Modular and Generic IoT Management on the Cloud

Konstantinos Douzis

Department of Electronic and Computer Engineering,
Technical University of Crete (TUC)
Chania, Crete, GR-73100, Greece
e-mail: kostasdouzis@gmail.com

Stelios Sotiriadis

Department of Electronic and Computer Engineering,
Technical University of Crete (TUC)
Chania, Crete, GR-73100, Greece
e-mail: s.sotiriadis@intelligence.tuc.gr

Euripides Petrakis

Department of Electronic and Computer Engineering,
Technical University of Crete (TUC)
Chania, Crete, GR-73100, Greece
e-mail: petrakis@intelligence.tuc.gr

Cristiana Amza

The Edward Rogers Sr. Department of Electrical and Computer Engineering,
University of Toronto, Bahen Centre for Information Technology
St. George Campus, 40, Toronto, ON M5S 2E4, Canada
e-mail: amza@ece.utoronto.ca

Abstract

Cloud computing and Internet of Things encompasses various physical devices that generate and exchange data with services promoting the integration between the physical world and computer-based systems. This work presents a novel Future Internet cloud service for data collection from Internet of Things devices in an automatic, generalized and modular way. It includes a flexible API for managing devices, users and permissions by mapping data to users, publish and subscribe to context data as well as storage capabilities and data processing in the form of NoSQL big data. The contributions of this work includes the on the fly data collection from devices that

is stored in cloud scalable databases, the vendor agnostic Internet of Things device connectivity (since it is designed to be flexible and to support device heterogeneity), and finally the modularity of the event based publish/subscribe service for context oriented data that could be easily utilized by third party services without worrying about how data are collected, stored and managed.

Keywords: Cloud computing, Internet of Things, FIWARE, Cloud services, Sensor data collector, Internet of Things Management

1. Introduction

Over the years various services and applications have been developed in the concept of future Internet (FI) (9), (17). In particular such services are available by different cloud platform nodes such as FIWARE lab[1] that is a non-commercial sandbox environment. The services follow the form of RESTFul architecture (14) that allow to talk to each other easily and in a decoupled way. In addition, the Internet of Things (IoT) involves various sensors that are embedded to every day devices ans monitor data produced by humans or by the environment in an automatic way (20). The combination of cloud computing and IoT generates a new opportunity for wide discovery (18) and such data, since more and more FI applications are available. The development of such applications that are using cloud resources becomes more efficient (scalable storage) and create a significant impact on the economic benefits e.g. because of cloud elasticity and pay on demand model (11). In addition, the data transmission speed and the large volume of data (since cloud has the ability to store and process it) makes it even more attractive.

In this work we focus on the FI concept and especially on the FIWARE platform[2] that offers public services followed by simple application programming interfaces (APIs) to facilitate the process of developing smart applications. FIWARE motivates new entrepreneurs and software developers to implement such applications in health, environment and smart city concepts by providing Generic Enablers (GEs) (3) that are the building blocks of FI applications (20). In the general concept of a smart city many IoT devices

[1]https://www.fiware.org/lab/
[2]https://www.fiware.org

and sensors are associated with cloud computing services. For example in cases of data processing produced in order to avoid natural disasters (fires, floods, etc.), control of environmental conditions, energy saving, control of patient status and other. The sheer volume of data generated by sensors, has forced the transfer of the Internet of Things concept entirely on cloud computing since traditional systems could not handle the large volume of data as well as to guarantee remote access to other systems, e.g. for integration purposes.

The FIWARE lab[3] provides software to developers that use such services to develop smart applications/services within the smart city and thus including idea of the Internet of Things. Already in recent years, the community of FIWARE has taken important steps in the development of such services that help in creating more complex applications, however all these services are oriented with vendors, IoT devices and protocols. Having said that, this work proposes a Sensor Data Collection (SDC) cloud service that focuses on the problem of collecting data from different devices and their sensors, thus moving to a vendor agnostic solution. SDC developed as a gateway among IoT devices and cloud, enabling the collection of the different sensor signals that are eventually send to various other services. The service is designed to be extensible and generalized, so IoT devices could be easily connect and communicate without any programming intervention. Also, it is modular based on the service oriented architecture (2), that allows (a) support of multiple sensors belonging to different domains (for example medical, environmental etc.), and (b) support of network gateway devices.

The work is organized as follows. Section 2 presents the motivation and Sections 3 the related approaches to this study, Section 4 demonstrates the architecture of the SDC service, Section 5 presents an analysis of the implementation aspects and demonstration of the service API and Section 6 present the experimental analysis based on the simulation of two IoT devices that are (a) the Netatmo environmental sensor[4] and the Zephyr HxM Bluetooth Heart Rate Monitor medical sensor[5]. Finally, in Section 7 we conclude with the summary of this work and the future research directions.

[3]https://www.fiware.org

[4]https://www.netatmo.com/en-US/site

[5]http://www.zephyranywhere.com

2. Motivation

This work is based on FIWARE that is a non-commercial platform that offers general purpose services called Generic Enablers (GEs) that are in the form of APIs. In particular, GEs are provided by cloud computing infrastructure as SaaS (4) and if combined can constitute a special-purpose service called Specific Enablers (SEs), which could be used for developing solutions for more complex problem. FIWARE enables developers to obtain services as infrastructure (IaaS), creating virtual machines and allocating computing resources in the FIWARE lab (23).

FIWARE lab is based on the Openstack (5) platform that is an open source software, which allows the creation of a cloud computing systems. The latter are designed according to Openstack standards, thus consisting of a centralized architecture encompassing various smaller pieces of services that are responsible for controlling and managing the high volume computing resources (16). In this work we utilize and OpenStack system and FIWARE GEs to propose an architecture for a sensor data collection service on the cloud. The solution is modular, decentralized and reusable (19) thus allows IoT devices to easily to attach over the service. We are motivated by the works in publish/subscribe systems in clouds and inter-clouds as in (1), (8), (22), (7), (10) and (15).The basic characteristic of the proposed service is the simplicity of use at any time requested by the user. Such reusable services are very important in a cloud computing because it allows developers to model complex systems. Another important advantage is the modularity, that is to say the replacement of one individual service (GE) in case of a new version or a failure.

