Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 4019858, 14 pages
https://doi.org/10.1155/2018/4019858

Research Article

A Multidomain Standards-Based Fog Computing Architecture for Smart Cities

Víctor Rampérez[1], **Javier Soriano**[1] and **David Lizcano**[2]

[1]*School of Computer Science, Universidad Politécnica de Madrid, Spain*
[2]*Open University of Madrid, Spain*

Correspondence should be addressed to David Lizcano; david.lizcano@udima.es

Received 6 June 2018; Accepted 1 September 2018; Published 26 September 2018

Guest Editor: Raquel Lacuesta

Many of the problems arising from rapid urbanization and urban population growth can be solved by making cities "smart". These smart cities are supported by large networks of interconnected and widely geo-distributed devices, known as Internet of Things or IoT, that generate large volumes of data. Traditionally, cloud computing has been the technology used to support this infrastructure; however, some of the essential requirements of smart cities such as low-latency, mobility support, location-awareness, bandwidth cost savings, and geo-distributed nature of such IoT systems cannot be met. To solve these problems, the fog computing paradigm proposes extending cloud computing models to the edge of the network. However, most of the proposed architectures and frameworks are based on their own private data models and interfaces, which severely reduce the openness and interoperability of these solutions. To address this problem, we propose a standard-based fog computing architecture to enable it to be an open and interoperable solution. The proposed architecture moves the stream processing tasks to the edge of the network through the use of lightweight context brokers and Complex Event Processing (CEP) to reduce latency. Moreover, to communicate the different smart cities domains we propose a Context Broker based on a publish/subscribe middleware specially designed to be elastic and low-latency and exploit the context information of these environments. Additionally, we validate our architecture through a real smart city use case, showing how the proposed architecture can successfully meet the smart cities requirements by taking advantage of the fog computing approach. Finally, we also analyze the performance of the proposed Context Broker based on microbenchmarking results for latency, throughput, and scalability.

1. Introduction

Today, more than half of the world's population live in urban areas [1–3]. This increase in the urban population together with rapid urbanization is creating great challenges for our society. Using the latest technological advances to make cities "smart" is an emerging strategy to address these challenges [4].

A smart city is an urbanized area where multiple sectors cooperate to achieve sustainable outcomes through the analysis of contextual, real-time information. Smart cities reduce traffic congestion and energy waste, while allocating stressed resources more efficiently and improving quality of life [5–7]. This is supported by a large-scale Internet of Things (IoT) system with widely deployed IoT devices (i.e., sensors and actuators) that generate a huge volume of data [8].

Cloud computing has been used as the supporting technology for the IoT (known as CloudIoT) due to its scalable and distributed data management scheme. However, some of the essential requirements of the smart cities such as low-latency, mobility support, location-awareness, bandwidth cost saving, large-scale, and geo-distributed nature of such IoT systems cannot be met by cloud computing technology [5, 8–11].

As a result, fog computing, which extends cloud computing [12–14], has emerged as a promising infrastructure to provide elastic resources at the edge of the network, and therefore it has been considered as an affordable and sustainable computing paradigm to enable smart city IoT services. Nevertheless, most of the existing fog computing frameworks nowadays define its programming model only based on their own private data model and interfaces, which

reduce openness and interoperability [8]. We argue that, in order to be able to take advantage of all the smart cities potential, fog computing frameworks and architectures must be as open and interoperable as possible.

To address this problem we have opted for a standard-based approach to enable it to be an open and interoperable solution. More specifically, we introduce the overall NGSI-based (Next Generation Service Interface, https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification) architecture for fog computing and also report its core technologies that support it. Furthermore, we introduce a concrete application example in a smart city environment to showcase how this architecture works and a performance evaluation of the main broker in terms of scalability and latency. The main contributions of this paper are highlighted as follows:

(i) *Standard-Based Architecture for Fog Computing*. We propose a fog computing architecture for the massive infrastructure of compute, storage, and network devices of the IoT services for smart cities based on the widely used standard NGSI. This leads some benefits such as the good interoperability and openness of the solution proposed for information sharing, data source integration, and context management; this is because NGSI is a standardized open data model and API and it has been widely adopted by more than 30 cities all over the world [12].

(ii) *Scalable and Low-Latency Context Management*. To overcome the limit of centralized context management and event processing, we propose a distributed context management and event processing to extend the cloud computing paradigm to the edge of the network. Moreover, we introduce a Context Broker to communicate different domains and our measurements shows that we can achieve much better performance than existing solutions in terms of throughput, latency, and scalability.

The rest of the paper is structured as follows. Section 2 shows a brief review of the related work, and Section 3 presents and describes the proposed architecture. Section 4 evaluates the proposed architecture through a use case for smart cities and also presents the performance evaluation of the Context Broker designed by the authors. Finally, we highlight the findings of this research and state out future goals in Sections 5 and 6, respectively.

2. Related Work

This section presents and summarizes the main contributions, advantages, and disadvantages of both academia and industrial related work, which has been taken as the starting point for the present study.

Many works in both academia and industrial literature claim that cloud computing is no longer a sustainable and economical model for the next generation of IoT smart city platforms and the new paradigm of fog computing, proposed by CISCO [12], which propose to extend the cloud computing to the edge of the network, is an affordable and sustainable computing paradigm to enable smart city IoT services [5, 8–10]. Despite the large number of publications on this subject, very few focus on the design of an open and interoperable fog computing architecture.

In [15], authors present a novel computational model for fog computing using Complex Event Processors (CEP) for achieving data intelligence and analytics. The use of a CEP is to identify meaningful events and then respond and take decisions quickly as possible. Deploying a CEP at the edge of the network allows the system to achieve low-latency and real-time responses. They also introduce a Fog to Cloud gateway that schedules data to the fog or to the cloud using a rule engine and system resource prediction.

According to [12], "subscribers model will play a major role in the fog computing", and therefore several works have focused on the use of the publish/subscribe communication paradigm for the fog computing. Publish/subscribe middleware is suitable for flexible data collection and dissemination in IoT services and also it can control the data acquisition process, saving bandwidth and reducing latency [16]. In [17], authors propose a fog computing architecture based on publish/subscribe model for Internet-of-Vehicles (IoV). They also claim that there is a need of a rich and common data model to express and propagate the knowledge and context information. Publish/subscribe paradigm is also used for fog computing in [18] to preserve privacy of customer energy usage data in smart grid environment using CoAP as the underlying application layer protocol. They also highlight the fact that "the publish/subscribe model was thought as a better model due to the lower network bandwidth and less message processing which in turn extending the lifetime of battery-run devices". Another example of usage of the publish/subscribe paradigm is presented in [16]. Authors present a two-tier publish/subscribe model based on a content-based and topic-based publish/subscribe middleware, called CUPUS. They also address the problem of context awareness in this scenarios of sensing process with data transmission from sensors through mobile devices into the cloud and vice versa. Another interesting feature of this work is that they introduce the idea of Cloud Brokers and Mobile Brokers to manage the context information and to elastically scale with the increase of data sources and consumers to cope with the latency and throughput requirements.

