M. Matos(✉)

Federal University of Paraiba, João Pessoa, Brazil
luizfernando@cc.ci.ufpb.br,
{jessicamaciel,iuryrogerio}@ppgi.ci.ufpb.br,
{eudisley,fernando}@ci.ufpb.br

Abstract. The implementation of the Internet of Things paradigm in the most diverse environments is already a reality. One of these environments is the smart hospital that uses several devices to improve patient care and streamline the internal processes of hospitals. However, an obstacle in the use of IoT systems is the difficulty encountered by their user to change devices behaviours as their needs changes. When the users can not update, the system may be outdated. This leads to problems in the hospital which is a critical environment and needs everything to be working properly every time. A way to customize the behavior of IoT devices is the use of police that can be described in high-level programming language. For this reason, this paper proposes a solution that uses policy refinement to assist IoT device management in smart hospitals. The proposed solution was implemented and tested, achieving favorable results for the hospital scenario.

Keywords: Internet of Things · Smart hospital · Policy management

1 Introduction

The Internet of Things (IoT) presents a set of devices connected to a network and interaction with each other. Those devices can be sensors, actuators, everyday objects, as refrigerators, smartphones, and so on. They have the ability to interconnect without human intervention and to obey pre-defined rules [1]. The IoT paradigm can be implemented in several domains seeking to improve the lives of its users. Some examples of environments are smart houses, smart offices and smart hospitals [2,3]. The smart environments are being implemented with a variety of functionalities, such as monitoring, recognition and smart management. The health care area can benefit from the implementation of IoT using the smart services that integrate the function of diagnosis, decision, management and treatment [4]. For this, sensors can be embedded in patients, employees, devices, environments in order to monitor, act and alert [5].

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 645–657, 2019.
https://doi.org/10.1007/978-3-030-24308-1_51

For the smart environment deliver the functionality for its users, its devices need to be configured to act as the context requires. In addition, if there are changes, it is necessary to reconfigure the devices so they can act according to the new rules of the environment. However, reconfiguring these devices can be a difficult task to lay people without programming experience [6]. Since users can not make the changes, it is necessary to request the assistance of a technician. This may lead to delays and/or disruptions in environmental services. Another problem caused by the dependence in specialized professionals to configure and reconfigure the devices is the hesitation to adopt the technology. Users are afraid to adopt a technology they do not fully understand or dominate [7].

Hospitals are critical environments. Thus, it is important that all systems work with the minimum of complications, including IoT systems [8]. Thereby, it is necessary to provide tools that facilitate the redefinitions of IoT devices behavior by users. One approach used to address this need is the use of policies. A policy is the representation of one or more rules that determine an action from an event. They are commonly represented in Event-Condition-Action (ECA) format [9] and can be used to dictate the behavior of IoT devices. The use of policy allows users to create and manage their own policies with more facilities instead of having to program each device in the environment [11]. And it brings more flexibility to the system and ensures that the behavior of the devices can be changed whenever necessary [9]. With this approach it is possible to provide tools so that users can create their own policies in a high level programming language. It can provide tools such as applications and users to create their policies [11]. As well as platforms across the most diverse visual language types to make it easier for the user to manage IoT systems and their devices [6].

However, to provide the creation of high level programming language policies it is necessary to refine the policy created by the user from a high to a low level of abstraction, so it can be enforced in IoT devices. It is important to ensure the consistency throughout the process. That is, it is necessary realize the translation as well as others requirements, such as policy validation according to the used resources [10]. Furthermore, in a smart hospital it is important to ensure that the policy follows the standards set by hospital and national and international health regulatory agencies. The standards can establish the temperature and humidity conditions inside the hospital rooms [17], medicine refrigerator temperature [18], illumination [19], among others.

This paper presents a solution to assist in the management of IoT devices using policy refinement in smart hospitals. It aims to provide to the IoT systems users with the autonomy they need to manage devices according to context changes. For this, it performs the policy translation from XML into a configuration file that can be executed by Middleware (MW). So, the Middleware uses the file to configure the behavior of IoT Devices. The solution also ensures policy validation in terms of hospitals standards and existing resources. The solution was implemented and validated with efficiency tests.

