Journal of
Sensor and Actuator Networks

MDPI

Article

How to Develop IoT Cloud e-Health Systems Based on FIWARE: A Lesson Learnt

Antonio Celesti [1,2,*], Maria Fazio [1,3], Fermín Galán Márquez [4], Alex Glikson [5], Hope Mauwa [6], Antoine Bagula [6], Fabrizio Celesti [7] and Massimo Villari [1,3]

1 Department of Mift, University of Messina, Viale F. Stagno d'Alcontres 31, 98166 Messina, Italy; mfazio@unime.it (M.F.); mvillari@unime.it (M.V.)
2 BIG DATA Laboratory - CINI, Via Volturno, 58, 00185 Rome, Italy
3 IRCCS Centro Neurolesi "Bonino Pulejo", Contrada Casazza, SS113, 98124 Messina, Italy
4 Telefónica S. A., Telefónica Research and Development (TID), Distrito Telefónica Edificio Oeste 1, Ronda de la Comunicació, 28050 Madrid, Spain; fermin.galanmarquez@telefonica.com
5 IBM Research, Abba Khoushy Ave 199, Haifa 3498838, Israel; gliksonn@il.ibm.com
6 University of the Western Cape, Robert Sobukwe Rd, Bellville, Cape Town 7535, South Africa; hmauwa@uwc.ac.za (H.M.); abagula@uwc.ac.za (A.B.)
7 Department of Biomedical and Dental Sciences and Morphological and Functional Images, University of Messina, Azienda Ospedaliera Universitaria Policlinico "G. Martino", Via Consolare Valeria 1, 98125 Messina, Italy; fabrizio.celesti@studenti.unime.it
* Correspondence: acelesti@unime.it; Tel.: +39-090-676-8577

Received: 23 November 2018; Accepted: 2 January 2019; Published: 10 January 2019

Abstract: Nowadays, the penetration of sensors and actuators in different application fields is revolutionizing all aspects of our daily life. One of the major sectors that is taking advantage of such cutting-edge cheap smart devices is healthcare. In this context, Remote Patient Monitoring (RPM) at home represents a tempting opportunity for hospitals to reduce clinical costs and to improve the quality of life of both patients and their families. It allows patients to be monitored remotely by means networks of Internet of Things (IoT) medical devices equipped with sensors and actuators that collect healthcare data from patients and send them to a Cloud-based Hospital Information System (HIS) for processing. Up to now, many different proprietary software systems have been developed as stand-along expensive solutions, presenting interoperability, extensibility, and scalability issues. In recent years, the European Commission (EC) has promoted the wide adoption of FIWARE technology, launching 16 Industrial Accelerators focusing on different application fields. One of these, i.e., FICHe, is specialized in healthcare, providing the guidelines on how to develop eHealth systems. This paper focuses on how to compose new cutting-edge IoT and Cloud-based Cyber Physical Health Sytem (CPHS) services and applications interconnected with remote medical sensors and actuators using FIWARE technology in the context envisioned by FICHe. In particular, we discuss the design and development of an RPM system implemented through the collaboration between the Istituto di Ricovero e Cura a Carattere Scientifico (IRCCS) "Bonino Pulejo" (i.e., a clinical and research healthcare centre specialized in the treatment of neuro lesions), University of Messina, IBM Research, Telefónica, and the University of the Western Cape in South Africa. The description of our best practice provides a model and guidelines for the development of lightweight and low cost RPM services for rural and isolated areas, with the expectation of expanding healthcare to the developing world and in general allows us to outline how to deal with the real adoption of the FIWARE technology in an e-health project.

Keywords: sensors; actuators; internet of things; cloud computing; e-health; FIWARE

1. Introduction

Nowadays, clinical centres are looking at Cloud computing and Internet of Things (IoT) technologies to develop new cutting-edge e-health services and applications. In this context, telehealth and, in particular, Remote Patient Monitoring (RPM) at home, represent a tempting opportunity for hospitals to reduce clinical costs and to improve the quality of life of both patients and their families. It allows patients to be monitored remotely in their homes by means networks of Internet of Things (IoT) medical devices equipped with sensors and actuators that collect healthcare data from patients and send them to a cloud-based Hospital Information System (HIS) for processing. RPM is also a powerful tool that can be leveraged by the developing world to provide remote access to patients' data in rural clinics. The expectation is to avail such data to medical experts worldwide, thus bridging the medical divide between developed and developing countries. RPM allows patients to be monitored remotely by means of IoT-based medical devices equipped with sensors and actuators that collect and send data to hospital cloud system providing services to the patients and clinical personnel. Up to now, many different proprietary software systems have been developed as stand-along, often expensive, solutions presenting interoperability, extensibility, and scalability issues.

In recent years, different tele-healthcare initiatives have been proposed. A holistic approach to design and implementation of a medical teleconsultation workspace is proposed in [1]. A system architecture implementation called TeleDICOM II based on Service Oriented Architecture (SOA) and Virtual Organization (VO) concepts is discussed. A mobile patient monitoring system that makes use of mobile computing and wireless communication technologies for continuous or periodic measurement and analysis of biosignals of patients is presented in [2]. In particular, a generic architecture associated terminology and a classificatory framework for comparing mobile patient monitoring systems are proposed, aimed at both healthcare and computer science professionals. A cloud-based mobile system to improve respiratory therapy services at home is proposed in [3]. The platform uses vital signs monitoring as a way of sharing data between hospitals, caregivers, and patients. Using an iterative research approach and the user's direct feedback, they show how mobile technologies can improve a respiratory therapy and a family's quality of life. A Framework for European Services in Telemedicine (FEST) is discussed in [4]. The main objective of FEST is to develop a framework of common understanding, which will assist those wishing to set up a Telemedicine service by providing structured guidance to the information required for such an endeavour. A software agent approach for telemonitoring of patients at home is presented in [5]. In particular, it is considered as an alarm raising system that addresses the issue of the increasing social, economical, and medical needs of maintaining people at home in loss of autonomy, while preserving privacy and quality of life. A preliminary study for the development and implementation of a national Taiwan's Telehealth Pilot Project (TTPP) for long-term care is presented in [6]. The system has three different models; the home-care, the community-care, and the residential-care model to assist the elderly in the pursuit of better healthcare and improved quality of life. The results revealed that both the home-care and community-care models facilitate timely medical responses if the enrolled patients have emergent conditions. An architecture for a continuous, user-driven, and data-driven application of clinical guidelines and its evaluation is discussed in [7]. Specifically, a realistic continuous guideline (GL)-based decision support architecture, i.e., PICARD that accesses a temporal reasoning engine and provides several different types of application interfaces is proposed. The feasibility and acceptability of using mobile phones as part of an existing Web-based system for collaboration between patients with diabetes and a primary care team is assessed in [8]. In design sessions, mobile wireless glucose meter uploads are tested along with two approaches to mobile phone-based feedback on glycemic control, highlighting how mobile diabetes management systems may present a strategy to improve the quality of diabetes care. However, this state of the art analysis highlights how most of existing solutions have been conceived as "stand-alone", adopting different technological approaches that require a considerable level of complexity with high design, development, and management costs. The adoption of the cloud computing technology could push down such costs. However, it is at an early stage in the

field of tele-health. A framework for supporting healthcare was proposed in [9], with the objective of providing a model and guidelines for deploying lightweight and low cost Cyber Physical Health Sytem (CPHS) in rural and isolated areas, with the expectation of expanding healthcare to the developing world.

Currently, Cloud technology is recognized as the enabling key technology for Future Internet. Several years ago, the European Commission (EC) envisioned a challenge for fostering a wide adoption of Cloud and IoT based systems, avoiding vendor lock-in and simplifying the composition of new services. The Future Internet initiative that involved Public and Private Partners (FI-PPP) has brought to the delivery of a new advanced European cloud platform, named FIWARE [10]. The FIWARE Accelerator Programme was part of the FI-PPP. It is investing 80 million euros to promote the use and adoption of FIWARE technologies and to help Smart and Medium Enterprises (SMEs) and entrepreneurs create innovative Future Internet applications [11,12]. The program included 16 Future Internet Accelerator projects covering different topics, such as smart cities, multimedia, agrifood, healthcare, etc.

Therefore, in recent years, several applications have been developed adopting the FIWARE technology in different application domains including information management [13,14], Cloud resource management [15], smart mobility [16–19], smart parking [20], smart agriculture [21,22], energy [23], security [24–27], human-computer interaction [28], earthquake prevention [29], and healthcare [30].

Moreover, in order to promote the wide adoption of a such an emerging technology, the EC launched 16 Industrial Accelerator programs focusing on different application fields. One of these, i.e., FICHe [31], was specialized on healthcare providing the guidelines on how to develop e-health systems. In its activity, it also boosted 20 innovative eHealth startups. However, how to use the FIWARE platform and its additional components, called Generic Enablers (GEs), for the development of e-health solutions is not trivial at all. In fact, currently, the development of a full e-health solution using the FIWARE components is not trivial at all because it is hard to understand how to effectively integrate them.

In this paper, we present the developement experience of an RPM system based on FIWARE in the context envisioned by FICHe. Such an experience was performed thanks to the collaboration among researchers belonging to both clinical, academinc and industrial partners including the Istituto di Ricovero e Cura a Carattere Scientifico (IRCSS) “Bonino Pulejo” (Italy) (i.e., a clinical and research healthcare centre specialized in the treatment of neuro lesions), University of Messina (Italy), IBM Research (Israel), Telefónica (Spain), and the University of the Western Cape (South Africa). In particular, we start from the preliminary RPM architectural model provided by IRCSS “Bonino Pulejo” previously discussed in [32], in order to describe how it was extended according to FICHe guidelines and how it was implemented considering the most recent FIWARE components. More specifically, the description of our case study allows us to outline how to deal with the real adoption of the FIWARE technology in an e-health project. The proposed solution aims at improving remote assistance to patients at home optimizing the management of the workflow of physicians, medical assistants, and operators involved in the service. We describe how such an e-health solution has been designed, composed of several FIWARE services and GEs, highlighting the main advantages in the modus operandi of the researchers involved in the project. Also, we provide a full description of the FIWARE ecosystem and of the specific GEs that were used in the RPM system in order to describe the main advantages in the agile software development of an e-health project. It can be a useful contribution for entrepreneurs or developers that would want to exploit the FIWARE platform to easily develop e-health projects. The solution is also expected to be implemented as a first fully fledged CPHS for the rural and underdeveloped areas of South Africa.