Finally, in [8] *B. Cheng et al.* propose a standard-based (i.e., NGSI-based) approach to design and implement a new fog computing-based framework, called *Fog Flow*, for IoT smart city platforms. Despite sharing some of the focus of this work such as the openness and interoperability of fog computing frameworks and therefore the use of a standard like NGSI, the main contribution of this work is a fog computing framework programming model and how tasks can be allocated in edge devices. As mentioned, authors opt for an open standard interface from Europe, open mobile alliance NGSI, and some of the reference implementations of the FIWARE platform (https://www.fiware.org/) generic enablers such as the Orion Context Broker to manage context information. Despite the similarities, we believe that there are several aspects of the proposed architecture that can be improved, as presented in this work. First, they claim

that Orion Context Broker developed by Telefonica is the most popular message broker supporting NGSI but it is not scalable due to the lack of distributed solutions and federation support. They apply two solutions to mitigate this issue: (i) scaling lightweight IoT broker up with shared IoT discovery and (ii) connecting different domains via federated brokers. We argue that this is not a real solution, and therefore we present in this work our implementation of the NGSI Context Broker specially designed to be elastic and to exploit context information. On the other hand, in this work authors consider each city a domain and we consider that this simplification is not realistic due the huge amount of devices connected in a real IoT smart city environment. For this reason, we propose a fog computing architecture in which each domain corresponds to an individual domain of a smart city (e.g., smart traffic, electricity, smart light, and waste management) and therefore we offer a communication model to integrate all the domains of a smart city in the fog computing architecture proposed.

3. Architecture

In this section we present the architecture that we propose to achieve openness and interoperability requirements of IoT smart city platforms. We propose a standards-based approach for designing the fog computing architecture using some of the core implementations of the FIWARE IoT platform.

3.1. NGSI. Open alliance Next Generation Service Interface or NGSI is an open standard interface from Europe used in industry and in large research projects, such as FIWARE. Around 90 cities from 19 countries in Europe, Latin America, and Asia-Pacific have signed up the Open and Agile Smart Cities (OASC) alliance (http://oascities.org/) for adopting the NGSI open standard in their smart city platforms [8]. Moreover, NGSI is a cornerstone in order to bring openness and interoperability to the proposed fog computing architecture due to its powerful and simple RESTful API which enables access to IoT context information. More specifically, it is designed to manage the entire lifecycle of context information, including registrations, queries, updates, notifications, and subscriptions. The data model offered by the NGSI describes the contextual information as context entities, which has an ID, type, and a set of attributes and metadata (e.g., source of information, location of IoT device, or observation areas).

3.2. FIWARE. FIWARE is an open standard platform for smart cities that offers a public, royalty-free, and truly open architecture and a set of open specifications that allow the development of digital services and innovative products to developers, service providers, enterprises, and other stakeholders for multiple domains, including Smart Cities and Industry 4.0. Moreover, the FIWARE Foundation (https://www.fiware.org/foundation/) provides an open-source reference implementation platform based on FIWARE standards, and a free experimentation environment, based on OpenStack (called FIWARE Lab). Nowadays, thousands of startups, small, and medium-sized enterprises (SMEs) and individual developers worldwide are working out their solutions based on FIWARE platform components at present.

In order to make it easier to connect to the Internet of Things, a rich set of open standard APIs and an enhanced multitenant CloudIoT environment based on OpenStack is provided by FIWARE. In particular, the FIWARE IoT Stack objective is to bring data-level interoperability to the complex plethora of standards and protocols in the world of IoT, Open Data, Data Analytics, and other related systems nowadays. More specifically, the FIWARE IoT Stack allows connecting devices, integrating all device protocols and connectivity methods, and receiving data, understanding and interpreting relevant information. In terms of access, security, and network protocols, the FIWARE IoT Stack isolates data processing and application service layers from the device and network complexity. Some of the key cornerstones of the FIWARE IoT Stack are (i) the IoTAgents (which provides support for well-known IoT standards including Ultralight2.0/HTTP, MQTT/TCP, LWM2M/CoAP, and SIGFox Cloud), (ii) the Connector Framework, (iii) the IoT Orchestrator, and (iv) the FIWARE Publish/Subscribe Context Broker component, which is directly related to this work.

The FIWARE catalogue of technologies offers the following:

(i) **Generic Enablers (GE):** a GE is a software component definition based on an open specification. For example, the FIWARE Context Broker is a GE specification based on the FIWARE-NGSI open specification.

(ii) **Generic Enabler Reference Implementations (GEri):** a software product that has been adopted as the open-source reference implementation of a FIWARE GE is said to be its FIWARE GEri. As an example, Orion is the product that has been adopted as the FIWARE Context Broker GEri.

(iii) **Generic Enablers Implementations (GEi):** any product that implements the specifications of a given FIWARE GE is considered a FIWARE GE implementation (GEi). Continuing with the example, the Context Broker that we present in this article is a GEi of the FIWARE Context Broker GE.

Internet of Things (IoT), or connecting things and objects, requires solving different problems in each of the layers of the communication model. Due to the lack of globally accepted standards and the wide variety of wireless communication technologies, the management of the data produced and/or needed by these devices usually requires the management of a very heterogeneous variety of communication protocols. Therefore, IoT platforms must offer a solution that allows this wide variety of protocols to be abstracted in order to allow the intermediation, gathering, and publication of data between the various devices in the most transparent way possible. Specifically, FIWARE's IoT platform proposes an architecture of components (i.e., generic enablers or GEs) that communicate and manage through a common and abstract data model of the devices, allowing to capture and

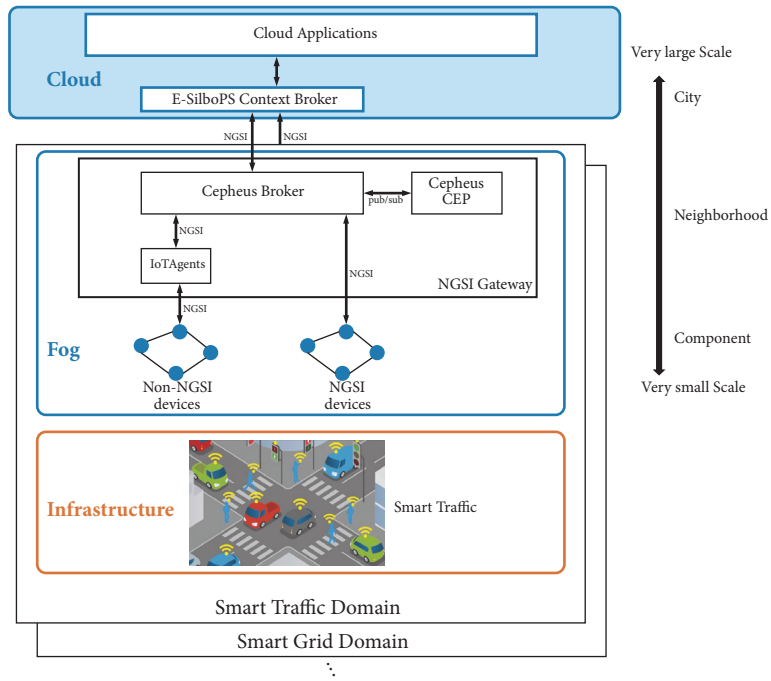

Figure 1: NGSI-based fog computing architecture.

act on this data from the IoT devices in a way as simple as reading/updating the value of attributes linked to context entities, which represent the data handled by each IoT device.

