CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

The EnerGAware Middleware Platform

Paulo Barbosa
António Barros*
Luis Miguel Pinho*

*CISTER Research Centre

CISTER-TR-170801

2017/10/29

The EnerGAware Middleware Platform

Paulo Barbosa, António Barros*, Luis Miguel Pinho*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: 1130648@isep.ipp.pt, amb@isep.ipp.pt, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

More and more cyber-physical systems and the internet of things push for a multitude of devices and systems, which need to work together to provide the services as required by the users. Nevertheless, the speed of development and the heterogeneity of devices introduces considerable challenges in the development of such systems. This paper describes a solution being implemented in the setting of a serious game scenario, connected to real homes energy consumption. The solution provides a publish-subscribe middleware which is able to seamlessly connect all the components of the system.

The EnerGAware Middleware Platform

Paulo Barbosa, António Barros, Luís Miguel Pinho
CISTER Research Center
School of Engineering of the Polytechnic Institute of Porto
Portugal

***Abstract*—More and more cyber-physical systems and the internet of things push for a multitude of devices and systems, which need to work together to provide the services as required by the users. Nevertheless, the speed of development and the heterogeneity of devices introduces considerable challenges in the development of such systems. This paper describes a solution being implemented in the setting of a serious game scenario, connected to real homes energy consumption. The solution provides a publish-subscribe middleware which is able to seamlessly connect all the components of the system.**

Keywords—internet of things; service-oriented architecture; publish-subscribe; middleware

I. INTRODUCTION

The increasing pervasiveness of Cyber-Physical Systems (CPS) and the Internet of Things (IoT), is transforming the world we live in a set of highly dense and heterogenous complex computing systems which are able to sense and actuate the physical environment, connecting it with the virtual world at our fingertips, on our computers or smartphones. This pervasiveness is being pushed both by technological advances, which allow cheap and widespread CPS/IoT devices, and by increasing user demand for additional services. Although with several advantages, this trend is leading to a multitude of systems being deployed, most of the times using incompatible technologies and communication systems.

In this regard, providing value requires that systems integrate seamlessly, with an easy and quick access to the end user, most of the times a non-expert on technology. This is more and more important in several application domains, even more in those targeting the general public, such as home energy-efficiency systems and applications.

The EnerGAware project [1,2] tackles one of such cases with a serious online game being used to enhance energy users' behavioral change through education and training. This follows from results that point out the effectiveness of serious games in domestic energy consumption [3], which concluded that gamification and serious game can be of value for energy consumption, conservation and efficiency. The project addresses existing houses, as the building sector currently accounts for 40% of energy use in most countries [4] and has the greatest energy saving potential [5]. As a consequence, the cornerstone of the European energy policy has an explicit orientation to the conservation and rational use of energy in buildings [6]. As buildings tend to have long lifetimes, to achieve significant impact in the short- and medium-term the challenge must be focused on the existing buildings.

The project has deployed a real-life conditions pilot in a set of houses located in Plymouth (United Kingdom). The energy consumption of houses, as well as the awareness, attitudes, engagement and self-reported behaviours of the tenants are being assessed both before and after the implementation of the serious game [1]. Once the project is finished, it will provide more quantitative empirical research on the effectiveness of serious games within the domain of domestic energy reduction.

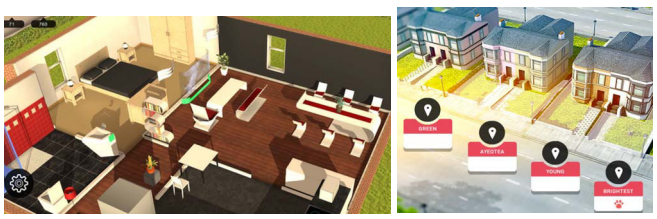

Fig. 1. Game scenarios

The serious game [7] is based on pseudo-realistic scenarios (Figure 1); the requirements analysis phase [8] and focus groups concluded that this would be better than a fantasy world (or sci-fi, or cartoon) and better than a fully-realistic simulation. The pilot includes energy (electricity and gas consumption) monitoring through an infrastructure installed in the homes, allowing the game to display to the user real energy savings (weighted according to the climate severity). Players making real energy savings receive rewards in the game.

