Employing an IoT Framework as a Generic Serious Games Analytics Engine

Luca Lazzaroni, Andrea Mazzara, Francesco Bellotti(✉), Alessandro De Gloria, and Riccardo Berta

DITEN, University of Genoa, Via Opera Pia 11A, 16145 Genoa, Italy
{francesco.bellotti,alessandro.degloria,riccardo.berta}@unige.it

Abstract. This paper proposes the use of a new data toolchain for serious games analytics. The toolchain relies on the open source Measurify Internet of Things (IoT) framework, and particularly takes advantage of its edge computing extension (namely, Edgine), which can be seamlessly deployed cross-platform on embedded devices and PCs as well. The Edgine is programmed to download from Measurify a set of scripts, that are periodically executed so to get data from sensors, pre-process them and send the extracted information to the Measurify APIs. Virtual sensors can be built in game engine scripts. This paper describes the implementation of the plug-in which deploys Edgine in Unity 3D, allowing an easy delivery of virtual sensor information to Measurify. Just as a proof of concept, we present the utilization of the whole chain within a trivial game scene, showing the application development efficiency provided by a tool which is made available open source to researchers and developers.

Keywords: Internet of Things · System architecture · Game development · Game engine plug-in · Virtual sensors · Reality-enhanced games

1 Introduction

Serious game analytics is an emerging research field, which aims at turning gameplay data into "valuable analytics or actionable intelligence for performance measurement, assessment, and improvement" [1]. Not only does this high-level goal require the need for developing serious games able to provide didactically relevant quantitative information, but also efficient integration of modules for analytics collection and management [2].

Dealing with big data is a general and common problem, particularly in the context of the Internet of Things (IoT), for which several solutions have been developed, especially using cloud-based frameworks. In this paper, we are interested in investigating the application to the serious game context of Measurify (previously known as Atmosphere), an open source IoT framework built around the concept of measurement, which looks quite relevant to the goal of assessing and supporting a learner [3].

Actually, that article already briefly reports the experience of a University spin-off company applying Measurify to design a data management model for the use case of a 3D virtual reality simulator for emergency room personnel instruction. The goal of the instructional tool was to assess the performance of a doctor by evaluating the effects of his interventions on the various patients. To this end, the Measurify data model was designed to record (i) the state of a patient, that is characterized by a set of time-evolving parameters; (ii) the actions performed by the doctor (e.g., how he interacted with the available simulation tools); and (iii) the events in the simulation (e.g., changes in medical equipment availability).

In this paper, we intend to go more in depth about the utilization of Measurify as an analytics engine, particularly investigating its extension towards edge computing [4], namely the recently released Edgine module [5], aimed at supporting a configurable provision of measurements from the field.

The remainder of the paper is organized as follows. Section 2 provides the related work, while Sect. 3 and 4 describe the Measurify and Edgine systems, respectively. Section 5 is devoted to the application of the IoT framework to a gaming environment. Section 6 draws the conclusions on the presented work and outlines possible directions for future research.

2 Related Work

Serious games analytics is a research field involving empirical research methodologies, including existing, experimental, and emerging conceptual frameworks, from various areas, such as: computer science, software engineering, educational data mining and statistics information visualization [1].

The literature provides several examples of serious game analytics (or game learning analytics) applications. [6] explored players' gameplay patterns to understand player dropout in the Quantum Spectre science game. Based on their results, the authors argue that modeling player behavior can be useful for both assessing learning and for designing complex problem solving content for learning environments. [7] present a novel method that suggests curricular sequencing based on the prediction relationship between math objectives. The authors argue that their method can potentially be applied to data from a wide range of games and digital learning platforms, enabling developers to better understand how to sequence educational content. [8] explore existing log files of the VIBOA environmental policy game. Our aim is to identify relevant player behaviours and performance patterns. The correlation analysis suggests to the authors a behavioural trade that reflects the rate of "switching" between different game objects or activities. The authors also established a model that uses switching indicators as predictors for the efficiency of learning. [9] developed a mobile game to support the transfer of theoretical knowledge on resuscitation training in case of cardiac arrest. To analyse a large and heterogeneous (in terms of sources and quality) data-set collected from 171 players, the authors applied different types of data modeling and analyses. This approach showed its usefulness and revealed some interesting findings.