3.3. Architecture Overview. The proposed fog computing architecture is illustrated in Figure 1. This architecture has been designed following the fog computing main idea [8, 12] of allocating computationally intensive tasks, such as big data analytics and big data visualization in the cloud servers, whereas some tasks, such as stream processing can be moved to the edge of the network in the edge devices (e.g., endpoint devices with computation capabilities or IoT gateways). The architecture is composed of multiple domains, which corresponds to the different smart cities domains, such as smart traffic, smart grid, and waste management. Each domain is divided into three logical sections: (i) infrastructure, (ii) fog, and (iii) cloud. The cloud layer is common to all the smart cities domains and responsible for communicating each of the domains through a Context Broker specifically designed by authors to be elastic, low-latency, and capable of exploiting the context information because of the underlying publish/subscribe middleware, called E-SilboPS.

At the edge of the network is the sensing network, composed by all the IoT devices such as sensors and actuators. These sensors should be noninvasive, highly reliable, and low cost in order to be widely distributed at various public infrastructures to monitor their condition changes over time [5]. This sensing network generates a massive data streams that must be processed as a coherent whole with minimal latency. IoT agents are directly connected to IoT devices (i.e., sensors and actuators) to solve the problem of the wide variety of heterogeneous protocols handled by each of these devices. To this end, these IoT agents act as mediators, translating each of the different protocols used by the devices to receive and send information (HTTP ultralight, MQTT, and OMA Lightweight M2M) in a common data format and model for the entire platform: FIWARE-NGSI.

In order to move the stream processing tasks to the edge of the network we propose the use of Complex Event Processors (CEP) in favour of low-latency, mobility, streaming, and real-time applications. CEP will filter, aggregate, and match events into higher level events that can be consumed by other applications. The main goal of CEP is to identify meaningful events and then respond and take decisions as quickly as possible in order to reduce latency [15]. In the proposed architecture, the CEP analyzes event data in real-time, generates immediate insights, and enables instant response to changing conditions. More specifically, the CEP used is Cepheus CEP and Cepheus Broker. The Cepheus CEP engine works by processing incoming events and generating outgoing events mapping these events to NGSI context entities. The CEP is configured using the ESPER Event Processing Language (Esper EPL, http://esper.espertech.com/release-5.2.0/esper-reference/html/epl_clauses.html). When the CEP engine receives a context entity update (as a consequence of a previous subscription of the CEP), it will fire the corresponding events and process the related EPL statements. If one or more outgoing events are fired by the CEP engine, the corresponding update will be called to notify the Context Broker. The Cepheus Broker is a lightweight broker (similar to the Context Broker GE) only supporting two kinds of operations: (i) requests forwarding by keeping track of which components register context entities and (ii) publish/subscribe requests for context entities. This broker acts as an intermediary between the IoT agents or NGSI devices and forwards their requests to a remote broker

(Context Broker) for long-term cloud computing applications (this lightweight broker does not store any information) while allowing the CEP to subscribe to updates in context elements to do stream data processing with low-latency. According to the authors, the main purpose of a CEP is to process data in real time and filter, aggregate, and merge real-time data from different sources represented by NGSI entities. Since this Cepheus Broker and the Cepheus CEP are lightweight components, they can be deployed in edge computing nodes to respond for anomalies with low-latency.

Finally, in the cloud layer, applications for global long-term analytics are deployed alongside the context broker GEi designed by the authors. The requirements of communication between the different domains of the proposed architecture (high throughput and low-latency large-scale publish/subscribe and exploration of the context information) make the communication systems used within the fog computing networks not valid for interdomain communication. To tackle these problems, we propose our implementation of the Context Broker GE, whose main goal is to maintain and deliver context information into the IoT platform components and external systems to communicate all the smart city domains. FIWARE IoT platform is able to expose all IoT devices information and commands using the OMA NGSI9/10 interfaces by means of the Context Broker component. By using these interfaces, clients can do several operations, including registration of producer applications, update context information, and being notified when changes on context information take place or with a given frequency and query context information. Despite the intrinsic importance of the Context Broker component in the IoT stack and in this fog computing architecture, the architecture of the current FIWARE reference implementation (Orion) has some performance and elasticity limitations as a consequence of the lack of an underlying publish/subscribe communication system that allows it to manage end-to-end context information [8]. For this reason, we have developed our implementation of the FIWARE Context Broker GE based on a middleware, also designed by the authors, called E-SilboPS [19].

3.4. Our Proposal for an Elastic Context Broker GE Architecture. In this section we present an innovative Context Broker based on the Context Broker GE (FIWARE-NGSI) specification. Our proposal is designed to leverage the cloud as infrastructure to support IoT by elastically scaling in/out. More specifically, our solution relies internally on E-SilboPS [19] due to its elasticity and its capacity to manage context information (context awareness).

3.4.1. E-SilboPS. E-SilboPS is an elastic context-aware content-based publish/subscribe (CA-CBPS) middleware specifically designed to support context-aware sensing and communication in IoT-based services. Since its internal state repartitioning is transparent for publishers or subscribers, the E-SilboPS middleware is able to provide an uninterrupted service to end users even during the scaling operation. Moreover, its scaling algorithm is time-bounded and depends only on the dimension of the state partitions to be transmitted to the different nodes [19].

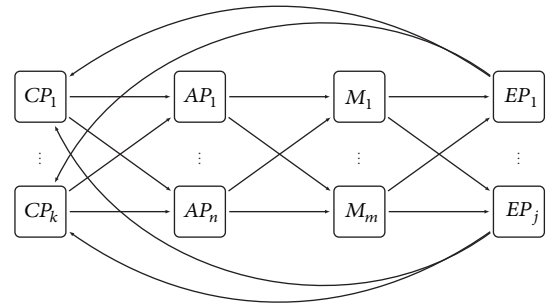

Figure 2: E-SilboPS architecture diagram. It is composed of 4 layers: Connection Point, Access Point, Matcher, and Exit Point. For clarity, this figure does not include the Distributed Coordinator because it has a direct connection to each instance of each layer.

As shown in Figure 2, from an architectural point of view, E-SilboPS is divided into four layers, which can be scaled in/out independently of the others. However, the system needs the support of a Distributed Coordinator (DC) to maintain state consistency. More specifically, E-SilboPS uses Zookeeper (https://zookeeper.apache.org/) to notify events of interest to each of the layers by creating and removing nodes. To take advantage of the resources provided by fully distributed environments, each operator instance can be deployed in a different node of a cluster or virtual machine (VM) of a cloud environment. In particular, these four operators layers are as follows:

(i) **Access Point (AP):** when receiving a subscription, it dispatches the subscriptions to the correct matcher. In the case of notifications, it broadcasts notifications to all the instances of the next layer (i.e., all the matchers).

(ii) **Matchers (M):** when receiving a subscription, it stores the subscription in its internal structure. In the case of notifications, it matches the notification against the set of subscriptions stored in its internal structure and sends the local result set to the next layer (i.e., Exit Point layer).

(iii) **Exit Point (EP):** when receiving notifications, it collects all partial results sets coming from the different matchers to merge them in the final result set and then send the notification to all the corresponding Connection Points.

(iv) **Connection Point (CP):** it is the entry and Exit Point of the system. Its purpose is threefold: on the one hand, it manages and handles the connections with clients. It also forwards subscriber subscriptions and publisher notifications to Access Points. Finally, it also sends the notifications received to interested subscribers.