We implement our service within the IoT concept based on a service centric architecture as in (6) that is based on the fact that a large problem can be solved optimally and efficiently if it is divided into smaller parts. The advantages of such modular architectures are:

i. The services are reusable and can be made available on a larger scale.
ii. It provides faster and more efficient debugging and leading to improved fault tolerance.
iii. It involves shorter time with regards to the distribution of new products and applications.
iv. The services are not bounded to the system, thus can be easily replaced.
v. In case of integrating to a new system it does not require changes to the internal procedures of the service.

The SDC service has been developed in the form of the so called protocol adapters[6] that are implementations developed specifically for communication protocol (for example Wi-Fi, ZigBee, Bluetooth, etc.) as well as for specific devices. These services provide APIs, with functionalities such as data sending and alerting in case that a stimulus is generated from the systems. Also, it is possible to retrieve the characteristics of a device. Usually, the resulting response in a method that calls the service API is a standard data JSON (JavaScript Object Notation).

In this work we aim to develop a more generalized service that will allow data collection and storage from various IoT devices without worrying about protocols or device specifications. Thus, we aim to "transform" sensors to flexible APIs so data could easily be flown over the Internet to other services. Two main issues that the SDC service is focused are as follows.

i. The issue of having many different communication protocols between devices and network gateway. The main communication protocols on modern sensors are the Wi-Fi, Bluetooth, ZigBee, etc. Thus there is a need for a service that implements interfaces according to these standards, so to allow easily integration and communication between services and IoT devices.
ii. There is a huge variety of devices because companies provide proprietary APIs to collect data from the sensors, so the implementation of a service for commercial sensors seems quite tricky.
iii. The large volume of data produced by devices requires a new solution for scalable data storage. In addition, big data that are collected from different devices have different schemas thus a more sophisticated way is required for data storage.

The motivation of our work is based on the fact that to the best of our knowledge there is not a FIWARE service capable of managing, storing and sharing information in such way. Users who use this service may be persons, services and applications developed in FIWARE and other development environments. The proposed solution manages users and sensors for the immediate updating and subscribes users on data updates for each sensor. The basic functions supported are (a) add, remove and update sensors by the administrator, (b) add, remove and update user subscriptions, (c) add,

[6]http://catalogue.fiware.org/enablers/protocol-adapter-mr-coap

remove and update user rights in sensors assistance from the administrator, (d) update subscribers' sensors, (e) identification and delete users from the administrator and (f) database support with historical data belonging to different sensors.

Having said that, the contribution of this work includes the following.

i. The proposed architecture is dynamic and expandable, for example it could be easily integrated with services such as data analysis and processing.
ii. The service it self it could be used easily, is generalized to support multiple IoT devices and protocols and provides a flexible RESTFul API (14). This allows third party services and users to take advantage of the cloud technology and subscribe to IoT devices and their data remotely and on demand.
iii. The service is modular by separating front-end (IoT devices) and back-end (cloud system) and supports big data storage since it includes a scalable NoSQL database.
iv. It is compatible with any any service that supports a REST API e.g. with FIWARE services, thus it hides the internal service implementation details, its stateless and therefore easily scalable and provides loosely coupling.

Next Section 3 presents the works and technologies directly related with our study.

3. Related Works

As mentioned before, FIWARE platform provides software developers the necessary tools to build FI applications within the context of the smart city and of the IoT. FIWARE offers important benefits over the traditional systems including (a) elasticity as the platform could allow various levels of resource provisioning, (b) there is no need for software updates and maintenance, (c) increase accessibility and collaboration in terms of availability of services to 3rd party users and developers, (d) centralized security offered by the FIWARE platform (21) , (e) remote access from everywhere and anywhere through its powerful API that allows technology shift to seamless application development, and (f) customization and user tailored orientation through user personalized features (e.g., shared cloud storage collections for users) as described in (19) and (13).

In this work we utilize the following FIWARE GEs in order to integrate our solution:

i. Identity Management GE: This service covers certain aspects of users' access to networks, services and applications. Moreover, it is used for the authentication of third party service so to gain access to personal data stored in a secure environment. The KeyRock identity management[7] includes REST API interfaces for use by application developers. In our case the SDC service must register to the the KeyRock identity manager. It provides secure and private authentication from users to devices, networks and services, authorization and trust management, user profile management, privacy preserving disposition of personal data, Single Sign-On (SSO) to service domains and Identity Federation towards applications. Identity Management is used for authorizing foreign services to access personal data stored in a secure environment.

ii. JSON Storage GE: This GE supports information storage in JSON format through more abstract base type Mongo DB[8]. The service includes an API designed based on REST architecture. This GE provides NoSQL database management services through a REST API. Its users can perform CRUD (Create-Read-Update-Delete) operations on resources by using the basic HTTP methods (POST, GET, PUT, DELETE). In addition, it allows users to query over the stored resources.

iii. Publish/Subscribe Context Broker-Orion Context Broker GE[9]: The Orion Context Broker provides a publish/subscribe mechanism. Using the Orion Context Broker, users can subscribe to context elements (e.g., a room whose temperature and atmospheric pressure are measured) and get updated on context changes. In addition, they can use predefined conditions (e.g., an interval of time has passed or the context element?s attributes have changed) so they get context updates only when the condition is satisfied. This module allows users to subscribe to other users gesture collections to use them for building applications or to subscribe to a user who is broadcasting information.
In this work we utilize the subscription request solution and we create

[7]http://catalogue.fiware.org/enablers/identity-management-keyrock

[8]https://www.mongodb.org

[9]http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker

a different entity for each sensor added to the system. Then users have the ability to make subscriptions to many features that are mapped to different sensors. In addition, by setting predetermined time interval or configure an on change paramater they can get data directly from the context broker. After a successful subscription, the user receives a response from the context broker with a unique subscription identification (id) since the system supports multiple requests from different users.

iv. Event Service Specific Enabler[10]: This SE is based on two entities namely as event senders and event receivers.

- Event Sender: are system components that issue events in order to inform other system components of user interactions or system changes.
- Event Receiver: are system components that receive events in order to react on user interactions or changes with other parts of the system.