[10] discuss a conceptual model (ecosystem and architecture) aimed to highlight the key considerations that may advance the current state of learning analytics, adaptive learning and SGs, by leveraging SGs as an suitable medium for gathering data and performing adaptations. [11] describe two key steps towards the systematization of game learning analytics: 1), the use of a newly-proposed standard tracking model to exchange

information between the serious game and the analytics platform, allowing reusable tracker components to be developed for each game engine or development platform; and 2), the use of standardized analysis and visualization assets to provide general but useful information for any SG that sends its data in the aforementioned format.

3 Measurify APIs

Measurify is a cloud-based Application Programming Interface (API) designed to support an efficient workflow for preparing different measurement-based data-rich applications, especially but not exclusively from IoT devices [Atmos]. The framework integrates APIs implementing Representational State Transfer (REST) services [12], that provide a platform-independent HTTP interface (e.g., [13]). Measurify exploits the open source MongoDB non relational database technology for data management, and on the NodeJS programming language, supporting by design seamless cross-platform portability, without being locked to a specific cloud vendor technology (e.g., Amazon Web Services for the Internet of Things (AWS IoT) [14] or Microsoft Azure cloud platform [15]), which is an important limit of current approaches. According to the RESTful programming paradigm, the API exposes a set of resources, A resource is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it. Measurify defines a limited but powerful set of resource types (Table 1), that have been suited to create applications in international research projects in the automotive and health domains [16–18].

Table 1. Outlook of the measurify resources

Resource	Description
Measurement	Is the actual data taken from the field (e.g., a temperature value)
Feature	Is the type of a measurement. A type could be complex, with several dimensions, of various numeric/text types (e.g., sets of position records). Measurements could be of different types
Device	Is the instrument (e.g., a thermometer) through which the measurement is obtained
Thing	Is the subject of a measurement (e.g., a weather station located on a mountain)
Tag	Labels that can be attached as attributes to other resources – typically measurements, features, things and tag themselves
Constraint	A generic relationship between two resources (e.g., to support automatic generation of graphical user interfaces)
Computation	Is the actual data taken from the field (e.g., a temperature value)

The first step of the workflow supported by Measurify consists of the domain modeling, where the field objects (i.e., the objects involved in the measurements) are mapped to the API's resources. In this phase, the IoT application designer has to define features devices, tags, and constraints. In the configuration (or deployment) step, the above designed model resources are straightforwardly encoded in a.json file, which is POSTed to the framework APIs, so to create the Application Database (ADB) structure. In the regime phase, the framework manages the ADB, allowing (i) dynamic insertion/update of users, things, field measurements and computation requests; and (ii) retrieval of results in terms of things, measurements and computation outcomes. The ADB structure can be updated during the operation as well, by POSTing/PUTting/DELETEing features, devices, and tags. All these actions happen only through the exposed resource routes (i.e., standard functions through which objects can be accessed), with the well known advantages of the RESTful approach in terms of scalability, encapsulation, security, portability, platform independence, and clarity of terminology and operations.

In the next section, we present the extension of Measurify in the direction of the edge computing, namely the Edgine (Edge Engine) module.

4 Edgine

Measurify is a cloud-based system designed to host applications receiving measurements data through a REST API. According to the edge computing paradigm, computation is being moved towards the edge, in order to optimize exploitation of resources. Availability of computing capabilities on the edge, close to the field sensors implies the possibility to implement sensor hubs, typically represented by microcontrollers, that can be configured from remote in order to deliver the dynamically needed information.