Clients first establish the connection to the Connection Points and then send the notifications and subscriptions. The Connection Points send these messages to the Access Points,

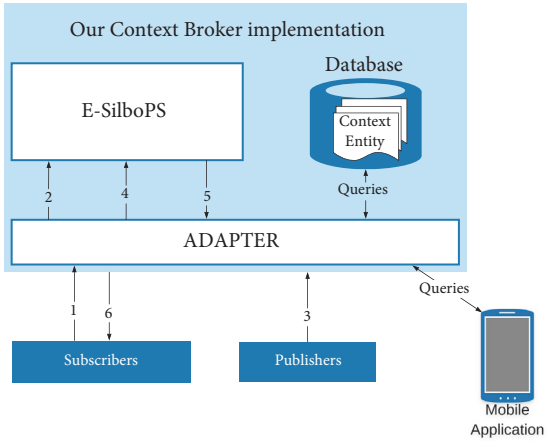

Figure 3: Architecture of our proposal for an elastic Context Broker GE.

which send them to the correct matcher instance (M1. . .Mn), selected using a selection algorithm. Once received by the matcher instance, it calculates the set of subscriptions that match the subscription and sends it to a specific Exit Point. Exit Points generate the final result set from the partial results sent by all the matchers and finally, they send the notification to the corresponding Connection Point instances to be sent to the interested subscribers.

3.4.2. The Architecture of Our Context Broker GE. As shown in Figure 3, the proposed architecture is composed of a database to store the context entities, E-SilboPS middleware to support the publish/subscribe model of communication, and an adapter to transform from NGSI to E-SilboPS message formats and vice versa and to ensure state consistency.

(1) First, interested systems send a subscription request to subscribe to context entities changes.

(2) The adapter receives the subscriptions and transforms them to E-SilboPS format. Moreover, the adapter stores the notification URL and the requested context attributes to be delivered when a notification matches the subscription condition.

(3) Systems send context entities updates with the new measured values of context attributes to the Context Broker. The adapter receives the request and updates the entities in the database.

(4) The context entities updates are forwarded to the E-SilboPS after being translated.

(5) E-SilboPS receives the notification and process them. If there is a match between the notification and a subscription, the generated subscription is sent to the adapter.

(6) The adapter transforms the notifications to the NGSI format, retrieves the notification URL and context attributes, and delivers them to the corresponding subscribers.

4. Evaluation

This section presents the evaluation results of the proposed fog architecture. First, a validation through a use case of the fog architecture is presented in Section 4.1, which explains the initial setup and the regular operation mode of the architecture for a use case of anomaly detection in a gas pipeline infrastructure for multidomain smart cities.

On the other hand, Section 4.2 shows the results of our experimental evaluation of our implementation of the Context Broker GE based on E-SilboPS, which is the cornerstone of the proposed architecture, in terms of throughput and latency compared to the FIWARE reference implementation of the Context Broker GE.

4.1. Use Case Validation. In this section, we discuss the proposed fog computing architecture through a use case of anomaly detection in a critical infrastructure for a city such as a gas pipeline and the propagation of that anomaly to other smart cities domains to a rapid and adequate reaction [20]. The main objective of this case study is to illustrate how the proposed architecture works and how it has been designed to support smart cities by taking advantage of the fog computing approach (i.e., low-latency, geo-distributed systems, and lightweight computation tasks moved to edge of the network) and the openness and interoperability offered by the NGSI standard data model and API interfaces.

The proposed use case is depicted in Figure 4. For clarity, the use case is focused on only two specific domains out of the large number of classic domains that cover smart cities. More specifically, it focuses on the domain of smart traffic and intelligent gas pipelines and how the proposed architecture is designed to exploit this multidomain approach to detect and provide a coordinated response between these domains with the lowest possible latency. More specifically, the set of interconnected devices in the gas pipeline domain consists of a series of sensors installed in the pipelines that collect information on the state of these pipelines in different sections (e.g., pressure, gas velocity, gas density, and the location of the sensor). This large amount of information collected periodically will be transmitted and processed by the corresponding NGSI gateways to detect anomalies, based on a previous configuration. An example of this configuration could be as follows: if the gas pressure reported by a sensor is greater than 70 bar for 10 minutes, an alert should be generated. Once the alert has been generated, the proposed architecture is capable of propagating that alert to other domains that may be interested with minimal latency. Continuing with the scenario outlined above, the alert previously generated could spread to the smart traffic domain, where the domain's interconnected devices are traffic lights and intelligent signals, which could cut off traffic on the streets affected by the gas incident and redirect traffic to safer routes, thus avoiding a possible catastrophe.

For the sake of clarity, the explanation of the use case validation has been divided into the initial setup configuration and the regular operation mode.

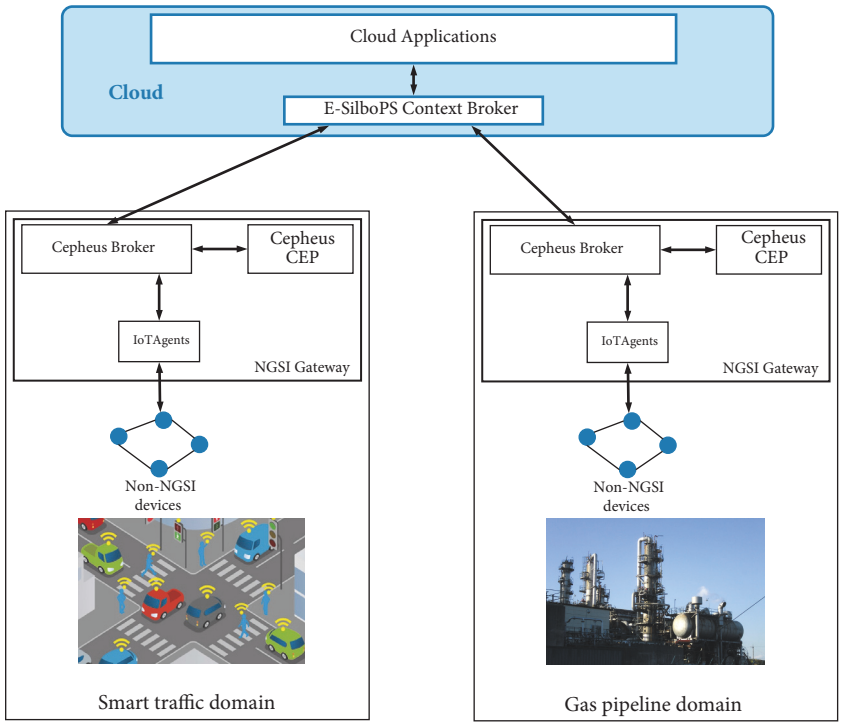

FIGURE 4: Adaptation of the proposed fog computing architecture to a multidomain use case scenario focused on smart traffic and gas pipeline domains. The proposed architecture allows detecting anomalies in a gas pipeline and creating an alert that can be broadcast to other interested domains such as the smart traffic domain to a rapid and adequate reaction with low-latency.

4.1.1. Initial Setup. Firstly, the components of the architecture require a minimum of configuration for communication between them to be as desired. More specifically, in this use case the configuration required is depicted in Figure 5.

(1) As shown in Figure 5, firstly, the IoT agents send requests to create and register a new entity for each gas pipeline sensor to the Context Broker, through the Cepheus Broker. It is important to highlight the fact that all the incoming requests to the Cepheus Broker will be forwarded to the Context Broker and the CEP, since the Cepheus Broker does not store any context entity, in order to be as lightweight as possible, and it only offers publish/subscribe operations. Listing 1 shows an example of a context entity representing a gas pipeline sensor. Context entities are composed of context attributes defined by the attribute name, type, value, and associated metadata. In this case the context attributes represents the different magnitudes measured by these sensors (i.e., `pressure`, `gas_velocity`, and `gas_density`) and their location (i.e., `location`).

