# A Methodology for Classification and Evaluation of IoT Brokers

Eddas Bertrand-Martínez*, Phelipe Feio*, Vagner Nascimento *, Billy Pinheiro* and Antônio Abelém*
*Federal University of Pará
Belém, Pará, Brazil
{eddasjbertrand, phelipefeio09}@gmail.com, {vagner, billy, abelem}@ufpa.br

***Abstract*—Since the term Internet of Things (IoT) was coined by Kevin Ashton on 1999, a bundle of middleware platforms has been developed to cope with important challenges such as the integration of different technologies. Is in this context of heterogeneous technologies that IoT message brokers become key elements for the proper function of smart systems and wireless sensor networks (WSN) infrastructures. This article proposes a methodology for classification and evaluation of brokers by using qualitative analysis, so to help in the selection of the more suitable brokers according to the given scenario and needs. The methodology uses the quality reference model described on the ISO/IEC 25010 normative from the SQuaRE set of standards published by the ISO/IEC conjunction. In the implementation case we developed the proposal with 9 different open source brokers so to validate the applicability and feasibility of our methodology.**

***Index Terms*—Internet of Things, message-oriented middleware, broker, quality, benchmarking, methodology**

## I. Introduction

As the Internet of Things (IoT) has gained an increasing attention during the last decade, a national plan has been implemented on Brazil for adopting public policies on its research since 2017 [1]. Initiatives such as InterSCity [2], FIoT [3], SmartMetropolis on the city of Natal [4] [5] or smart campuses as the implemented on the State University of Campinas [6] reflect the interest of the giant of South America and its public institutions on the development of smart environments and smart cities as those efforts have also been made by countries such as China [7], Hong Kong [8], United States [9] and various countries in Europe among others [10].

This worldwide interest has been followed by the development of middleware platforms with the intent of integrating heterogeneous technologies. As a communication technology, protocols such as MQTT[1], CoAP[2], AMQP[3], STOMP[4] and HTTP[5] are used for exchanging messages on middleware platforms and inside wireless sensor network (WSN) infrastructures as well [11].

Message brokers, a central component present in middleware platforms, are the main responsible for exchanging data among network nodes, so be them sensors, actuators, services or even other platforms. They are in charge of publishing information from network nodes and making it available to see for other nodes. This kind of communication model is known as the publish/subscribe pattern, in which sensors and actuators act as publishers and subscribers, respectively, and application users can act as publishers or subscribers depending on the logic of each application [12].

According to the requirements of each application, features such as quality of service (QoS), robustness, performance, adaptability and others can be fundamental when selecting an IoT broker. Due to the great number of alternatives offered, it can become a daunting task to choose the most suitable brokers depending on the presented scenario.

With the purpose of facilitating a guide to choose among different IoT broker options, we propose a methodology to evaluate them and classify them. This methodology takes some considerations from the ISO/IEC 25000 family of standards for Systems and software Quality Requirements and Evaluation (SQuaRE), specifically the ISO/IEC 25010 norm which describes a quality reference model. In addition, our methodology proposes a set of steps to be followed as for choosing the brokers to be benchmarked, defining the requirement specifications according ISO/IEC 25010 norm, so to evaluate and classify the brokers.

The remainder of this paper is structured as follows: Section II presents a shallow overview of theoretical concepts concerning the proposal such as the definition of Internet of Things, a generic architecture for it and why the topic of brokers is relevant in this commission. Then Section IV describes our proposed methodology, which is complemented by Section V where an implementation case is unfold. Finally, Section VI closes the paper with some conclusions for this work.

## II. Internet Of Things and Brokers

The Internet of Things is a paradigm that describes a complex interaction between the physical and the virtual world where ordinary things like televisions, cars, thermostats and other objects can communicate with people, with other things or with services. Fig. 1 depicts different possibilities for IoT scenarios and how a digital ecosystem can be created integrating a plethora of domains.

As a consequence, one of the major challenges to address and likewise one of the majorly disputed topics is the establishment of a standardized architecture which encompasses all

[1]MQTT - Message Queueing Telemetry Transport
[2]CoAP - Constrained Applications Protocol
[3]AMQP - Advanced Message Queueing Protocol
[4]STOMP - Streaming Text Oriented Messaging Protocol
[5]HTTP - Hypertext Transfer Protocol

978-3-903176-23-2 © 2019 IFIP

Fig. 1. IoT interaction domains
Source: [12]

the required layers for smart applications and can be adapted to different scenarios and domains, therefore it is mandatory for it to be versatile enough. A considerable number of architecture proposals include models consisting of three or five layers [13], which are equivalent in addressing the same needs but differ on the level of granularity.