The rest of this article is organized as follows. Section 2 presents the related works. Section 3 describes the proposed solution and its operation. The Sect. 4

details the implementation and results achieved. Finally, Sect. 5 presents the final considerations of the article.

2 Background

In [11] the authors present about a IoT device programmable language for home automation. They show the architecture developed for this as well as the tests that were performed. Although they do not use the term policy refinement, they do the translation and validation of the policy, verifying that the devices used are available and can perform the required tasks. As it is a proposal to be deployed in a residence, they use a centralized architecture. In [7] a flow language for assisted monitoring is presented in order to facilitate the programming of the devices in houses. However, the presented language is not accessible for lay users, which makes it impossible to be used in more complex scenarios, such as a hospital. As the papers in the literature are generally focused on IoT system applications for residential environments, and in this context it is not necessary to obey standards, they do not present validation of standards.

In work [12] is presented a performance management model for IoT as a service for e-health caregivers. The model is based on low complexity policies, they are implemented as a program code that runs on the gateway and are inspired by an approach that is constructed as event-condition-action. The work is focused on constructing policies to manage the performances of devices and not on customizing their behavior.

The paper [10] presents a Survey on methods of policy refinement for sustainable networks. Even though the work focuses on refinement for the scenario of computer networks, it served as the basis for defining the necessary requirements in a refinement method that could be implemented in the IoT devices scenario. From the requirements presented by the authors, we consider relevant for the scenario of this work the following: Translation, Resource, Verification/Coverage and Conflict. Besides these, with the survey of requirements the need for a new

Table 1. Requirements summary.

Requirements	Summary
Translation	Should have the steps that ensures the translation of a high level policy to a low level
Resource	During the transformation process it is necessary to ensure that the resources in the IoT system environment are taken into account
Coverage	Verify if the refined policies meet the requirements of the original policy
Conflict	Deal with conflicts, detecting and resolving them
Standard	During the transformation process it is necessary to ensure that the hospital standards are taken into account

requirement for hospital application was identified. The new requirement is Standards. The explanation of all these requirements can be found in the Table 1. The Fig. 1 shows the requirement and those that are circled are those that were considered in the development of this work (Standard, Translate and Resource).



Fig. 1. Refinement requirements

3 Solution Proposed

The proposed solution aims to assist in the management of IoT devices using policy refinement in smart hospitals. So, the Fig. 2 presents the solution architecture integrated to a IoT Framework. The Policy Refiner and Data Manager modules are part of the proposal and Front-end, Middleware and IoT Devices of

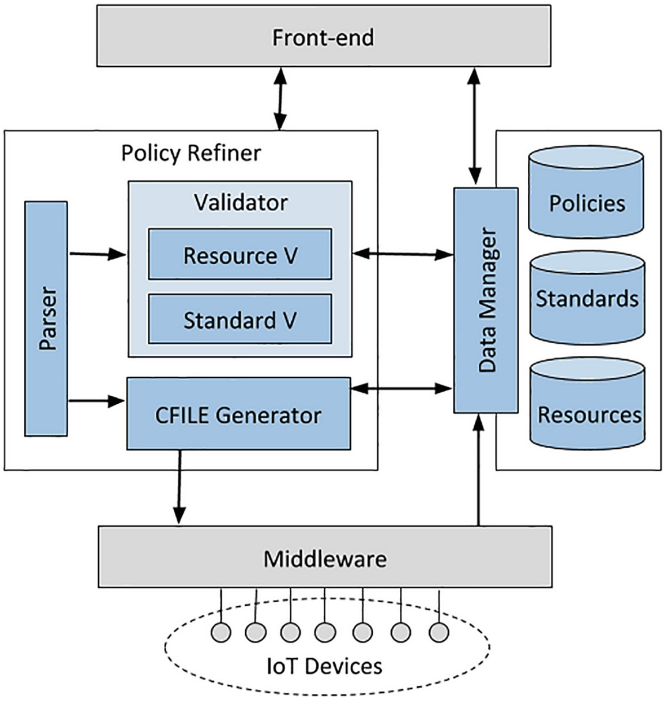


Fig. 2. Solution architecture

the framework. The Front-end module is responsible for presenting the platform to the user. He will use it to create, edit and view existing policies. The fronte-end will perform the transformation of the policy from a visual language to an XML file.