(2) The CEP should be configured using the ESPER Event Processing Language (Esper EPL) [21]. Listing 2 presents a configuration for the CEP to create a notification event when the average pressure reported by gas sensors is greater than 70 bar in a time window of 10 minutes.

(3) Moreover the CEP should send a subscription request to the Cepheus Broker in order to be notified when there are updates in the gas pipeline entities.

(4) Finally, the interested actuators send subscription requests to the Cepheus Brokers corresponding to its local domain in order to be notified when the CEP generates a new gas pipeline alert. More specifically,

   (a) Gas pipeline actuators send a subscription request to the Cepheus Broker in the gas pipeline domain, to be notified when the CEP generates a new gas alert in order to respond accordingly (e.g., closing the valves of the affected pipeline gas section).

   (b) Smart traffic actuators (e.g., smart traffic lights) also send a subscription request to the Cepheus Broker in their domain (i.e., smart traffic domain) to be notified when the CEP generates a new gas alert and to respond accordingly. Note that, in this case, the Cepheus Broker in the smart traffic domain must have sent a similar subscription to the Context Broker in advance, since the Context Broker is responsible for communicating all the domains.

4.1.2. Regular Operation Mode. After setting up the architecture correctly, as explained above, the system can function normally. As show in Figure 6 and continuing with the use case,

```
POST http://CepheusBrokerAddress:1026/v2/entities
{
 "type": "gas_pipeline_sensor",
 "id": "101615A",
 "pressure": {
  "value": 50,
  "type": "double",
  "metadata": {
   "unit": {
     "value": "bar"
   }
  }
 },
 "gas_velocity": {
  "value": 20.89,
  "type": "double",
  "metadata": {
   "unit": {
     "value": "m/s"
   }
  }
 },
 "gas_density": {
  "value": 0.743,
  "type": "double",
  "metadata": {
   "unit": {
     "value": "Kg/m3"
   }
  }
 },
 "location": {
  "value": "41.3763726, 2.1864475",
  "type": "geo:point",
  "metadata": {
   "crs": {
    "value": "WGS84"
   }
  }
 }
}
```

Listing 1: Example of a POST request to create a new context entity representing a gas pipeline sensor. The entity consists of a type, an identifier and a set of context attributes (i.e. pressure, gas_velocity, gas_density and location), each of which is defined by a value, type and additional metadata.

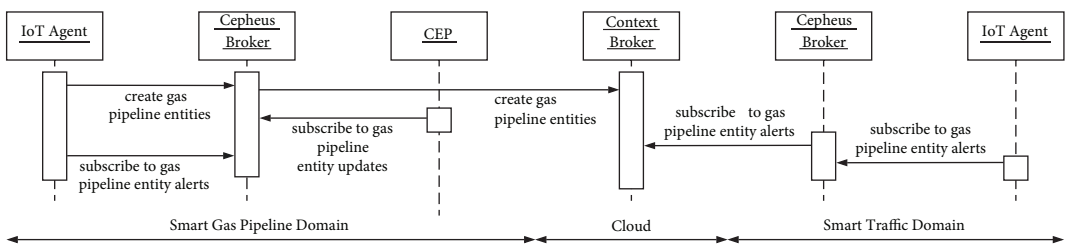

Figure 5: Interaction diagrams of the elements of the architecture for its configuration according to the use case presented.

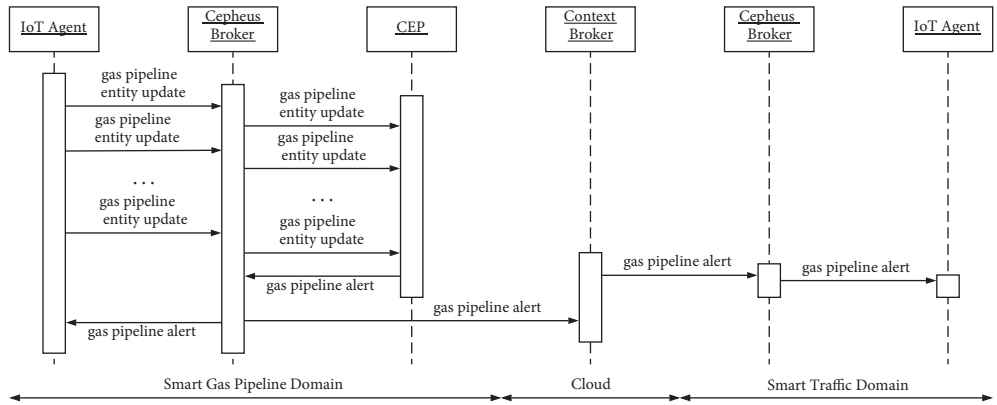

Figure 6: Diagram showing the interactions between the different elements of the proposed architecture for the use case of smart traffic and gas pipeline domains.

```
{
...
 "statements": [
     "INSERT INTO GasAlert
     SELECT avg(pressure) as pressure, gas_velocity , gas_density , location
     FROM gas_pipeline_sensor.win:time(10 min) HAVING avg(pressure) > 70"
   ]
}
```

Listing 2: Example of CEP configuration using ESPER Event Processing Language (Esper EPL). The configuration gist creates a notification event when the average pressure reported by a gas sensor is greater than 70 bar in a time window of 10 minutes. Note that the configuration is based on the context attributes of the context entity representing a gas pipeline sensor presented in Listing 1.

(1) After all the previous configuration, the IoT devices start to send context entity updates to the Cepheus Broker reporting the measured data of the gas pipeline sensors, as depicted in Listing 3.

(2) The Cepheus Broker forward the received updates from the IoT agents to the CEP, due to its previous subscription to track updates to context entities, and to the Context Broker. The Context Broker stores all the information received to be processed later by data-intensive cloud applications such as Big Data Analytics applications. Additionally, requests can be made to interrogate the status of each context entity, as shown in Listing 4.

(3) When any of the conditions of the CEP configuration are met, it creates a new event that will be sent to Cepheus Broker for dissemination to all subscribers of these events. In this case, when the average pressure of any gas pipe is greater than 70 bar for a period of 10 minutes, the CEP will generate a gas alert that will be sent to the Cepheus Broker. When this new alert arrives to the Cepheus Broker, it will forward the notification to (i) the Context Broker and (ii) to the gas pipeline actuators (previously subscribed as seen in the configuration section) to close the valves of the pipeline sections adjacent to the one with the high pressure.

(4) When the Context Broker receives the gas alert notification, it matches the subscription of the Cepheus Broker in the smart traffic domain and therefore sends the notification to the Cepheus Broker in the smart traffic domain.

(5) Finally, the Cepheus Broker of the smart traffic domain will disseminate the notification to all the subscribers. In this case, the notification will be forwarded to the smart traffic actuators (e.g., smart traffic lights) in order to modify the flow of traffic to prevent vehicles from circulating in the area close to that of the possible incident. Additionally, if other services in other domains have been subscribed, they would also receive the notification, such as emergency services.