The paper is organized as follows: In Section 2, we provide an overview of the FIWARE technology and we discuss the method we adopted to design the RPM architecture; in Section 3, we discuss how can be possible to developed a FIWARE-based RPM system. In particular, we will focus on how to setup RPM IoT devices, how to collect and permanently store patients’ health data over

Cloud, also addressing security issues. Moreover, we will explain in detail how we integrated GEs. A discussion on the overall design and development experience and conclusions are provided in Section 4.

2. System and Software Architecture

2.1. FIWARE Platform and Software Components Overview

At present, a few big cloud players in the world, such as Google, Amazon, Facebook, Apple (see *Les GAFA* [33]), Rackspace, Saleforce, have a prominent position in the business on cloud computing. They are actuating the policy of *Vendor Lock-In*, where it is easy to *go in* their systems, but it is a nightmare to *go out* of their cloud systems. For example, Amazon gives its clients the possibility to import data in its cloud for free, but it is necessary to pay to export data and services out. As described in this section, FIWARE might represent the redemption of Europe in cloud offerings as it represent a meaningful alternative to speed up new ICT applications under the Future Internet ecosystem in a safe and secure way thanks to its *Openness* and free availability of the cloud infrastructure.

The aim of FIWARE is to provide an open standard platform and an open, sustainable, global ecosystem. Being a cloud-based platform, FIWARE heavily relies on its cloud hosting layer. Both the reference architecture and the reference implementation of FIWARE Cloud are based on OpenStack [34], the leading open source cloud middlewere that has been developed collaboratively and widely adopted by the industry. The following mainstream OpenStack modules are being adopted:

- OpenStack Compute Service (Nova)—used to provide and manage Compute resources (including virtual machines and Linux containers), hosting the various runtime components of the software stack comprising the application;
- OpenStack Image Service (Glance)—used to store and manage virtual machine images, which encapsulate the pre-installed operating system and the software stack to be deployed on the individual virtual machines hosting the application;
- OpenStack Volume Service (Cinder)—used to provide and manage block storage resources (storage volumes), which can be attached to Compute resources as data disks in order to keep the persistent state of the application;
- OpenStack Network Service (Neutron)—used to provide and manage virtual networks, which can be attached to Compute resources so that the different application components can interact between themselves as well as with external users/entities;
- OpenStack Orchestration Service (Heat)—used to orchestrate the provisioning of complex multi-resource configurations, as well as computation, storage and network artifacts associated with a particular application infrastructure;
- OpenStack Identity Service (Keystone)—used to manage authentication, authorization and access control for the various cloud components;
- OpenStack Application Management Service (Murano)—used to manage the provisioning and life cycle of the individual software components comprising the application, including dependencies between them and on the underlying infrastructure;
- OpenStack Object Storage Service (Swift)—allows the application to hold BLOBs and associated metadata, in a large scale, highly available, and low cost storage facility built on commodity hardware.

OpenStack is the buiding block of FIWARE that can take advantage of its large community involving developers and companies. The complementary interaction with the OpenStack community makes the FIWARE Cloud sustainable over time, allowing the developers to leverage best-of-bread capabilities in cloud infrastructure and in other areas, where FIWARE is providing innovative GEs that can be hosted on FIWARE Cloud seamlessly.

The FIWARE Reference Architecture includes a set of general-purpose platform functions available through APIs that are GEs. GEs provide advanced and middleware interfaces to networks and devices, advanced Web-based user interfaces, application/services ecosystems and delivery networks, cloud hosting, data/context management, IoT service enablement, and security. FIWARE considers GE Open Specifications (that are public and royalty-free) and their implementations (GEi). There might be multiple compliant GEi(s) of each GE open specification. At least, there is one open source reference implementation of FIWARE GEs (FIWARE GEi(s)) with a well-known open source license.

As described by Hierro et Al. in [35], FIWARE is working in the following directions:

- the technical approach for implementing use case and trial projects using FIWARE generic enablers on available infrastructures;
- the operational approach for deploying and running the experiments and trials;
- the way social, legal, and economic requirements and constraints are addressed.

The FIWARE Ecosystem is rather comprehensive and it is composed of the elements and stakeholders reported here:

- **Small and Medium-sized Enterprises (SMEs)**, i.e., the players that are able to provide new ICT services to a wider audience. FIWARE with its Ecosystem tries to increase new business for SMEs.
- **Platform Providers (PPs)** hosting the FIWARE Cloud infrastructure.
- **Service Provider (SP)**, i.e., a company that provides organizations with consulting, legal, real estate, education, communications, storage, processing, and many other services. In general in FIWARE, SPs can be PPs, but also creators and providers of GEs and SEs.
- **FIWARE-LAB**, i.e., a live instance of FIWARE available to developers for free experimentation with the technology.
- **FI-OPS**, i.e., a collection of tools that eases the deployment, setup and operation of FIWARE instances by PPs.
- **Experimentation Environment**, i.e., the FIWARE Cloud infrastructure hosted by PPs that can be either academics or industrial.
- **Production Environment**, i.e., the FIWARE Cloud infrastructure hosted PPs aimed at Business purposes typically created by industrial providers like industrial companies.
- **XIFI**, i.e., project facilitates the uptake, deployment and federation of several instances of such a common platform to pave the way for a unified European marketplace that is crucial for enabling commercial exploitation of FI resources.

The FIWARE Ecosystem is shown in Figure 1. Starting from the left part of the picture, the GE repository holds the code of available GEs. Different geometric shapes for GEs remark the variety of GEs already available in FIWARE. GEs can be instantiated in the FIWARE infrastructure and executed in the RunTime environment (RT). RT is included in the FIWARE platform and it can be a component of an isolated cloud (such as in Production Environments), or part of federated clouds, where different RT interact with each other by means of XIFI specifications (such as in Experimentation Environments). In the example of Figure 1, *University A (PP)* and *University B (PP)* are PPs federated following XIFI agreements, whereas *Cloud Provider C* and *Cloud Provider D* manage isolated clouds. However, due the open nature of FIWARE, GEs can be moved from one RT to another and vice versa (see the dashed-lines) seamlessly. Even if PPs present different Cloud interfaces, users, SMEs and IoT devices manage GEs in the RT in the same way. That is why FIWARE represents the *Standard de Facto* for future systems.

For the best of our knowledge, apart from the solution discussed in [30], there are not other scientific works focusing on the adoption of the FIWARE technology for healthcare applications. The best practice described in this paper aims at contributing to improve the state of the art in this context.

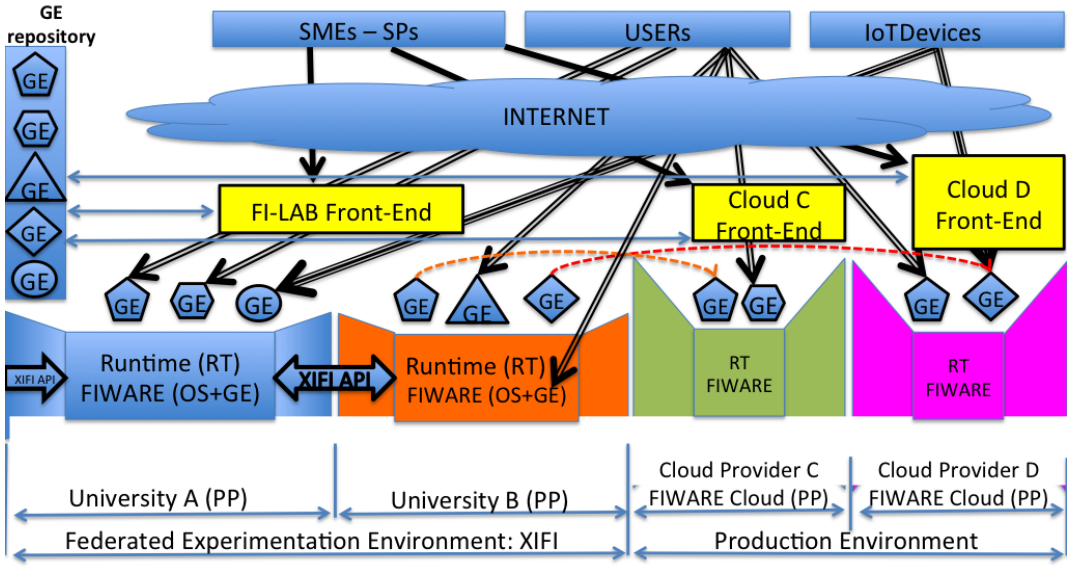

Figure 1. FIWARE Ecosystem.

2.2. Case Study: An e-Health System for Remote Patient Monitoring (RPM)

In order to describe how the FIWARE technology can be used in a real healthcare use case, we focus on a solution for RPM at home. RPM is becoming a challenging solution for hospitals and healthcare organizations, which aims at offering a high quality care service, keeping costs down, and increasing their potential market. Actually, patients with multiple chronic diseases consume a high quantity of healthcare resources and, often, they have to frequently come back to the hospital for clinical treatments. Providing health assistance to patients at home allows health providers to reduce hospitalizations and allows patients to stay in a more homely environment.