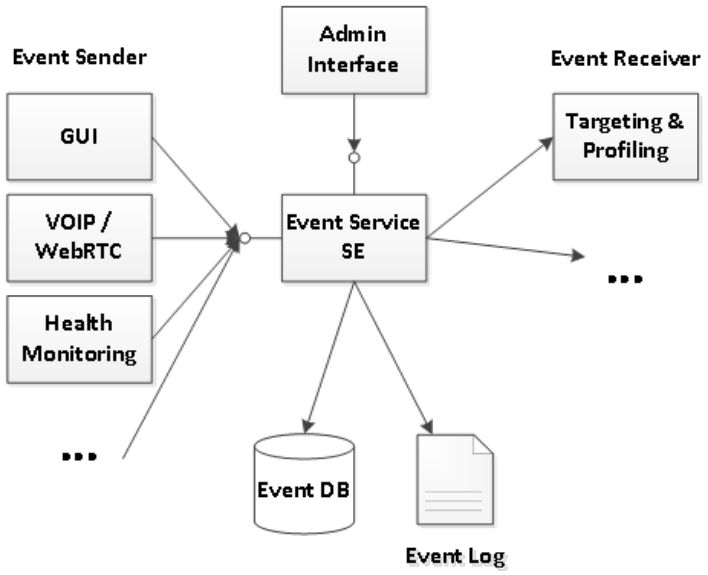

Figure 1: Event Service SE modular architecture

The Event Service SE is a web service that provides a RESTful interface

[10]http://fistarcatalogue.fiware.eng.it/enablers/event-service

to be accessed by the event senders and receivers. Additional a XML-RPC interface has been envisioned to extend the supporting interfaces. The interfaces follow the notation of the OMA NGSI-9/10 standards. It forwards events to the embedded event database, log file(s), and all subscribed event receivers. It provides a web-based interface for the administrator to subscribe/configure event receivers and to check on the event database.
However, security considerations have resulted in the extension of the service, as the current version does not guarantee the security of potentially sensitive data and how is managed and stored.

Table 1: Application Programming Interface (API) of Event Service SE

Interface	**Description**
/NGSI10/updateContext	Create/update new event/context
/NGSI10/queryContext	Query for existing event/context
/NGSI10/subscribeContext	Subscribe to event/context
/NGSI10/updateContextSubscription	Update subscription to event/context
/NGSI10/unsubscribeContext	Unsubscribe to event/-context
`/oauth2/access_token`	Get OAuth2 access token

Also, it does not necessarily ensure data security due to non-use encryption algorithms (AES) for local storage. By contrast, during the conceptualization of the SDC model we use specific services offered by the FIWARE platform for the identification of users and data storage to ensure system security. By using remote storage e.g. utilizing the JSON Storage GE the SDC service made possible the creation of a public repository, where all instances can access through the SDC provided API. In this public repository is stored information about all the sensors' schemas available on the market (e.g Kinect,Zephyr,Atmo etc). This was not possible in the Event Service SE because of the local storage. Therefore, the main differences of the SDC service with service Event Service

SE are:

i. Identification of users who have an account in the FIWARE, by external authorized FIWARE services. In compliance with the state-of-the-art authentication techniques.
ii. Remote Storage data service using a special storage service with a public repository where every instance of the service can access.
iii. JSON Storage GE uses No-SQL database Mongo DB. So no further processing time for parsing and then insertion the parsed JSON data to a relational SQL database.
iv. Ability to restrict users from specific sensors, as not every user can subscribe to specific sensors that determine the "administrator" of the instance.
v. When a new instance is been deployed is ready to use without any further configuration.

4. Architecting the Sensor Data Collection (SDC) Cloud Service

This section presents the reference architecture (Section 4.1) as a best practice model for IoT systems, the SDC architecture that derives from the reference model (Section 4.2) and the identification of the possible users (Section 4.3).

4.1. Reference Architecture

The reference architecture follows the service centric conceptualisation model as presented in (13). The reference architecture modules include producers (IoT devices), front-end (data collectors), back-end (resources for data storage and analysis) and consumers (third party users) and it is demonstrated in Figure 2.

The architecture is divided into four parts that interact with each other in order to implement a generic IoT solution. The architecture parts are (a) the producers that include sensors that generate the information (e.g. data collection), (b) the front-end side, that plays the role of a gateway between the data sent by the sensor and the data managed by the application, (c) the back-end is the actual system and includes the general purpose services for user authentication, create subscription, data storage application and other (these integrate the application logic, which makes use of standards, controls and conditions for the transfer of information on individual services) and (d) the consumers are either end users or other applications that communicate

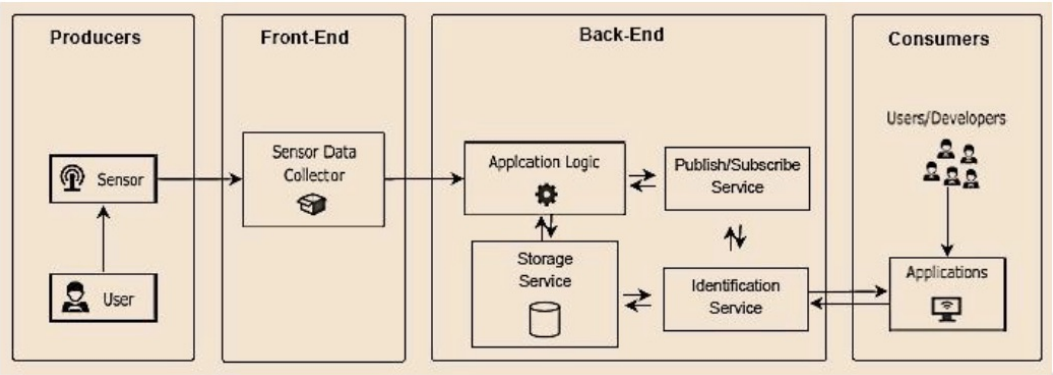

Figure 2: The reference architecture for sensor data collection in IoT systems

with the SDC API. The architecture of the SDC service is derived from the reference model and is described bellow.

