Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 130 (2018) 318–325

Procedia Computer Science

www.elsevier.com/locate/procedia

The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)

A physical access control system on the cloud

Filippos Antonolpoulos[a], Euripides G.M. Petrakis[a]*, Stelios Sotiriadis[b], Nik Bessis[c]

[a]School of Electrical and Computer Enginnering, Technical University of Crete, Chania, Greece
[b]Department of Computer Science and Information Systems, Birkbeck, University of London, London, UK
[c]Department of Computer Science, Edge Hill University, UK

Abstract

We present an automated Physical Access Control System (PACS) as a cloud service for controlling users' activity, navigation and access in large residential infrastructures (e.g. apartment buildings, shopping malls). The main idea is to install on buildings beacon Bluetooth radio transmitters that broadcast an identifier to nearby devices (e.g. users' mobile phones) and from there, together with users' information, to transmit this information safely to a private cloud. The service monitors peoples' movements and overall traffic in buildings and public facilities and offers prompt response on cases of critical events (i.e. overcrowding, health incidents, attacks etc.). PACS provides a variety of services aiming to inform the infrastructure manager for possible increase of users' activities or access requests that require permission based on subscriptions or authorization criteria. These services are deployed over secure private clouds capable of dealing safely with sensitive information while ensuring users' privacy. Collectively, anonymous history (log) data are sent to a public cloud for analysis.

Keywords: Cloud computing; Fog computing; Internet of Things; Bluetooth Beacon sensors; Physical access control system, PACS;

1. Introduction

The idea of the Internet of Things (IoT) combined with cloud computing, opens new horizons in the field of real time data collection and analysis. The use of wearable sensors and mobile devices and their capability for Internet connectivity provides significant benefits in applications areas that require fast and continuous monitoring of user data from anywhere (e.g. activity monitoring in health care, smart cities etc.). In real-life applications, huge amounts

* Corresponding author. Tel.: +302821037229; fax: +302821037542.
E-mail address: petrakis@intelligence.tuc.gr

of data are collected and analyzed (e.g. for scientific or business purposes). The solutions have to be scalable (to deal with the ever-increasing number of users and size of data), cost-effective, respond within reasonable time (e.g. taking into account the time constraints of the application) and, address concerns related to users' privacy and data safety. Cloud is the ideal environment for IoT applications design and implementation due to reasons related to its affordability (no up-front investment, low operation costs), ease of deployment (taking advantage of IaaS and PaaS solutions already available in the market by many cloud vendors), low maintenance costs (easy software updates and enhancements), scalability (compute resources can be added on demand) and accessibility (IoT services can be accessed anytime from anywhere over the Web).

Cloud computing has some inherent disadvantages and limitations. Even though a cloud may offer virtually unlimited compute resources, internet bandwidth may impede application performance or, business or regulatory requirements of the application mandate for hosting resources in a specific place. This is typically the case with user data where severe restrictions in regards to data transfer and storage to public locations may apply (e.g. military, medical, people activity monitoring applications). Another limitation is with regards to security; although many attempts towards increasing the security on the cloud have emerged lately, users are not always willing to send their data to the cloud due to the unknown storage location and also, due to the perceived "distance" between them and their data. In general, the user or application owner has limited control on cloud infrastructures that are managed only by the service providers. This "distance" is also the main reason for the long delays that are experienced sometimes between the client devices and the services locations[1]. Examples for delay intolerant applications are physical access control systems, health monitoring and factory automation. To address these limitations, the paradigm of fog computing has lately emerged, starting from Cisco[3]. Fog computing can be assumed as an extension of cloud computing, bringing virtualized services closer to the edge of the network (i.e. close to the user on gateways or on user devices[4]). Fog brings the benefits of low latency (due to the close proximity), location awareness and increased security, privacy and availability. Efforts to standardize architectures and platforms for extending the cloud to support functional edge nodes are currently underway[†]. However, since the paradigm of fog computing has emerged only lately, architectures and platforms for extending the cloud to support functional edge nodes have not been fully designed yet.