The monitoring infrastructure uses a proprietary communication system, being the data available through a specific server (Concordia [9]). Considering that the energy consumption is related to the weather conditions, daily weather parameters, especially air temperature, are also retrieved from a weather platform.

To connect all systems, and aggregate and distribute all the data (energy consumption, weather and game experience), an IoT middleware was designed. The middleware uses Web of Things technologies providing generic and flexible REST-based APIs with messages based on URIs/URLs (as web services) [10]. This allows that a small number of generic methods can create a consistent, interoperable API [11].

[1] Preliminary results show a daily electricity saving ranging from 3 to 10% [1]

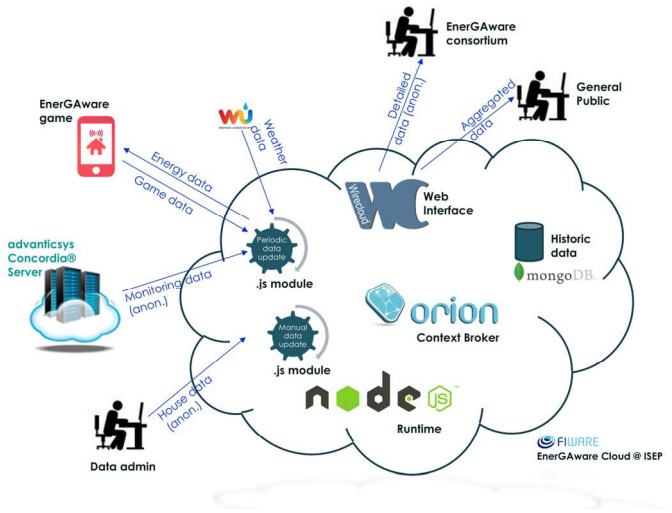

Fig. 2. System Architecture

With respect to service management, the middleware considers an architecture consisting of a set of interconnected components which may be co-located and/or distributed over a network of communicating computer nodes. The approach tackles the limitation of Service Oriented Architectures (SOA) using a Publish-Subscribe approach, removing the strong coupling of the client/server paradigm [12].

Although the middleware was specified and built for the purpose of the project, it was designed to be generic and support a multitude of house devices, and multiple applications or games, being also scalable in terms of load. This paper provides an overview of the main components of this middleware and how it is being used in the scope of the EnerGAware pilot.

The paper is structured as follows. Section II presents an overview of the architecture of the middleware. Afterwards, Section III describes the publish-subscribe approach used, while Sections IV and V describe the core and application modules. Section VI provides an evaluation of the implementation. Finally, Section VII concludes the paper.

II. The EnerGAware Middleware

A. Project Requiremnts

The EnerGAware Middleware is required to maintain a repository of different types of data, retrieved from different sources (as represented in Figure 2):

- Energy consumption readings, available from the Concordia Proprietary Server. This server is responsible to collect data received from the monitoring infrastructure and provides a set of services to export the energy consumption.
- Local (in this case Plymouth) weather data, available from weather services (currently Weather Underground). The weather data can be provided by automatic online sources or by periodic files.
- Pilot households' game experience are available from the Serious Game Server. This server harvests the gaming experience data, and provides a set of services that aggregates and exports the relevant game experience data.

Besides collecting and storing the mentioned sets of data, the Middleware must export the same data in a suitable format for a post analysis conducted by the specialist partners of the EnerGAware consortium. These data are anonymized, such that it allows partners to discriminate between pilot homes, but it is impossible to identify the physical houses to which the data refers to.

The game incorporates a system of player rewards determined by energy savings in the real, physical world, such that these rewards can be used by the player to improve his progress in the game. The EnerGAware Middleware must compute these real-world energy savings and provide this data to the Game Server.

Furthermore, the EnerGAware Middleware must provide web services to conveniently export relevant data:

- To the Game Server: computed values of energy savings for each individual home.
- To the EnerGAware consortium partners: aggregated energy meter readings, weather data and game play experience.

