Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices

Lucas M. C. e Martins
Cybersecurity INCT, Electrical Engineering Dept., University of Brasilia
Brasilia, DF, Brazil
lucas.martins@redes.unb.br

Francisco L. de Caldas Filho
Cybersecurity INCT, Electrical Engineering Dept., University of Brasilia
Brasilia, DF, Brazil
francisco.lopes@uiot.org

Rafael T. de Sousa Júnior
Cybersecurity INCT, Electrical Engineering Dept., University of Brasilia
Brasilia, DF, Brazil
rafael.desousa@redes.unb.br

William F. Giozza
Cybersecurity INCT, Electrical Engineering Dept., University of Brasilia
Brasilia, DF, Brazil
giozza@unb.br

João Paulo C. L. da Costa
Cybersecurity INCT, Electrical Engineering Dept., University of Brasilia
Brasilia, DF, Brazil
joaopaulo.dacosta@ene.unb.br

ABSTRACT

Internet of Things (IoT) has left the experimental field and is reaching the final consumer in areas such as residential automation, health, transportation and government support. Since these are applications intrinsically linked to the physical world, they require more attention in aspects related to their dependability. Focusing on its availability and reliability, we present a proposal to apply the Microservices architectural pattern in an IoT middleware with web services in the monolithic architecture. We describe the reengineering that must be done in this middleware and, finally, we analyze the advantages and disadvantages of this approach, highlighting the availability improvement, optimization of the infrastructure resources, the ease of maintenance and evolution, as well as the inclination to the elasticity that the architecture allows.

CCS CONCEPTS

• **Software and its engineering** → **Message oriented middleware**; **Abstraction, modeling and modularity**; • **Information systems** → *RESTful web services*; Service discovery and interfaces; • **Computer systems organization** → *Cloud computing*; *Reliability*;

KEYWORDS

Internet of Things (IoT); Microservices; Cloud Computing; Dependability; Middleware; SOA.

ACM Reference Format:
Lucas M. C. e Martins, Francisco L. de Caldas Filho, Rafael T. de Sousa Júnior, William F. Giozza, and João Paulo C. L. da Costa. 2017. Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices. In *UCC'17 Companion: UCC '17: 10th International Conference on Utility and Cloud Computing Companion, December 5–8, 2017, Austin, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3147234.3148092

UCC'17 Companion, December 5–8, 2017, Austin, TX, USA

ACM ISBN 978-1-4503-5195-9/17/12…$15.00
https://doi.org/10.1145/3147234.3148092

1 INTRODUCTION

Network connections have revolutionized the way our society interacts. This transformation became stronger since the advent of the Internet in the 1980s and the emergence of the Web in the following decade. First, the Internet was adopted in the academic and military circles, then in commercial and business environments and finally in interpersonal relations and entertainment.

This gradual growth of the Internet can be characterized by its structure and functioning: static pages, static web pages with markup languages and style; dynamic sites, pages generated dynamically by script languages and lightweight databases; Web 2.0, in which the user used to participate more actively in content construction; and social networks, organizations of people in a computational network where relationships and interactions between people are built up. In addition, the popularization of smartphones and tablets, from the mid-2000s, enabled and boosted mobile computing.

In turn, [15] emphasizes that in the Internet of the Future, "real world" objects will be part of the network, providing information and actions for other resources with access to the worldwide network. With the integration of so-called "smart objects" to the Internet, the term Internet of Things (IoT) was born. According to [22], IoT will allow all kind of objects be connected to the Internet and interact each other with minimum human intervention.

The dependability of this new Internet needs close attention because, among other things, it will be capable of interacting with the physical environment that we are in. In his seminal work, [2] defines dependability of a system as ability to have service failures in acceptable frequency and severity. It also presents the dependability attributes that need to be addressed, among which we highlight availability and reliability.

This Section 1 includes also three subsections which highlights the key concepts about IoT (Subsection 1.1), Cloud Computing

(Subsection 1.2) and Microservices (Subsection 1.3). In Section 2, we present related work and its relation with our proposal. In Section 3, we present the proposal overview and, in Section 4, the scenario that motivates this proposal. We finish in Section 5, with a brief conclusion.

1.1 IoT

IoT is a construction paradigm of computational systems. [15] defines IoT as a paradigm where some of the objects around us will somehow be in the network in order to extend the capabilities of the environment. [22] highlights that IoT allows people, things and environment to be connected with no constraint of time or place, in any available interaction way.