In Fig. 2 we illustrate a widely used reference architecture consisting of three layers, namely, *perception layer*, *network layer*, and *application layer*. The perception layer is the one responsible of capturing data and dealing with aspects of the physical world including the transmission of such captured data for it to be abstracted by the network layer and have it processed into valuable information at the application layer.

The aforementioned abstraction is mainly done by the network layer, where the IoT middleware platforms integrate all the data received by different sources and big data processes begin. A particularly important type of middleware due to its extensive use on distributed systems and in web services is the Message Oriented Middleware (MOM). The MOMs are event-based middleware, i.e., they act according to the received messages [14].

Brokers are the principal agent on the MOMs implementation, they are in charge of the dissemination of data between nodes, where Fig. 3 illustrates the generic mechanism for them. They are based on the publication-subscription model which obeys the principle of providing information only for those components that have previously subscribed to a particular data type. The usage of brokers leads to an efficient data manipulation and a more scalable architecture, at the expense of adding a degree of overall complexity. Some examples of brokers for IoT are ActiveMQ, RabbitMQ, Mosquitto, ZeroMQ, Orion Context Broker and YAMI4 among others.

## III. Related Work

There are works that evaluate IoT brokers, in fact in most proposals concerning new mechanisms for optimization, enhancement on security or so on, a benchmarking is done to ensure that the proposed solution does not compromise performance. Efforts as the ones made on [15], [2], [16] and [17] propose middleware that was subject to performance analysis, thus proving the performance improvement and benefit for IoT applications. On [18], [19] and [20], the authors proposed mechanisms for data dissemination, clustering and update, respectively, where they evaluated brokers performance to ensure the feasibility of these new mechanisms. However all these works were focused on proposing new middleware implementations or improvement, the performance evaluation was just a tool to reassure the contributions.

The work done on [21] and [22] both benchmark diverse publicly available MQTT brokers. The first one measures subscription throughput (productivity) and the time it takes to send messages from the broker (responsiveness), both aspects relate to the speed of the broker. The second work measures productivity and responsiveness as well, but adds resource utilization evaluation. These works emphasize the importance on comparing MQTT brokers, but there are aspects such as availability or reliability that could be analyzed too and are not considered. Although each benchmarking must be designed according to different needs and goals, having no necessity to evaluate every possible aspect, it is important for IT professionals and researchers to know which aspects are not being evaluated and why.

Finally the contributions made on [23] and [24] mention the importance of analyzing both qualitative and quantitative aspects for middleware solutions. The first work mentions functional aspects that are crucial for MOM to be chosen as ideal solutions, i.e., messaging pattern they use, filtering techniques, QoS semantics, etc. The second work propose

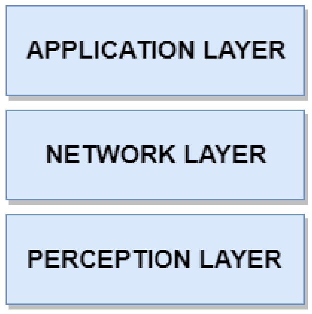

Fig. 2. IoT reference architecture
Source: [12]