Finally, data such as overall energy consumption and achieved reduction may be displayed to the general public, after being anonymized and aggregated, such that individual homes cannot be discerned nor identified.

B. Technical Architecture

The Middleware is built upon a FIWARE platform [13] which establishes a set of Application Programming Interfaces (API) which facilitates the development of applications for the IoT. The FIWARE platform is supported by the Future Internet Public-Private Partnership (FI-PPP) project of the European Union [14], and provides in public access an open-source reference implementation of each of its components. Typical IoT applications acquire data from multiple and diverse sources types but all related to a specific context, which is then processed and registered, such that applications can answer requests on the working context. Nevertheless, and although FIWARE provides a multitude of components, the scalability and interoperability requirements considered for the middleware required the development of a set of new service modules.

Therefore, the EnerGAware Middleware functionality is achieved by several components (represented in Figure 3):

- FIWARE modules: the Orion context broker (manages the information including registrations and subscriptions of publishers/subscribers of data), the mongoDB persistent database and the Wirecloud web user interface;
- Service modules, specific to the middleware, handling connections and data acquisition and export.

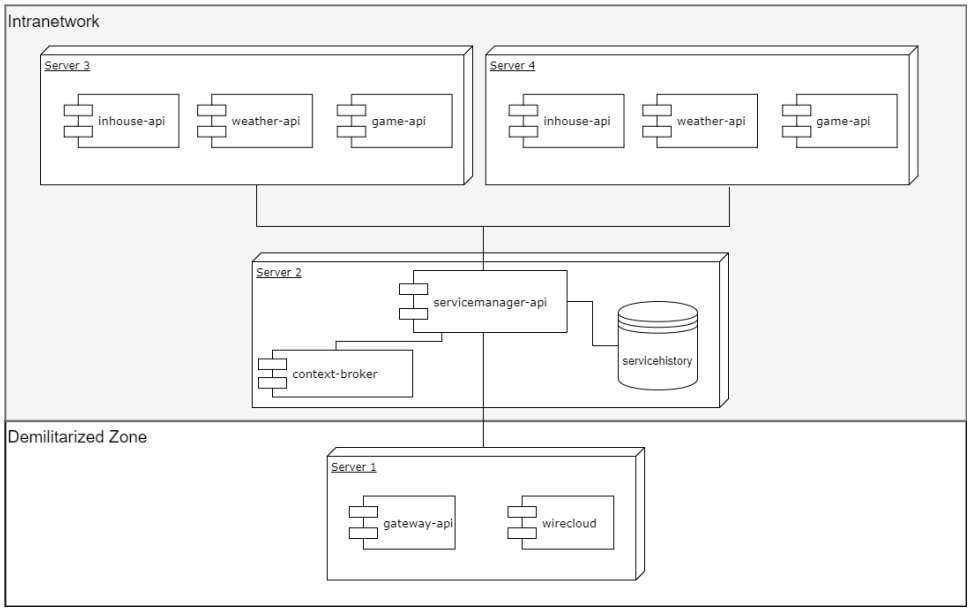

Fig. 3. Midddleware architecture

The service modules are responsible for the acquisition and processing of data from the energy meter readings, the game experience and the weather conditions. In addition, they handle the requests of specific data (either bulk data for energy consumption behaviour analysis or game rewards based on energy savings). These services execute on top of a Node.js [15] runtime environment. The Node.js is an open-source event-driven Javascript runtime that executes on the server side. Node.js executes on a single thread, and I/O calls are treated asynchronously (non-blocking), such that other concurrent operations can be executed; when a call finishes, a callback function is called to process the results. This characteristic allows Node.js to support multiple concurrent requests without the execution cost of thread context switching and the possibility of deadlock, as no locks are used.

The Middleware stores the acquired data persistently in a MongoDB [16] database. MongoDB is an open-source document-oriented Database Management System (DBMS). Unlike traditional relational databases that organise data into tables and relations, MongoDB organises data in JSON documents (i.e. records) that are gathered in collections, instead. Furthermore, MongoDB allows dynamic schemas, such that documents in the same collection do not need to present the same format. The middleware is independent of the database configuration (centralized, decentralized, clustered, etc.) although for the specific implementation of the project, a centralized databased (accessed by all services) is used.