RPM is based on the continuous assessment of patience's health status and allows ill patients or patients with disabilities to interact with the medical personnnel [36]. It may include physiological monitoring (e.g., temperature, weight, glucose, blood pressure, etc) and motor exercise tracking tools. Nowadays, most of hospitals do not provide RPM services, but thanks to the new ICT technologies, in our opinion, they can be easily developed in order to offer advanced heath care services. Our use case is focused on a cloud system providing RPM services at home. Specifically, it proposes a support system to optimize the workflow of medical and paramedical personnel according to available health resources based on the real needs of patients.

Such a scenario includes many different actors who are classified as:

- Physicians: Medical personnel who are responsible to take care of patients through the analysis, diagnosis, and treatment of diseases. They can be typically general or specialist medical practitioners focused on specific disease categories and methods of treatment.
- Operators: Medical personnel who is responsible for visiting (periodically or on demand) patients at home and to forward information to physicians if a risk is detected. They represent the direct link between patients and the hospital/caregiver services.
- Patients: People with cronical disease or who need a long time convalescence. Their biological activities, habits, and environmental factors are continually monitored in order to give medical personnel updated information on the patients' health status.

RPM at home is possible thanks to the large adoption of many health monitoring devices along with new tele-healthcare services. To this end, cloud computing can be very convenient for the development of reliable and scalable solutions that can be employed in many different contexts. In particular, the exploit of FIWARE technologies is strategic for several reasons. FIWARE allows developers to reduce the time necessary to set up the whole solution and to increase the modularity,

scalability, and flexibility of the final product. The basic cloud-based components used in this initiative are aimed at data storage and processing, brokerage, and security. Moreover, software modules for collecting data are really important, due the massive data that need to be treated. The advantages of using FIWARE components is that they can be easily instantiated and interconnected using well defined Application Program Interfaces (APIs). IoT devices equipped with sensors and actuators integrated with the FIWARE platform play a fundamental role in development of Future Internet applications. The FIWARE platform can easily orchestrate all the necessary services of a RPM scenario that is rather critical in terms of medical IoT device management (including sensors and actuators), data management (information have to be processed and exchanged between physicians and medical operators) and patients' privacy.

2.3. Design Steps

To face the development of a new project, it is necessary to perform well defined independent steps. They are:

- *functional analysis*—it describes what the system does and identifies all the functions that must be performed by the system in order to meet the operational requirements. Each of these functions is decomposed into sub-functions, and the requirements allocated to the function are each decomposed with it. This process is iterated until the system has been completely decomposed into basic sub-functions and each sub-function at the lowest level is completely, simply, and uniquely defined by its requirements.
- *technology investigation*—it describes how the system works and identifies possible technologies useful to implement the sub-functions in the system.
- *service and software development*—it includes the effort in developing new services and software components that are specifically designed for the project.
- *integration*—it provides an abstract definition of the system to capture the key processes, data relationships, and data sources necessary to the whole system. Such abstraction maps one or a set of functions and sub-functions in modules. Then, modules can be organized in an ordered structure, where interactions among different modules are fully defined, as well as the interfaces towards the external world.

Whereas Step 1 is strictly related to the specific project, FIWARE provides great support for Step 2 of the project development, offering a useful set of technologies and packages that can be easily adopted. Thus, in the following, we briefly describe the functional analysis of our specific use case (Step 1). Then, in the next Sections, we deal with the technology investigation process using FIWARE technologies (Step 2). Step 3 is out of the scope of this paper, since it includes design and implementation strategies for proprietary software developed for the e-health solution. We will describe the Step 3 results in a future work. About Step 4, we provide a skeleton of an integration structure, where the main components identified for the development of the project are interconnected.

With reference to the specific use case, the RPM solution need to implement the following main functionalities:

- Input data management: RPM solutions are based on data gathered by remote sensors or measurement systems necessary to detect specific physiological and environmental parameters. Data gathering can be performed by small healthcare devices either clipped onto patients' clothings or directly attached on the patient's body by means of smartphones, smartwatches. Moreover, devices placed in the patient's home (e.g., equipped with motion sensors) can also automatically collect data coming from the surrounding environment without the need of any specific action performed by the patient. Other devices can be explicitly used by patients to manually insert data.
- Data storage: Data collected at the patient's homes need to be transmitted from healthcare devices to a central data repository placed in the clinical centre for benchmarking [37]. It is hosted in a

remote server that is accessed by authorized users (e.g., medical personnel responsible for nursing of the patient) for further processing or history tracking.

- Service development: Specific e-health services need to be developed to process information according to users' configurations and event occurrences. They include alarm detection, medical activities scheduling, personnel management, patients cross-information processing for statistics, etc. Moreover, additional services need to be deployed to interconnect different components aimed at specific goal.
- Application development: It helps patients and medical personnel to interpret data coming from medical devices and equipments in order to plan the workflow for RPM at home. Software applications can be developed as mobile apps for smartphones/tablets or as web applications.
- Secure data access: Security is crucial in healthcare solutions. Thus, specific functionalities for data authentication and access management are necessary. The implementation of specific access policies specifies how different users can use information on patients. For example, physicians can modify monitoring configurations and visualize statistics, whereas operators cannot.

In order to proceed with the development of the e-health solution, we investigated on how to implement each of the above functionalities using the FIWARE technologies, trying to respond to the question: How can we do that with FIWARE? Our approach and the results we achieved are described in the next sections, where each section addresses a specific question.

3. Results

3.1. How to Setup RPM Devices at Patients' Home

Recent technological advantages allow to setup personal body networks of low-cost IoT medical devices equipped with sensors and actuators able to collect parameters related to the health status of patients directly in their homes enabling tele-monitoring. In this context, patients can use both wearable and wireless IoT medical devices able to monitor their health status (heart rate, speed, respiratory rate, single-lead ECG, and training load in real-time). Such devices, being connected over the Internet send monitored health data to the hospital Cloud.

Up to now the healthcare market has been dominated by vendor lock-in societies providing expensive solutions that make difficult their integration with third-party software systems. However, current trend of creating very low-cost programmable IoT devices is opening towards new frontiers in RPM. Indeed, in the last period we are witnessing to a real battle among Single Board Devices (SBD) makers that are pushing down costs. Such players includes Raspberry, Arduino, Realtek, and Espressif, among others. Emerging IoT devices are all characterized not only by a remarkable low cost, but even by their standalone capabilities, such as computation resources (micro-controller based products are very powerful), RAM (at least 1 MByte of memory), communications&protocols (e.g., WiFI, Bluetooth, Ethernet, IP, UDP, TCP, CoAP), and many GPI/Os (e.g., PWM, PCM, ADC, DAC etc.) for interfacing them with any external physical/electrical transducer. In acquiring and converting analogue physical/electrical data (ADC), the GPIOs of such devices present a considerable sampling rate in terms of frequency (up to 100 khz, that is also good for ultrasound scans), along with a good depth of bits per sample (at least 10 bits).

In the context of human health monitoring, RPM tools can be arranged using such IoT devices equipped with proper sensors and actuators able to collect bioelectrical signals from patients. Differently from existing Commercial off-the-shelf (COTS) human health monitoring devices, medical IoT devices allow the development of a plethora of very cheap RPM solutions that can revolutionize healthcare, not only in more developed countries, but also in developing ones and isolated rural ares. In fact, patients can benefit of many Internet-attached medical IoT devices able to monitor and assist them at home, such as: A low cost treadmill, a low cost exercise bike, a low cost adult walker aid, a bluetooth scale, a blood monitor (for pressure, glucose, oxygen, etc.), a step counter (pedometer), a bluetooth Heart Rate Monitor, a game-based interaction with SmartPhones,

a game-based interaction with Smart TVs and Remote Controllers. In some cases, cheap smart sensing devices measure physical parameters interfacing them with physical/electrical transducer (e.g., pedalling electronic measurement, running electronic measurement, indoor positions in home, etc.). In other cases, they may represent the sinks conveying all the data generated by wireless systems (e.g., smart health monitoring devices with remote controllers) [38].

Since each medical IoT device has its own local computing and storage capabilities, it is able to pre-process the collected raw health data and send only required aggregated pieces of information to the hospital Cloud system, hence reducing the communication overhead in term of data transmission. Furthermore, each personal body network can be interconnected with other ones forming a network of personal body network accessed by the hospital Cloud system.

3.2. How to Collect Data Over the Cloud

As previously specified, in the RPM use case, it is fundamental to collect data at the patient's homes and move them into the Cloud. To this aim, we identified Orion in the GE repository as key solution. Orion is the reference implementation of the publish/subscribe context broker GE. Orion implements the context management functionality in FIWARE, based on the OMA's NGSI standard [39]. A good introduction to context management in FIWARE can be found in [40]. Context management includes a set of API to create, update, read, or remove context information. A piece of context information consists of a set of *attributes* that characterize the *entities* of an application. In the e-health scenario, for example, each single entity can represent patients, whereas attributes can represent related diseases.

As shown in Figure 2, Orion includes a set of operations used by *context producers* (e.g., a medical sensor) to generate and update context information that are provided to *context consumers* (e.g., e-health services). Context consumption can be carried out by means of synchronous or asynchronous tasks. Context data production and consumption are decoupled. It is interesting to notice that producer and consumer are independent. Some complex applications have some parts playing the producer role and others playing the consumer role to provide a global service. For example, an e-health application could have two parts: The first one runs in the medical sensors (context producers) and the second in the doctors' smartphones (context consumers) to provide real-time clinical information about patients.