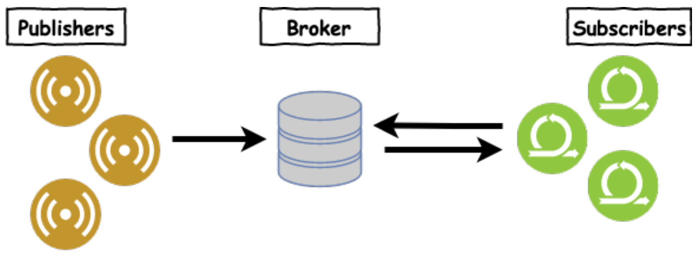

Fig. 3. IoT broker communication model
Source: Created by the Author

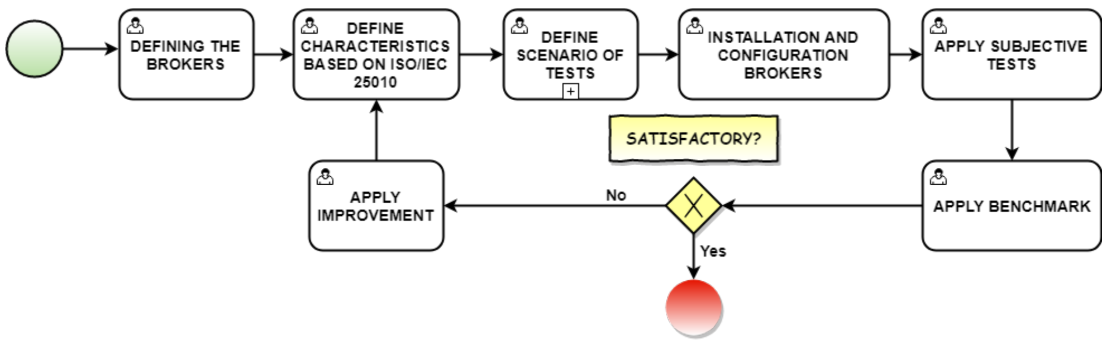

Fig. 4. Methodological process
Source: Created by the Author

non-functional aspects to be considered such as availability and clarity of technical documentation so to assure support continuity, also the compliance of the middleware platforms to follow IoT-A reference model architecture for IoT solutions. They both mention performance metrics and unfold comparison cases, but as happens too with previous works, they do not propose a formal methodology or framework so to permit researchers to implement different comparisons.

Under this context is that this paper aims to set a base for elaborating a methodological and systematic process for performing broker evaluations according to the stakeholders needs, at the end simplifying the process of choosing among diverse IoT broker options.

## IV. Evaluation and Classification Methodology

To structure our methodology we defined a logical process in which the steps are to be followed sequentially, but taking in consideration that in any part of the process the performer may return to a previous stage to reconfigure the process as needed. Fig. 4 illustrates the process to be followed. It is valid to state that the application of this process is continuous and can be reapplied as the tendencies on the technology industry change and so IoT brokers evolve. The steps and the description of each are as follows:

1) **Choose the brokers to be evaluated:** The first step is to choose the brokers to be evaluated. In this step some basic selection criteria can be established, such as communication protocol(s) supported, operating system(s) in which they run, open source or closed source, etc.]
2) **Choose features to be evaluated according to the ISO/IEC 25010 quality reference model:** In this step, the characteristics to be evaluated are chosen taking as a base the ISO/IEC 25010 software product quality reference model [25], shown on Fig. 5. This model describes a set of categories and sub-categories, and this last ones could be as well defined by metrics. The available characteristics are:
   - Functional Suitability;
   - Performance Efficiency;
   - Compatibility;
   - Usability;
   - Reliability;
   - Security;
   - Maintainability;
   - Portability.
3) **Define the test scenarios:** This step aims to create test scenarios for the previously selected characteristics and its corresponding sub-characteristics, in this way we ensure a complete test coverage. As a general consideration, each test scenario proposed should be bound to, at least, one of the sub-characteristics and it should be measured by metrics. It is also important to note that before creating test scenarios that verify more than one requirement at once, it is mandatory to ensure that there already exist test scenarios for each sub-characteristic in isolation.
4) **Installation of brokers on a controlled environment:** The deployment and installation of the brokers on a controlled environment is a primary way to prove the features that, normally, the software documentation should describe. Here, characteristics like compatibility, portability and usability could be initially evaluated as they are possible aspects to note during the configuration and installation phase. Ideally, the environment should be a controlled one in which the experimenter has the capability to monitor, configure or even re-install in a needed case.
5) **Apply subjective tests:** One of the most important evaluations to be done must be the subjective tests regarding user experience. These tests should consider if technical documentation was understandable for other users, the easiness of the installation for them and configuration and operation of the broker by them. An important point to keep in mind is the level of expertise the users have, it could be convenient to group users by expertise level so to normalize the subjectivity of the evaluations.
6) **Perform benchmarking and classification:** At this step, metrics must be calculated so to evaluate the selected sub-characteristics per broker. The classification should be made according to the mutual specifications that the brokers share. The resultant groups from the classification should be organized by the same experi-