4.2. Architecture of the Sensor Data Collection (SDC) Cloud Service

This section presents the architecture of the SDC service that is based upon the following concepts.

i. Requires to maintain user data security based on validated user identification service.
ii. Provides scalable remote storage of user data, sensors and licenses by using a JSON storage service for secure and quick retrieval of data stored in the service.
iii. It is build upon the RESTFul architecture for efficient and flexible communication with other services. This will facilitate the development of more complex services and applications with extra functionality. Also it uses the JSON standard for information exchange.
iv. It exploits the benefits of cloud computing in the development of services such as elasticity, flexibility, and low infrastructure and maintenance costs.

To design the system we use the top-down approach, that is to begin with the overall system architecture and then analyze in detail the subsystems that compose it (2). The overall architecture is derived from the reference architecture and consists of different modules. The user interface (Front-End) connected directly to the system management interface (Back-End) and follow Users (Users) application following the conceptual model of Figure 2.

Starting with the front-end this part is responsible for the connection of sensors and export of data to the cloud. As mentioned in the introduction, the communication protocol used by the sensors to communicate with the SDC service varies from sensor to sensor, thus it is necessary to use a mechanism that will properly encode the data according to the standards set by the SDC service. This part includes the service functions implemented as insert/delete/update description/schemas of sensors and connect/disconnect of sensor. It further includes the method by which data are sent and related with administrator rights.

The back-end part is the system management interface that consists of general purpose services that we develop for the processing and storage of data and are the same for all sensors that interact with the system. More specifically, these are the Publish/Subscribe Orion Context Broker GE and JSON Storage GE (12), which is responsible for managing user subscriptions and storing information and data respectively. Furthermore, this part contains the mechanism for identification of users that enter the application, for example the KeyRock Identity Management GE. Finally the Application Logic orchestrates the transferring of the information to the appropriate individual parts (storage, identification and information manager). Figure 3 demonstrate the detailed architecture of the service. The four segments include user identification (authentication process), management information framework (context management), application logic and storage.

The modules of Figure 3 are described as follows.

i. The user identification (authentication) takes place on two levels, first through KeyRock Identity Management GE and then through the SDC service. In fact, the system administrator is responsible for identifying the consumer to access the functions of the SDC. This is due to the fact that the administrator should have full control of the instance which was created.
ii. The context nanagement module ensures all the necessary processes such as creation of entity sensor (entity creation process), entity framework renewal (update context), create/update/delete user subscriptions and entity deletion/sensor. All these functions are included in RESTFul API offered by the publish subscribe Orion Context Broker GE.
iii. The application logic module includes the control, regulation and necessary API calls that the system is operating.
iv. The storage is used to store information about users (administrator and

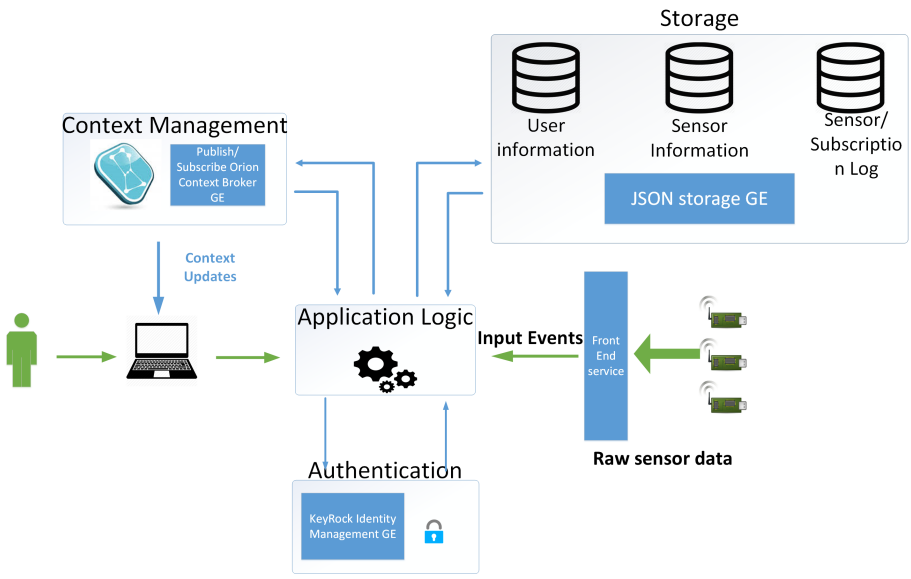

Figure 3: The sensor data collection service architecture

consumers), sensors and additional stored data history that have been sent from a sensor and user subscriptions history.

An important aspect of a generic IoT system is the orchestration of users with regards to their privileges for data accessibility. Next Section 4.3 details such conceptualization.

4.3. Users Orchestration

Here we identify the various user catagories that interact with the SDC and are either persons or IoT devices. Also we include services and applications that require to integrate functions of the SDC service in their own functionality offered via the API. In particular, users entering the system are twofold; the administrator and the consumer. The first user who enters the system automatically receives administration rights, that is responsible for entering data into the system, as well as the management of sensors and users. The administrator is therefore unique for each instance of the SDC service. On the other hand, the consumer is able to collect data from several sensors simultaneously or when sensors change some features (attributes) either on a threshold change or based on an interval. The user accessibility

requirements are divided among administrator and consumers and depending on the functionality that the developer want to map to each user.

Initially, the administrator is enrolled into the cloud platform and is the first user who logs into the system. The role of the administrator includes the following parameters.

i. He is unique for each instance of the SDC service and is responsible for the identification of users after the users have been identified by KeyRock IDM GE. If the user is not identified by the administrator has no access to any function the API.
ii. He has the ability to allow certain users to make subscriptions to specific sensors. Permission of each sensor can include multiple users simultaneously and is unique.
iii. He has the ability to create a subscription. He can also renew or delete the subscription. In case of renewal can only change the type of subscription and not the sensors' data that wants to be informed.
iv. He is responsible for sending the sensors' data to the system.
v. He can create a new sensor schema, update or delete a particular sensor schema.
vi. He can authorize a user with administration rights.

The consumers are also enrolled into the cloud platform and their role includes the following parameters.

i. To access any of the following functions to be initially identified by KeyRock Identity Mangement GE with the email address and password, and secondly from the administrator of the instance.
ii. To make a sybscription to the sensor's data first need to have the permission of the administrator.
iii. Has the ability of subscription to all sensors depending on the administrator's aproval. He can also create, renew or delete the subscription. In case of renewal can only change the type of subscription and not the sensors' data that wants to be informed.