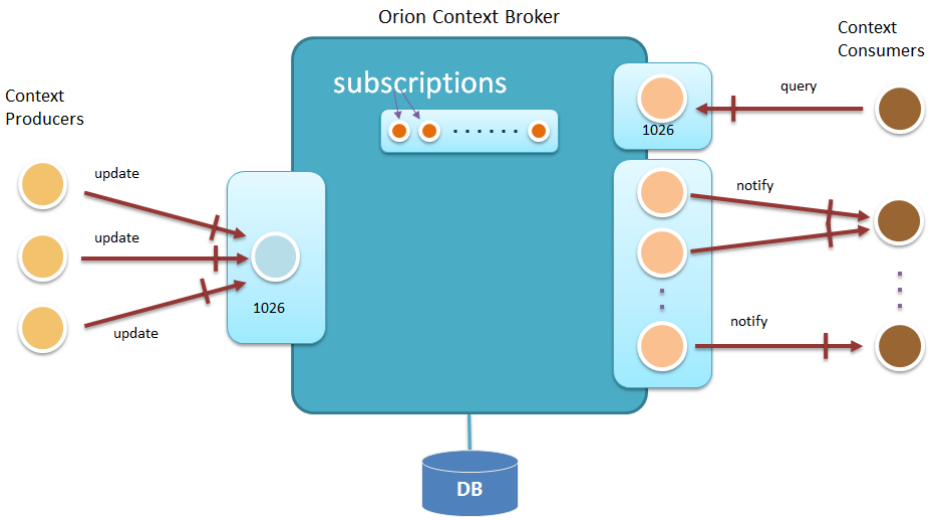

Figure 2. Orion Context Broker in a nutshell.

Considering the Orion API, we note that the OMA standard is abstract, thus not directly implementable by any piece of software. For this reason, FIWARE has defined a concrete syntax by means of a RESTful binding [41] that adopts HTTP as transport protocol and JSON as payload encoding format. The API is divided into two kinds of sub-APIs: the first for *context management* itself (i.e., OMA's NGSI10) and the second for *context availability management* (i.e, OMA's NGSI9). The difference among

them is that the latter does not manage pieces of context information themselves (e.g., the attribute A of entity E), instead the *context provider* of such pieces of information (e.g., the provider of attribute A of entity E is the context provider at URL P). This enables interesting scenarios in which context broker delegates context management to such providers and acts basically as a proxy, providing transparency to context consumers (i.e. if the provider of a given piece of context information changes, the change does not have any impact on context consumers), context provision infrastructure hiding (the context broker acts as single entry point for context management operations), and hierarchical scalability (one context broker can acts as context provider of a high-level context roker).

Both NGSI9 and NGSI10 sub APIs present a parallel structure and they include:

- Task to create/update context information (NGSI10) or context provider registrations (NGSI9).
- Task to query context information (NGSI10) or context provider registrations (NGSI9) in a synchronous fashion.
- Task to subscribe a context information (NGSI10) or context provider registration (NGSI9) so as to receive notifications in an asynchronous fashion.

From a RESTful perspective, the API implements two "families". One family (named *standard operations*) has been designed to be as closer as possible to the OMA specification, thus each resource in the RESTful API models an operation in the OMA API and the only verb used is POST. There is another family (named *convenience operations*) with a richer set of resources and verbs in which resources model context concepts (entities, attributes and subscriptions) and all the usual verbs are used (GET, POST, PUT, and DELETE). It is important to remark that both families are equivalent and are provided to enhance the flexibility of the enabler, so that the developer using Orion can chose the one the developer prefers.

Other functionalities that Orion Context Broker provides are the following:

- Horizontal scalability. The Orion Context Broker logic is basically stateless, so it can scale horizontally. The persistence layer used by Orion is based in MongoDB, which also scales horizontally using *shards*.
- Geolocation awareness. On the one hand, entities can be marked with a location (in GPS coordinates) at context broker. On the other hand, context broker is able to process queries scoped to geographic areas (e.g., inside/outside a circle or a polygon), which response takes into account the location information associated to entities.
- Pagination. Context information base can be very large, e.g., an e-health application managing information of 1,000,000 patients, each patient modelled as a different entity. Orion Context Broker API takes this into account and provide mechanism to query context information by blocks (named *pages*).
- Multitenancy. A single instance of Orion Context Broker may manage context information sets belonging to different parties (usually referred as *tenants*) isolated at DB level, so operations done by one tenant only take into account the context information belonging to that tenant.

Orion Context Broker could be compared with messaging systems (messaging queues/buses, service buses, etc.). In addition, it could be compared with conventional databases to store the same entity/attributes context information. In general, Orion Context Broker is *simpler* and *more flexible* than other kinds of system. Regarding simplicity, given that everything is about entities and attributes, no complex modelling (e.g., relational modelling) is needed. Moreover, entities and attributes are concepts that naturally arise during the analysis and design steps of an application. In addition, as explained before, the API itself includes a small number of operations that are easy to master. Regarding flexibility, context is a rather generic concept, so it is suited for many applications and not only to applications related to e-health. As part of this flexibility, take into account that an entity does not necessarily models things in the real world (such as a medical sensors). It can also model things in the virtual world, such as an "alarm" (which does not have any physical representation and only exists within the IT system, which manages alarms).

Specifically regarding the messaging system, they also allow actors (context consumer) to consume the information published by other actors (context producers). However, messaging systems used to process messages in an opaque way. Thus, the advanced functionality that Orion implements (such as geolocation awareness or context management delegation in context providers) is not possible with a messaging system. However, a messaging system could be a good complement to Orion, covering messaging bridging functionality (e.g., RESTful to Web-sockets), notification retries policies, etc.

Regarding conventional databases, they are general purpose systems, not specifically oriented to context management. Thus, although in some sense they are more powerful (e.g., queries and aggregations), they have a steep learning curve, compared with using the Orion RESTful API. Moreover, the lack of push functionality (all databases work under the pull paradigm) can have horizontal scalability problems (depending on the technology) and involve modeling complexity (typically, relational modelling, which is not needed with the entity-attribute approach).

Finally, there is an intrinsic advantage of using Orion Context Broker compared with the previous systems: Its seamless integration in the core of the FIWARE platform, thus opening the door to added value functionality directly or with out-of-the-box connectors. Orion Context Brokers sits in the heart of the platform, pumping data from/to several GEs or, in other words, being the "glue" to integrate different FIWARE GEs together.

3.3. How to Permanently Store Data

The huge amount of collected context data that can be defined as "big data" is stored in the Object Storage GE reference implementation in order to track the history of patients' diseases for possible further data analytics tasks. The Object Storage GE on one hand can be easily accessed to upload data into the cloud from external sources (e.g., IoT) and to store it in a scalable and resilient manner and on the other hand, it can be easily used as a source for complex processing and analytics workflows implemented using the various Data/Context Management GEs. Such combination provides a strong foundation for Big Data analytics, including evolution of existing GEs as well as introduction of new GEs (e.g., leveraging results of complementary development efforts).

The basic concepts of Object Storage are Containers, which must possess a unique name and contain the real data Objects. Users can create several Containers up to a certain limit. An Object also possesses a unique name within its Container and contains the real data, e.g., a document. An Object can be understood as a file having additional metadata. Some metadata is predefined such as HTTP-Metadata (Last-Modified, ETag, Content-Length, Content-Type, Content-Encoding, Content-Language etc.). Users can also attach user-defined metadata. Since the Object name may contain a separator '/', Objects can be organized in a form of directory hierarchies.

Typical operations for Object Storage are CRUD operations on Containers and Objects and the setting of properties or metadata attached to an Object. Addressing Containers and Objects is done by means of special Uniform Resource Identifications (URIs). The URI "http://tributary-gallery.s3.amazonaws.com/sinwaves.png" addresses an Object sinwaves.png in a Container named tributary-gallery. The element s3.amazonaws.com is a predefined host name to be used.

Access is given by the REST web services. REST is an HTTP-based protocol, which uses HTTP operations such as POST, GET, DELETE, and PUT for performing corresponding CRUD data manipulations. Since handling HTTP operations in a programming language is not trivial, specific APIs for various programming languages are provided, thus releasing programmers from the complicated HTTP programming APIs.

The Object services typically maintain multiple copies of an Object, in general, 3 redundant copies with eventual consistency. As already mentioned, Objects can have system-defined and user-defined metadata attached to them. A GET operation also retrieves the associated metadata.

Here, the idea of the e-health use case is the usage of FIWARE Object Storage as an archive for generic and medical data, more precisely, the storage of medical imaging data. For this use case, many types of file format are considered, like PDF, JPEG, DOCX, etc. and for this domain specific

such as the DICOM file format, HL7, etc. In particular, considering DICOM file format, it does not only contain a set of metadata values but also involves a data model that reflects the handling of medical imaging data in the e-health domain. For example, different images are part of a series, and this relation between the series should also be reflected in the storage. Therefore, the Object Storage needs to deal with the following capabilities:

- Support for the metadata as existing in the e-health domain: The idea here is to have a first test on what is required for the automated extraction of DICOM file metadata for FIWARE Object Storage GE.
- Support of the data model relations: in the e-health domain, different images stand in relation to each other and such relations should be reflected. The relations should not only be stored in the way that this information is retrieved when retrieving the data item. The useful implementation of this functionality is to quickly identify Objects belonging together.

The Object Storage GE on one hand can be easily accessed to upload data into the cloud from external sources through RESTful API (e.g., IoT) and to store it in a scalable and resilient manner and on the other hand, it can be easily used as a source for complex processing and analytics workflows implemented using the various Data/Context Management GEs leveraging data and metadata.

3.4. How to Instantiate New Software Components in the FIWARE infrastructure

FIWARE heavily relies on its cloud hosting layer that is based on OpenStack, as discussed in Section 2.1. FIWARE clouds typically offer "ready to deploy" packages including independent GEs, popular combinations of GEs and packaged pieces of software by using the innovative FIWARE Cloud tools. This makes extremely easy to setup complex development and deployment environments. Similarly, our RPM system is packaged using FIWARE Cloud tools, hence ensuring seamless compatibility and portability.