In this work, we focus on the problem physical access control and activity monitoring of people using (i.e. living in or visiting) large public facilities (e.g. shopping malls, sport arenas or smaller areas such as cinemas, restaurants etc.) and residential infrastructures (e.g. apartment blocks). Access to public areas is granted, controlled or monitored by administrative personnel in order prevent emergency events (e.g. due to overcrowded areas, attacks, fire, health-related incidents etc.). Possible solution to dealing with situations such as the above, would be to warn or prevent users from accessing these sites or, to provide temporary access to user groups on demand or, based on authorization criteria and for specific time periods. Such procedures are often time-consuming and costly since they require the involvement of well trained personnel. For example, administrators of large buildings, may benefit by using computerized services capable of monitoring users' activities and informing them on events taking place in specific areas. In addition to dealing with critical events and ensuring users safety, installing and applying services facilitating access to areas where large numbers of people are concentrated, may also improve users experience (e.g. to avoid overcrowding or queuing, customers are advised to visit a place or use a facility in a later time). Furthermore, the analysis of information collected from large numbers of users (eventually this information becomes big) can lead to important conclusions in relation to users' behavior or the cause of critical events. Also, the analysis of this information may provide the means to real estate owners and tenants to improve their plan for more efficient, safer and profitable management of their facility.

The physical access application requires that services capable of dealing with information acquired from users (e.g. their mobile phones) and sensors run on a fog device (e.g. a local installation providing virtualized resources for data processing and storage). Alongside, to mitigate concerns of user data protection and users' privacy, we opt for

[†] https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Architecture-Overview-WP-2-2016.pdf

anonymous data collection (with user consent) and services installed locally on a private rather than on a public cloud. In this case, the role of fog node resembles that of a private cloud. Off-the shelve fog or private cloud devices providing the desired functionality are currently becoming available on the market at affordable prices (e.g. SixSQ's NUVLA Box[‡]). This way, anonymous user data pertaining to users' information are stored locally (i.e. no data are exposed to the internet or stored on a public cloud). However, the analysis of big amounts of anonymous user data (i.e. data analytics) resort to services running on a public cloud. The combined use of private and public cloud is referred to as Hybrid Cloud[2].

In the following, we present a Physical Access Control System (PACS) which implements the physical access scenario and principles of fog/private cloud systems referred to above. In our scenario, each building has a private cloud system installed that provides greater guarantees of protection of personal or business data as this remains in control of the infrastructure owner. To allow users monitoring and to control their access to facilities, we install Bluetooth Low Energy (BLE) proximity detection sensors in all shared or private areas. We use Estimote Beacon sensors[§] that are available on the market on reasonable cost. The building infrastructure manager registers the areas (with their installed Beacon sensors) in the system (i.e. a fog node running the access management services). Before a user is granted access to a building or facility, he/she must be logged into the system using his/her mobile device (i.e. phone). An application running on the mobile device receives area codes from Beacon sensors in its proximity. If access is granted by the administrator or by the system, user's activity is being monitored constantly as the mobile device transmits the area codes it receives from Beacons sensors to the fog each time a new area is accessed. By interacting with the Beacon sensors, users are informed in real-time whether or not they have access to a facility. At the same, the system administrator receives updates about heavily populated areas and (if required) can inform the visitors of an area accordingly. Toast messages are posted to their mobile devices or a message is sent to the staff or to any interacting administrator to take actions. The private cloud (fog node) communicates with a public cloud to send anonymous infrastructure usage data for further analytics.

PACS is a novel Future Internet (FI) cloud service for data collection from IoT devices in an automatic, generalized and modular way. We develop a two-fold solution, based on micro-services for the IoT (users' smart devices) and the cloud side (that includes the back-end services). PACS services are implemented on FIWARE[**], an open-source distributed cloud infrastructure funded by the EU. PACS encompasses FIWARE and IoT-A[6,7] design principles in an attempt to develop an innovative IoT platform that supports generic services and IoT devices (i.e. independent of connectivity and not coupled to specific IoT protocols). IoT-A[5] proposes an Architecture Reference Model (ARM) defining the principles and guidelines for generating IoT architectures, providing the means to connect vertically closed systems in the communication layer (i.e. how devices interact with the system) and service layer (i.e. how services are integrated). IoT-A and Open Fog compliant architectures may assure that generated knowledge will be modular and reusable across domain or use-case specific boundaries.

2. Design and architecture

We followed a valid design approach that identified (a) the functional components and their interaction, (b) the information that is managed and how this information is acquired, transmitted, stored and analysed, (c) the physical software entities that support the functional and information activities, (d) the requirements for assuring data, network and user security and privacy. PACS (reference) architecture is described by a set of UML diagrams[8] including (a) information (class) diagrams describing information that is handled by the system, (b) activity diagrams describing flowcharts for several types of user actions the most important of them being, system login request (user authentication and authorization), request for new account, request to access a facility or area and, event handling (i.e. handling cases of overcrowded areas and critical events) and, (c) an architecture diagram.