The web user interface is built upon Wirecloud [17], a FIWARE application mashup component. Wirecloud is a web mashup platform that allows an end user to create a personal dashboard from independent widgets. Wirecloud widgets are developed in HTML and Javascript, and must comply with a set of rules, such that they can be registered into a Wirecloud widget repository where users can select and load preferred widgets into his/her dashboard.

Services that compute large quantities of data necessarily have to manage each client request effectively and distribute computational tasks to provide a decent response-time. As such, the idea of the proposed system architecture is to transfer the responsibility of solving these problems from each service system to a single system, to ease the development of services.

The designed architecture provides different services, such as energy consumption to weather statistics, from different sub-systems in a centralized manner. Acting as a platform, the architecture allows an effective management of service requests recurring to load-balancing operations.

The service modules architecture, depicted in Figure 3, is composed by two types of modules:

- Core: the core modules (*gateway*, *servicemanager*) enable all the inherent features of the architecture, mainly the management of service requests and inter-operability of services.
- Applicational: the applicational modules (currently *inhouse*, *game*, *weather*) provide the domain logic for the end-user, such as energy consumption or game statistics. The services provided by the applicational modules are provided through the core systems.

Since the core modules act as proxies and are capable of interrupting and modifying services, they hold fundamental responsibilities for the services consumption. Therefore, it is important to guarantee the security (integrity and availability) of the services.

The designed solution divides the deployment of all modules in two environments:

- Intra-network: this environment is inaccessible to the end-user, and only the deployed modules in the demilitarized zone have access to this environment. The *servicemanager* and the publish/subscribe module,

context-broker and the remaining applicational modules are deployed in the intra-network environment. By restricting the access from the end-users, especially malicious users, to this environment it limits the exposure of vulnerabilities.

- Demilitarized Zone: this is the only environment accessible to the end-user and exposes only the front-end modules, in this case the *gateway* and *wirecloud* modules.

The middleware supports security both by HTTP over TLS (HTTPS), with configured accounts, as well as IP filtering, for more restricted accesses.

III. Publish-Subscribe Architecture

The proposed architecture incentives the deployment of the same application modules in different machines, which requires ensuring the data consistency between these databases. By using the FIWARE publish/subscribe *context-broker*, the application components do not have the responsibility of updating their databases.

When the *servicemanager* integrates new services, it automatically subscribes the service's module to the *context-broker* if it is necessary. In case of an *inhouse* module, whenever it is integrated into the *servicemanager*, it will automatically receive daily request posts with the most recent energy consumption measurements.

Regarding subscription, as represented in Figure 4, only the *servicemanager* has the responsibility of subscribing modules in the *context-broker*. Whenever a new module joins the *servicemanager*, if necessary, the *servicemanager* automatically subscribes the joined module to the *context-broker*.

Scripts, deployed in the *service-manager* module, request the most recent content from their content providers (e.g: the content provider of the *inhouse* module is the meter data collection system) and in success immediately update the *context-broker* with the new values. Finally, the *context-broker* notifies every subscribed module, as represented in Figure 5.

Currently there are three available publication services:

- Energy publication service: every *inhouse* module needs to be subscribed;
- Game publication service: every *game* module needs to be subscribed;
- Weather publication service: every *weather* module needs to be subscribed.

IV. Core Modules

As explained, there are two core modules, the *gateway* and *servicemanager*, both acting as proxies, rerouting all service requests from the user to the requested service module.