The Policy Refiner module is responsible for performing the refinement process of the policy described in XML for a configuration file that will be used to customize the behavior of the devices. The Parser module receives the file and parse of all the information using a object-oriented programming language. Moreover, the parse tree is created and contains the condition defined in the policy. Such elements will be used by the validator and the Configuration File (CFILE) Generator. Validator is responsible for the validation of resources and standards. The Resource Validator is responsible for ensuring that the IoT devices, which are being used to perform the event capture or the resulting action, exist and are available for use. So, the role of the Policy Validator is to verify that the policy complies with the hospital standards. The CFILE Generator uses the information generated by the Parser to create the configuration in a language that can be executed by the Middleware.

The Data Manager is in charge of managing the data that is inserted or requested to the databases (DBs). There are three DBs: Policies, Standards and Resources. The first one stores the policies that have been submitted by the users and have already been validated and accepted. The policies are stored in XML format because if users need to view or change them, they can be easily converted to the visual programming language that the users uses to create the policies. The second maintains the hospital standards. For example: which category of patient can enter what room or what is the minimum and maximum temperature allowed in the hospital rooms. The third one is used to store the information of all the IoT resources. Such as which devices are available at the hospital, their location, and what services they offer. Furthermore, the Data manager is also responsible for organizing and inserting into the database the new resources discovered by the Middleware.

And last, there is the MW. It has the function to coordinate the existing IoT devices in the hospital. So, its receive the policies configuration file and when an event happens the MW interprets the corresponding policy and define the behavior of the responsible devices, as the one presented in [9]. And the IoT devices are the sensors and actuators that act in the hospital environment, such as lights, sensors of vital signs, luminosity, temperature, among others.

So, the policy refinement consists of the entire policy transformation process presented in this section, from its defined form in XML file to the configuration file that will be used by the MW configure the devices. It is guaranteed that the policy can dictate the behavior of the hospital's IoT devices by ensuring that the translation is performed correctly, maintaining all its initial characteristics. As well as ensuring that it complies with all hospital standards and that the policy will be executed without error by ensuring that all IoT devices required for its implementation exist.

3.1 Solution Operation

The solution uses refinement to from high level policies generate low level policies. For this, it is necessary to ensure that the policy can be translated correctly, assuring that all its initial characteristic. Also, during the refinement process it is important to take into consideration the hospital standards and existing resources in the environment. Figure 3 shows the operating scheme of the solution. Demonstrating the process since the submission of the policy by the user to the configuration of the IoT system and its devices.

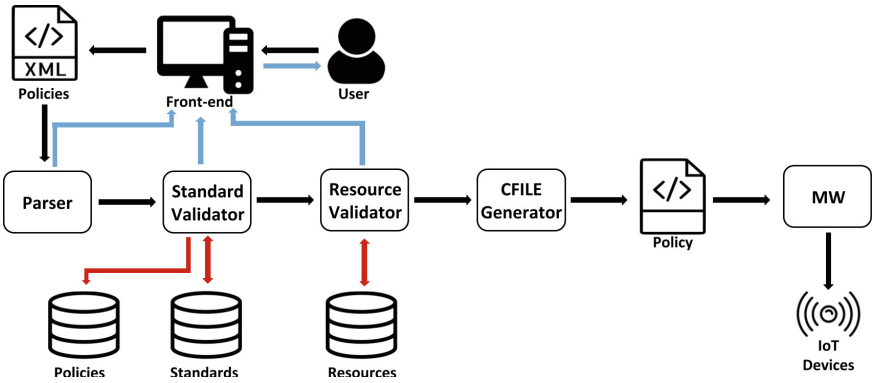


Fig. 3. Solution operation

When the user creates a new policy or edited an existing one, the policy in visual programming language is submitted and the front-end translates it to a XML file. The XML contains all the policy information and also of the user that submitted it. This file is the input artifact for the beginning of the policy refinement process. So, the Parser parses the policy in XML format for a programming language object and creates the parser tree. The parser tree will be used for the validation process and creating the configuration file. If any error is found, the information is returned to the user.