Table 2: Users' interactions in API's functionality

Functionality	**Administrator**	**Consumer**
Retrieve all the schemas of available sensors.	✓	✓
Retrieve a specific sensor's schema.	✓	✓
Create, update and delete a sensor's schema.	✓	-
Retrieve a specific connected sensor or all the connected sensors.	✓	✓
Delete a specific connected sensor and retrieve the logged data of a specific sensor	✓	-
All the methods for user management (described in the next section 4).	✓	-
Create, update and delete a subscription.	✓	✓
Retrieve all the subscriptions of the users, retrieve a specific subscription or delete a subscription.	✓	-
Retrieve the logged data of all the subscriptions of the user.	✓	✓
Retrieve the permission of a specific sensor.	✓	✓
Create, update or delete a permission for a specific sensor.	✓	-

Functionality	Administrator	Consumer
Retrieve the context notifications of sensors' data in real time.	✓	✓
Sending sensor's data to the system.	✓	-

Next Section 5 details the implementation aspects and configurations of the SDC service.

5. Implementation of the Sensor Data Collection Cloud Service

This section presents the implementation of the solution and the RESTFul API. We detail the methods and the functionality that each of which performs. To characterize the API, we classify its functionalities into five categories similar to the requirements analysis phase. These categories of the API methods are (a) the management of sensors, (b) the management of users, (c) the management of subscriptions,(d) the management of licenses/permissions and (e) the management of sensor data. Finally, we present a discussion of the implementation of the user authentication functionality.

5.1. Management of sensors

This section presents the functionalities of the module related with the management of devices (and their sensors) e.g. on how to create, read, update and delete sensors. These are presented in the form of the method (e.g. GET /sensors) and the explanation of its functionality.

Table 3: Interfaces for sensor management

Method (HTTP)	Interface	Description
GET	/sensors	Retrieve all the schemas of available sensors.
GET	/sensors/sensorId	Retrieve the specific sensor's schema with id sensorId.

Method (HTTP)	**Interface**	**Description**
POST	/sensors	Create a new sensor's schema, stored in the public repository.
PUT	/sensors/sensorId	Update the schema of a stored sensor.
DELETE	/sensors/sensorId	Delete the schema of a stored sensor.
GET	/connected	Retrieve the schemas of all connected sensors.
GET	/connected/sensorId	Retrieve the schema of a specific connected sensor.
DELETE	/connected/sensorId	Delete/Disconnection of a specific sensor.
GET	/log/sensorId	Retrieve of the data been sent from the sensor with id sensorId.

For example, if the administrator of the instance wants to create a sensor Atmo with attributes temperature and pressure then he will send an HTTP POST request to the address 147.27.50.119/api/sensors with payload:
{ "name": "Atmo", "attributes":[{"name":"temperature","type":"celsius"} ,{"name":"pressure","type":"bar"}] }
Also, along with the payload user must send additional info in the header of the request. Content-Type must be "application/json" and Authorization must be Basic base64(username:password). The base64[11] function encodes the string username:password into another string which we have to attach in Authorization header.

[11]https://en.wikipedia.org/wiki/Base64

Response	Description
JsonException	The payload is not in JSON format.
DataFormatException	The payload data is not in the proper format like above or the name of attributes are not unique.
CurlException	Some error occured in the exchange of data between the services.

5.2. Management of users

This section presents the functionalities of the module related with the management of users e.g. on how to retrieve, authorize and delete users. These are presented in the form of the method (e.g. GET /users) and the explanation of its functionality.

Table 4: Interfaces for user management

Method (HTTP)	Interface	Description
GET	/users	Retrieve all the users of the instance.
GET	/users/userId	Retrieve the specific user with id userId.
DELETE	/users/userId	Delete the specific user with id userId.
POST	/users/admin	Give a consumer administrator rights.
GET	/users/authorized	Retrieve all the authorized users of the instance.

Method (HTTP)	Interface	Description
GET	/users/unauthorized	Retrieve all the unauthorized users.
POST	/users/authorize	Authorize a user to be able to access the instance .

For example, if an unauthorized user wants to access the interfaces of the API, administrator of the instance must send an HTTP POST request to the address 147.27.50.119/api/user/authorize with payload:
{ "email": "123@emailservice.com" }
The email provided in the payload maps to the unauthorized user.

Response	Description
JsonException	The payload is not in JSON format.
NotFoundException	No user with this email.
NotAllowedException	If the user has already been authorized or the email belongs to the administrator of the instance.
CurlException	Some error occured in the exchange of data between the services.

5.3. Management of subscriptions

This section presents the functionalities of the module related with the management of subscriptions e.g. on how to create, retrieve, update and delete user's subscription. These are presented in the form of the method (e.g. GET /subscriptions) and the explanation of its functionality.

Table 5: Interfaces for users' subscriptions management

Method (HTTP)	**Interface**	**Description**
GET	/subscriptions	Retrieve all the users' subscriptions.
GET	/subscriptions/subId	Retrieve the specific subscription with id subId.
GET	/subscription/log	Retrieve all the logged data from subscriptions of the user.
DELETE	/subscriptions/subId	Delete the specific subscription with id subId.
GET	/subscription	Retrieve the subscription of the user who calls the interface.
POST	/subscription	Create a new subscription for the user.
PUT	/subscription	Update the subscription for the user.
DELETE	/subscription	Delete the subscription for the user.

For example, if a user wants to watch two attributes of sensor test1 whenever the attribute a1 changes, then he will send an HTTP POST request to the address 147.27.50.119/api/subscription with payload:
{ "subAttributes":[{"name":"a1","sensorid":
"test1"},{"name":"a2","sensorid":"test1"}], "subType":"ONCHANGE",
"condAttributes":[{"name":"a1","sensorid":"test1"}] }

Otherwise, if the user wants to watch the two attributes of sensor test1 every 3 seconds then he sends a payload:
{ "subAttributes":[{"name":"a1","sensorid":"test1"},{"name":"a2","sensor
id":"test1"}], "subType":"ONTIMEINTERVAL", "interval":"3" }

Also, along with the payload user must send additional info in the header of the request. Content-Type must be "application/json" and Authorization must be Basic base64(username:password).