[28] cites three possible scenarios in which these interactions may occur: temporarily, concomitantly and in a mobile way. Considering these scenarios, [28] exemplifies the following situation: when a person moves with their smartphone from home to work, the phone can be part of their home IoT network, their vehicle, their body (network built from devices in their clothing), their route (network in the path) and their work.

Furthermore, [3], [15] and [22] emphasize that IoT differs from sensor networks because it brings intelligence to its domain, turning information and actions into knowledge. In this way, this knowledge feeds back the network, creating new actions and information.

The growth in the number of connected devices and the search for devices that act on people's daily life signal that IoT will represent a significant portion of the use of computing resources and Internet. According to [15], the number of devices connected to the Internet is expected to jump from 9 billion in 2013 to 24 billion in 2020. In addition to this growth in the number of devices, they will flood Internet traffic with information and operations at every moment.

An inherent feature of IoT is that much of the communication between smart objects and middleware is performed by Machine-to-Machine (M2M) communication, usually via web services in Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) way. This gives the middleware a lot of influence of the Service-Oriented Architecture (SOA).

1.2 Cloud Computing

Cloud Computing (CC) is defined by [10] as a form of distributed computing in a remote computing environment, with physical (hardware) and virtual (software) resources to be allocated and used on demand by its various clients. Despite being envisioned in 1961 by scientists like John McCarthy, Cloud Computing has been made possible in recent years from the maturation of technologies such as Clustering, Grid Computing and Virtualization. [10]

Since the mid-2000s, Cloud Computing has spread and consolidated as a new way of planning and using IT resources, and can be seen as commodities, that is basic consumer services such as energy and telephony. Thus, the relationship between the cloud service provider and its customers is guided by two elements: the services use measurement and the quality with which this service was made available. For the client, this means that all the complexity of a computational environment is abstracted with simple units of measurement such as the volume of data stored or the amount of processing used.

An interesting strategy in using Cloud Computing is the *elasticity* that can be obtained, since it is possible to scale in, adding new instances of the resource, and scale out, turning off resources that are no longer in use. This mechanism can be applied to different types of computing resources such as authentication services, content delivery and application hosting. However, this is only possible when the service is built to take advantage of this feature.

1.3 Microservices

Microservices, also called uServices, is an architectural style, inspired by the agile practices, which emerged in the software industry with the goal of increasing the development teams capacity to build and maintain large applications in corporate environments.

In this style, an application is built by the composition of several microservices. A microservice is a small and autonomous service that communicates over a light network and protocols infrastructure. "Small" means that the service must have a single responsibility. "Autonomous" means that the service must have its own independent infrastructure.

[9] considers that Microservices is the second generation of service-oriented architectures, the evolution of SOA. He also highlights the concepts obtained from SOA by dividing the application implementation into services and services orchestration and choreography. They also point out the fundamental differences that separate SOA Micro-Services: the size of the service should be small, with a single responsibility; the *bounded context*, imported from Domain-Driven Design (DDD), which combines related functionalities in a single business capability and the operational independence between services in which each part of the service must be built and executed independently of the others.

[19] point out that the channel is another big difference between two generations of service-oriented architectures. SOA relies on middlewares such as Enterprise Service Bus (ESB) to act as a means of service integration. ESBs are products with many features and facilities such as message routing, data transformation, protocol transformation and business rules application. For Microservices, the channel is only a means of transport. [19] explain this approach with the expression "*smart endpoints and dumb pipes*".

The innovation of Microservices is more clearly perceived if its architecture is compared to the architecture widely used in the corporate environment: the monolithic architectural style. In [9], a monolithic application is defined as being "*a software application whose modules cannot be executed independently*". [20] highlights that the entire application is built on the same set of technologies and as a consequence it also uses the same servers in its infrastructure. In [20], it is noted that a problem derived from this feature is that any addition or change of language, *framework* or database affects the entire application.