This is the founding idea of Edgine, an edge system designed to support efficient development of integrated IoT applications. Edgine features an HTTP communication interface with the cloud (particularly with the Measurify APIs) in order to download configuration settings and scripts that are executed locally.

The system code consists of two main parts: an initial one and a loop one. In the start phase, which is implemented inside the setup() function, the Edgine software authenticates itself and connects to the APIs to download its configuration parameters and the list of scripts to be executed. Then, during the infinite cycle loop, the software continuously executes each assigned script in sequence, processing the data and sending them to the Measurify APIs.

Each device associated with Measurify is described by a JSON (JavaScript Object Notation) file which includes, among others, the feature field, describing the expected type of the measurements to be delivered, and the scripts field, including all the scripts to be downloaded and executed by the particular device. A script consists of a sequence of instructions that typically conclude with the delivery of the processed data to Measurify. The raw data processing instruction types currently include simple arithmetic functions, computation of simple statistic operations (e.g., min, max, mean, median, stdev), and accumulation of values in a (sliding) window.

A key design requirement for Edgine is platform independence, in order to make this paradigm widely available across edge devices. Thus, we strived to keep the system as independent as possible from the hardware. To that end, classes have been created to allow developers to switch from Windows/Linux/Mac PC platforms to Arduino through the use of C preprocessor macros.

The differences between the two main platform types concern the Internet connection. In the Arduino case, an automatic connection to a WiFi network (predefined in the code) is performed, and the system is also designed to perform a reconnection in case of signal loss. In the Edgine version for PC-type machines, on the other hand, the system waits until a network connection becomes available, before trying to perform the initial authentication operations. At runtime, data that cannot be sent due to lack of connectivity are accumulated in a buffer, waiting for a connection.

Up to now, Edgine has been used in some IoT applications, concerning the monitoring of environmental parameters such as temperature, humidity, light conditions [19].

5 Serious Game Application

While Edgine has a clear application to edge devices, such as microcontrollers, mobile phones and automotive electronic central units (ECUs), its concept is abstract and can be applied to any device processing raw data. Thus, the Edgine can also be employed in a serious game with a goal to get measurements from it and pre-process them according to some basic scripts. Everything is dynamically programmed from remote. Conceptually, this corresponds to applying the IoT sensor measurement concept not only to reality but also to a game's virtual reality.

To achieve this goal, it is necessary to insert the Edgine system inside the target serious game. This can be achieved by integrating Edgine inside a game engine. This is a general solution, that could be seamlessly applied to any game implemented with a given game engine.

For the case of Unity 3D, a widely used game engine, the idea is to develop a plug-in that would wrap Edgine, making it available to any game. For instance, it would be possible to configure the Edgine so to send to the measurement API (i.e., the generic analytics engine) information such as: the average distance covered by a virtual character in the last minute, the number of collected objects, the number of correctly answered quizzes, etc. The key advantage – especially in a software engineering perspective of efficient development - is the abstract and generic architectural approach, which makes the solution generally applicable.

In the following, we describe the implementation of Edgine in Unity 3D. In this game engine, C# scripts are usually employed to create components and specify their behaviour. Scripts can exploit libraries of additional functionalities, made available in Unity as plug-ins. Plug-ins are platform-specific native code libraries that can access operating system functions and other third-party libraries, that would not otherwise be available to Unity developers.

A Unity plug-in is equivalent Dynamic-Link Library (DLL), and can be efficiently developed with Visual Studio, which is quite similar to the Unity Integrated Development Environment (IDE). In the DLL, we implemented a single class, which wraps the methods intended to be exposed to Unity C# scripts. Inside the header, it is necessary to define such functions with the declspec(dllexport) attribute. Moreover, through the extern "C" label, it is indicated to the compiler that the C linking conventions should be employed for such function. This is necessary for a correct export, because, otherwise, C++ compilers would perform the mangling process, making them unreachable from Unity. The mangling is a technique used to distinguish functions that have the same name (overloading). The technique manipulates the name during compilation, so that it becomes unique. But this would make the final name unknown to the Unity developer. The C language, on the other hand, does not support function overloading, thus does not perform the mangling. Thus, the binary file generated by the compilation process contains the original method names. However, since the C linking convention is adopted, it is necessary that the input and output data types are those of the C, even if the body of the function is written in C++.