When the parser is completed successfully, the process of validating the policy is initiated. First, it is verified that the policy is in accordance with the hospital standards. To do this, the Standard Validator requests the Data Manager the current standards of the hospital. It is then checked if the submitted policy is in accordance with the standards. If it is, the process continues. If not, an error message is returned to the user. After this, the resource validation is done. So, it is necessary to check whether the services and devices used to capture the event, check the condition and perform the action exist in the hospital and have the ability to perform what was required. Therefore, the Resource Validator requests information from the Data Manager about the services and devices available in the hospital and their location in the hospital. The Data Manager returns the requested information to the Resource Validator. Then it is checked if there are

the resources for performing the event and the action in the room defined in the policy. If it does not exist, an error message is sent to the user.

Once the policy validation is successfully completed, the XML file is saved in the Policy DB. Now it is necessary to complete the translation process for a language that will be easier to be processed by MW. Therefore, the Configuration File Generator uses the policy object and also the parser tree to construct the file in a programming language that can be interpreted by the MW to determine the behavior of the devices. This file contains the information of the sensors and actuators used to execute the policy, such as the event and the condition and action. The file is then sent to the MW so that it can manage and order the behavior of the devices contained in the hospital network.

4 Prototyping and Results

The policy refinement process was implemented to prove that the proposed solution can be applied in the scenario of a smart hospital. This section presents the implementation process, the tests and results obtained from the execution and simulation of the proposal. For this, the programming language Python, version 3.x was used. The choice of the use of this language was due to some advantages that it provides, such as rapid development, easy integration with new modules, even other languages and also because it is an interpreted language that has portability for several operating systems (Windows, Linux and Windows). Moreover, Python has been used as a programming language for projects in the context of IoT [13–15]. The use of Python as a programming language influenced how the data was represented in the solution. The DBs were implemented using CSV files, because the Pandas library was created for manipulation and data processing in Python and is a very intuitive tool for this purpose [16].

The first step was to define the XML policy file. This is, what information the file would contain and how they would be arranged. An example of the file can be seen in Fig. 4. The XML file is organized as follows: Line 3 to 6 presents general information about the policy. In line 3 has the policy identification number (ID) and name. Lines 4 and 5 are about the author and present the author ID and the id of author's department, respectively. The policy event is described between line 7 and 10. The type of the event is shown in Line 7. Lines 8 and 9 bring the local that the policy will be applied, the department and room, respectively. Line 10 describes if any medical equipment, such as ECG, it is necessary in policy implementation. Lines 12 to 17 present the condition for applying the policy. It condition element can have a several operation elements where is described in 'op' what type of operation it has to perform and inside it have the two value that will be applied the operation. Lastly, the action is defined between lines 18 and 20. Line 18 shows the action that will be performed if the condition is positive. The file presented in Fig. 4 represents a policy that by identifying that there is a new person with a patient band (Line 7) on the door of a specific room (Line 9), that band is read and the category of that patient is checked. If it is orange or red (lines 14 and 15), the door will open for him (Line 19).

```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
    <info id="1" name="Porta 100">
        <id_author>15</id_author>
        <id_dep>1</id_dep>
    </info>
    <event type="new_band">
        <at_dep>1</at_dep>
        <at>100</at>
        <equip>None</equip>
    </event>
    <condition>
        <operation op="or">
            <operation op="equal">band;orange</operation>
            <operation op="equal">band;red</operation>
        </operation>
    </condition>
    <action>
        <do>open_door</do>
    </action>
</policy>
```

Fig. 4. Policy in XML example

With the policies xml template ready, the Parser was implemented. The cElementTree library was used to read the XML file. It creates a tree with all elements of the document. The parser is then responsible for finding and mapping all the data contained in the XML to its corresponding variable in the policy class. Furthermore, there is also the creation of the parser tree, in this step the operations contained in the condition (for example: Fig. 4, lines 12–16), as well as the values, are identified and the parser tree is built.

Once the file has been parsed, it is necessary to validate the policy. Before checking the standards and resources, it is important to check the user information contained in the policy. So, the existence of the room defined in the policy is checked and it is checked if the employee can create a policy for that location.