Response	Description
JsonException	The payload is not in JSON format.
DataFormatException	The payload data is not in the proper format like above.
CurlException	Some error occured in the exchange of data between the services.
SubscribeException	If the user hasn't got permission to subscribe to the specific sensor/s.

5.4. Management of permissions/licenses

This section presents the functionalities of the module related with the management of permissions e.g. on how to create, retrieve, update and delete sensors' permissions. These are presented in the form of the method (e.g. GET /permissions) and the explanation of its functionality.

Table 6: Interfaces for sensors' permissions/licenses management

Method (HTTP)	Interface	Description
GET	/sensors/sensorId/permission	Retrieve the permission for the sensor with id sensorId.
POST	/sensors/sensorId/permission	Create the permission for the sensor with id sensorId.
PUT	/sensors/sensorId/permission	Update the permission for the sensor with id sensorId.
DELETE	/sensors/sensorId/permission	Update the permission for the sensor with id sensorId.

For example, if the administrator gives permission to a consumer with email=d@e.com so to subscribe to the sensor with sensorId Atmo, the HTTP POST request is : "users":[dem2@emailservice.com]

Response	Description
JsonException	The payload is not in JSON format.
NotFoundException	The user or the sensor cannot be found.
CurlException	Some error occured in the exchange of data between the services.
PermissionException	The sensor have already a permission attached.

5.5. Management of sensors' data

This section presents the functionalities of the module related with the management of sensors' data e.g. on how to get context notifications and send sensor's data as an event. These are presented in the form of the method (e.g. GET /contextNotifications) and the explanation of its functionality.

Table 7: Interfaces for sensors' data

Method (HTTP)	Interface	Description
GET	/contextNotifications	The user is informed in real time for sensor data to which he has been subscribed.
POST	/event	Administrator has the ability to send sensor data to the system.

For example, if a user has a subscription with subid=553e93a498702c804cc and the subscribed attributes are the temperature and pressure of the Atmo sensor, the JSON with which the system response is like below:

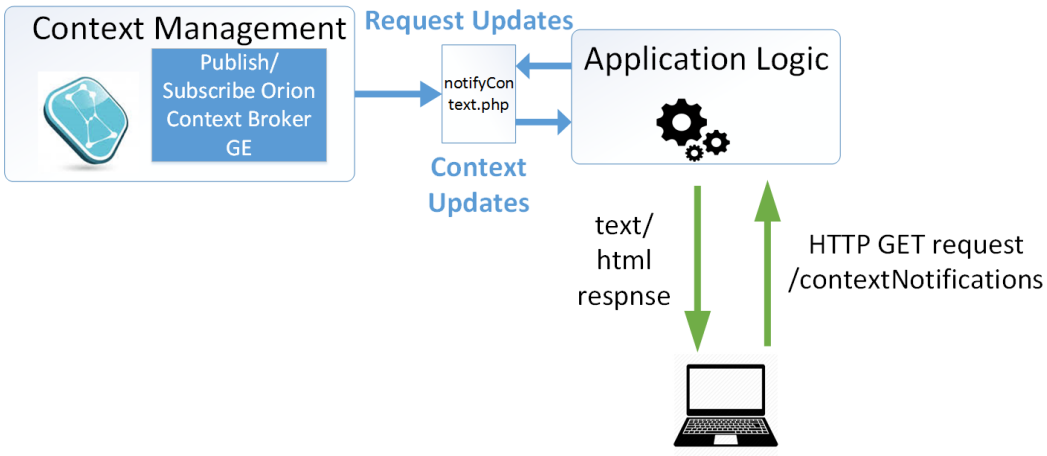

Figure 4: Context notifications on real time

{”subscriptionId”:”553e93a498702c804cc”,”originator”:”localhost”,
”contextResponses”:
[{”contextElement”:{”type”:”Sensor”,”isPattern”:”false”,”id”:”Atmo”,
”attributes”:
[{”name”:”temperature_Atmo”,”type”:”celsius”,”value”:”24”},
{”name”:”pressure_Atmo”,”type”:”bar”,”value”:”70”}]},
”statusCode”:{”code”:”200”,”reasonPhrase”:”OK”}}]}

Otherwise, if the administrator (or a front-end service) sends data from the sensor Atmo about the attributes temperature and pressure, then he will send an HTTP POST request to the address 147.27.50.119/api/event with payload:

{ ”id”: ”Atmo”, ”attributes”:[{”name”:”temperature”,”type”:”celsius”,
”value”:”20”},
{”name”:”humidity”,”type”:”percentage”,”value”:”75”}] }

Also, along with the payload user must send additional info in the header of the request. Content-Type must be ”application/json” and Authorization must be Basic base64(username:password).

Response	Description
JsonException	The payload is not in JSON format.
NotFoundException	The user who wants to get the notifications has no subscription stored or the sensor from which the administrator wants to send data cannot be found.
CurlException	Some error occured in the exchange of data between the services.

5.6. Implementation of the user authentication functionality

This sections presents the user authentication middleware that is offered by the SDC service. The middleware is called *HttpBasicAuth* and allows us to identify the user at every call of a method of API. Whenever the user makes call to a method of the API, it requires to provide credentials (in the form of username and password). The username and password are encoded in the form of base64 (username: password) and be placed as a header to the HTTP request made by the user. We implement the SDC to utilize two middlewares, (a) the first is used for identification and (b) the second for caching.

The authentication middleware consists of functions to authenticate and store the user who use a method of the provided API. Originally the middleware specifies that the requested address URL/api/ does not require user authentication. After a function of the middleware is called to authenticate the user through the appropriate method of the KeyRock Identity Management GE `https://account.lab.fiware.org/api/v1/tokens.json?email=123@gmail&password=123` using the user's email and password. According to the API of the KeyRock the call this method has resulted in the identification of the user and gives a response to the user with a session token and a status code 200 OK.