Developers can enjoy the standardized approach to package their applications (e.g., in VM images), which can be then deployed on any FIWARE Cloud seamlessly. Moreover, since FIWARE Cloud is compatible with OpenStack, the same applications can be deployed on any OpenStack-compliant cloud, private or public seamlessly.

For complex applications (e.g., multi-tier), FIWARE Cloud offers the abilities to implement logical network segregation, allowing to isolate individual application components from each other and from the external users/entities easily. The network virtualization capabilities are delivered following the modern Software Defined Networking (SDN) principles, available via OpenStack Network Service. Moreover, the SDN foundation enables the innovation in the network function virtualization area, where individual advanced network functions, such as load balancing or firewalls, are delivered flexibly within a unified cloud infrastructure, spanning compute, network, and storage resources. Such innovations are likely to be easily adopted by future FIWARE Cloud implementations, based on emerging standards and the ongoing work to make them compatible with OpenStack.

FIWARE Cloud provides seamless integration with the FIWARE Identity Management GE, enabling single-sign-on between the cloud GEs and the GEs hosted on the cloud. This way developers can maintain a single set of credentials for the different parts of their applications, including those delivering services to end-users and those controlling the life cycle of the application deployment itself, hosted on the FIWARE cloud.

In addition to capabilities provided by "Vanilla" OpenStack, FIWARE Cloud provides several innovations beyond what is currently available in mainstream OpenStack deployments. This includes the following capabilities:

- Support for LinuX Containers (LXC) as a novel mechanism to provide compute resources, enabling higher density, agility, and flexibility. With LinuX Containers (such as Docker), the application developer can enjoy a much more lightweight compute resource, hosting only the incremental

parts of the particular application component, as opposed to full operating system and the heavy-weight software stacks typically deployed in virtual machines.

- Support for execution of processing tasks called "storlets" within the Object Storage infrastructure, reducing the need to transfer large amounts of data between the data cluster and the Compute cluster. By using storlets, the application developer can easily offload many data-intensive tasks such as media transcoding or data anonymization to the Object Storage facility, leaving mostly the heavy-weight Compute-intensive processing within the Compute cluster.
- Support for template-based holistic management of complex application deployments, involving composite configurations of infrastructure resources (compute, storage, network) as well as software components installed and configured within the individual virtual machines or Containers. This technology enables DevOps-style approach to application development, where the individual provisioning and configuration steps across the different application components are fully automated, using easy to use declarative or prescriptive interfaces.

3.5. How to Develop Application Front-End

An e-health scenario requires a Graphical User Interface (GUI) that is easy to used by the clinical personnel and patients. FIWARE, by means of particular GEs, offers developers and end-users the possibility to build HTML5-based GUI to develop their front-end software components that are commonly used to arrange Software as a Service (SaaS). We can distinguish between developer-oriented and user-oriented GEs.

By means of developer-oriented GEs, developers can build pre-customized front-end software components. These GEs facilitate the developemnt of 2D/3D User Interfaces (UI), providing advanced tools for image rendering, interface design, synchronization, animation, etc. GEs belonging to such category include: 2D-UI, 2D/3D Capture, 3D-UI-XML-3D, 3DUI-Web Tundra, Cloud Rendering, GIS-Data Provider - Geoserver-3D, Interface Designer, POI Data Provider, Real Virtual Interaction, Synchronization, and Virtual Characters. Using such GEs, developers are responsible for defining the application logic and the interaction with back-end components (e.g., third-party software systems and other GEs).

Besides software developers, the FIWARE technology aims to enable "common" users (who do not have programming skills) to acquire built-in software components and combine them with a drag-and-drop approach in order to customize their applications. This is possible by means of user-oriented GEs. Such a new methodology opens FIWARE users to a new flexible software delivery model; users can easily acquire pieces of front-end software, each one controlling a particular back-end component and combine mash-up applications. GEs that enable such a scenario include: Application Mashup (whose reference implementation is Wirecloud), Marketplace (whose reference implementation is SAP RI), Repository SAP RI, Revenue Settlement and Sharing System, Store (whose reference implementation is WStore). In particular, Wirecloud allows both developers and end-users to build their own application by integrating different widgets in order to compose mashups. A widget is a graphical HTML5-based front-end that allows to control a particular software component back-end. Different from developer-oriented GEs, user-oriented GEs allow users to integrate different widgets according to a drag-and-drop approach. Wirecloud is aimed at both desktop and mobile users.

Wirecloud is the reference implementation of the Applicatin Mashup GE based on Rich Internet Application (RIA) and semantic technologies. It offers a web platform that allows users (without programming skills) to easily compose "on-fly" with a "drag and drop" approach their own dashboard/cockpit applications according to their specific needs. In fact, Wirecloud allows to develop an application logic by integrating heterogeneous data and UI components called "widgets". They can be chosen by users from a vast, ever-growing catalogue and integrated by means of a piping editor in order to develop dashboard/cockpit applications.

Wirecloud suits well the requirements of the RPM front-end. In fact, it allows medical personnel to customize their dashboard/cockpit applications easily in order to monitor the health status of patients.

This is possible by developing different widgets that can be integrated (wired) by the medical personnel according to their needs. Application mashups can be built, e.g., to address particular doctors and patients' needs and to monitor thresholds related to different patients' biological parameters. Let us think, for example, a doctor who is able to compose "on-fly" different widgets in order to build a dashboard that provides a global picture of the health status of a patient, or let us think of a doctor who is able to modify "on-fly" the same dashboard by removing or adding widgets according to his/her needs. In such a scenario, widgets can be the front-end component of a medical device, monitoring particular biological parameters of a patient (heartbeat, blood pressure, neurological activity, and so on). In this way, doctors can build customized dashboard/cockpit that can help them to make complex diagnosis, e.g., combining different widgets related to different observations in order to identify possible inter-dependent aspects that can condition a specific disease. In addition, considering an RPM scenario, doctors can access the real-time observations of a patients.

3.6. How to Address Security Issues

Security and privacy in e-health raise several issues. In order to address such issues, FIWARE offers several GEs, such as the Identity Management GE (whose reference implementation is KeyRock), the PEP Proxy GE (whose reference implementation is Wilma), and the Authorization PDP GE (whose reference implementation is AuthZForce).

Keyrock provides Single-Sign-On (SSO) authentication as a service. Users' identity attributes are typically managed by a trusted party. Keyrock, acting as Identity Provider (IdP) provides authentication services to different Service Providers (SP) acting as relying parties via open standard protocols such as OASIS SAML (Security Assertion Markup Language) version 2.0 [42] and OAuth. It covers attributes for both users and IoT devices, managing also the identity of "things" themselves (attributes, location, history, and so on).

Wilma is based on the eXtensible Access Control Markup Languages (XACML) [43]. It is responsible for controlling resource access. When a third party software component would like to access a resource, it includes in the requests a token issued by KeyRock that identifies the user that is going to perform the action. It is also possible to add specific access authorization policies to resources by means of AuthZForce. In a typical configuration, Wilma is integrated with the Keyrock and AuthZForce in order to carry out both authentication and authorization services.

Considering our RPM solution, we used the KeyRock to provide user authentication to both patients and medical personnel. When a widget receives a request from another widget from a user, the process of authentication and authorization for accessing the resources and/or services of the widget back-end begins. According to this approach, it is safe the access the resources and/or services belonging to other GEri(s) (e.g., Object Storage and Orion Context) or other third-party software systems (e.g., legacy systems belonging to hospitals). This approach is very smart because the security of an application mashup in not a monolithic but can be configured with a modular approach on each involved widget. Figure 3 depicts the authentication and authorization processes required to access both resources and services of a single widget back-end. The authentication process takes place by means of KeyRock that issues a OAuth token that then is sent to the Wilma. If the token is not expired, the request is forwarded to AuthZForce that verifies if the user holds the rights to access the widget back-end. If so, the widget back-end serves the request.

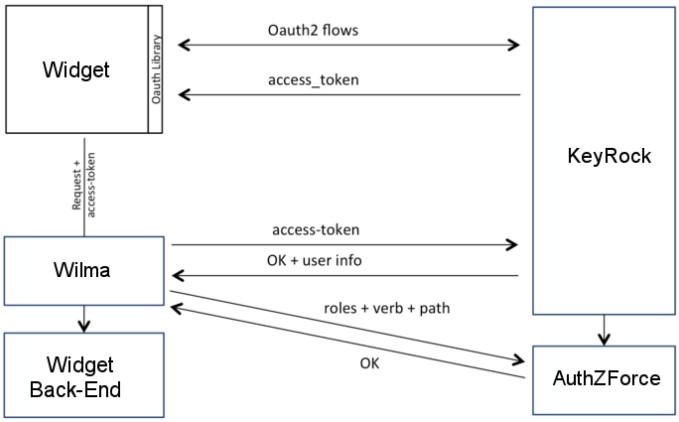

Figure 3. Integration between IdM, Authorization PDP, and Proxy Generic Enablers (GEs).

3.7. FIWARE-Based RPM Architecture

To integrate the above technologies in an efficient e-health solution, we need to organize components in a whole system, specifying how they are interconnected and the type of information they process. Figure 4 shows the skeleton of our solution, that describes how information flows in the system across different components. Rectangles in the picture represent the components implementing the main functionalities necessary to develop the e-health solution. Inside each rectangle, we identify the specific GEs useful for the related functionality. All the components and GEs in the system can be deployed in FIWARE VMs. Monitoring data collected from patients are forwarded across the Internet to the FIWARE-based hospital Cloud, where the *Data Collection component*, implemented by the Orion Context Broker GE, is devoted to organize data according to the context. Then, health data are processed by specific services arranged by the *service deployment* component, implemented by the Infrastructure as a Service (IaaS) GE (i.e., OpenStack), or simply stored by the *Data Storage* component implemented by the Object Storage GE. Raw and processed health data can be accessed by end users through applications, arranged by the Wirecloud GE, characterized by different features according to the particular involvement of the end user in the healthcare workflow (e.g., he/she is a doctor, operator, health assistant, and so on). In the end the *Secure Data Access* component implemented by IdM, PDP, and PEP GEs guarantees security. Focusing on the specific interaction among GEs, we provide some details that can be useful to develop the whole system.