Then, it made the validation of standards. It is verified what type of event, condition and action is described in the policy. After this is checked if there is any current standard that applies in politics. If there is, it is necessary to check the condition of the policy to verify that the policy can be applied. These information checks and norms are sequential, so if the policy is not validated in any of them, the process is interrupted and an error message that describes the reason for the validation failure is sent to the user. One example of standard is that the temperature of the medicine refrigerator should remain between 2 °C and 8 °C [18]. Therefore, if a policy is created for this equipment, the validator checks if this standard is met, that is, if the user attempts to implement a temperature outside that range for that equipment, the policy will not be validated and an error message will be displayed.

So, it is need to ensure that the resources used exist in the defined location. There are two steps to do this. The first is to verify that the event defined in the policy can be performed at the desired location and the second is whether the action can be performed or not. To do this, it is verified which services are required to perform the event and the defined action. Having the service information, a search is performed on the resources DB to check if there is any device that performs such events on the site. If it is found, then it follows the process to the data manager, but if not, a message is returned to the user stating the error. For example, for an event that identifies the presence of a new band (bracelet), it is necessary to have a sensor that can read the patient band. Or, to perform an action that opens the door, you must have an automatic door that can perform the action.

After verifying that the policy does not violate any hospital standard and that all resources used exist, a configuration file as shown in Fig. 5 is generated. It shows the information about the sensor identifier number (line 1) and actuator (line 2) that are used to execute the policy. Line 4 shows when this policy will be verified, then when the described event occurs. Line 5 presents the condition and Line 6 the action that will be performed if the condition is positive. These last two lines are written in the Python language and can be easily executed.

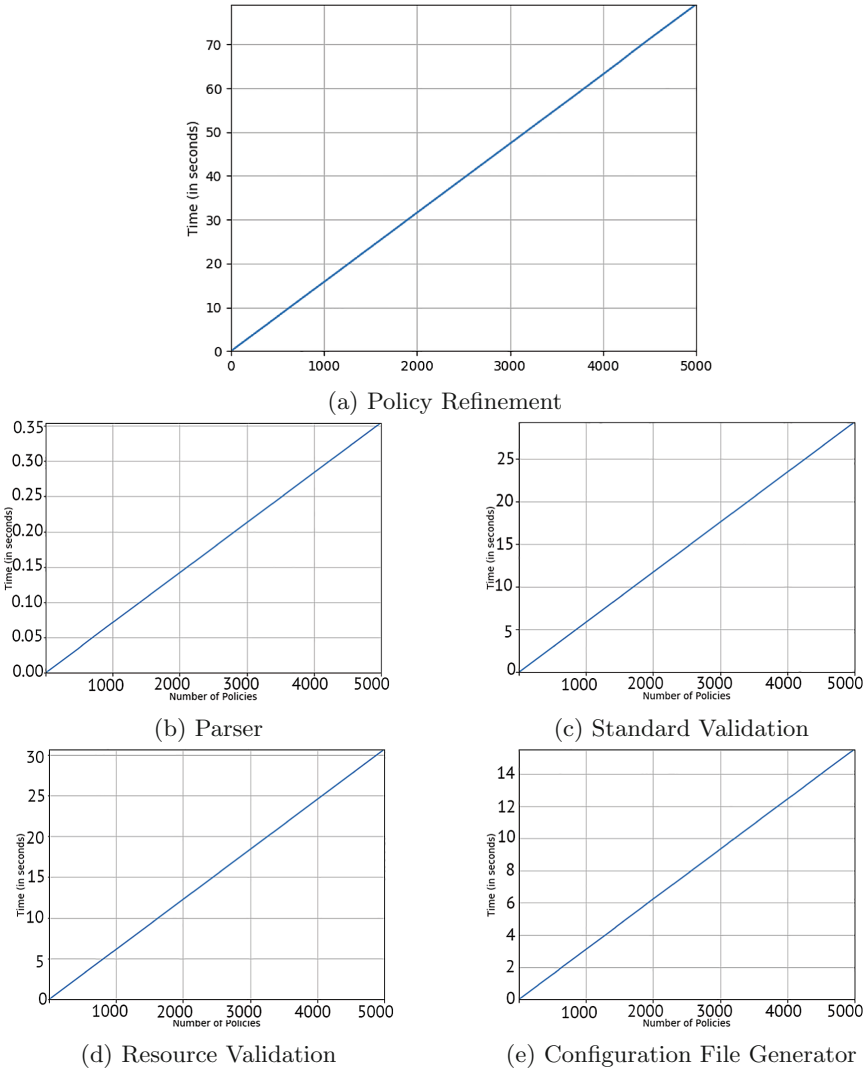
```
1  SENSOR_ID: 3
2  ACTUATOR_ID: 2
3
4  when new_band
5  if (( band equal orange ) or ( band equal red )):
6      answer = 'open_door'
```