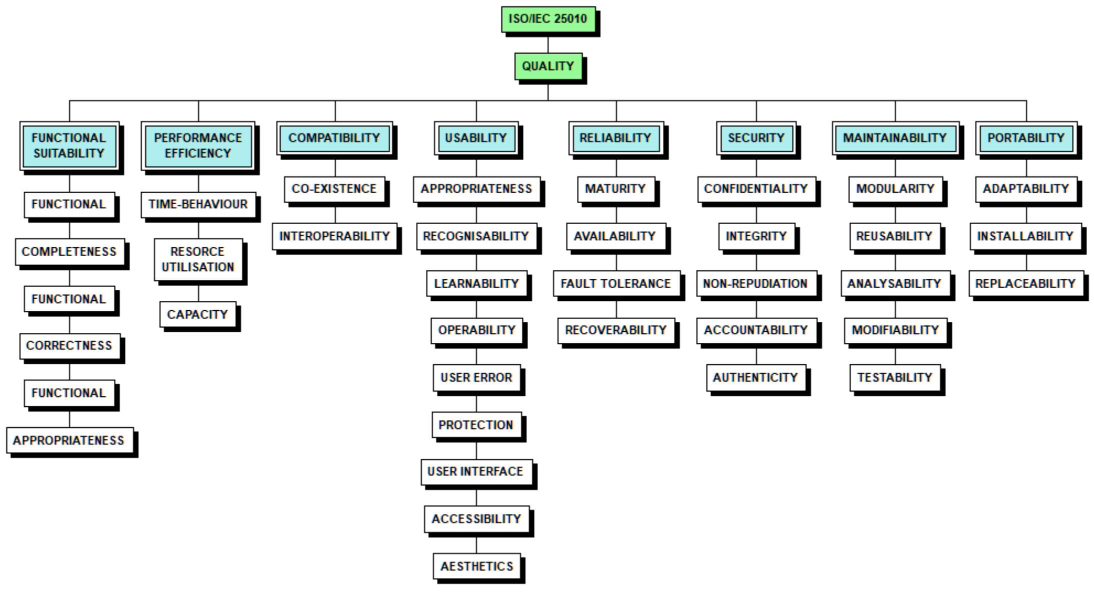

Fig. 5. ISO/IEC 25010 quality reference model
Source: Created by the Author

menter, according to the needs of the evaluation.

In this way, we have a sequential and logical process than can be followed to evaluate brokers according to the ISO/IEC 25010 software product quality model; this is a guideline for the research community on how to implement evaluations on a systematic way.

## V. Implementation Case of the Methodology

We performed an initial research on the Internet about available brokers, basing our initial criteria according to the most common features mentioned in their technical documentation. A step-by-step approach is described, showing how the methodology was carried out.

### A. Brokers selection

As communication protocol criterion, we selected brokers supporting MQTT due that it is one of the more widely used protocols; additionally we chose them to be open-source code as there are manifold solutions. The selected brokers were: Mosquitto, Emqttd, RabbitMQ, ActiveMQ, Apollo, Mosca, HBMQTT and Moquette.

Because they support MQTT, the brokers also support different QoS levels, being them:

- **QoS 0 - At most once**: It is known as best-effort delivery, as it gives no guarantee of the message being delivered and the receiver does not acknowledge reception of messages.
- **QoS 1 - At least once**: At this level the sender stores the message sent for a certain amount of time as there is acknowledgement by the receiver regarding the messages delivery.
- **QoS 2 - Exactly once**: This is the highest level of quality, a four-part handshake is made by both communicating ends to assure messages are delivered, it is the slower service though.

### B. Choosing broker features to be evaluated

For the implementation case we chose the following features to evaluate: operating system support, available programming languages, communication protocol(s) and supported QoS levels. We considered these features to be related with portability, maintainability, functional suitability and reliability, respectively.

### C. Test scenario setup

We adopted a general scenario for basic test communication among publishers, brokers and subscribers. The goal was to capture the basic inter-communication of each layer. Fig. 6 illustrates the proposed use case scenario, it uses a simple configuration but suitable for testing basic functionalities.

The publishing/subscribing clients were deployed using a Macbook Air notebook, processor Intel Core I5 1.4GHz, 4GB RAM, Graphical Card Intel HD Graphics 5000 with 1536 MB memory and a MacOS HighSierra v10.13.2 as operating system; we used Oracle VirtualBox to emulate the test environment.

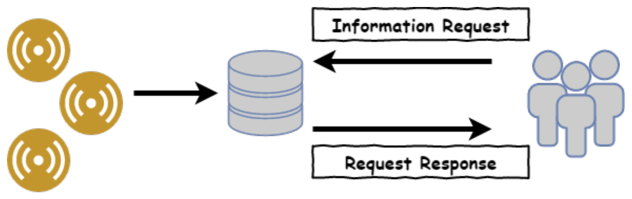

Fig. 6. Testing scenario
Source: Created by the Author

| **Broker** | **Supported protocols** | **Programming language integration** |
|---|---|---|
| RabbitMQ | MQTT, AMQP, STOMP, HTTP | C, C++, Java, Python, Ruby, PHP, JavaScript, Golang, Erlang, Objective-C, Swift, Rust, Haskell |
| Apollo | MQTT, AMQP, STOMP, OpenWire | C, C++, Java, Python, Ruby, PHP, C#, Perl |
| ActiveMQ | MQTT, AMQP, STOMP, HTTP, OpenWire | C, C++, Java, Python, Ruby, PHP, C#, Perl |
| Mosca | MQTT | C, C++, JavaScript |
| Mosquitto | MQTT | C, C++ |
| Emitter | MQTT | JavaScript, Golang |
| Moquette | MQTT | Java |
| HBMQTT | MQTT | Python |
| Emqttd | MQTT, CoAP, STOMP, HTTP | Erlang |