Figure 1 highlights the difference between the structure of the architectural approach of a monolithic application and an application with microservices. While monolithic uses the same infrastructure for all functionalities, microservices have a complete separation of their entire technology stack, running in separate processes and / or environments so that communication via HTTP/HTTPS is the

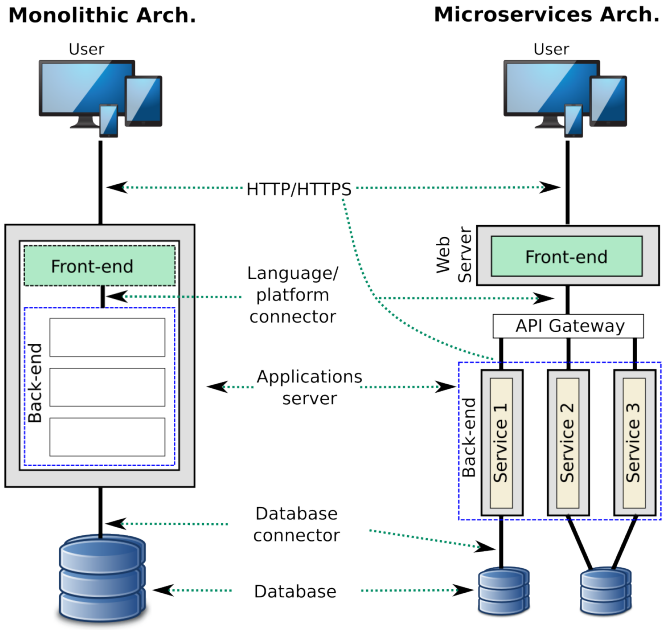

Figure 1: Monolithic Architecture versus Microservices.

only link between the microservices. As highlighted in the figure, each microservice may have its own data storage infrastructure or may share with other microservices.

2 RELATED WORK

Works such as [1], [15], [22], [26] and [29] present surveys about the current IoT research scenario. These works provide an overview of how the paradigm has been treated in the most different aspects, including the traditional architecture. However, they do not detail the solutions or how they are implemented.

In [5], [6, 7], [8], [11], [17], [21], [23] and [30], projects and prototypes involving IoT middlewares are shown. In spite of presenting IoT from some perspective focused on some use, none of these works focus on how to solve problems of architecture scalability, nor on techniques to increase the system dependability. When addressing the subject of dependability, the focus is on issues related to architecture security.

[1], [3], [5], [15], [16], [22] and [29] highlight and advocate the use of Cloud Computing as a way to give processing power to the applications built in the IoT paradigm. They argue that the volume of data generated by IoT applications and the need to turn this data into useful information within their context perfectly fit the benefits offered by Cloud Computing. These works discuss the benefits and limitations provided by the combination of IoT and Cloud Computing. However, they bring conventional proposals to the way IoT applications and network components are built and deployed in the cloud. Therefore, they do not address techniques and ways to optimize applications and services architecture to get the best use of the capabilities offered by the cloud.

The literature on Microservices covers the most diverse points of architecture. However, its common focus is the software industry, as well as the impact of Microservices on the software ecosystem, including the smartphone applications industry.

Works such as [18], [25] and [31] present convergence between the two research areas. In the following paragraphs, we detail this convergence a bit.

Thinking about the need to provide robustness and the ability to evolve over time, [18] reports the experience of using Microservices on a platform of a Smart City project. The experiment is directed to the topic of energy efficiency, as well as details the use of services with the focus on providing semantic information about services.

The work suggests the use of Microservices to meet IoT challenges, considering each microservice as a base element that can be used and reused as needed. As in this paper, they argue about advantages such as technology heterogeneity, the resiliency and scalability that can be achieved with Microservices and the possibility of a better organizational alignment. However, it is an initial work, tested in a context of simplified use and with few results to present.

The work of [25] presents a prototype platform for a smart building, using Microservices and cloud computing to implement this IoT environment. However, the architecture is described at a high level as well as focuses on the use of Jolie[1], a language focused on implementation of micro-services.

[31] presents an IoT application framework with microservices as its REST API. Thinking of benefiting from the flexibility of structure and agility for development, they are concerned about making use of all the benefits of Microservices to build a structure that can evolve over time, being adaptable to the new scenarios. The paper outlines the architecture of this proposal, using IoT concepts and Microservices.

However, its proposal has limitations. The first is that it focuses on the tools and support structures of the IoT network and leaving the device aside without investigation. In this way, it is already assumed that the device is connected to the Internet to participate in the IoT network The second disadvantage is that, despite being based on Microservices, its proposal still has monolithic characteristics. For example, there is a direction for using a centralized data storage service. Centralization causes dependence on another resource and this is exactly what is not wanted in the microservice approach.