Fig. 5. Configuration file example

4.1 Results

Based on the work of [11] two experiments were performed: (i) validation of the policy refinement process; (ii) performance test focused on the response time of the proposal. The tests were performed on a Desktop computer with an Intel i5 processor and 8 GB RAM, as it is a machine with processing power that a hospital can easily have. To test the translation, validation of resources and standards, the implemented solution was executed and the output was analyzed. All the policies and situations that were envisaged during the proposal for an intelligent hospital were described in a XML file and later translated into the configuration file that contains the executable code, succeeding in policy coverage and in all implemented translation steps.

To test the standards and resource validation, situations where the submitted policy violated hospital standards were tested. During the execution of these policies, the solution successfully identified the violations and returned the error by showing which standard was infringed. The situations in which the resource



(a) Policy Refinement

(b) Parser

(c) Standard Validation

(d) Resource Validation

(e) Configuration File Generator

Fig. 6. Policy refinement test graphics

described in the policy was not existing was also tested. And, just as validation of standards, the solution was able to correctly identify all policies that use non-existent resources and returned the error message pointed out which was the resource that was unavailable.

Moreover, the time needed for policy refinement was verified as well as for each separate step. For this, the policies were executed 30 times to calculate the average time used and to have a fair result. The policy refinement was performed with 1 policy to 5000 policies to ensure the scalability of the proposal. The Fig. 6

shows the graphs of the test results. The first one, Fig. 6a show the time required to run the policy refinement from the input of the XML file to the output of the configuration file. The other figures are from each step of the refinement process: Parser (Fig. 6b), Standard Validation (Fig. 6c), Resource Validation (Fig. 6d) and Configuration File Generator (Fig. 6e). The total time for policy refinement was 5000 policies was 79.15 s, that is, the time required for a policy was only 0.0158 s. Analyzing the total time of the process, it is possible to verify that the time used by the parser is only 0.44% of the whole process. While standards and resources validation spend 37.04% and 38.77%, respectively. This is due to access to standards and resources databases. Lastly, the file generation process consumes 19.63% of the process, as it is also necessary to access databases, writing and reading.

Therefore, the results obtained in the tests of 0.0158 s to perform the entire refinement process for a policy are satisfactory. We can notice that the return time of success or error message to the user will not be time consuming and the policies can be modified as soon as the user needs. Based on the results, the solution proposed in this article can be implemented in a critical environment, such as the hospital. These results ensure that the hospital is always functioning properly without major problems. In addition to the low execution time, we have the guarantee of the successful refinement process. This allows the user to perform policy creation at a more abstract level, such as in a visual programming language. And by doing this, it ensures that users are able to create policies that will be used to reconfigure the IoT devices in the hospital. This ensures that the hospital is always functioning perfectly and that the system is not out of date.