Fig. 7. Implemented communication protocols and integration with programming languages
Source: Created by the Author

### D. *Installation and configuration of brokers*

To do the installation and configuration the brokers were set up using virtual machines running Oracle VirtualBox v5.0.26, with the following settings:

- Operating System: Ubuntu Server 16.04 64 bits;
- Memory: 512 MB;
- Storage: 8GB;
- Network: Network interface running on Bridge mode;
- Video Memory: 12 MB.

### E. *Evaluation results from installation and configuration*

This subsection shows the results considering the following features: supported operating systems, supported programming languages, supported communication protocols and supported QoS levels.

An important insight is to consider the communication protocols implemented and the different programming languages that could be used for integration of the brokers and their functionalities. Fig. 7 illustrates that most brokers work with more than one programming language being C/C++ the most widely used, whereas Java/Python rank second in use and after that we have integrations with Ruby, PHP, JavaScript and others as described on the table.For communication protocols implemented, MQTT is implemented by all of them. The second communication protocol implemented was STOMP, being followed by AMQP, HTTP and lastly OpenWire.

On Fig. 8 we can see that 67% of the brokers support three families of operating systems, so they could be installed in heterogeneous environments.

When publish/subscribe clients establish a first connection with the brokers the QoS level is set, therefore defining the service delivered; Fig. 9 shows the distribution according to the levels of QoS provided and the percentage of brokers corresponding to it.

### F. *Subjective tests application*

On the subjective tests phase, the selected characteristics to be evaluated were directly related to usability, portability and functional suitability as mapped on the ISO/IEC 25010 software product quality reference model. It was done as follows:

- Documentation – Learnability (Usability)
- Broker installation – Installability (Portability)
- Broker coverage – Completeness (Functional Suitability)
- Broker functionality – Correctness (Functional Suitability)
- User experience – Appropriateness (Usability)

Documentation is as a key factor that can be evaluated by the conceptual consistency detailed in the documentation, these feature was related to the learnability characteristic. Broker installation was related to the installability sub-characteristic that defines the software easiness to be installed.

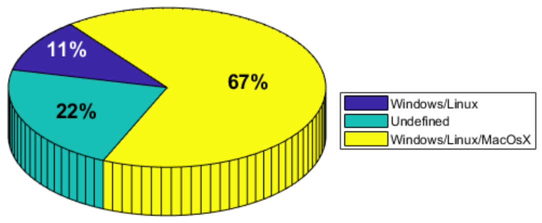

Fig. 8. Supported operating system
Source: Created by the Author

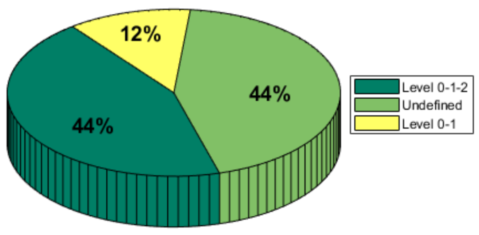

Fig. 9. Implemented QoS levels
Source: Created by the Author

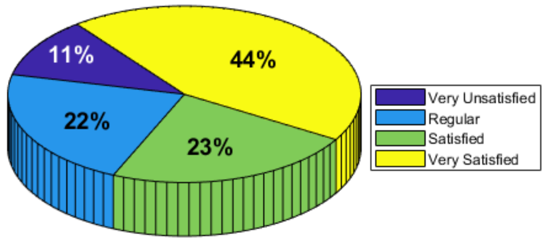

Fig. 10. Documentation
Source: Created by the Author

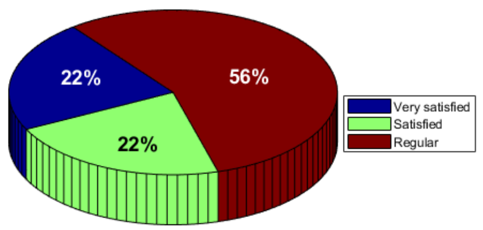

Fig. 12. Broker coverage
Source: Created by the Author

Coverage concerns the completeness of brokers which is defined as the degree to which the they cover most different scenarios as possible. Broker functionality was related to correctness, the degree to which the functions provides the correct results with the needed degree of precision. Finally, user experience was associated to appropriateness, which describes the degree to which users can recognize whether a product or system is appropriate for their needs.