According to the Edgine programming model presented in the previous section, the class exposes two methods: Setup() and Action(). The first one will be used at start-up, the second one will be employed inside the game engine's update loop. For the Unity implementation, we let that the service descriptive features (device, thing, feature, username e password) can be specified by the Unity user as parameters of the two exposed functions and are not hard-wired in the source code, which is the approach employed in the edge/embedded environment. Intuitively, the user can specify the names of the resources related to a correct delivery of measurements to Measurify. The Action() function, takes in input the name of the feature and the measurement value, which will need to be delivered to the cloud.

The compiled Edgine project produces five dll files: NativeCppLibrary.dll, pcred.dll, PocoFoundationd.dll, PocoNetd.dll, zlibd1.dll, of which the first one is the dynamic library containing the two functions mentioned above. In order to use the plug-in inside Unity, it is necessary to copy all these .dll files in the project directory.

As a proof of concept, we implemented a very simple application example of collection of data from a game scene and their delivery to the cloud. Particularly, we tracked the collisions of a ball with the borders of a squared region (Fig. 1). Every time a collision is detected with the wall, a variable is incremented. When the number of collisions becomes a multiple of 5, the value, added with the time stamp, is sent to the Measurify APIs, from where it can be manipulated, also in real-time, by other applications and/or user interfaces.

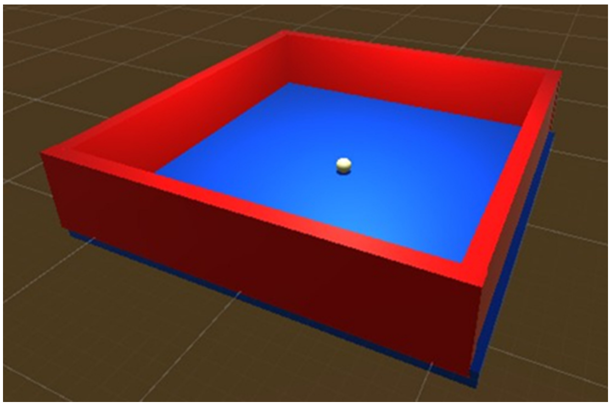


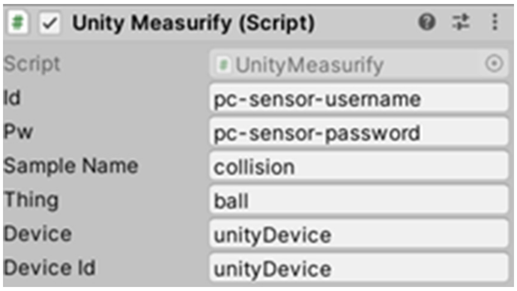
Fig. 1. Simple game scene in Unity3D

This goal is achieved by following the standard steps of the workflow of an Edgine-extended Measurify application. The first step consists in POSTing to Measurify (e.g., through a common collaboration platform for API development, such as Postman) the names and, particularly for the Script resource, the contents of the resources needed to configure the environment so that it will be able to receive measurements from our application. In our case, the names are reported in the second column of Table 2. The Measurify APIs will create the corresponding resources, that will be available at the URLs in the third column, where the base url is http://students.atmosphere.tools/

Table 2. Measurify resources for the example application

Resource type	Resource	Resource URL
Thing	Ball	url/v1/things/ball
Feature	Collision	url/v1/features/collision
Device	Unity	url/v1/devices/Unity
Script	Collisions-count-send	url/v1/scripts/collisions-count-send

The next step concerns the development of the C# script that detects the collision and exploits the Edgine plug-in to transmit the corresponding value to Measurify. As anticipated, for the sake of flexibility, in the Unity implementation the Setup() and Action() functions are parametric, thus we allow the Unity user to specify through the Inspector module (Fig. 2) the values that will be passed to such functions.