5 Conclusion

With the evolution of technology and the implantation of the IoT paradigm and smart environments, it is important to develop tools that allow users to customize their IoT devices. Providing these tools to users, they will be more confident and will be easier to them to adopt the technologies. Aiming it, This article presented a solution that facilitates the customization of IoT devices in smart hospitals. For this, we used policy refinement to ensure a user-created policy will be translated correctly and validated according to existing resources and hospital standards. With the implementation of the proposal it was possible to verify that it works and can accomplish what was proposed. The results were satisfactory and showed that the solution can be implemented in a hospital. Thus, it is expected that in conjunction with the front-end application, hospital employees can manage their own policies without the need to request a technician, ensuring that the system is always functioning correctly and as expected. As future work, we hope to develop Front End and a visual programming language so that users can create their policies. We also want to implement a hospital scenario in which tests with real devices can be performed.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Kafle, V.P., Fukushima, Y., Harai, H.: Internet of things standardization in ITU and prospective networking technologies. IEEE Commun. Mag. **54**(9), 43–49 (2016)
3. Ahmed, E., Yaqoob, I., Gani, A., Imran, M., Guizani, M.: Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. IEEE Wirel. Commun. **23**(5), 10–16 (2016)
4. Dhariwal, K., Mehta, A.: Architecture and plan of smart hospital based on Internet of Things (IOT). Int. Res. J. Eng. Technol. **4**, 1976–1980 (2017). e-ISSN 2395-0056
5. Barroca Filho, I.M., de Aquino Junior, G.S.: Proposing an IoT-based healthcare platform to integrate patients, physicians and ambulance services. In: Gervasi, O., et al. (eds.) ICCSA 2017. LNCS, vol. 10409, pp. 188–202. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62407-5_13
6. Reisinger, M.R., Schrammel, J., Fröhlich, P.: Visual languages for smart spaces : end-user programming between data-flow and form-filling, pp. 165–169 (2017)
7. Edgcomb, A.D., Vahid, F.: MNFL: the monitoring and notification flow language for assistive monitoring. In: Proceedings of the 2nd ACM SIGHIT Symposium International Health Informatics, IHI 2012, pp. 191–200 (2012). https://doi.org/10.1145/2110363.2110387
8. Ullah, M., Fiedler, M., Wac, K.: On the ambiguity of quality of service and quality of experience requirements for eHealth services. In: 2012 6th International Symposium Medical Information and Communication Technology, pp. 1–4 (2012). https://doi.org/10.1109/ISMICT.2012.6203030
9. Singh, J., Bacon, J.M.: On middleware for emerging health services. J. Internet Serv. Appl. **5**, 1–19 (2014). https://doi.org/10.1186/1869-0238-5-6
10. Riekstin, A.C., Januario, G.C., Rodrigues, B.B., Nascimento, V.T., de Brito Carvalho, T.C.M., Meirosu, C.: A survey of policy refinement methods as a support for sustainable networks. IEEE Commun. Surv. Tutorials. **18**, 222–235 (2016). https://doi.org/10.1109/COMST.2015.2463811
11. Ponte, F.R.P., Gomes, R.L.: IoT device programmable language customization for home automation. In: IEEE Symposium on Computers and Communications, pp. 168–173 (2018)
12. Ha, M., Lindh, T.: Distributed performance management of Internet of Things as a service for caregivers. In: 2017 IEEE 19th International Conference e-Health Networking, Applications and Services, pp. 1–6 (2017). https://doi.org/10.1109/HealthCom.2017.8210765
13. Patchava, V., Kandala, H.B., and Babu, P.R.: A smart home automation technique with Raspberry PI using IoT. In: International Conference on Smart Sensors and Systems. IEEE (2015)
14. Shete, R., Agrawal, S.: IoT based urban climate monitoring using Raspberry PI. In: International Conference on Communication and Signal Processing. IEEE (2016)
15. Quadri, S.A.I., Sathish, P.: IoT based home automation and surveillance system. In: International Conference on Intelligent Computing and Control Systems. IEEE (2017)
16. McKinney, W.: Python for Data Analysis: Data Wrangling with Pandas, Numpy, and Ipython, 1st edn. O'Reilly, Sebastopol (2012)

17. Anvisa: RESOLUTION-RE N° 9. http://portal.anvisa.gov.br/documents/33880/2568070/RE_09_2003.pdf
18. Ebserh: Controle de Temperatura da Geladeira de Medicamento. http://www2.ebserh.gov.br/documents/147715/0/ROP_controle_temp_geladeira.pdf
19. ABNT: Iluminancia de interiores. http://ftp.demec.ufpr.br/disciplinas/TM802/NBR5413.pdf