### G. Subjective tests results

This sub-section describes the results applied to the subjective tests. The size of the sample was of one (1) user; the method used to implement these subjective tests was a survey on each of the brokers, listing the features as described:

- **Documentation:** Range: (1) Very unsatisfied, (2) Unsatisfied, (3) Regular, (4) Satisfied and (5) Very satisfied.
- **Broker installation:** Range: (1) Very difficult, (2) Difficult, (3) Easy, (4) Very easy.
- **Broker coverage:** Range: (1) Very unsatisfied, (2) Unsatisfied, (3) Regular, (4) Satisfied and (5) Very satisfied.
- **Broker functionality:** Range: (1) Very Unsatisfied, (2) Unsatisfied, (3) Regular, (4) Satisfied and (5) Very Satisfied.
- **User experience:** Range: (1) Very unsatisfied, (2) Unsatisfied, (3) Regular, (4) Satisfied and (5) Very satisfied.

On Fig. 10 we can observe the evaluation made on technical documentation for all 9 brokers. 44% of the documentation was highly satisfying; 22% was only essential with a concise and straight approach; 23% of the documentation was merely satisfactory, meaning that there were more details when compared to an essential level, like including more examples on its manuals.

After documentation, the next feature compared was the broker installation. Depending on the broker this step took more or less time, as there was a lack of information offered by documentation of some, therefore compromising the installation process. To accomplish this task additional information was obtained from websites. During the installation, Ubuntu operating system was used.

According to Fig. 11, 56% of brokers' installation process was very easy, considering that the process took a few steps; 11% of brokers were classified as very difficult to be installed. A recurrent problem found in the installation process was the absence of some commands related to software dependencies, additionally some of the step-by-step guides were not precise or clear enough.

Concerning broker coverage, the brokers were evaluated whether they were easy and practical to use or adapt in different application scenarios. Fig. 12 shows the statistic of coverage for the brokers, representing 56% of the result for regular coverage, i.e., working just as a single broker with no clustering capabilities. 22% of them were classified as satisfying, including RabbitMQ and Apollo because of clustering and other types of applications like database management. Emqttd and HBMQTT are part of the remaining 22% represented by the very satisfied level. Emqttd also can be used in different strategies like clustering, collaboration with other brokers and as database. HBMQTT can interoperate with other brokers like Mosquitto for message publication.

Broker functionality describes the broker's correctness, that is, the degree to which the broker provided the correct results. Most of the brokers, according to Fig. 13, met the user requirements. Based on the publisher/subscriber model the brokers achieved 78% of highly satisfied level; 11% (viable) show that