‡ http://sixsq.com/products/nuvlabox/
§ https://estimote.com
** https://www.fiware.org

PACS offers a cloud deployment platform of a complete working set of services which cover all design aspects of public, hybrid cloud and edge processing services that are foreseen in the physical access application scenario. PACS architecture design is focusing on the communication of IoT with the cloud system for fast data collection and data storage in a secure way. We aim to develop a framework that transforms users and their devices to data sources, where streams are forwarded to the cloud, so other applications and users can subscribe to cloud services and perform useful actions. Firstly, we target real-time IoT data collection from smartphones and sensors, with a particular interest on Bluetooth Low Energy (BLE) devices (i.e. Beacon sensors installed in facilities). These are low power sensors that support mobile application connectivity and connections to handheld devices which are connected to the Internet through WiFi or GSM networks. Secondly, we target a private cloud system deployment with scalable data storage capabilities and interfaces to external users and systems. If required, user activity (log) data are forward to a public cloud for evaluation (data analytics take place on a public cloud).

Building upon principles of service-oriented design and driven by the key requirements of today's IoT systems for adaptability, low-cost and scalability, PACS architecture is modular, expandable and generic (as it can be adapted to diverse application domains and use case scenarios). The implementation relies on modular services which are available on the cloud and implement fundamental functionalities. They offer important benefits such as, connectivity of smart devices, scalability, openness, re-usability, multi-tenancy, increased accessibility and security. Some of the service modules foreseen in PACS architecture are available as online as re-usable Generic Enablers (GEs) on FIWARE catalogue[††] (the rest were developed by the authors). Each module is designed as a RESTful service which is able to interact with other RESTful services resulting in seamless application integration. This design highlights significant advantages that include (in addition to the comprehensive structure and re-usability referred to above), easy replacement of modules and flexible system configuration that suits the needs of the application. Fig. 1 illustrates the two-fold solution described here with its two system nodules namely, front-end and back-end. Application intelligence is implemented partially on the cloud back-end (fog node) and partially on the gateway.

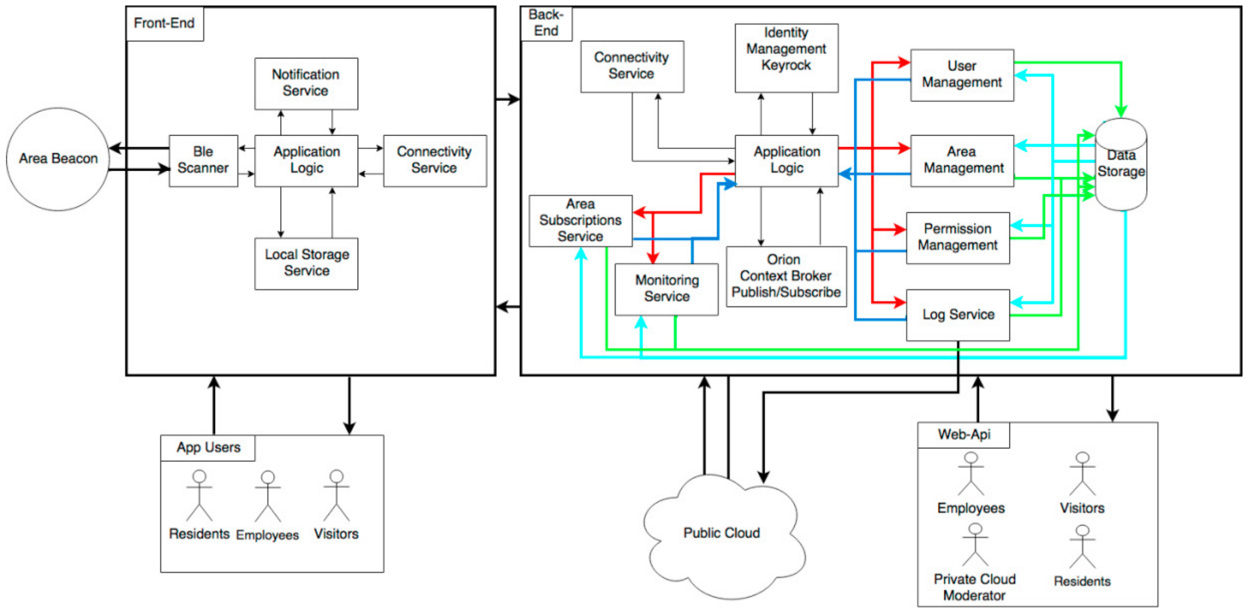

Fig. 1. PACS architecture.

