COLLECT: COLLaborativE ConText-aware Service Oriented Architecture for Intelligent Decision-Making in the Internet of Things

Alfonso García-de-Prado*, Guadalupe Ortiz, Juan Boubeta-Puig

UCASE Software Engineering Research Group, University of Cádiz

Escuela Superior de Ingeniería. Avda. de la Universidad de Cádiz 10,

11519 Puerto Real, Cádiz, Spain

{alfonso.garciadeprado, guadalupe.ortiz, juan.boubeta}@uca.es

Abstract

Internet of Things (IoT) has radically transformed the world; currently, every device can be connected to the Internet and provide valuable information for decision-making. In spite of the fast evolution of technologies accompanying the grow of IoT, we are still faced with the challenge of providing a service oriented architecture, which facilitates the inclusion of data coming together from several IoT devices, data delivery among a system's agents, real-time data processing and service provision to users. Furthermore, context-aware data processing and architectures still pose a challenge, in spite of being key requirements in order to get stronger IoT architectures. To face this challenge, we propose a COLLaborative ConText Aware Service Oriented Architecture (COLLECT), which facilitates both the integration of IoT heterogeneous domain context data — through the use of a light message broker — and easy data delivery among several agents and collaborative participants in the system — making use of an enterprise service bus —. In addition, this architecture provides real-time data processing thanks to the use of a complex event processing engine as well as services and intelligent decision-making procedures to users according to the needs of the domain in question. As a result, COLLECT has a great impact on context-aware decentralized and collaborative reasoning for IoT, promoting context-aware intelligent decision making in such scope. Since context-awareness is key for a wide range of recommender and intelligent systems, the presented novel solution improves decision making in a large number of fields where such systems require to promptly process a variety of ubiquitous collaborative and context-aware data.

Keywords: Service Oriented Architecture, Complex Event Processing, Collaborative Internet of Things, Context Awareness, Intelligent Decision-Making.

* Corresponding author: Alfonso García-de-Prado (alfonso.garciadeprado@uca.es)
Phone number: +34 956 483465

Publisher version:

Garcia-de-Prado, A., Ortiz, G., & Boubeta-Puig, J. (2017). COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things. *Expert Systems with Applications*, *85*, 231–248. https://doi.org/10.1016/j.eswa.2017.05.034

1. Introduction

Internet of Things (IoT) was introduced by Kevin Ashton in 1999 in the field of supply chain (Ashton, 2009), but since then the term has been applied in many other domains and used in a wider sense, referring to interconnected devices which can obtain and share information across platforms, providing added value to innovative applications. Definitely, the IoT definition has evolved towards concepts where service integration plays a relevant role (Khodadadi, Dastjerdi, & Buyya, 2016). IoT architectures have to provide essential elements such as sensor devices, offered services, communication networks and event context processing, interoperability being one key requirement for such IoT architectures, so standard interfaces must be provided to facilitate information submission from such devices. Besides, reliability and scalability are also key requirements (Buyya & Vahid Dastjerdi, 2016).

Faced with that situation, a clear statement surfaces: one of the remaining challenges in this scope is the design of a *Service Oriented Architecture* (SOA) for IoT, which facilitates the inclusion of data coming from several IoT devices as well as facilitating the delivery of such data among system agents, which can process such data and provide services to the users (L. D. Xu, He, & Li, 2014). Previously outlined requirements can unquestionably be provided by a SOA: interoperability is one of SOA's principles; besides, SOAs let us build systems based on loosely coupled modules, so that system maintenance and reliability are guaranteed, as well as scalability, through the use of an *Enterprise Service Bus* (ESB) or a federation of them (Papazoglou, 2012). In addition, *Complex Event Processing* (CEP) is the ideal technology for real-time event processing, which is an inherent need for current IoT architectures and can be used in conjunction with SOAs to improve decision-making in multiple domains (Boubeta-Puig, Ortiz, & Medina-Bulo, 2015).

Concerning the matter of sharing information across platforms, collaborative architectures for data sharing in the scope of the IoT are an essential requirement de facto for giving additional value to any decision-making process (Behmann & Wu, 2015). The huge amount of data obtained from multiple smart devices can indeed be shared to provide streaming events in an information system with enriched semantic, therefore offering a wider global knowledge of the domain in question. Collaborative IoT (C-IoT) takes into account the IoT in a scope of heterogeneous elements and domains, where sensors, gateways and services can interact at different levels; sensors provide their sensed data; gateways add intelligence to them and take actions or communicate information to a higher level, and services make use of the information provided by the gateway to improve people's life quality or business processes.

It is obvious that with the aim of offering a collaborative SOA in the scope of the IoT, it is necessary to search for solutions and protocols that allow data exchange among IoT low computation capacity devices. Such devices have been called *fog* devices since they are closer to the things that produce IoT data than to the cloud computational level; any computing device with connection to a network and some storage capacity can be a fog device (CISCO, 2015). Therefore, IoT can benefit from *fog computing*, where the processing of local data is not performed in the cloud, but can be performed in smart *edge* devices (Dastjerdi, Gupta, Calheiros, Ghosh, & Buyya, 2016). That is, these devices process all the local events and only send the most relevant information to the cloud, where such relevant information can be processed together with the information provided by other nodes. Besides, these devices can subscribe to the relevant information provided by other nodes and exchange data with them in a collaborative way, with no need to send the data to the cloud forward and back, so that system resources can be saved.