4.2. Performance Evaluation. This section includes our experimental evaluation of our implementation of the Context Broker GE based on E-SilboPS. The different evaluations that have been carried out, which are presented in this section and whose results are also discussed in this section, focus on the study of the system performance in terms of throughput and latency, since they are two of the key performance indicators in the fog computing paradigm. More specifically, we compare our system and the reference implementation of the FIWARE Context Broker (Orion), both following the

```
PATCH http://CepheusBrokerAddress:1026/v2/entities/101615A/attrs
{
   "pressure": {
     "type": "double",
     "value": 75.78,
     "metadata": {
       "unit": {
         "type": "Text",
         "value": "bar"
        }
      }
    }
}
```

Listing 3: Example of a PATCH request to update a context entity representing a gas pipeline sensor. More specifically, the request updates the pressure value of the entity with id=101615A to 75.78 bar.

```
GET http://ContextBrokerAddress:1026/v2/entities?id=101615A
GET http://ContextBrokerAddress:1026/v2/entities?type=gas_pipeline_sensor
GET http://ContextBrokerAddress:1026/v2/entities?georel=near
GET http://ContextBrokerAddress:1026/v2/entities/101615A/attrs?attrs=pressure
```

Listing 4: Examples of entities queries. Note that each query is made by means of a GET request and the query params of the request can be used as a filter based on the context attributes of the entities.

FIWARE-NGSI specification (Context Broker GE), in terms of notification throughput and scalability. Additionally, we present the latency results of the proposed system.

We ran our experiment on a cluster composed of three machines with Linux 4.9, OpenJDK 1.8.0_121, and the following specifications:

(i) Intel Core i7-4790K 4.00GHz and 16 GB of RAM.

(ii) Intel Core i7-920 2.67GHz and 3 GB of RAM.

(iii) Intel Core i5-3550 3.30GHz and 16 GB of RAM.

In all the evaluations that have been carried out and presented in this section, prior to measuring performance and to ensure that the creation time of the objects did not affect the measured throughput, the context brokers have been loaded with one context entity with 100 context attributes and 1 M subscriptions. Moreover, 100 k notifications were created that always macheted with at least one subscription of the previously loaded ones, to be able to constantly measure the throughput of notifications. This scenario presents the worst case scenario, as it requires that all notifications be sent to at least one subscriber. On the contrary, in a real scenario, many notifications would not be sent as they would not match with any of the subscriptions. In particular, this workload has been designed and generated taking into account the estimations and the type of scenario presented in [22] to make the city of Barcelona (Spain) a smart city monitoring a huge network of different sensors (e.g., gas, temperature, noise, container sensors, or air quality).

When our GEi (our Context Broker implementation) was deployed with a 1-1-1 E-SilboPS configuration (1 Access/Connection Point, 1 Matcher, and 1 Exit Point) and after the system received 30 k subscriptions, notification throughput started to slow down, as shown in Figure 7. From 40 k subscriptions onwards, the measured throughput starts to fall steadily because the processing time is dominated by the matching process. Therefore, the measured throughput remains stable with a small amount of subscriptions (left side of the graph) due to network saturation, limiting the number of notifications that the operator can process [23]. On the other hand, Figure 7 also shows how the throughput of the Context Broker reference implementation (i.e., Orion), deployed with a single instance, slows down in proportion of the number of subscriptions just after the system was loaded with 1 subscription. It is noteworthy that while the initial throughput of our system is 30 k notifications/s, the initial throughput of the FIWARE reference implementation is 900 notifications/s.

From a scalability point of view, Figure 8 depicts how the proposed implementation of the Context Broker benefits from the E-SilboPS architecture specifically designed to scale, which is a prerequisite for elasticity [24]. As can be seen, the lower value of the throughput of notifications is obtained with the configurations that use the largest number of slices (1-6-2 and 1-6-1, with an initial throughput of 8000 notifications/s and 7000 notifications/s, respectively). However, the most important thing about this graph is that it shows how the architecture of the E-SilboPS is able to scale out. This is demonstrated because those configuration deployments with

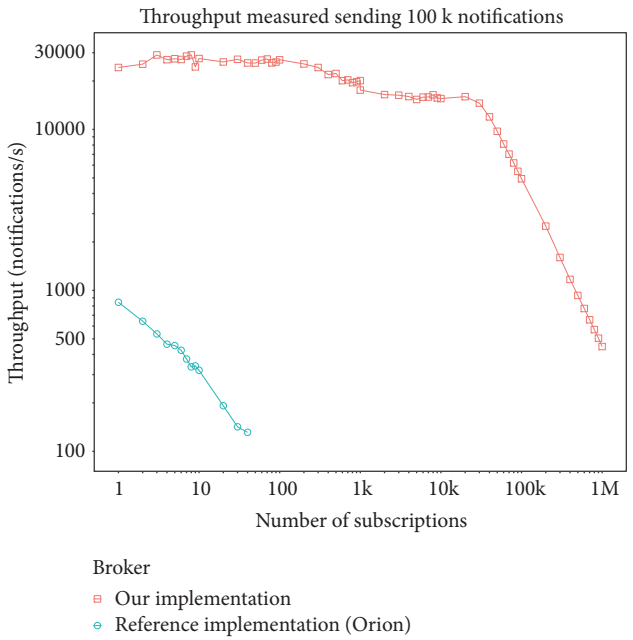

Figure 7: Notification throughput comparison between our solution and the FIWARE reference implementation (Orion).

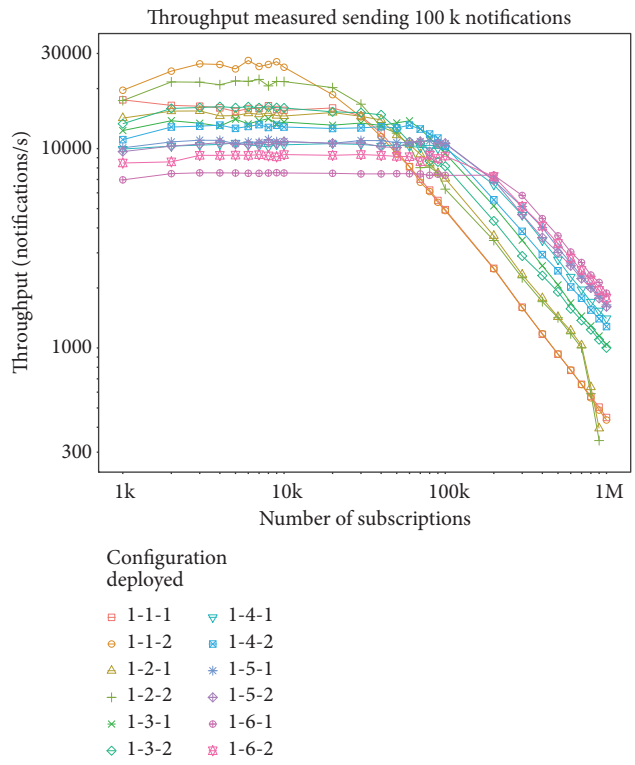

Figure 8: Notification throughput measured with different number of layer instances deployed. The legend represents the number of instances deployed for each layer (i.e., Access Point, Matcher, and Exit Point).

more matchers have a better throughput when the matching time is dominant (right side of the graph). This is because the status of each matcher instance is smaller, as there are a greater number of matcher instances for the same total number of subscriptions, and therefore it takes less time to calculate the subset of subscribers. For example, when the system is loaded with 200 k subscriptions, the deployment configurations 1-1-1 and 1-1-2 have a throughput notification of 2500 notifications/s, whereas for the same number of subscriptions, a deployment configuration of 1-6-1 or 1-6-2 maintains their initial throughput notification of 7000 or 8000 notifications/s.