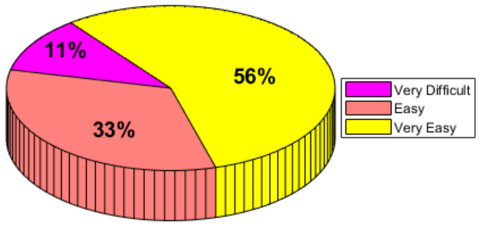

Fig. 11. Broker installation
Source: Created by the Author

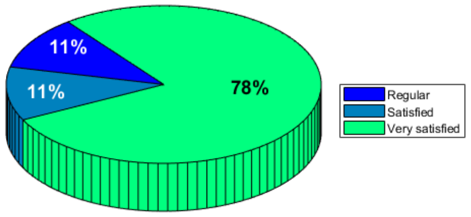

Fig. 13. Broker functionality
Source: Created by the Author

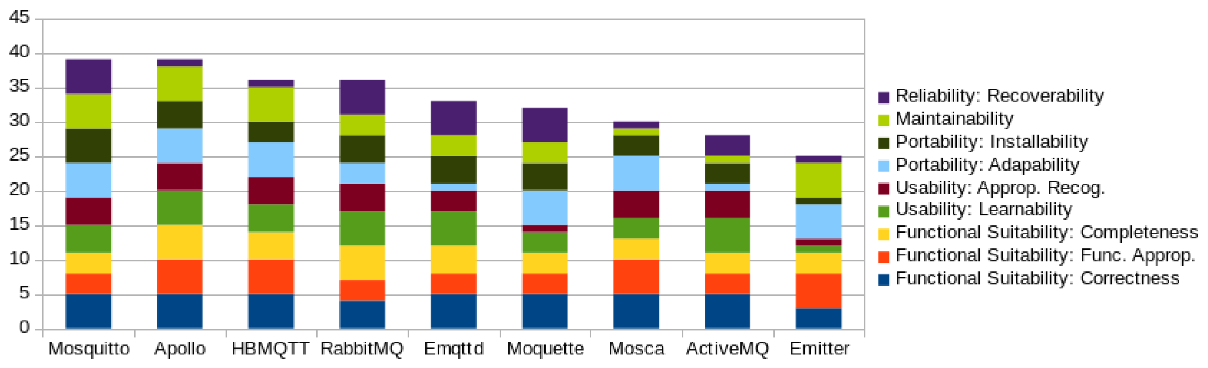

Fig. 14. IoT brokers benchmarking
Source: Created by the Author

the functionality could not have worked properly, which may be due to problems during the execution or errors generated by commands; the other 11% meant that the evaluation had no problem but a careful attention was required by the user to make it work appropriately.

User experience evaluated the overall experience of the user with the broker. Considering the large number of existing brokers and their complexity, the user's satisfaction level was 67% for most of them. Still, according to Fig. 15 it could be observed that the general experience was enjoyable. Nonetheless, 22% was dissatisfying in the experience, probably because of problems during the installation and execution; 11% represent the brokers that could be used but with difficulty.

### H. Benchmarking and classification

From the two previous stages of this research we could take different features for each broker. In the first part we analyzed results regarding portability and functional suitability. Taking in consideration the operating system portability, 67% of brokers provided three options of operating system (Windows, Linux and MacOS). Regarding programming language, 44% of them (Mosquitto, Emitter, RabbitMQ and ActiveMQ) can be implemented with different languages like: C, C++, Ruby, Java, Pyhon, Go, .NET, Javascript and PHP.

Several problems were found relating QoS. Most of them support all three levels but a majority of the brokers presented

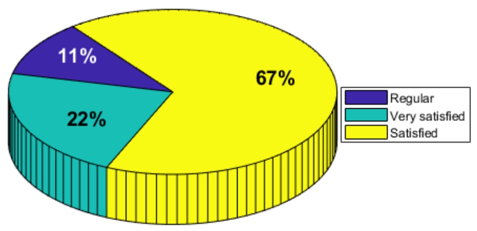

Fig. 15. General user experience
Source: Created by the Author

issues, which are mentioned in their proper documentation. We did the classification as follows::

- **Group 1 - Portable in various operating system:** Mosquitto, Emqttd, Emitter, RabbitMQ, ActiveMQ and Apollo.
- **Group 2 - Support different programming languages:** Mosquitto, Emitter, RabbitMQ and ActiveMQ.
- **Group 3 - Support different communication protocols:** Emitter, RabbitMQ, ActiveMQ and Apollo.
- **Group 4 - Support different levels of QoS:** Emqttd, Mosca, HBMQTT and Moquette.

Fig. 14 shows the benchmarking done with. From these results, can be observed that functionality was a satisfactory feature in most of them. Documentation is not standardized given that these brokers have a poor documentation with incomplete information. Regarding usability the evaluation varied from broker to broker, as well as in coverage. Mosquitto is considered one of the most popular due to its simplicity in installation, operating system portability and lightweight by being done in few lines of code.

## VI. Conclusions and Future Works

Defining widely accepted middleware solutions for Internet of Things is still a big challenge due to the lack of standardization on the evaluation processes and comparisons among the different proposals. [24]

This methodology intends to help the tech community in setting an initial guideline to perform different evaluations but taking a standardized approach by using an international and well known norm, the ISO/IEC 25010 software quality reference model. Our goal is to develop a more complete and defined methodology so it would be possible for different stakeholders to implement their own evaluation and choose among the more suitable broker options.

Among the main points in our proposal are (1) gathering basic information from technical documentation, (2) using the ISO/IEC 25010 software product quality model for guiding on the characteristics selection and (3) defining a logical process to do the evaluation and classification.

For future works we intend to improve this methodology with a more specific scope on doing performance evaluation for brokers, establishing a more clear process on how to define metrics and prove their validity. As the implementation case we show on this paper only considers subjective evaluations by relying on user perception, we intend to develop a form of calculating performance metrics so they present more objective results.

## References

[1] MCTIC, "MCTIC and BNDES present a study of the national IoT plan with 76 actions for the sector," http://www.mctic.gov.br/mctic/opencms/salaImprensa/noticias/ (Portuguese), 2017, [Online; Accessed on November 10th, 2017].