First of all, widgets and operators wishing to use the Javascript bindings provided by Wirecloud for accessing the FIWARE NGSI Open RESTful API in order to seamlessly interoperate with the Orion Context Broker GE must add the NGSI feature as a requirement into their description files (config.xml files). Listing 1 shows an example of connection query. In order to create queries, it is required to enable the NGSI support to widgets/operators (XML). The following is an example of a widget description using the XML flavour of the Workflow Description Language (WDL):

Listing 1. Example of connection query.

```
connection.query([{
      isPattern: true,
      id: MashupPlatform.prefs.get('id')
    }],
    null,
    {
      flat: true,
      onSuccess: function (discovered){
      ...
      }
    }
);"
```

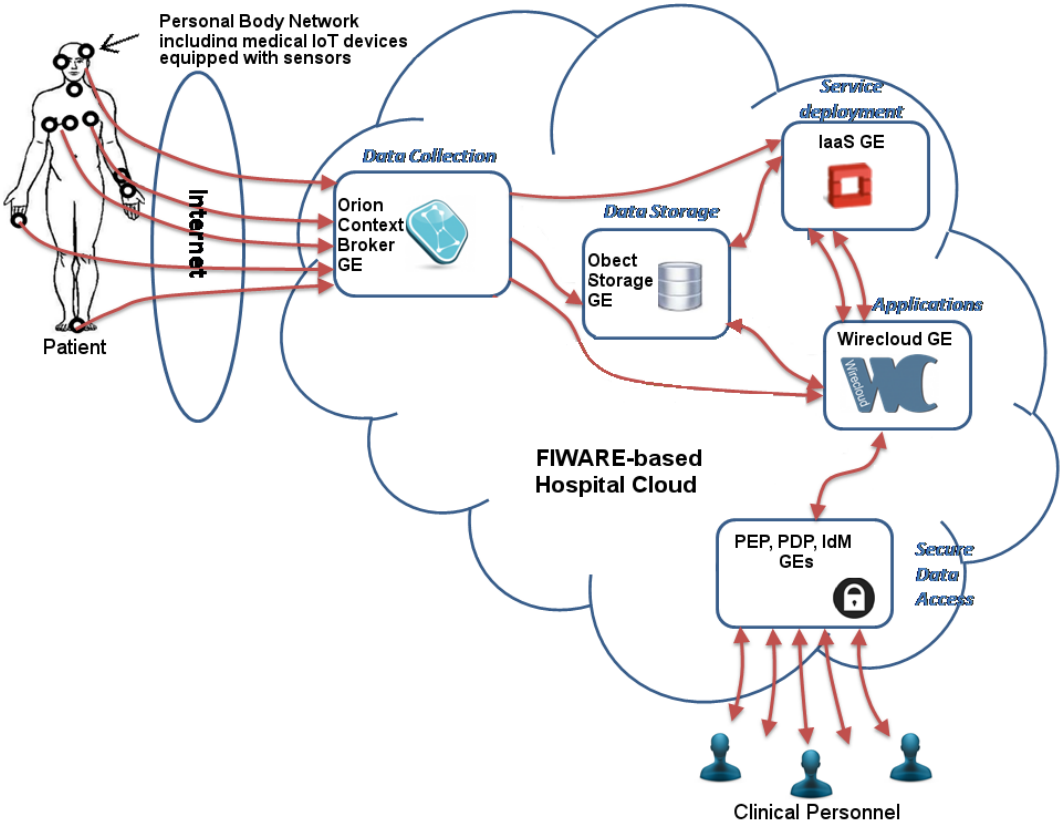

Figure 4. Skeleton of GE composition.

Listing 2. Example of Widget description using the XML flavour of the workflow description language (WDL).

```
<?xml version='1.0' encoding='UTF-8'?>
<widget xmlns="http://wirecloud.conwet.fi.upm.es/ns/macdescription/1"
        vendor="CoNWeT" name="observation-reporter"
        version="1.0">
  <details>
    <title>Observation Reporter</title>
    <authors>aarranz</authors>
    <email>aarranz@conwet.com</email>
    <image>images/catalogue.png</image>
    <smartphoneimage>
        images/smartphone.png
    </smartphoneimage>
    <description>
        Creates a new observation
    </description>
    <doc>http://www.envirofi.eu/</doc>
  </details>
  <requirements>
    <feature name="NGSI"/>
  </requirements>
  <wiring/>
  <contents src="index.html"
            contenttype="text/html"
            charset="utf-8"
            useplatformstyle="true"/>
  <rendering height="20" width="5"/>
</widget>
```

It is also possible to enable the NGSI support to widgets/operators (RDF).

In order to create a connection from Wirecloud Widgets to NGSI Context Brokers, first of all, the developer has to make use of the NGSI API by creating a connection with the NGSI Context Broker the developer is going to use. This can be accomplished with the following code:

Listing 3. Connection with the NGSI Context Broker.

```
var ngsi_connection =
    new NGSI.Connection(ngsi_server, options);
```

The *ngsi_proxy_url* option is required for being able to create subscriptions handled by widgets/operators. Also, if you are connecting to a Orion Context Broker using the IdM authentication, you will need to pass the required authentication credentials. This can be accomplished in two ways:

- Making use of the *request_headers* option and passing directly the required authentication header;
- Making use of the *use_user_fiware_token* option to make the NGSI API user the FIWARE's OAuth2 token of the current user (obtained by Wirecloud from the IdM). Any request made by a connection using this option will fail if the current users does not have a valid token (take into account that anonymous users and users authenticated using other authentication back-ends fall into this category). If you are worried about security, take into account that the OAuth2 token is injected in the request by the Wirecloud's proxy

The code below shows a NGSI connection creation using the resources available at FIWARE Lab:

Listing 4. NGSI connection creation.

```
var ngsi_connection =
    new NGSI.Connection(
      'http://orion.lab.fiware.org:1026',
      {use_user\_fiware_token: true,
       ngsi_proxy\_url:
          'https://ngsiproxy.lab.fiware.org'
});
```

Once created the connection, you will be able to use the NGSI API bindings (in the example, through the *ngsi_connection* variable).

Queries are the most basic operations that can be executed in a Orion Context Broker. This operation can be accessed thought the query method of the connection object:

Listing 5. Connection query.

```
var entityIdList = [
    {type: 'Patient', id: '.*', isPattern: true}
];
var attributeList = ['current_position'];
var options = {
    flat: true,
    onSuccess: function (data) {
        ngsi_subscriptionId = data.subscriptionId;
    }
};
ngsi\_connection.query(entityIdList, attributeList, options);
```

The first parameter is the list of entities you are interested on. In our case, we are interested in all *Patient* entities. This is accomplished using the *isPattern* option that allows us to use a regular expression that matches with any *id*.

The second one is the list of attributes you are interested in. In our case, we are interested only in the *current_position* attribute. However, you can pass null or an empty list to indicate that you are interested in all the attributes of the selected entities. Finally, all the methods support a last parameter called options that should be used to pass the callbacks and the extra options. Any method of NGSI.connection supports at least the following callbacks:

- *onSuccess* is called when the request finishes successfully. The parameters passed to this callback depends on the invoked method. In the case of the query operation, the first parameter will contain the data returned after querying the context broker.
- *onFailure* is called when the request finishes with errors.
- *onComplete* is called when the request finishes regardless of whether the request is successful or not.

The *query* method also supports other extra options. The *flat* options is used for simplifying the data structure used for representing the returned data. This simplification relies in making the following assumptions about the returned entry set: given an entity id there is only one value for the entity's type parameter; entities do not have attributes called id or type; attribute types do not matter or are already known; attribute metadata does not matter or is already known. For example, this is the value of the data parameters passed to the onSuccess callback when using the flat option:

Listing 6. Example of parameters.

```
{
    "Patient1": {
        "id": "Patient1",
        "type": "Patient",
        "current_position":
                "43.47557, -3.8048315"
    },
    "Patient2": {
        "id": "Patient2",
        "type": "Patient",
        "current_position":
                "43.47258, -3.8026643"
    },
    "Patient3": {
        "id": "Patient3",
        "type": "Patient",
        "current_position":
                "43.47866, -3.7991238"
    }
}
```

One of the most important operations provided by the context broker is the support for creating subscriptions, in this way our system can obtain "real time" notifications about the status of the entities of our system. Subscriptions are very similar to queries. The main difference between queries and subscriptions is that queries are synchronous operations. Moreover, the Orion Context Broker will send a first notification containing the data that would be returned for the equivalent query operation. This way, you will know that there is no gap between the current values and their notified changes. Both widgets and operators can create subscriptions through the *createSubscription* method. In the previous example, this call to *createSubscription* will make the context broker to call the *onNotify* function each time the *current_position* attribute of the entities of type *Patient* is changed. You must take into account that the Orion Context Broker evaluates patterns at runtime, so using patters, one is able to receive notification about new entities provided that the notify conditions are meet.

This subscription expires after 3 h, from which time the context broker stops sending notifications. Anyway, widgets/operators can renew those subscriptions by using the *updateSubscription* method,

even if they have expired. Subscriptions can be cancelled using the *cancelSubscription* method making the context broker release any info about the subscription. In any case, Wirecloud cancels any subscription automatically when widgets/operators are unloaded. As with the query operations, one can make use of the *flat* option when creating subscriptions. The assumptions made by the *createSubscription* method will be the same as the ones used by the *query* method. The only thing that changes is that this affects the parameter passed to the notification callback instead of the success callback.