For this paper, we consider the Universal Internet of Thing (UIoT) from University of Brasília. After discussions in [12], [14] and [13], [11] proposes an IoT middleware architecture to "*control and notify the current state of generic devices*". In his proposal, he specifies the physical and logical components of that architecture that allow interaction with these smart objects, as well as defines its Application Programming Interface (API) and the use of Universal Plug and Play (UPnP) for device discovery. In [27], this view is refined and evolved, consolidating the architecture protocols and the API in REST Abstract Service Layer (RAISe). In [28], It is proposed to replace the UPnP by a self-registration mechanism in which the device itself, aware of its context, becomes responsible for the initiative of its participation in the network.

[1] http://www.jolie-lang.org/

3 PROPOSAL FOR IOT WITH MICROSERVICES

The software dependability encompasses faults handling and threats handling. These are ordinary and key concepts that must drive its design and implementation. Furthermore, we propose special attention on *availability* and *reliability* to increase the dependance of IoT middleware. In [2], availability is a dependability attribute defined as the "readiness" of the service and reliability is another dependability attribute defined as the "continuity" of the service. We rely on software elasticity to make it highly available and reliable.

To provide its M2M services, RAISe is a monolithic software which means that concentrates all its services into a single bundle, within an integrated and homogeneous infrastructure. To deal with the increasing demand, its elasticity (scale in) is made by adding new instances of the application. When there is a reduction in demand, simply remove the instances that are underutilized by doing the (scale out).

The Microservices architecture allows a better manipulation of this type of scenario, since it allows to differentiate the treatment given to the various services that compose the application. Thus, we suggest performing RAISe reengineering[2] to separate each of its services according to the Microservices architecture. That way, RAISe would be composed of several microservices, as described in [27], each one executed independently in its own infrastructure. As a consequence, elasticity would occur individually for each middleware service. Figure 2 suggests the RAISe microservices which are involved in the self-registration process, from [28].

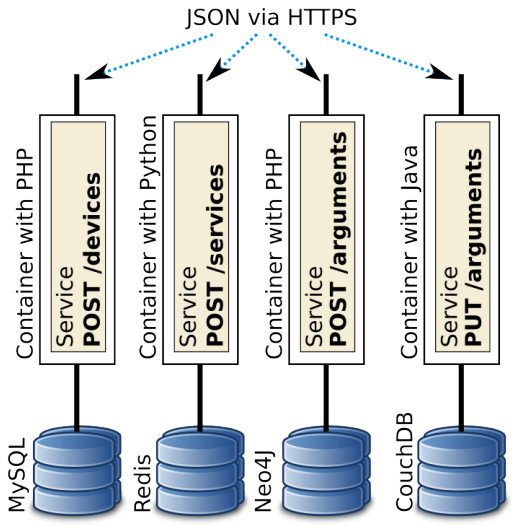

Figure 2: Microservices suggested for RAISe.

Figure 2 depicts several microservices with different technologies. It was illustrated in this way to make explicit that the choice of the technologies of each microservice is irrelevant to the system as a whole, as long as the signature and the services behavior are maintained. It is worth mentioning that this technological variety appears only for didactic purposes and that this work does not intend to review the UIoT implementation choices

Although making a big change in the RAISe's structural area, this reengineering must not affect any service interface so that its clients will not be affected. It means the Microservices REST API will have the same behavior of the monolithic one.

We also suggest the use of Infrastructure Automation practices, as defended in [20], using containers to build the microservices execution environment. In this way, it is intended to simplify all the complexity and details of deploying and managing these several microservices, since it is possible to build deployment and infrastructure components that can be reused among several microservices that have similar characteristics. Figure 3 ilustrates the composition of images based on other pre-existing images. For example, you can build an image with basic features for the functioning of the operating system. You can then create a derived image with the necessary resources to function as a web server. In it, modules and libraries necessary for the operation of PHP applications can be added. At last, the microservice image is built on it. This same logic is used for the construction of the other images such as for microservices built in Python technology and for services such as database and messaging.

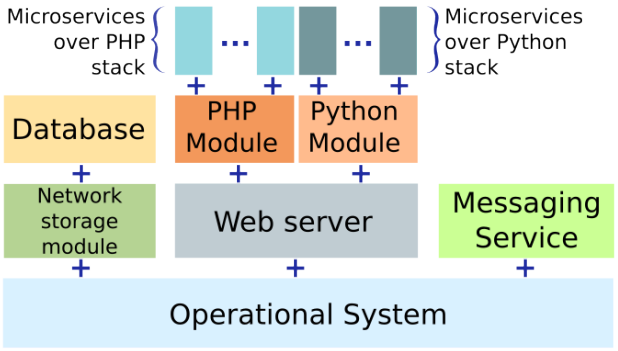

Figure 3: Containers images.