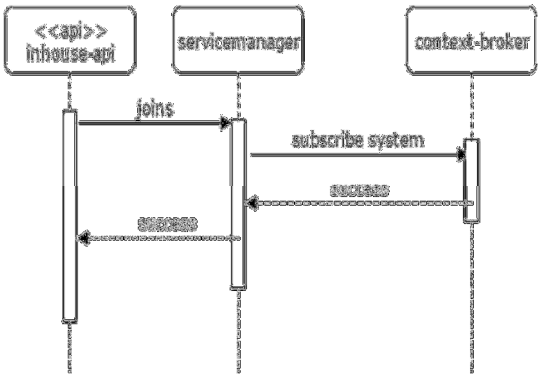

Fig. 4. Subscription sequence diagram

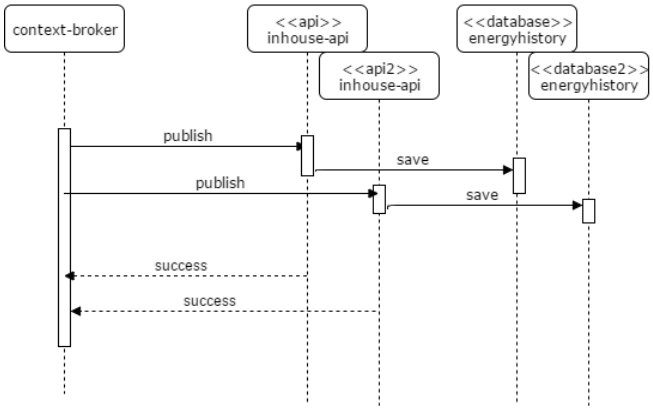

Fig. 5. Publish sequence diagram.

A. Gateway Module Description

The *gateway* consists in a reverse proxy and consequently is the only module that is directly interacted by the user. Each service request received is rerouted to the *servicemanager* module deployed in a different machine than the *gateway*. The main purpose of the *gateway* is to hide the existence of the other modules, including the *servicemanager* to avoid any security vulnerability exposure.

Furthermore, the *gateway* can also have features like:

- Monitoring of client requests: by having an history of IP addresses it is possible to monitor the behaviour of each user;
- Application Firewall: since all requests pass through the same point, in order to avoid common cyber-attacks, like Denial of Service, the *gateway* can protect the applicational modules in run-time by blocking IP addresses;
- Encryption: the gateway can also encrypt (SSL) the data retrieved from the applicational modules, like *inhouse*, guaranteeing integrity and confidentiality security properties.

B. Service Manager Module Description

The most relevant module of the architecture is the *servicemanager*. It is the module that aggregates all applicational modules, and distributes all service requests recurring to load-balancing operations. At the same time, it allows the integration of new REST services without the need of interrupting itself.

Regarding the calculations made for load balancing, currently the *servicemanager* uses Round-Robin load balancing algorithm. Though it is not one of the most accurate distribution methods of traffic, it is a very simple method to implement. The applicational service's modules are grouped in a list and the algorithm loops the list and distributes each request from the top module to the lower module located in the list. Whenever the algorithm finally distributes the requests to every module located in the list, it loops again the list, always in the same order (top-down). It should be noted that other load balancing algorithms can be used in the service-manager.

Regarding the interoperability of services, the *servicemanager* can aggregate any REST service without any restriction and interruptions. This is done using the Dynamic Routing characteristic of the Node.js web application framework, Express [18]. Periodically, the module searches for any "json" file, to dynamically add and update the services rerouted by the *servicemanager*.

V. Applicational Modules

As the middleware is currently deployed to support the EnerGAware pilot, it provides three application modules to handle the different types of data services: *inhouse*, *game* and *weather*.

A. Inhouse Module Description

The *inhouse* module (Figure 6) provides the readings of real energy consumption, both electricity and gas, which are automatically collected periodically. With this data, the module is capable of providing consumption analysis such as the evolution of energy consumption over time, for each house.

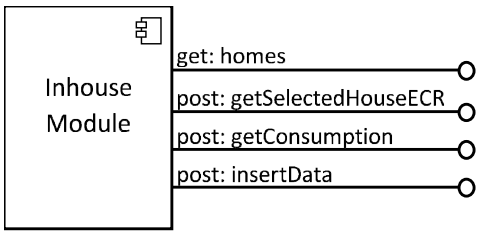

Fig. 6. Services provided by the inhouse module

Regarding data persistency, the consumption readings are collected every 15 minutes for each house, collecting at the end of each day around 100 signals per house. When dealing with only 100 houses, in just over a month there will be around 300 thousand signals that should be stored and rapidly processed by the *inhouse* module. To guarantee scalability, each time period signal of each house is stored in a separate document.