Widgets and operator can update entities using the *updateAttributes* and *addAttributes* methods. The *updateAttributes* and *addAttributes* methods use the same format for their parameters. The main difference is that *addAttribute* method will create new attributes/entities if needed, whereas *updateAttributes* will fail if the referred entities/attributes does not exist. For example, the code in Listing 7 updates the attribute *position* of the *Patient1* entity if it exists, or creates the attribute or the entity if one of them does not exist. The *response_data* parameter of the *onSuccess* callback is a summary of the accepted changes as returned by the Orion Context Broker. This info can normally be ignored, as it will be very similar to the one provided to the *updateAttributes/addAttribute* methods when the request ends successfully. If everything goes fine, the *unaccepted_changes* parameter will contain an empty array. If something goes wrong, the *unaccepted_changes* parameter will contain all the information about what changes were rejected. It is very important to take this into account as the *onFailure* callback will not be called for reporting unaccepted changes as they are treated individually by the Orion Context Broker.

Listing 7. Example of attribute update.

```
ngsi\_connection.addAttributes([{
        entity: {id: 'Patient1', type: 'Patient'},
        attributes: [
            {
                type: 'string',
                name: 'current\_position',
                contextValue: coordinates
            }
        ]
    }], {
        onSuccess: function
          (accepted\_changes, unaccepted\_changes) {
            // The Orion Context Broker processed
            // the request successfully
            if (unaccepted\_changes.length === 0) {
                // Patient created/updated successfully
                ...
            } else {
                // Something went wrong
            }
        }.bind(this),
        onFailure: function (error) {
            // General failure when
            // creating/updating the Patient
        },
        onComplete: function () {
            //
        }.bind(this)
    }
);
```

Since widgets are developed using HTML5, CSS, and Javascript, besides the FIWARE NGSI Open RESTful API, it is possible to develop other types of asynchronous interaction with other third-party systems and back-end GEs. To this end, cutting-edge technologies including Asynchronous Javascript and XML (AJAX), Web Sockets, and eXtensible Message Presencce Protocol (XMPP) can be used to achieve such communications. Figure 5 shows how the communication between a widget and a system back-end can take place.

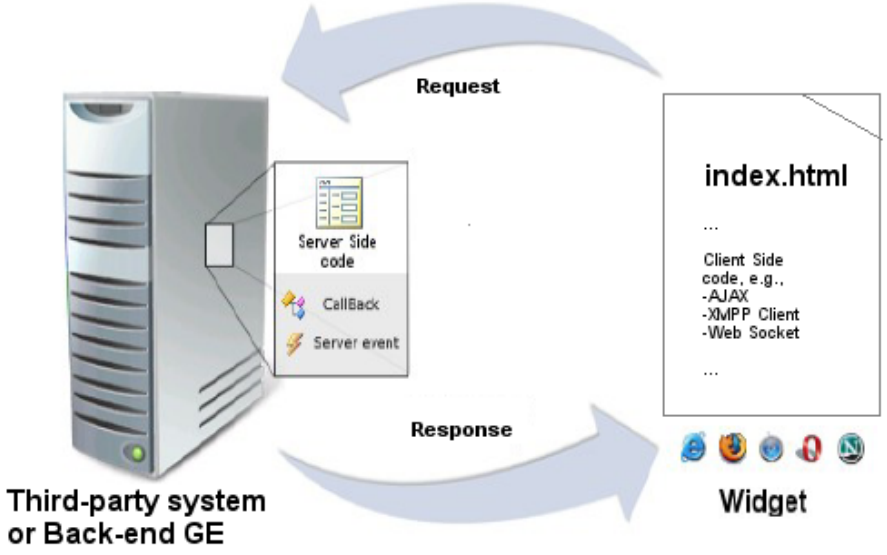

Figure 5. Interaction of a widget with a third-party system or a GE.

In the following, we provide a brief description on how to accomplish such communication mechanisms. AJAX is a programming technique that allows web application to asynchronously load contents by means of a particular Javascript function, i.e., *ActiveXObject("Microsoft.XMLHTTP")* for IE6 and IE5 and *window.XMLHttpRequest* for IE7+, Firefox, Chrome, Opera, Safari. By using these functions in the client-side code embedded in the index.html file of a widget, it is possible to asynchronously load contents processed by server-side codes placed in remote systems, e.g., legacy systems or back-end GEs. The system interface back-end can be developed by using different server-side programming languages including PHP, Asp, Asp.NET, JSP, Ruby on Rails, etc.

Communication should be immediate, secure, private, and without hurdles. The XMPP already provides the first three. But in today's world, most users will be in the browser, so even the most basic application installation can be a major hurdle. Javascrip XMPP Client (JSXC), Strophe.js, and Candy are some of the major projects that allow developers to add a few simple lines to a client-side code of a Web application in order to enable users to immediately communicate over organisation's XMPP server. They allow full-featured XMPP client in the browser, no client installation required; easy integration into existing Web page or application; one-to-one chats among people and systems, e.g., legacy systems or back-end GEs. By using these libraries in the client-side code embedded in the index.html file of a widget, it is possible to asynchronously communicate server-side codes placed in remote systems. The back-end interface can be developed using an XMPP client library such as the Smack written in Java.

WebSocket is a protocol providing full-duplex communications channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C. WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. A client-side code embedded

in a widget can be developed using the native HTML5 WebSocket library. On the server-side, Web sockets interfaces for the communication with legacy systems or GEs can be developed using different programming languases including Java, C#, and PHP.

4. Discussion and Conclusions

Nowadays, clinical centres are looking at cloud computing technology to develop new cutting-edge tele-health services and applications. In this context, Remote Patient Monitoring (RPM) at home represents a tempting opportunity for hospitals to reduce clinical costs and to improve the quality of life of both patients and their families.

However, currently, existing tele-health solutions have been conceived as "stand-alone", adopting different technological approaches that require a considerable level of complexity with high design, development, and management costs. The adoption of the cloud computing technology could push down such costs, however, it is at an early stage in the field of tele-health.

This work deals with the adoption of the FIWARE for the development of RPM and, in general, for tele-health projects aiming at supporting clinical centres interested in adopting the cloud computing technology. Specifically, we presented a step-by-step approach to develop a a RPM solution, investigating how the FIWARE platform and Generic Enablers (GEs) could be adopted and integrated. Our requirement analysis brought us to define a skeleton of the whole RPM architecture based on the integration of FIWARE GEs.

From our experience, the advantages of using FIWARE in the tele-medicine context are multiple. Hospitals are not restricted to vendor lock-in solutions. In fact, FIWARE allows software architects and developers to adopt an agile software development approach. Moreover, FIWARE allows to re-use and integrate a set of reliable cloud components that require only minimal customization in order to fit the tele-health requirements. In addition, clinical centres do not have to take care of system management tasks because these are demanded to the cloud service provider hosting the tele-health system. As far as security, FIWARE offers a set of features allowing hospitals to preserve the patients' privacy. In any case, for security reasons, clinical centres have the chance to setup theirs own "in-home" FIWARE platform where they can deploy their tele-health systems since it is based on open source code. Moreover, hospitals can rely on a wide FIWARE community that actively maintain and develop GEs. Apart from open source GEs, it is also raising a market of proprietary Specific Enables (SEs), and the offerings of FIWARE-based cloud service provider are also rising. Engineering (Italy), Telefónica (Spain), and Orange (France) have been pioneers in this sense. As far as the effort required for clinical centers in order to use FIWARE platform, we can state that it was designed to enable an agile development strategy and that grantees of the FIWARE acceleration programmes developed their applications with 100.000,00 Euros funding roughly in 12 months.

The solution discussed in this paper is also expected to be implemented as a first fully fledged CPHS for the rural and underdeveloped areas of South Africa.

With this paper, we hope to succeed in stimulating the e-health community for the adoption of FIWARE.

Author Contributions: A.C. and M.F. designed the archtecture of the system; F.G. and A.G. technically supported the FIWARE implementation; H.M. and A.B. analysed the case study in developing coutry; F.C. analysed the RPM scenario; M.V. supervised the paper.

Funding: This research was funded by the Italian Healthcare Ministry, project name "Do Severe acquired brain injury patients benefit from Telerehabilitation? A Cost-effectiveness analysis study", grant number GR-2016-02361306.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Łukasz C.; Malawski, F.; Wyszkowski, P. Holistic approach to design and implementation of a medical teleconsultation workspace. *J. Biomed. Inform.* **2015**, *57*, 225–244.