[††] https://catalogue.fiware.org

The gateway (mobile device) is implemented in Apache Cordova[‡‡] and supports the following services:

- *Application logic*: It runs service orchestration that controls services interconnection and applies user specific rules for handling access rights (i.e. subscriptions to facilities on a fee per time interval or access rights granted by an administrator on demand). It synchronizes with user rules stored in the cloud back-end (where rules are created and access rights are granted by a system administrator).
- *BLE Scanner*: Supports discovery of new Beacon sensors that are associated with a facility. It is a responsibility of the system administrator to install and register the sensors in a database in the cloud back-end. The front-end need not be aware of this information. Only sensors in the proximity of the mobile device are discovered automatically and paired with access rights or rules for granting or prohibiting access to users.
- *Local storage*: The gateway stores the necessary data to automatically access the back-end. It also supports temporary storage in case of non-Internet connectivity. It is implemented using HTML Web storage[§§].
- *Connectivity service*: Checks the local storage for login data and attempts to automatically connect users with the cloud back-end. Applies data (Base64) encryption (or decryption) before data transmission.
- *Notification service*: Allows messages to be sent to users and system administrators (e.g. for taking actions).

Services running on the cloud back-end (private cloud or fog node) are implemented in PHP and Slim[***] and support the following actions:

- *Application Logic*: Implements the orchestration of the various services.
- *Identification and authorization*: It implements (a) identification services for users and manages their profiles and, (b) authorization services supporting access control based on user roles and access policies on services. It is implemented using the Keyrock Identify management GE[†††] of FIWARE which provides Single Sign On (SSO) service of users to services. All services above are protected by an OAuth2.0 mechanism[‡‡‡].
- *Monitoring service*: This service is aware of user activities, movements and requests and applies user specific rules for handling access rights (granted by a system administrator).
- *Publish and subscribe* services for associating facilities to Beacon sensors and for assigning user (gateway) information to administrators (responsible for user monitoring and providing advises). It is responsible for updating PACS on events (e.g. changes in the population of the sites in real time). It acts as a media broker (i.e. passing information to users subscribed to events) to inform or alert people (e.g. stay alert because of critical conditions, forward advertisements and new offers to users visiting a shopping mall, be informed of scheduled maintenance or access restrictions applying in buildings). It is implemented using the Publish/Subscribe Context Broker GE[§§§] GE of FIWARE.
- *Notification service*: Applies the information channel (e.g. toast, voice messages, e-mails etc.) through which employees can be informed of any incidents occurring. The services receive information from the Publish and Subscribe service.
- *Storage* services for user actions and access rights, sensors, messages, rules, events (e.g. rule violations) and actions undertaken by system administrators in response to events and user requests. It is implemented as a non-SQL database (i.e. a JSON storage service) using on MongoDB[****].
- *Connectivity services* for establishing secure network connections between cloud back-end and gateway.

[‡‡] https://cordova.apache.org
[§§] https://www.w3schools.com/html/html5_webstorage.asp
[***] https://www.slimframework.com
[†††] https://catalogue.fiware.org/enablers/identity-management-keyrock
[‡‡‡] https://oauth.net/2/
[§§§] https://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[****] https://www.mongodb.com

- *Interface*: This service allows system administrators to access all system data and to communicate with the PACS services and the users. It can run on mobile devices and can be accessed using a Web browser.

As shown in Fig. 1, there are different types of users that interact with the system. Each type of user is associated with different services that handle different types (classes) of information:

- *End-User Class*: Describes users interacting with the system. Each user is identified by ID, email, and a name being displayed. This class is associated with methods for importing and deleting users.
- *Permanent User Class*: Describes permanent users (i.e. registered users who have been accepted by the system administrator). Permanent users can issue access or assistance requests to employees.
- *Pending User Cla*ss: Describes temporary users (i.e. users who have applied for registration in the system but their application has not been accepted yet).
- *Access Log Class*: This information relates to the permanent users and their history data (access data).
- *User Location Class*: The objective of the class is to provide information about the user's current location.
- *Employee Class*: It is a subclass of permanent users. These are permanent users who are also employees. It keeps information about whether the employee has a fixed position in the building, and its location.
- *Area Class*: Defines categories of areas each one characterized by an identifier and its description. It is associated with methods for adding new areas, updating or deleting existing ones and also with methods for correlating an area with a Beacon sensor or for updating its Beacon sensor.
- *Public Area Class*: This class contains public areas (i.e. accessible by most users). It is described using several parameters that define the current population in the area, its capacity, a limit of completeness which if surpassed, a critical event is generated (e.g. dispatch personnel to deal with this situation or alert messages are sent to users).
- *Area Beacon Class*: Describes a Beacon sensor which is associated with an area.
- *Permissions Class*: Describes access rights for users visiting a facility. Access rights can be created, updated and lapse (e.g. if outdated).
- *Dispatch Request Class*: A user may issue a service request to the administrator of a facility. This class describes the request (e.g. a request has an identifier, a description that may specify an area identifier, an event, the sender and the receiver). It is associated with methods for creating and handling the request. This class is related to the Users class and the Employee class.
- *Alert Class*: Describes information about an event (i.e. an event identifier, an area). It is associated with methods for creating, manipulating, and ending the event.

3. Performance evaluation

In this section we study system scalability (i.e. how system response time increases with the number of connected users). We run an exhaustive set of experiments and we analyze the performance limits of the services running on the cloud back-end. All services run on the same VM, with the exception of the Publish/Subscribe Orion Context Broker that runs on a separate (second) VM, the JSON storage service that runs on a third VM and, the Identification – Authorization service (Keyrock Identity Management) that runs on a fourth VM. Therefore, the cloud back-end runs on four VMs. The Identity Management service is not expected to cause any performance bottlenecks as it is addressed only once by each user at login. Therefore, in the experiments below, the time for system login is not taken into account. However, all other VMs may receive up to a large number of simultaneous requests as the number of users and their requests increase. All measurements of time below account also for the time spent for the communication between VMs or between services within the same VM.

All VMs, with the exception of Orion Context Broker that runs on a shared VM and is installed on a remote FIWARE node (FIWARE is a federation of distributed cloud infrastructures), are hosted on the FIWARE node of TUC and have the following features: One virtual processor (x86_64 processor architecture, 2,800 Mhz, first level cache size 32 KB, second layer 4,096 KB cache size), 2,048 MB RAM, 20 GB hard drive capacity. Each VM runs Ubuntu Operating System 14.04 and Apache HTTP server.

The computational resource usage metrics are measured using the Linux top command[††††]. We use ApacheBench[‡‡‡‡] (Apache HTTP server benchmarking tool) to send multiple simultaneous requests to PACS. In ApacheBench we are opted to define the total number of requests and how many of them will be executed simultaneously. We start by sending 200 and 2,000 requests in a sequence (concurrency = 1). Table 1 and Table 2 summarize the results for each experiment.

Table 1: 200 requests (concurrency = 1)

Percentage of requested served	50%	65%	75%	80%	90%	95%	100%
Time (ms)	488	530	579	611	871	1,055	1,647

Table 2: 2,000 requests (concurrency = 1)

Percentage of requested served	50%	65%	75%	80%	90%	95%	100%
Time (ms)	485	505	551	583	986	1,095	1,254

We notice a very low resource usage when performing the above measurements. This is expected, as only one request is executed at any time. The data transfer rate is 0.61 Kbytes/s. In the following experiment we measure performance for concurrent requests (concurrency > 1). We execute 2,000 requests with concurrency = 40 and concurrency = 120. Table 3 summarizes the results of the experiment.

Table 3: 2,000 requests (concurrency = 40)

Percentage of requested served	50%	65%	75%	80%	90%	95%	100%
Time (ms)	1,908	2,101	2,238	2,338	2,708	2,859	2,478

Here the average CPU usage is 50% and the use of RAM is 150 MB. The data transfer rate is 7.04 KB/s. The average time per request is 1,039 ms. We notice a delay in the execution of our system, however it is still at a tolerable level. The delay is due to communication between services which is implemented using CURL[§§§§]. This results in a sharp increase in the required computing resources. Together with the need to simultaneously serve requests, the server is forced to switch to multitasking at the same time. Then we study how the processor usage increases, depending on the increase of simultaneous requests (Table 4).