The EnerGAware project operates with two different consumption readings: electricity and gas consumption. Both signals are collected at the same instant and therefore are stored in the same database document. Average size of a stored document is 500 bytes, thus one year of collected readings of only one house accounts for 17.52 megabytes (MB). Since the EnerGAware project monitors 80 dwellings, each month the database size expands 119 MB and currently holds consumption readings since 2015. The data size of the services can range from few hundred bytes to larger values, dependent on the requested period (e.g., the service *getConsumption* has a data size of 27KB per requested day).

B. Game Module Description

The *game* module (Figure 7) is responsible for registering the overall progression of each user on the game by home. Every day this module updates its collection with the most recent game progressions of all users.

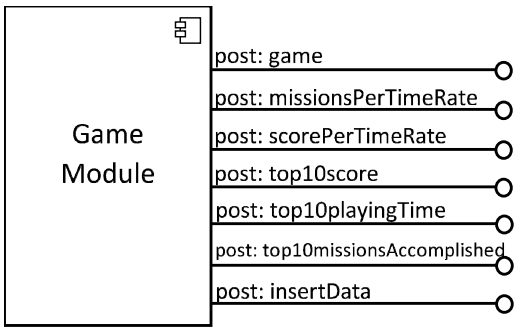

Fig. 7. Services provided by the *game* module

Regarding data persistency, the module stores each user's game progression in a separate document, in the MongoDB database.

C. Weather Module Description

The *weather* module (Figure 8) has the responsibility of providing weather data of every day, such as temperature and humidity, from a specific location.

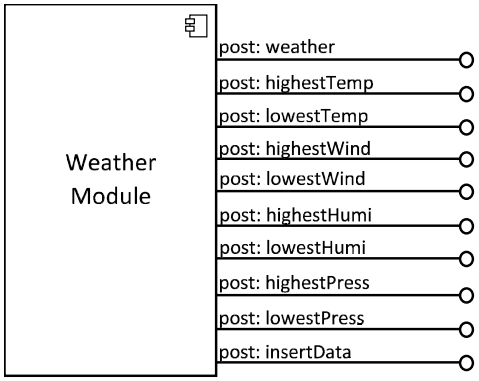

Fig. 8. Services provided by the w*eather* module

Regarding data persistency, the module stores daily weather measurements, mandatory for the service *getSelectedHouseECR* of the *inhouse* module, to compensate the effect of outside temperature in the energy consumption of a house. Each daily weather data is stored in separate database documents.

VI. Evaluation

In order to showcase the feasibility and scalability of the middleware, an evaluation was performed by increasingly add servers to process incoming requests. The evaluation was performed in a Google Cloud Platform[2] virtualized environment with:

- The service manager service is provided by one instance (8 cores @ 2.3 GHz)
- The applicational services are provided by 1 to 9 virtualized servers (each with 2 Intel Xeon @ 2.30 GHz)

Requests were simulated using the Apache Benchmark tool [3], configured for 100K requests (1K simultaneous); multiple runs were executed, with mean values for processed request/second being presented in Figure 9 (standard deviation ranging from 1 to 8%).

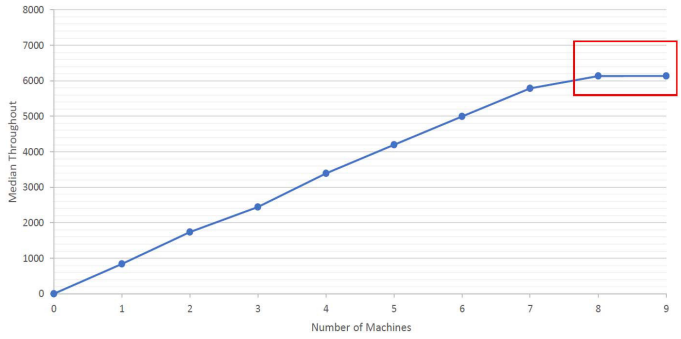

Fig. 9. Evaluation of middleware scalability

The analysis shows how the system is able to scale linearly with the number of applicational services up to the limit of full occupancy of the service manager server (when 9 applicational servers are used).