Once the user is identified it checks if this user has already been stored in the *Users* collection through the caching middleware. This middleware utilizes the higher chance of the administrator's credentials to be checked for a small period of time after the first sensor's update to the SDC. With this methodology we save time for the administrator's credentials to be retrieved from the cached data instead of the stored data in JSON Storage GE. If the user is already been stored after the identification, then the execution of middleware ends and the program shifts to the execution of the API. If

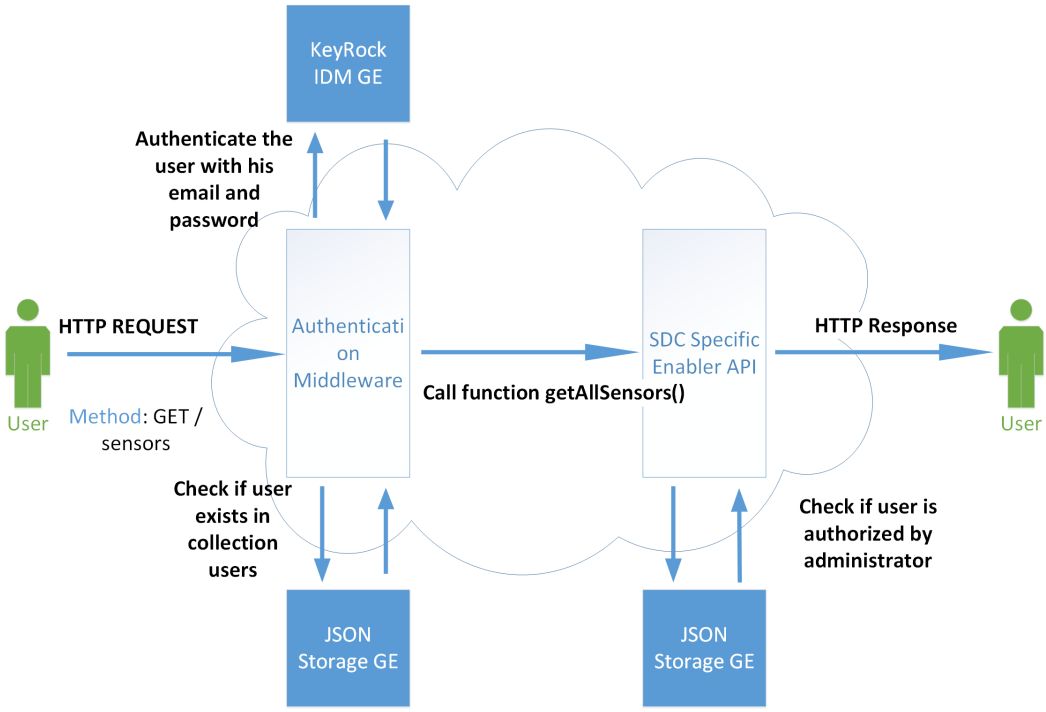

Figure 5: Utilization of authentication middleware for user identification

the user is not stored in the *Users* collection then first a check is made to determine whether it has rights as administrator or not and then is being stored in the collection. At that point the execution of the middleware ends and the API calls the requested method for execution.

6. Experimental Analysis

This section presents the experimental analysis of the work in order to demonstrate the effectiveness of the proposed solution. We utilize two IoT devices that are (a) the Netatmo (Environmental sensor)[12] and (b) the Zephyr HxM Bluetooth Heart Rate Monitor (medical sensor)[13]. The Netatmo sensor is monitoring climatic changes (temperature, humidity, etc. pressure). The device supports services for storage and processing of data to a private cloud

[12]https://www.netatmo.com/en-US/site
[13]http://www.zephyranywhere.com

infrastructure. The sensor is met in applications for remote environmental control and anticipation of natural disasters such as fire or frost. The Zephyr HxM Bluetooth Heart Rate Monitor uses the Bluetooth communication protocol, and sends to a mobile device medical data of the person who wears it. It can take measurements such as heart rate, speed, distance and time between two peaks in an electrocardiogram.

We test the performance of the SDC by simulating the two IoT devices and we check whether can the SDC to respond fully functional on a continuous sensor data. The experiment we conducted is performed in four phases. Each phase includes two users who subscribe to data provided by two sensors in the system, the Zephyr is medical sensor and measures the heart rate in bpm (beats per minute) and environmental Atmo sensor, which measures the temperature in centigrades and humidity in percentage.

We decided that the experimental analysis will apply for the method POST /event of the provided API. This method transfers the incoming data to the most internal services (e.g JSON Storage, Identity Management etc.), so we know that this method will have the slowest response time of all the methods in the provided API. Hence, it is so crucial to measure the performance of the proposed solution. To simulate the operation of these sensors we created a simulation software for each sensor and we send sensor data to a 5 minutes window at a time been set by each phase of the experiment.The time data refresh intervals of the sensors cover a wide range of 15 seconds in the first phase until 2 seconds in 4th phase. Table 8 demonstrates the experimental analysis and results.

Table 8: Experimental results for simulations of sensors Zephyr and Atmo. All the measurements are in seconds.

Device	Zephyr	Atmo	Results Zephyr	Results Atmo
1st phase	15	15	1.6	1.6
2nd phase	10	10	1.6	1.6
3rd phase	5	5	1.7	1.7
4th phase	Random (2 to 4)	Random (2 to 4)	1.8	1.8

Notably, the lower limit to determine the time was 2 seconds as a shorter period would mean unexpected behavior of the SDC (the execution of the method POST /event have an average response time of 1.6 to 1.8 seconds), probably data loss or maybe memory overflow of the server. The Publish/-

Subscribe Context Broker GE by default is configured not to use a thread pool for the incoming requests, so if many incoming requests need to be handled in a narrow time window, equivalent number of threads needs to be created. That extreme situation, would probably mean that the server could crash out of memory.

Below we present the results of the experiment, which is the time difference between the time that the call is made in the SDC (call POST /event) for sending sensor data and timing data that is displayed to the user (GET /contextNotifications). So "Results Zephyr" column indicates the time difference between the call POST /event have been made and the time that the users see the Zephyr's updated data in the browser.

With regards to the response times of Table 8 , we conclude that there are small time differences between "Zephyr" and "Atmo" among the four phases. That difference can be justified due to the continuous data burden. The more incoming requests are pushed into the service the more likely is the time difference to be wider. The average response time for the method POST /event to be executed is around 1.7 seconds.This delay can be justified by the sequential requests been made to the Context Broker GE with an average response time 700ms and the two requests been made to the JSON Storage GE with an average time of 400ms. Finally, if we include in these the delays due to the network it is apparently that the time delay of 1.6 to 1.8 seconds could be considered as realistic.