Table 4: 2,000 requests (concurrency = 120)

Percentage of requested served	50%	65%	75%	80%	90%	95%	100%
Time (ms)	5,766	6,224	6,576	6,713	7,183	10,045	13,621

The data transfer rate is 7.61 Kb/s while the processor usage is almost maximum (~ 95%). The use of RAM is 420 MB. It is clear that the system has is handly responding to requests that are being sent continuously. Various tests have shown that this is due to the use of many CURL requests, that is, requests for communication between different services via the HTTP protocol, resulting in a high number of active requests. The above disadvantage, coupled with the slow communication, means that requests had to be active for a very long time, thus committing the available resources. This is also confirmed by the fact that an increase in the maximum number of available Apache threads (MaxRequestsWorkers parameter) is needed to further increase the number of parallel requests.

ApacheBench does not provide information about the time consumed by each VM. Table 5 shows this result. We run 2,000 requests (concurrency = 1). The average response time is 437 ms which accounts for the time spent on

[††††] https://www.linux.com/blog/15-simple-top-command-examples-linux-monitor-processes
[‡‡‡‡] https://httpd.apache.org/docs/2.4/programs/ab.html
[§§§§] https://curl.haxx.se/docs/manpage.html

Publish/Subscribe Context Broker of FIWARE, JSON storage and the core VM that implements the rest functionality on cloud back-end. Almost 75% of the response time is consumed on Publish/Subscribe Context Broker VM. This VM runs on a remote FIWARE node while, all other VMs run on the FIWARE node of TUC. We don't apply any optimization (we cannot influence operations on this VM as this is a shared VM servicing many applications at the same time on the cloud). Presumably, speeding-up the response time of this VM would require installation on the same FIWARE node together with all other VMs running PACs.

Table 5: Time consumed per VM (concurrency = 1)

VM	Network Delay	Core VM	Context Broker	JSON Storage
Time (ms)	46	17	331	43

Processing capacities may increase exponentially or raise restrictions for space or bandwidth (especially for concurrency > 1). A PACS may produce big amounts of data and requests, requiring large processing capabilities, which can surpass the capacities that our experimental system set-up is able to provide. In this set-up, most PACS core services are implemented in a single VM thus overloading the VM when the number of concurrent service requests exceeds a limit. An obvious solution to dealing with performance would be to employee additional VMs each running a single service (or a small group of services). Alongside, we can also allocate additional VMs implementing the same service (or groups of services) thus having more than on VM sharing the load.

4. Conclusions

PACS monitors peoples' activity and overall traffic in large residential infrastructures and provides prompt response in cases of critical events (i.e. overcrowding, health incidents etc.). The experimental results suggest that, usage capacities and response times of PACS can be improved significantly by applying certain optimizations on the implementation and placement of services on the cloud. As a future extension, we can implement a natural mechanism to provide or deny access to facilities (i.e. doors locks controlled by Bluetooth proximity sensors). Implementation of data analytics functionality on a public cloud is also an interesting direction for future work.

References

1. Zhang Q, Cheng L, Boutaba R. Cloud computing: State-of-the-art and research challenges, *Journal of Internet Services and Applications* 2010; 1(1):7-18.
2. Sotomayor B, Montero RS, Llorente IM, Foster IT. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing* 2009; 13(5):14-22.
3. Bonomi F, Milito RA, Zhu J, Addepalli S. Fog computing and its role in the internet of things. *MCC@SIGCOMM, ACM* 2012; p.13-16.
4. Soultanopoulos T, Sotiriadis S, Petrakis EGM, Amza C. Data management of sensor signals for high bandwidth data streaming to the cloud. *Sarnoff Symposium, IEEE* 2016; p. 53-58.
5. Bassi A, Bauer M, Fiedler M, Kramp T, Kranenburg Rv, Lange S, Meissner S. Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model, Springer*, Heidelberg*; 2013.
6. Preventis A, Stravoskoufos K, Sotiriadis S, Petrakis EGM. IoT-A and FIWARE: Bridging the Barriers between the Cloud and IoT Systems Design and Implementation. *International Conference on Cloud Computing and Services Science (CLOSER)*, 2016; p.146-153.
7. Douzis K, Sotiriadis S, Petrakis EGM, Amza C. Modular and generic IoT management on the cloud. *Future Generation Computer Systems* 2018; **78**:369-378.
8. Antonopoulos F. Access Control System for Large Residential Infrastructures on the Cloud. *Diploma Thesis, School of Electrical* and Computer Engineering, Technical University of Crete March 2017; http://www.intelligence.tuc.gr/show_publications.php?arg=PERSON_S&id=1&ptype=8.