2. Pawar, P.; Jones, V.; van Beijnum, B.J.F.; Hermens, H. A framework for the comparison of mobile patient monitoring systems. *J. Biomed. Inform.* **2012**, *45*, 544–556. [CrossRef] [PubMed]
3. Risso, N.A.; Neyem, A.; Benedetto, J.I.; Carrillo, M.J.; Farías, A.; Gajardo, M.J.; Loyola, O. A cloud-based mobile system to improve respiratory therapy services at home. *J. Biomed. Inform.* **2016**, *63*, 45–53. [CrossRef] [PubMed]
4. Gerneth, M. FEST: Framework for European services in telemedicine. *Comput. Methods Progr. Biomed.* **1994**, *45*, 71–74. [CrossRef]
5. Rialle, V.; Lamy, J.B.; Noury, N.; Bajolle, L. Telemonitoring of patients at home: A software agent approach. *Comput. Methods Progr. Biomed.* **2003**, *72*, 257–268. [CrossRef]
6. Hsu, M.H.; Chu, T.B.; Yen, J.C.; Chiu, W.T.; Yeh, G.C.; Chen, T.J.; Sung, Y.J.; Hsiao, J.; Li, Y.C.J. Development and implementation of a national telehealth project for long-term care: A preliminary study. *Comput. Methods Progr. Biomed.* **2010**, *97*, 286–296. [CrossRef] [PubMed]
7. Shalom, E.; Shahar, Y.; Lunenfeld, E. An architecture for a continuous, user-driven, and data-driven application of clinical guidelines and its evaluation. *J. Biomed. Inform.* **2016**, *59*, 130–148. [CrossRef] [PubMed]
8. Harris, L.T.; Tufano, J.; Le, T.; Rees, C.; Lewis, G.A.; Evert, A.B.; Flowers, J.; Collins, C.; Hoath, J.; Hirsch, I.B.; Goldberg, H.I.; Ralston, J.D. Designing mobile support for glycemic control in patients with diabetes. *J. Biomed. Inform.* **2010**, *43*, S37–S40. [CrossRef] [PubMed]
9. Bagula, A.; Mandava, M.; Bagula, H. A framework for healthcare support in the rural and low income areas of the developing world. *J. Netw. Comput. Appl.* **2018**, *120*, 17–29. [CrossRef]
10. FI-WARE Cost-Effective Creation and Delivery of Future Internet Applications. Available online: http://www.fi-ware.eu/ (accessed on 1 October 2014).
11. Celesti, A.; Tusa, F.; Villari, M.; Puliafito, A. How the Dataweb Can Support Cloud Federation: Service Representation and Secure Data Exchange. In Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications, London, UK, 3–4 December 2012; pp. 73–79.
12. Fazio, M.; Celesti, A.; Ranjan, R.; Liu, C.; Chen, L.; Villari, M. Open Issues in Scheduling Microservices in the Cloud. *IEEE Cloud Comput.* **2016**, *3*, 81–88. [CrossRef]
13. Li, W.; Privat, G.; Cantera, J.M.; Bauer, M.; Gall, F.L. Graph-based Semantic Evolution for Context Information Management Platforms. In Proceedings of the 2018 Global Internet of Things Summit (GIoTS), Bilbao, Spain, 4–7 June 2018; pp. 1–6.
14. Steinmetz, C.; Schroeder, G.; dos Santos Roque, A.; Pereira, C.E.; Wagner, C.; Saalmann, P.; Hellingrath, B. Ontology-driven IoT code generation for FIWARE. In Proceedings of the 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), Emden, Germany, 24–26 July 2017; pp. 38–43.
15. Zahariadis, T.; Papadakis, A.; Alvarez, F.; Gonzalez, J.; Lopez, F.; Facca, F.; Al-Hazmi, Y. FIWARE Lab: Managing Resources and Services in a Cloud Federation Supporting Future Internet Applications. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC), London, UK, 8–11 December 2014; pp. 792–799.
16. Carnevale, L.; Galletta, A.; Fazio, M.; Celesti, A.; Villari, M. Designing a FIWARE Cloud Solution for Making Your Travel Smoother: The FLIWARE Experience. In Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA, USA, 18–20 October 2018; pp. 392–398.
17. Carnevale, L.; Celesti, A.; Pietro, M.D.; Galletta, A. How to Conceive Future Mobility Services in Smart Cities According to the FIWARE frontierCities Experience. *IEEE Cloud Comput.* **2018**, *5*, 25–36. [CrossRef]
18. De Fátima Pereira Marquesone, R.; de Brito Carvalho, T.C.M.; Guimarães, L.B.; Dias, E.M. A FIWARE-Based Component for Data Analysis in Smart Mobility Context. In Proceedings of the 2017 IEEE First Summer School on Smart Cities (S3C), Natal, Brazil, 6–11 August 2017; pp. 25–30.
19. Souza, A.; Pereira, J.; Oliveira, J.; Trindade, C.; Cavalcante, E.; Cacho, N.; Batista, T.; Lopes, F. A data integration approach for smart cities: The case of natal. In Proceedings of the 2017 International Smart Cities Conference (ISC2), Wuxi, China, 14–17 September 2017; pp. 1–6.
20. Araújo, A.; Kalebe, R.; Girão, G.; Filho, I.; Gonçalves, K.; Melo, A.; Neto, B. IoT-Based Smart Parking for Smart Cities. In Proceedings of the 2017 IEEE First Summer School on Smart Cities (S3C), Natal, Brazil, 6–11 August 2017; pp. 31–36.

21. Ferreira, D.; Corista, P.; Gião, J.; Ghimire, S.; Sarraipa, J.; Jardim-Gonçalves, R. Towards smart agriculture using FIWARE enablers. In Proceedings of the 2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC), Funchal, Portugal, 27–29 June 2017; pp. 1544–1551.
22. Corista, P.; Ferreira, D.; Gião, J.; Sarraipa, J.; Gonçalves, R.J. An IoT Agriculture System Using FIWARE. In Proceedings of the 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Stuttgart, Germany, 17–20 June 2018; pp. 1–6.
23. Andriani, P.; Briguglio, L.; Lombardo, L.; Nigrelli, M.; Pellegrino, D.; Torres, J.S.; Voulkidis, A. FIWARE generic enablers as building blocks of a marketplace for energy. In Proceedings of the eChallenges e-2015 Conference, Vilnius, Lithuania, 25–26 November 2015; pp. 1–10.
24. Barreto, L.; Celesti, A.; Villari, M.; Fazio, M.; Puliafito, A. Identity management in IoT Clouds: A FIWARE case of study. In Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS), Florence, Italy, 28–30 September 2015; pp. 680–684.
25. Oliveira, C.T.; Moreira, R.; de Oliveira Silva, F.; Miani, R.S.; Rosa, P.F. Improving Security on IoT Applications Based on the FIWARE Platform. In Proceedings of the 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 16–18 May 2018; pp. 686–693.
26. Omosebi, O.; Sotiriadis, S.; Bessis, N. An Openstack Based Accounting and Billing Service for Future Internet Applications. In Proceedings of the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, Switzerland, 23–25 March 2016; pp. 618–623.
27. Oh, S.; Kim, Y. Development of IoT security component for interoperability. In Proceedings of the 2017 13th International Computer Engineering Conference (ICENCO), Cairo, Egypt, 27–28 December 2017; pp. 41–44.
28. Preventis, A.; Stravoskoufos, K.; Sotiriadis, S.; Petrakis, E.G.M. Personalized Motion Sensor Driven Gesture Recognition in the FIWARE Cloud Platform. In Proceedings of the 2015 14th International Symposium on Parallel and Distributed Computing, Limassol, Cyprus, 29 June–2 July 2015; pp. 19–26.
29. Omosebi, O.; Sotiriadis, S.; Asimakopoulou, E.; Bessis, N.; Trovati, M.; Hill, R. Designing a Subscription Service for Earthquake Big Data Analysis from Multiple Sources. In Proceedings of the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Krakow, Poland, 4–6 November 2015; pp. 601–604.
30. Sotiriadis, S.; Vakanas, L.; Petrakis, E.; Zampognaro, P.; Bessis, N. Automatic Migration and Deployment of Cloud Services for Healthcare Application Development in FIWARE. In Proceedings of the 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Crans-Montana, Switzerland, 23–25 March 2016; pp. 416–419.
31. The Future Internet CHallenge eHealth (FICHe) Acelerator Project. Available online: http://www.f6s.com/fiche (accessed on 30 September 2015).
32. Fazio, M.; Celesti, A.; Márquez, F.G.; Glikson, A.; Villari, M. Exploiting the FIWARE cloud platform to develop a remote patient monitoring system. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 264–270.
33. Europe Targets U.S. Web Firms. Available online: http://www.wsj.com/articles/french-german-officials-call-for-fresh-look-at-internet-giants-1417110508 (accessed on 30 Novembre 2018).
34. OpenStack Open Source Cloud Computing Software. Available online: www.openstack.org (accessed on 1 October 2018).
35. Surridge, M.; Alvarez, F.; Carrillo, M.; Salvadori, E.; Hierro, J.; Bohnert, T. Trade-offs and responsibilities in phases 2 and 3 of the FI-PPP Program. In Proceedings of the Invited presentation at the FI-PPP Phase II Information Day, European Commission, Brussels, Belgium, 11–12 September 2012.
36. Mulfari, D.; Celesti, A.; Puliafito, A.; Villari, M. How Cloud Computing Can Support On-demand Assistive Services. In Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility, Rio de Janeiro, Brazil, 13–15 May 2013; pp. 27:1–27:4.
37. Persico, V.; Pescapé, A.; Picariello, A.; Sperlí, G. Benchmarking big data architectures for social networks data processing using public cloud platforms. *Future Gener. Comput. Syst.* **2018**, *89*, 98–109. [CrossRef]
38. Fazio, M.; Bramanti, A.; Celesti, A.; Bramanti, P.; Villari, M. A Hybrid Storage Service for the Management of Big e-Health Data: A Tele-Rehabilitation Case of Study. In Proceedings of the 12th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Floriana, Malta, 13–17 November 2016; ACM: New York, NY, USA, 2016; pp. 1–8.

39. OMA. NGSI Context Management v1.0. 2012. Available online: http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf (accessed on 1 October 2018).
40. Galan, F. Orion Context Broker: Introduction to Context Management (I). 2015. Available online: http://www.fiware.org/2015/02/19/orion-context-broker-introduction-to-context-management-i/ (accessed on 1 October 2018).
41. FIWARE. FIWARE NGSIv2 Specification. 2018. Available online: http://fiware.github.io/specifications/ngsiv2/stable (accessed on 1 October 2018).
42. OASIS. Security Assertion Markup Language (SAML). Available online: http://saml.xml.org/saml-specifications(accessed on 1 October 2018).
43. OASIS. eXtensible Access Control Markup Language (XACML). Available online: https://www.oasis-open.org/committees/xacml (accessed on 1 October 2018).