VII. Conclusions

This paper provided an overview of the architecture and implementation of the EnerGAware Middleware, a scalable service-oriented system which is used to interoperate between energy monitoring systems and a serious online game. Although implemented for a specific application it has been designed to be generic, allowing easy extension and scalability to be able to handle, even if simultaneously a multitude of systems and applications. The middleware is currently being used to support a pilot for a European R&D project for energy efficiency.

[2] https://cloud.google.com/

[3] https://httpd.apache.org/docs/2.4/programs/ab.html

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No 649673 (EnerGAware); also supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the Portugal2020 Program, within the CISTER Research Unit (CEC/04234).

References

[1] M. Casals, M. Gangolells, M. Macarulla, A. Fuertes, V. Vimont, L.M. Pinho, "A serious game enhancing social tenants' behavioral change towards energy efficiency", 2017 GIoTS Workshop on Energy Efficient Solutions Based on IoT - EESIoT 2017, June 2017

[2] EnerGAware project, "Energy Game for Awareness of energy efficiency in social housing communities", EU funded project, contract number: 649673, 2016. Available at: http://energaware.eu/, last accessed September 2017.

[3] D. Johnson, E. Horton, R. Mulcahy, M. Foth, "Gamification and serious games within the domain of domestic energy consumption: A systematic review", Renewable and Sustainable Energy Reviews, vol. 73, pp. 249-264, 2017

[4] European Union, "Directive 2012/27/EU of the European Parliament and of the Council of 25 October 2012 on energy efficiency, amending Directives 2009/125/EC and 2010/30/EU and repealing Directives 2004/8/EC and 2006/32/EC", 2012. Available at: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32012L0027, last accessed September 2017.

[5] International Energy Agency, "Energy performance certification of buildings. A policy tool to improve energy efficiency", 2010. Available at: http://www.iea.org/publications/freepublications/publication/buildings_certification.pdf, last accessed September 2017.

[6] E. Asadi, M. Gameiro da Silva, C. Henggeler Antunes, and L. Dias, "Multi-objective optimization for building retrofit strategies: A model and an application", Energy and Buildings, vol. 44, pp. 81-87, 2012

[7] Energy Cat: the House of Tomorrow, http://energycatgame.com, last accessed September 2017.

[8] EnerGAware project, "D2.3. Game requirements", 2016. Available at: http://www.energaware.eu/downloads/EnerGAware_D2.3_Game_Requirements_r1.pdf, last accessed September 2017.

[9] Concordia Cloud Platform, https://www.advanticsys.com/services/lot-of-options/, last accessed September 2017.

[10] Erik Wilde, "Putting Things to REST", UCB iSchool Report 2007-015, School of Information, UC Berkeley, November 2007

[11] D. Guinard, I. Ion, S. Mayer, "In Search of an Internet of Things Service Architecture: REST or WS-*?" A Developers' Perspective", Mobile and Ubiquitous Systems: Computing, Networking, and Services. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering Volume 104, 2012.

[12] Albano, M, Ferreira, L, Sousa, J, "Extending publish/subscribe mechanisms to SOA applications", Work in Progress Session, 12th IEEE World Conference on Factory Communication Systems (WFCS 2016). 3 to 6, May, 2016. Aveiro, Portugal.

[13] FIWARE, http://www.fiware.org, last accessed September 2017.

[14] Future Internet Public-Private Partnership Programme, http://www.fi-ppp.eu/, last accessed September 2017.

[15] Node.js runtime, https://nodejs.org/, last accessed September 2017.

[16] MongoDB, https://www.mongodb.com, last accessed September 2017.

[17] Application Mashup – Wirecloud, https://catalogue.fiware.org/enablers/application-mashup-wirecloud, last accessed September 2017.

[18] ExpressJS, https://expressjs.com/, last accessed September 2017.