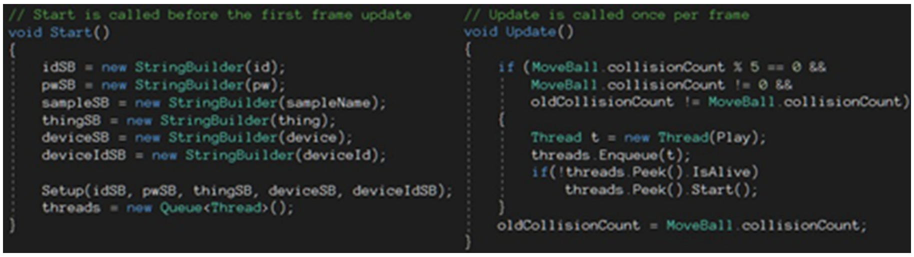

Unity Measurify (Script)

Script	UnityMeasurify
Id	pc-sensor-username
Pw	pc-sensor-password
Sample Name	collision
Thing	ball
Device	unityDevice
Device Id	unityDevice

Fig. 2. Variables set by the user inside the Unity Inspector

The Unity scripts are organized in a structure that strictly corresponds to the Edgine programming model, with an initial Start() method, executed only once at the beginning of the programme, and an Update() method, which is continuously called, at each frame's update. The mapping for an Edgine application in Unity is thus straightforward: the Setup() method is called inside the Start(), while Action() inside Update(), (Fig. 3).

Start() accomplishes the task of initializing all the StringBuilder public variables involved in the script. Once such user-defined parameters are collected, the plug-in's Setup() is executed, which performs the authentication in Measurify, getting the Json



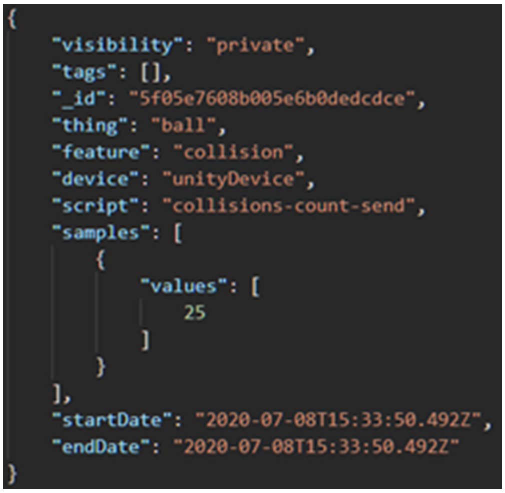
```
// Start is called before the first frame update
void Start()
{
    idSB = new StringBuilder(id);
    pwSB = new StringBuilder(pw);
    sampleSB = new StringBuilder(sampleName);
    thingSB = new StringBuilder(thing);
    deviceSB = new StringBuilder(device);
    deviceIdSB = new StringBuilder(deviceId);

    Setup(idSB, pwSB, thingSB, deviceSB, deviceIdSB);
    threads = new Queue<Thread>();
}
```

```
// Update is called once per frame
void Update()
{
    if (MoveBall.collisionCount % 5 == 0 &&
        MoveBall.collisionCount != 0 &&
        oldCollisionCount != MoveBall.collisionCount)
    {
        Thread t = new Thread(Play);
        threads.Enqueue(t);
        if(!threads.Peek().IsAlive)
            threads.Peek().Start();
    }
    oldCollisionCount = MoveBall.collisionCount;
}
```

Fig. 3. Start and update methods in the Unity script

Web Token (JWT) that will be used in all the subsequent accesses to the cloud APIs. A Queue<Thread> object is also initialized, that will be useful during the game loop. The creation of a Thread object, in fact, is necessary when sending data to the cloud, in order not to block the continuous game update cycle. The threads are inserted in a queue to simplify their management, as they are created, executed once at a time and destroyed in order, making use of the Enqueue(), Peek() and Dequeue() methods.