Once these images are built, they can be easily deployed in cloud platforms. The deployment is made by a container instance initialization hence the new instance of the service is available.

To support the RAISe elasticity mechanism and to simplify access by smart objects, [24] suggests the use of a design standard called API Gateway. For [31], the API Gateway handles outside requests and forwards them to the microservice single instance that will process it, and also lightens the interaction between the microservices.

On the client side, the API Gateway address would be the single point of contact that customers have with middleware microservices. On the side of the RAISe implementation ecosystem, several instances must be made available and unavailable at any time, so that it becomes impracticable for the client to directly access any of them.

[2] According to [4], Reengineering is rebuilding a software in a new way. They also emphasize that reengineering seeks to reorganize the structure and maintainability of the software without changing its behavior.

4 PROPOSAL JUSTIFICATION

To assess the suitability and whether UIoT can really benefit from this proposal, we will consider the ideal behavior of the network and the smart objects, as well as focus the analysis on the devices self-registration process in the RAISe, as described in [28].

Considering $t_{validity}$ as a fixed period of time in which the smart object's identification token remains valid and $t_{interval}$ as the time interval between each sending of data to RAISe, Equation 1 defines $q_{\max}$ as the maximum number of interactions that can be performed in the token validity cycle of a device.

$$q_{\max} = \frac{t_{validity}}{t_{interval}} \tag{1}$$

Equation 2 describes the number of services performed in an interaction i, with d, s, a and a' corresponding to the REST operations described in table 1 and n corresponds to the number of services offered by the smart object. In the first interaction, it is necessary to perform self-registration, making all possible requests for this procedure. Then, as long as the token is valid, requests are made regarding the services offered by the device. Finally, when the token expires, only one request is received that gets an access denied information, signaling to the device the need to rrepeat the registration.

$$f(i) = \begin{cases} d + n \times s + n \times a + n \times a', & \text{if } i = 0 \\ n \times a', & \text{if } 0 < i < q_{\max} \\ a', & \text{if } i = q_{\max} \end{cases} \tag{2}$$

At last, Equation 3, derived from the foregoing, describes q_{op} as the total number of operations of an smart object during a validity cycle of its token.

$$q_{\text{op}} = f(0) + \overbrace{f(1) + \cdots + f(q_{\max} - 1)}^{\sum_{i=1}^{q_{\max}-1} n * u} + f(q_{\max}) \tag{3}$$

Based on these equations, it can be realized a projection of the number of operations the device would perform during the validity of its token. For the projection, it was considered that the token had 2 hours of validity, that the device sent to RAISe information of its two services every ten seconds and that the network and the intelligent object had stable behavior.

Service	Symbol	Formula	Qt.	%
POST /devices	d	1	1	0,0007%
POST /services	s	n	2	0,0014%
POST /arguments	a	n	2	0,0014%
PUT /arguments	a'	$(q_{\max} + 1) \times n + 1$	1.443	99,9965%
Total	q_{op}	–	1.448	–

Table 1: Quantity of services performed simulation.

According to Table 1, `PUT /arguments` service is used in more than 99,99% of the time. This result is expected, since once inserted in the network in an authorized manner, the intelligent object begins to interact according to its purpose of sending and/or receiving data from that network.

Table 1 spotlights some RAISe's components have higher demand than other and it is important to note all services shares the same resources and the same infrastructure. Also, each service may have different non-functional attributes. For example, the unavailability of the `POST /devices` service prevents new devices from entering the IoT network and sending information when the token expires. However, this unavailability should not prevent the device with a valid token from using the `PUT /arguments` service and keeping the IoT network powered by its information or feeding it from the IoT network information.

5 CONCLUSIONS AND FUTURE WORK

This work proposed the reengineering of the UIoT middleware to the Microservices architectural standard. In addition, we suggested the combination of Microservices with Infrastructure Automation and the use of Cloud Computing. This proposal is feasible because there is no impact for RAISe clients, since no change should be made to the subscription of the existing services.