[2] A. Esposte, F. Kon, F. Costa, and N. Lago, "InterSCity: A scalable microservice-based open source platform for smart cities," in *6th International Conference on Smart Cities and Green ICT (SMARTGREENS)*, Porto, Portugal, 2017, pp. 1–12.

[3] N. Nascimento and C. Nascimento, "FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the Internet of Things," *Information Sciences*, vol. 378, pp. 161–176, 2017.

[4] C. Batista, P. Silva, E. Cavalcante, T. Batista, T. Barros, C. Takahashi, T. Cardoso, J. Neto, and R. Ribero, "A middleware environment for developing Internet of Things applications," in *5th International Workshop on Middleware and Applications for the Internet of Things (M4IoT 2018)*, Rennes, France, 2018, pp. 41–46.

[5] E. Cavalcante, "Smart Campus as a Smart City," https://eventos.rnp.br/sites/default/files/activity/activity-presentation/painel10b_evertoncavalcante.pdf (Portuguese), 2016, [Online; Accessed on May 24th, 2019].

[6] Unicamp, "Smart Campus - Unicamp," http://smartcampus.prefeitura.unicamp.br/ (Portuguese), [Online; Accessed on May 24th, 2019].

[7] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of IoT: Applications, challenges, and opportunities with China perspective," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 349–359, 2014.

[8] R. Ma, P. Lam, and C. Leung, "Potential pitfalls of smart city development: A study on parking mobile applications (apps) in Hong Kong," *Telematics and Informatics*, vol. 35, no. 6, pp. 1580–1592, 2018.

[9] J. Ellsmoor, "Smart Cities: The Future Of Urban Development," https://www.forbes.com/sites/jamesellsmoor/2019/05/19/smart-cities-the-future-of-urban-development/#7684ffa12f90, 2019, [Online; Accessed on May 25th, 2019].

[10] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart Cities in Europe," *Journal of Urban Technology*, vol. 18, no. 2, pp. 65–82, 2011.

[11] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Vienna, Austria, 2017, pp. 1–7.

[12] S. Kraijak and P. Tuwanut, "A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends," in *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*, Shanghai, China, 2015, pp. 1–6.

[13] P. Sethi and S. Sarangi, "Internet of Things: Architectures, protocols and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 1–25, Jan 2017.

[14] M. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A survey," *IEEE Internet of Things Journal*, vol. 3, pp. 70–95, 2016.

[15] A. Bhawiyuga, D. Kartikasari, and E. Pramukantoro, "A Publish Subscribe based Middleware for Enabling Real Time Web Access on Constrained Device," in *2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Phuket, Thailand, 2017, pp. 1–5.

[16] R. Giambona, A. Redondi, and M. Cesana, "Demonstrating MQTT+: An Advanced Broker for Data Filtering, Processing and Aggregation," in *21st ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, Quebec, Canada, 2018, pp. 357–358.

[17] W. Pipatsakulroj, V. Visoottiviseth, and R. Takano, "muMQ: A Lightweight and Scalable MQTT Broker," in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Osaka, Japan, 2017, pp. 1–6.

[18] R. Banno, J. Sun, M. Fujita, S. Takeuchi, and K. Shudo, "Dissemination of edge-heavy data on heterogeneous MQTT brokers," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Tokyo, Japan, 2017, pp. 1–7.

[19] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, and D. Kobayashi, "A Scalable and Low-Cost MQTT Broker Clustering System," in *2017 2nd International Conference on Information Technology (INCIT)*, Nakhon Pathom, Thailand, 2017, pp. 1–5.

[20] M. Neumann, C. Bach, A. Miclaus, T. Riedel, and M. Beigl, "Always-On Web of Things Infrastructure using Dynamic Software Updating," in *WoT 2016: The Seventh International Workshop on the Web of Things*, Stuttgart, Germany, 2016, pp. 1–6.

[21] B. Mishra, "Performance evaluation of MQTT broker servers," in *18th International Conference on Computational Science and Applications (ICCSA 2018)*, Melbourne, Australia, 2018, pp. 599–609.

[22] Scalagent, "Benchmark of MQTT servers," 2015. [Online]. Available: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

[23] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT platform requirements with open pub/sub solutions," *Annals of Telecommunications*, vol. 72, pp. 41–52, 2017.

[24] C. Pereira, J. Cardoso, A. Aguiar, and R. Morla, "Benchmarking pub/sub IoT middleware platforms for smart services," *Journal of Reliable Intelligent Environments*, vol. 4, pp. 25–37, 2018.

[25] ISO/IEC, *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, 2011, International Standard ISO/IEC 25010:2011.