The Update() method first makes a check on the number of accumulated collisions (a measurement is sent to the cloud only when a multiple of 5 is hit). Then, if the sending condition is met, a thread is created, consisting of the Play() function calling the plugin's Action(), and added to the queue. Once created, a thread is ready to be executed as soon as no other thread is still alive. In this way, the application is not overloaded with threads. If some threads are still in the queue when the game is completed and quitted by the player, the OnApplicationQuit() method launches the missing threads before the actual end of the program execution. Finally, Fig. 4 shows an example of measurement received by Measurify in JSON format.



```
{
    "visibility": "private",
    "tags": [],
    "_id": "5f05e7608b005e6b0dedcdce",
    "thing": "ball",
    "feature": "collision",
    "device": "unityDevice",
    "script": "collisions-count-send",
    "samples": [
        {
            "values": [
                25
            ]
        }
    ],
    "startDate": "2020-07-08T15:33:50.492Z",
    "endDate": "2020-07-08T15:33:50.492Z"
}
```

Fig. 4. The measurement received from the Unity game at the Measurify's end

6 Conclusions and Future Work

This paper has proposed the use of a new data toolchain for serious games analytics. The toolchain relies on the open source Measurify IoT framework for measurement management, and particularly takes advantage of its edge computing extension (Edgine), which can be seamlessly deployed cross-platform on embedded devices and PCs as well. The Edgine is programmed to download from Measurify a set of scripts, that are periodically executed so to get data from sensors, pre-process them and send the extracted information to the Measurify APIs.

Thanks to its powerful abstractions, this model can be seamlessly employed also in the virtual reality and gaming domain. Virtual sensors can be built in game engine scripts. This paper has presented the implementation of the plug-in which deploys Edgine in Unity 3D, allowing an easy delivery of virtual sensor information to Measurify. Just as a proof of concept, we have presented the utilization of the whole chain within a trivial game scene. We plan to employ and test the system with more complex serious game analytics, and this opportunity is possible for every researcher and practitioner thanks to the open source release of the whole Measurify framework [20].

There is another aspect beside the general applicability of the toolchain to any virtual sensors. In fact, the same software engineering approach (and actual Edgine tool) could be used also for measuring real world parameters, connected to the game. This opportunity is particularly relevant to augmented reality games and reality enhanced serious games (e.g., [21–23]). Moreover, the same system can be used also the physiological parameters of the player. To the best of our knowledge, no tool exists that is able to process information from such heterogeneous sources as physical and virtual sensors. For instance, the Edgine toolchain could be used to simultaneously get data about a game session and a player's physiological status.

References

1. Loh, C.S., Sheng, Y., Ifenthaler, D. (eds.): Serious Games Analytics. AGL. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-05834-4
2. Alonso-Fernández, C., Pérez-Colado, I., Freire, M., Martínez-Ortiz, I., Fernández-Manjón, B.: Improving serious games analyzing learning analytics data: lessons learned. In: Gentile, M., Allegra, M., Söbke, H. (eds.) GALA 2018. LNCS, vol. 11385, pp. 287–296. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11548-7_27
3. Berta, R., Kobeissi, A., Bellotti, F., De Gloria, A.: Atmosphere, an open source measurement-oriented data framework for IoT. IEEE Trans. Ind. Inf. https://doi.org/10.1109/tii.2020.2994414
4. Lin, L., Liao, X., Jin, H., Li, P.: Computation offloading toward edge computing. Proc. IEEE **107**, 1584–1607 (2019)
5. https://github.com/measurify/edge
6. Hicks, D., Eagle, M., Rowe, E., Asbell-Clarke, J., Edwards, T., Barnes, T.: Using game analytics to evaluate puzzle design and level progression in a serious game. In: Proceedings of the Sixth International Conference on Learning Analytics & Knowledge (LAK 2016), pp. 440–448. ACM, New York (2016). https://doi.org/10.1145/2883851.2883953