It is expected, with this change, to increase the avaiability and the reliability of the application, for reasons such as the following:

Uninterrupted execution. The resources offered by the cloud are virtually unlimited. By using load balancing and client redirection strategies, it is possible to provide middleware services uninterruptedly to smart objects. This feature is especially useful because it eliminates the time between failures as well as the need for deployment window.

Failure isolation. As each microservice runs independently on its own infrastructure, the failures of one microservice do not affect the execution of the other microservices. This type of independence also makes it possible to isolate the failures of an instance of a microservice, so that other instances are also protected.

Middleware elasticity. Infrastructure automation is a key tool for providing service elasticity. In this way, it is possible to scale services on demand, within levels of service quality, allowing smart objects to be always attended properly. The infrastructure can be resized to meet the specific needs of each microservice without impacting another microservice. Thus, it is possible to optimize the infrastructure, as well as to apply the elasticity in a specific way to the `PUT /arguments` service, which is the great consumer of infrastructure resources;

Environmental reproducibility. The use of containers ensures the reproducibility of the software in the manner and environment it has been defined. This is possible because the images contain every software, configuration, and library required for its execution. Another interesting aspect is that this allows the standardization of the service execution environment since the images are constructed from the aggregation of other images.

As future work, we envisage to make a detailed comparison between the performance of RAISe implemented with microservices with its monolithic version, exploring the behavior of both in scenarios of increase of demand and change of version. We also envisage evaluating the applicability of Microservices and containers in devices with reduced architecture, such as the ARM architecture.

The use of this type of platform allows deciding whether or not to use the embedded solutions project.

ACKNOWLEDGMENTS

This research work has the support of the Brazilian research and innovation Agencies CAPES – Coordination for the Improvement of Higher Education Personnel (Grant 23038.007604/2014-69 FORTE – Tempestive Forensics Project), CNPq – National Council for Scientific and Technological Development (Grant 465741/2014-2 Science and Technology National Institute – INCT on Cyber Security), and FAPDF – Research Support Foundation of the Federal District (Grant 193.000976/2015), as well as the Brazilian Ministry of Planning, Development and Management (Grants 005/2016 DIPLA – Planning and Management Directorate, and 11/2016 SEST – State-owned Federal Companies Secretariat) and the DPGU – Brazilian Union Public Defender (Grant 066/2016).

REFERENCES

[1] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials* 17, 4 (2015), 2347–2376.

[2] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing* 1, 1 (2004), 11–33.

[3] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescape. 2014. On the Integration of Cloud Computing and Internet of Things. IEEE, 23–30. https://doi.org/10.1109/FiCloud.2014.14

[4] Pierre Bourque and Richard. E Fairley (Eds.). 2014. *SWEBOK: Guide to the software engineering body of knowledge, Version 3.0.* IEEE Computer Society. http://www.computer.org/portal/web/swebok/swebokv3 OCLC: 926093687.

[5] Francesco Carrino, Elena Mugellini, Omar Abou Khaled, Nabil Ouerhani, and Juergen Ehrensberger. 2016. iNUIT: Internet of Things for Urban Innovation. *Future Internet* 8, 2 (May 2016), 18. https://doi.org/10.3390/fi8020018

[6] Soumya K. Datta and Christian Bonnet. 2014. Smart M2M gateway based architecture for M2M device and Endpoint management. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE.* IEEE, 61–68.

[7] Soumya K. Datta, Christian Bonnet, and Navid Nikaein. 2014. An IoT gateway centric architecture to provide novel M2M services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on.* IEEE, 514–519.

[8] Alfredo D'Elia, Lucca Roffia, Guido Zamagni, Fabio Vergari, Paolo Bellavista, and Alessandra Toninelli. 2010. Smart applications for the maintenance of large buildings: How to achieve ontology-based interoperability at the information level. In *2010 IEEE Symposium on Computers and Communications (ISCC).* 1077–1082. https://doi.org/10.1109/ISCC.2010.5546639

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2016. Microservices: yesterday, today, and tomorrow. *arXiv:1606.04036 [cs]* (June 2016). http://arxiv.org/abs/1606.04036 arXiv: 1606.04036.

[10] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. 2013. *Cloud computing: concepts, technology & architecture.* Pearson Education.

[11] Hiro G. C. Ferreira. 2014. Arquitetura de Middleware para Internet das Coisas. (2014).