7. Conclusions

The SDC service developed to allow efficient, fast and on the fly IoT data collection and storage to a cloud system. Our solution allows the efficient management of users and sensors and an on the fly updating of the subscribers (users) with regards to data updates of each sensor. We take advantage of the benefits offered from the combination of these technologies. The next list demonstrates the basic functionality of the services.

- The service allows to add/remove/update sensors by the administrator.
- It allows to add/remove/update user subscriptions easily using the API.
- Administrator can add/remove/update user rights for sensors.
- It includes a database with historical data for system sensors.

- The service is expandable and can be added to other services that require to expand their functionality such as data analysis tools.
- The architecture is modular and the solution is easy to use, based on the RESTFul API that is available for utilization over the Internet.
- The solution is generalized, easy to use and compatible with FIWARE thus could be easily integrated to other cloud applications.

The future research directions include the definition of patterns to specific sensor data thus the users could be notified according to patterns and rules. For example, if the temperature of an internal space continuously rises over 5 minutes then the user is alerted. Another aspect is to ensure further security in the data management by the service. Finally, we aim to explore scaling behaviour of the JSON storage module in order to experiment how big data affects performance of the system.

8. References

[1] A. Antonic, M. Marjanovic, K. Pripui, and I. P. arko. A mobile crowd sensing ecosystem enabled by cupus: Cloud-based publish/subscribe middleware for the internet of things. *Future Generation Computer Systems*, 56:607 – 622, 2016.

[2] J. Bih. Service oriented architecture (soa) a new paradigm to implement dynamic e-business solutions. *Ubiquity*, 2006(August):4:1–4:1, Aug. 2006.

[3] J. Brogan and C. Thuemmler. Specification for generic enablers as software. In *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, pages 129–136, April 2014.

[4] V. Chang. The business intelligence as a service in the cloud. *Future Generation Computer Systems*, 37(0):512 – 534, 2014. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.

[5] A. Corradi, M. Fanelli, and L. Foschini. {VM} consolidation: A real case based on openstack cloud. *Future Generation Computer Systems*, 32(0):118 –

127, 2014. Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures.

[6] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[7] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione. Interconnecting federated clouds by using publish-subscribe service. *Cluster Computing*, 16(4):887–903, 2013.

[8] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione. A knowledge-based platform for big data analytics based on publish/subscribe services and stream processing. *Knowledge-Based Systems*, 79:3 – 17, 2015.

[9] A. Galis and A. Gavras. *The Future Internet: Future Internet Assembly 2013 Validated Results and New Horizons.* Springer Publishing Company, Incorporated, 2013.

[10] X. Ma, Y. Wang, Q. Qiu, W. Sun, and X. Pei. Scalable and elastic event matching for attribute-based publish/subscribe systems. *Future Generation Computer Systems*, 36:102 – 119, 2014. Special Section: Intelligent Big Data ProcessingSpecial Section: Behavior Data Security Issues in Network Information PropagationSpecial Section: Energy-efficiency in Large Distributed Computing ArchitecturesSpecial Section: eScience Infrastructure and Applications.

[11] D. Petcu. Consuming resources and services from multiple clouds. *J. Grid Comput.*, 12(2):321–345, June 2014.

[12] A. Preventis, K. Stravoskoufos, S. Sotiriadis, and E. G. M. Petrakis. Interact: Gesture recognition in the cloud. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, pages 501–502, Washington, DC, USA, 2014. IEEE Computer Society.

[13] A. Preventis, K. Stravoskoufos, S. Sotiriadis, and E. G. M. Petrakis. Personalized motion sensor driven gesture recognition in the fiware cloud platform. In *Proceedings of the 2015 14th International Symposium on Parallel and Distributed Computing*, ISPDC '15, pages 19–26, Washington, DC, USA, 2015. IEEE Computer Society.

[14] S. Schreier. Modeling restful applications. In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, pages 15–21, New York, NY, USA, 2011. ACM.

[15] A. Sfrent and F. Pop. Asymptotic scheduling for many task computing in big data platforms. *Information Sciences*, 319:71 – 91, 2015. Energy Efficient Data, Services and Memory Management in Big Data Information Systems.

[16] S. Sotiriadis and N. Bessis. An inter-cloud bridge system for heterogeneous cloud platforms. *Future Gener. Comput. Syst.*, 54(C):180–194, Jan. 2016.

[17] S. Sotiriadis, N. Bessis, and N. Antonopoulos. Towards inter-cloud schedulers: A survey of meta-scheduling approaches. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, pages 59–66, Oct 2011.

[18] S. Sotiriadis, N. Bessis, Y. Huang, P. Sant, and C. Maple. Towards decentralized grid agent models for continuous resource discovery of interoperable grid virtual organisations. In *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, pages 530–535, July 2010.

[19] S. Sotiriadis, N. Bessis, and E. Petrakis. An inter-cloud architecture for future internet infrastructures. In F. Pop and M. Potop-Butucaru, editors, *Adaptive Resource Management and Scheduling for Cloud Computing*, Lecture Notes in Computer Science, pages 206–216. Springer International Publishing, 2014.

[20] S. Sotiriadis, E. Petrakis, S. Covaci, P. Zampognaro, E. Georga, and C. Thuemmler. An architecture for designing future internet (fi) applications in sensitive domains: Expressing the software to data paradigm by utilizing hybrid cloud technology. In *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, pages 1–6, Nov 2013.

[21] A. G. Vázquez, P. Soria-Rodriguez, P. Bisson, D. Gidoin, S. Trabelsi, and G. Serme. Fi-ware security: Future internet security core. In *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ServiceWave'11, pages 144–152, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] X. Xie, H. Wang, H. Jin, F. Zhao, X. Ke, and L. T. Yang. Dta: Dynamic topology algorithms in content-based publish/subscribe. *Future Generation Computer Systems*, 54:159 – 167, 2016.

[23] T. Zahariadis, A. Papadakis, F. Alvarez, J. Gonzalez, F. Lopez, F. Facca, and Y. Al-Hazmi. 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 '14, pages 792–799, Washington, DC, USA, 2014. IEEE Computer Society.