7. Peddycord-Liu, Z., Cody, C., Kessler, S., Barnes, T., Lynch, C.F., Rutherford, T.: Using serious game analytics to inform digital curricular sequencing: what math objective should students play next? In: Proceedings of the Annual Symposium on Computer-Human Interaction in Play (CHI PLAY 2017), pp. 195–204. ACM, New York (2017). https://doi.org/10.1145/3116595.3116620
8. Westera, W., Nadolski, R., Hummel, H.: Serious gaming analytics: what students' log files tell us about gaming and learning. Int. J. Serious Games **1**(2) (2014). https://doi.org/10.17083/ijsg.v1i2.9
9. Lukosch, H., Cunningham, S.: Data analytics of mobile serious games: applying bayesian data analysis methods. Int. J. Serious Games **5**(1) (2018). https://doi.org/10.17083/ijsg.v5i1.222
10. Baalsrud Hauge, J.M., et al.: Learning analytics architecture to scaffold learning experience through technology-based methods. Int. J. Serious Games **2**(1) (2015). https://doi.org/10.17083/ijsg.v2i1.38
11. Alonso-Fernandez, C., Calvo, A., Freire, M., Martinez-Ortiz, I., Fernandez-Manjon, B.: Systematizing game learning analytics for serious games. In: 2017 IEEE Global Engineering Education Conference (EDUCON), Athens, pp. 1111–1118 (2017). https://doi.org/10.1109/EDUCON.2017.7942988
12. Solapure, S.S., Kenchannavar, H.: Internet of Things: a survey related to various recent architectures and platforms available. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, pp. 2296–2301 (2016). https://doi.org/10.1109/ICACCI.2016.7732395
13. Jiong, S., Liping, J., Jun, L.: The integration of azure sphere and azure cloud services for Internet of Things. MDPI J. Appl. Sci. **9**(13), 2746 (2019). https://doi.org/10.3390/app9132746
14. Amazon Web Services AWS IoT. https://aws.amazon.com/iot/solutions/industrial-iot
15. Azure IoT, Microsoft (2019). https://azure.microsoft.com/en-us/overview/iot/
16. Cirimele, V., et al.: The fabric ICT platform for managing wireless dynamic charging road lanes. IEEE Trans. Veh. Technol. **69**(3), 2501–2512 (2020)
17. Hiller, J., et al.: The L3Pilot data management toolchain for a level 3 vehicle automation pilot. Electronics **9**, 809 (2020)
18. Monteriù, A., et al.: A smart sensing architecture for domestic moniotring: methodological approach and experimental validation. Sensors **18**(7) (2018). https://doi.org/10.3390/s18072310
19. http://students.atmosphere.tools/
20. https://github.com/measurify
21. Massoud, R., Bellotti, F., Poslad, S., Berta, R., De Gloria, A.: Towards a reality-enhanced serious game to promote eco-driving in the wild. In: Liapis, A., Yannakakis, G.N., Gentile, M., Ninaus, M. (eds.) GALA 2019. LNCS, vol. 11899, pp. 245–255. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34350-7_24
22. Massoud, R., Poslad, S., Bellotti, F., Berta, R., Mehran, K., De Gloria, A.: A fuzzy logic module to estimate a driver's fuel consumption for reality-enhanced serious games. Int. J. Serious Games **5**, 45–62 (2018). https://doi.org/10.17083/ijsg.v5i4.266
23. Fijnheer, J.D., van Oostendorp, H.: Steps to design a household energy game. Int. J. Serious Games **3**(3) (2016). https://doi.org/10.17083/ijsg.v3i3.131