[12] Hiro G. C. Ferreira, Edna D. Canedo, and Rafael T. de Sousa Jr. 2013. IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and Arduino. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob).* 53–60. https://doi.org/10.1109/WiMOB.2013.6673340

[13] Hiro G. C. Ferreira, Edna D. Canedo, and Rafael T. de Sousa Jr. 2014. A ubiquitous communication architecture integrating transparent UPnP and REST APIs. *International Journal of Embedded Systems* 6, 2/3 (2014), 188. https://doi.org/10.1504/IJES.2014.063816

[14] Hiro G. C. Ferreira, Rafael T. de Sousa Jr., Flávio E. G. de Deus, and Edna D. Canedo. 2014. Proposal of a secure, deployable and transparent middleware for Internet of Things. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI).* 1–4. https://doi.org/10.1109/CISTI.2014.6877069

[15] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[16] Zaheer Khan and Saad Liaquat Kiani. 2012. A cloud-based architecture for citizen services in smart cities. In *Proceedings of the 2012 IEEE/ACM fifth international conference on utility and cloud computing.* IEEE Computer Society, 315–320.

[17] Peter Kostelnik, Martin Sarnovsk, and Karol Furdik. 2011. The semantic middleware for networked embedded systems applied in the internet of things and services domain. *Scalable Computing: Practice and Experience* 12, 3 (2011), 307–316. http://www.scpe.org/index.php/scpe/article/view/726

[18] Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti. 2015. Designing a Smart City Internet of Things Platform with Microservice Architecture. IEEE, 25–30. https://doi.org/10.1109/FiCloud.2015.55

[19] James Lewis and Martin Fowler. 2014. Microservices. (March 2014). http://martinfowler.com/articles/microservices.html Accessed: 2017-02-23.

[20] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly, Sebastopol, CA, USA.

[21] Edoardo Patti and Andrea Acquaviva. 2016. IoT platform for Smart Cities: Requirements and implementation case studies. IEEE, 1–6. https://doi.org/10.1109/RTSI.2016.7740618

[22] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials* 16, 1 (2014), 414–454. https://doi.org/10.1109/SURV.2013.042313.00197

[23] Dragos Repta, Ioan Stefan Sacala, Mihnea Moisescu, and Aurelian Mihai Stanescu. 2013. Semantic Middleware Architecture. *Applied Mechanics and Materials* 436 (Oct. 2013), 488–496. https://doi.org/10.4028/www.scientific.net/AMM.436.488

[24] Chris Richardson. 2014. API gateway pattern. (Sept. 2014). http://microservices.io/patterns/apigateway.html Accessed: 2017-02-21.

[25] Dilshat Salikhov, Kevin Khanda, Kamill Gusmanov, Manuel Mazzara, and Nikolaos Mavridis. 2016. Microservice-based IoT for Smart Buildings. *arXiv:1610.09480 [cs]* (Oct. 2016). http://arxiv.org/abs/1610.09480 arXiv: 1610.09480.

[26] Sabrina Sicari, Alessandra Rizzardi, Luigi A. Grieco, and Alberto C. Porisini. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks* 76 (Jan. 2015), 146–164. https://doi.org/10.1016/j.comnet.2014.11.008

[27] Caio C. de M. Silva, Hiro G. C. Ferreira, Rafael T. de Sousa Jr., Fábio Buiati, and Luis J. García Villalba. 2016. Design and Evaluation of a Services Interface for the Internet of Things. *Wireless Personal Communications* (Jan. 2016). https://doi.org/10.1007/s11277-015-3168-6

[28] Caio C. M. Silva, Francisco L. de Caldas, Felipe D. Machado, Fábio L. L. Mendonça, and Rafael T. de Sousa Jr. 2016. Proposta de auto-registro de serviços pelos dispositivos em ambientes de IoT. Santarém-PA.

[29] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. 2017. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications* 24, 3 (2017), 10–16.

[30] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of Things for Smart Cities. *IEEE Internet of Things Journal* 1, 1 (Feb. 2014), 22–32. https://doi.org/10.1109/JIOT.2014.2306328

[31] Herwig Zeiner, Michael Goller, Víctor Juan Expósito Jiménez, Florian Salmhofer, and Werner Haas. 2016. SeCoS: Web of Things platform based on a microservices architecture and support of time-awareness. *e & i Elektrotechnik und Informationstechnik* 133, 3 (June 2016), 158–162. https://doi.org/10.1007/s00502-016-0404-z