For its part, Figure 9 shows the scalability results for the Orion Context Broker reference implementation. The deployment for this evaluation is composed of one mongoDB instance, N Orion Context Broker instances, and a load balancer (e.g., haproxy), configured following the performance tunes specified in the documentation (e.g., notification mode and only error log level). The configuration deployment with only one instance of the Orion Context Broker provides a notification throughput of 850 notification/s and it slows down at constant rate, since this is the configuration deployment already presented in Figure 7. With a configuration deployment of 2 or 3 instances of the Orion Context Broker, the initial throughput increases slightly until it reaches 1500 notifications/s, but it also decreases proportionally with the number of subscriptions. However, despite the huge gap between the initial throughput of both systems (independently of the configuration deployment used), the most relevant fact illustrated in Figure 9 is that there is no throughput improvement deploying more than two instances of the Orion Context Broker, and therefore, the architecture is neither scalable nor elastic.

Since low-latency is a fundamental prerequisite, we replicate the previous performance study of throughput for the proposed implementation of the Context Broker, but this time analyzing how the average notification latency value behaves with different amounts of subscriptions and notifications. Figure 10 shows the latency (in milliseconds) for different number of subscriptions and notifications. For the same number of notifications, the value of the average notification latency remains constant and low (5, 9, and 60 ms for 10, 100, and 100 notifications, respectively) and starts to increase from 100 k subscribers. This behavior of the notification latency, maintaining itself constant until a high number of subscriptions, is the desired behavior for this component, since its main objective is to communicate abnormal events or anomalies (low number of notifications) among the large number of domains of a smart city (large number of subscribers).

Additionally, it is important to highlight the fact that E-SilboPS architecture allows a minimum initial deployment with one Access/Connection Point, 1 Matcher, and 1 Exit Point and scale in/out the different operators depending on the different type of needs. On the other hand, this is not possible in the Orion Context Broker deployment because in order to scale a whole new instance of the Orion must be deployed.

According to [22], there are almost 1800 sensors installed as part of the IoT platform to make the city of Barcelona

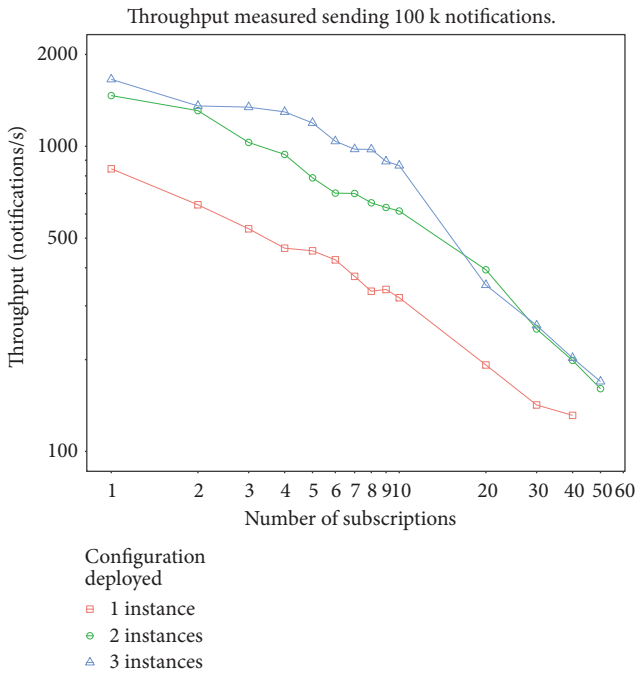

Figure 9: Notification throughput with different number of Orion instances.

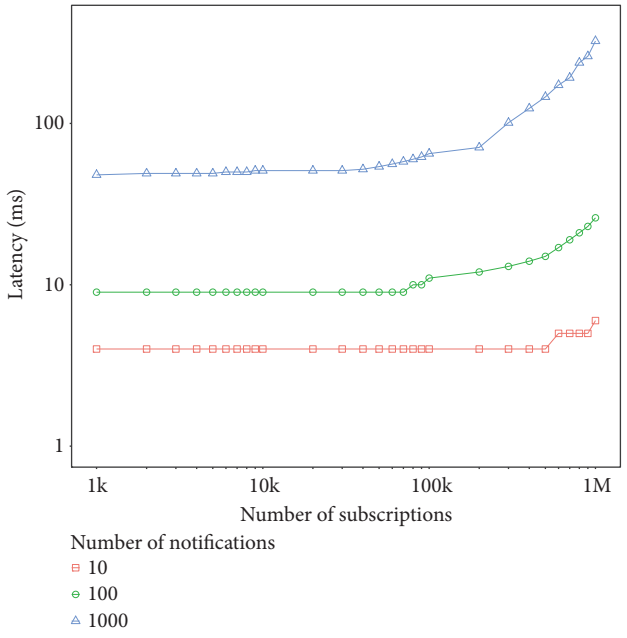

Figure 10: Average notification latency with different number of subscribers and notifications.

(Spain) a smart city. Additionally, authors claim that it would take more than 320.000.000 sensors for a whole coverage of the city and each sensor would send, at least, an information update every 15 minutes, which means a notification throughput of 355 k notifications per second and more than 8GB of data transfer per day.

It is clear that in order to monitor a network of sensors of these characteristics and be able to make decisions in real time is essential to use an elastic, low-latency, and good-performance Context Broker, as the one presented in this work.

5. Conclusions

In this section we explain the main findings and implications of the work presented, highlighting its relevance and importance.

In this article we have presented a novel fog computing architecture based on the FIWARE-NGSI standard specification to achieve openness and interoperability. The proposed architecture logically divides the smart cities environments in different domains which are communicated by a Context Broker specially designed to be elastic and low-latency. The proposed Context Broker relies internally on E-SilboPS, an elastic context-aware content-based publish-subscribe middleware (CA-CBPS). Moreover, the stream processing tasks have been moved to the edge of the network using lightweight CEP and context brokers that can be deployed in edge computing nodes to respond for anomalies in real time. We validate the proposed architecture through a use case to monitor gas pipelines and to provide a rapid response by other domains of a smart city to an incident at such gas installations. As shown in the evaluation section, the proposed Context Broker implementation overcomes the performance and elasticity problems of the FIWARE reference implementation (Orion) with low-latency. This is due to the elasticity, low-latency, and context awareness of the E-SilboPS.

In conclusion, in this paper we show how the proposed fog computing architecture can be successfully used as an architecture to meet the fog computing challenges and characteristics. Nevertheless, this is just one use case to show the overall operating mode. This solution can be deployed to monitor and make informed decisions for all the domains of smart cities due to the openness and interoperability offered by the unified data model abstraction of the FIWARE-NGSI standard. Moreover, the elasticity, scalability, and low-latency of the E-SilboPS permit the architecture to be deployed in environments with low workloads with a minimum resource consumption and to efficiently scale out to cope with huge workloads. However, there is some work to be done in order to achieve proper autoscaling for the E-SilboPS.

6. Future Work

This section states out future goals of this research and describes the main features that we believe would add more value to this work.

On the one hand, the Context Broker implementation presented in this article depends on an external system to adapt the FIWARE-NGSI notification and subscriptions to the E-SilboPS format, as described above. Nowadays, this system does not cover the whole expressiveness defined by the FIWARE-NGSI specification, such as regular expressions.

However, since these types of constructions have a high computational costs, it could be interesting to firstly analyze the benefits and how often these types of constructions are used in real scenarios.

On the other hand, there are some improvements that can be done to E-SilboPS, since it is the core element of the Context Broker GEi presented. Although this system is elastic, as demonstrated in the evaluation presented, the system lacks the functionality necessary to decide autonomously when to scale in or out, to adjust the number configuration of the system to fluctuations in workloads so there is neither degradation in the quality of service (QoS) nor unnecessary economic costs. However, the decision of when the system has to scale in/out is not trivial. Classical approaches are based on low-level metrics such as CPU usage or memory usage (e.g., autoscaling systems offered by major cloud providers). Although these classic approaches are very intuitive and easy to understand, they have multiple disadvantages, such as the gap or difficulty of mapping the criteria and objectives of the service provider, called high-level metrics (e.g., low-latency or high throughput), to the low-level metrics used by the autoscaling systems (e.g., CPU usage or memory usage). Therefore, the autoscaling system of the E-SilboPS must be aware of the high-level metrics to be able to predict peaks of usage, in addition to taking into account the monetary cost and performance efficiency of the system configuration, which should be optimal [25–30].

Another possible improvement of the E-SilboPS system would be to allow the replacement of a certain instance of an operator, since currently the instances of each of the operators are managed as if they were a stack. In other words, in order to be able to replace a certain instance, the system needs to scale in until that particular instance is no longer used and can therefore be removed.

Another feature that would improve system performance would be to make use of network multicast, because it would not only reduce the total number of messages sent between the different E-SilboPS instances, but also reduce the CPU usage.

Other potentially features that would add value to E-SilboPS are the ability to independently scale in/out all layers in a single scaling transaction (i.e., multilayer scalability). However, this functionality can greatly increase the complexity of the system's scaling algorithm, and therefore it should be analyzed beforehand whether the benefit obtained is sufficiently relevant to compensate for the possible increase in the complexity of software implementation.

Finally, FIWARE-NGSI v2 does not support NGSI 9 (context information discovery); however, it would be interesting to update the proposed architecture in this work when it becomes available, as FIWARE-NGSI v2 greatly simplifies this standard and interfaces.

Data Availability

The performance data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

This article expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The research leading to these results was partially funded by the European Commission 7th Research Framework Programme FI-WARE & FI-CORE Projects and the European Commission Horizon 2020 Research Framework Programme FI-NEXT Project under Grant Agreements nos. 285248, 632893, and 732851, respectively.

References

[1] S. Dirks and M. Keeling, "A vision of smarter cities: how cities can lead way into a prosperous and sustinable future," *IBM Global Business Services*, p. 18, 2009.

[2] S. Dirks, C. Gurdgiev, and M. Keeling, "Smarter cities for smarter growth," *IBM Global Business Services*, p. 24, 2010.

[3] S. Dirks, M. Keeling, and J. Dencik, *How Smart is Your City? Helping Cities Measure Progress*, IBM Global Business Services, 2009.

[4] H. Chourabi, T. Nam, S. Walker et al., "Understanding smart cities: an integrative framework," in *Proceedings of the 45th Hawaii International Conference on System Sciences (HICSS '12)*, pp. 2289–2297, January 2012.

[5] B. Tang, Z. Chen, G. Hefferman et al., "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial Informatics*, no. 99, 2017.

[6] I. García-Magariño and R. Lacuesta, "ABS-SmartPriority: An Agent-Based Simulator of Strategies for Managing Self-Reported Priorities in Smart Cities," *Wireless Communications and Mobile Computing*, vol. 2017, Article ID 7254181, 9 pages, 2017.

[7] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart cities in Europe," *Journal of Urban Technology*, vol. 18, no. 2, pp. 65–82, 2011.

[8] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities," *IEEE Internet of Things Journal*, 2017.

[9] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Proceedings of the 3rd Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015*, pp. 73–78, USA, November 2015.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[11] M. Chiang and T. Zhang, "Fog and IoT: an overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the 1st*

ACM Mobile Cloud Computing Workshop, MCC 2012, pp. 13–15, Finland, August 2012.

[13] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[14] P. Sethi and S. R. Sarangi, "Internet of things: architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, Article ID 9324035, pp. 1–25, 2017.

[15] M. B. A. P. Madumal, D. A. S. Atukorale, and T. M. H. A. Usoof, "Adaptive event tree-based hybrid CEP computational model for Fog computing architecture," in *Proceedings of the 16th International Conference on Advances in ICT for Emerging Regions, ICTer 2016*, pp. 5–12, Sri Lanka, September 2016.

[16] A. Antonić, M. Marjanović, K. Pripužić, and I. P. Žarko, "A mobile crowd sensing ecosystem enabled by CUPUS: cloud-based publish/subscribe middleware for the Internet of Things," *Future Generation Computer Systems*, vol. 56, pp. 607–622, 2016.

[17] S. Chun, S. Shin, S. Seo, S. Eom, J. Jung, and K. Lee, "A Pub/Sub-Based Fog Computing Architecture for Internet-of-Vehicles," in *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 90–93, Luxembourg City, December 2016.

[18] J. Huang, P. Tsai, and I. Liao, "Implementing publish/subscribe pattern for CoAP in fog computing environment," in *Proceedings of the 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 198–203, Vancouver, BC, October 2017.

[19] S. Vavassori, J. Soriano, and R. Fernández, "Enabling large-scale IoT-based services through elastic publish/subscribe," *Sensors*, vol. 17, no. 9, 2017.

[20] I. García-Magariño and C. Gutiérrez, "Agent-oriented modeling and development of a system for crisis management," *Expert Systems with Applications*, vol. 40, no. 16, pp. 6580–6592, 2013.

[21] Esper, "Esper EPL reference," 2018, http://esper.espertech.com/release-5.2.0/esper-reference/html/epl_clauses.html.

[22] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin et al., "Estimating Smart City sensors data generation," in *Proceedings of the 15th IFIP Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2016*, Spain, June 2016.

[23] K. C. Lazowska, D. Edward, J. Zahorjan, and et al., *Quantitative system performance: computer system analysis using queueing network models. 1em plus 0.5em minus 0.4em*, Prentice-Hall, Inc, 4em, 1984.

[24] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," in *Proceedings of the Presented as part of the 10th International Conference on Autonomic Computing*, pp. 23–27, 2013.

[25] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, ACM, New York, NY, USA, November 2011.

[26] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD '12)*, pp. 423–430, IEEE, Honolulu, Hawaii, USA, June 2012.

[27] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 67–78, USA, May 2013.

[28] J. M. Galloway, K. L. Smith, and S. S. Vrbsky, "Power Aware Load Balancing for Cloud Computing," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. I, pp. 122–128, 2011.

[29] J. Galloway, K. Smith, and J. Carver, "An empirical study of power aware load balancing in local cloud architectures," in *Proceedings of the 9th International Conference on Information Technology, ITNG 2012*, pp. 232–236, USA, April 2012.

[30] P. Nguyen and K. Nahrstedt, "Resource management for elastic publish subscribe systems: A performance modeling-based approach," in *Proceedings of the 9th International Conference on Cloud Computing, CLOUD 2016*, pp. 561–568, USA, July 2016.