Context-awareness is key for a wide range of recommender and intelligent systems (Sundermann, Domingues, Conrado, & Rezende, 2016) since it provides privileged information which is crucial for intelligent decision support systems (Kwon, 2006). One of the main open challenges for context-aware intelligent and expert systems is how to obtain the contextual information (Sundermann et al., 2016) in the scope of IoT, where multiple data can be obtained from several sources in the context of the domain application, the opportunities increase rapidly and context awareness becomes fundamental (Perera, Zaslavsky, Christen, & Georgakopoulos, 2014). Not surprisingly, the European Union identifies, among the Horizon 2020 challenges, research and development for context-aware IoT computation (European Research Group in the Internet of Things, 2012; The Alliance for Internet of Things Innovation, 2015). This is why in order to fulfill the described outstanding challenges, in this paper we provide COLLECT: a COLLaborativE ConText-aware Service Oriented Architecture for the Internet of Things. Such an architecture, which is the main contribution of this paper, facilitates context spreading and sharing among the nodes in the architecture, therefore improving and speeding up intelligent decision-making in countless domains. COLLECT will optionally be formed of two types of node: cloud and fog nodes. Both types of node implement an *Event-Driven Service Oriented Architecture* (ED-SOA or SOA 2.0) (Papazoglou, 2012) which combines benefits from the use of CEP in a SOA, where proposed fog nodes architecture is suitable for limited capacity devices. CEP is widely used in intelligent decision-making systems (Bhargavi, Pathak, & Vaidehi, 2013) and it will play a key role in our architecture. Besides, the mentioned SOA 2.0 has been enriched with a Context Broker.

As a result, bearing in mind the requirements previously described, COLLECT permits: (1) Implementing collaboration among several nodes through a collaborative ED-SOA. (2) Ensuring system scalability through the opportunity of federating ESBs in the cloud and through distributed CEP and low consumption communications among the fog nodes. (3) Facilitating an architecture for fog devices that allows processing information and publishing and subscribing to distributed complex events of interest in the context of the application. And (4) Facilitating communications between several layers of the C-IoT in the proposed architecture: sensor data, the intelligence added by the gateway (our ESB) and the actions launched through the latter to send notifications and information of higher semantic meaning. The proposed architecture is illustrated and evaluated through the implementation of a case study from the field of healthcare IoT, one of the key domains for C-IoT applications (Behmann & Wu, 2015; Islam, Kwak, Kabir, Hossain, & Kwak, 2015).

The rest of the paper is organized as follows: Section 2 presents the background required for understanding the technologies and paradigms used in this paper. Then, related work is examined in Section 3. Afterwards Section 4 explains the proposed collaborative context-aware SOA. The case study description and implementation follow in Section 5 and the architecture and case-study evaluation can be found in Section 6. Finally, conclusions and future work are summarized in Section 7.

2. Background

In this section, we introduce the most relevant technologies and knowledge in order to facilitate understanding of the paper: context awareness, SOA, CEP, message brokers, IoT and fog computing.

2.1. Context Awareness

Dey context definition in (Dey, 2001) is specially well-known, where "*Context is any information that can be used to characterize the situation of an entity*"; such entity can be almost anything — an object, a person, et cetera —, that can be useful to improve the interaction between a user and an application as well as the application functionality itself. Context information is specific to each system and can rarely be generalized, therefore one specific type of information can be considered as part of the context in a given system but not in a different one.

Context awareness supports the fact that context information, obtained from the system environment, is properly used by the system so as to improve its quality; that is, using information such as location, social attributes and other information to foresee the system's needs so that we can offer more personalized systems. Therefore, a system is context-aware if it uses the context to provide relevant information or services to the user or to the system itself, adapting its system behavior to the particular needs of such specific user or system (Abowd et al., 1999). Context and context awareness have become a key issue for decision-making in general and for real-time decision-making in particular (Burstein, Brézillon, & Zaslavsky, 2011).

It is important to highlight that when talking about real time throughout this paper, we refer *to quasi real time*. This term differs from the strict traditional definition of real-time computation, where real-time responses are expected to be received in the order of milliseconds or even microseconds. Generally, the term quasi real time refers to a short-time response from a system according to its needs, it might be in the order of milliseconds or maybe in seconds. For instance, as we will later see in the case study, if we need to warn a citizen about current air quality, a millisecond or even a second difference in the response time is not relevant (an hour delay would). Therefore, such systems respond rapidly to the occurring events but do not require strict under millisecond response.

2.2. Service Oriented Architecture

SOA consists of a paradigm for the design and implementation of loosely coupled distributed systems which use services for their implementation. These architectures provide easy interoperability among third-party systems in a flexible and loosely coupled way, so that the focus can remain on the business process rather than on the technologies. This way, system maintenance and evolution are facilitated when the system requires changes, and costs are reduced (Papazoglou, 2012). Therefore, the service orientation concept is based on the idea of offering a well-defined interface which provides communications based on a standard protocol, where currently the provider and the consumer are completely decoupled.

With the growth of service components and processes in service oriented applications, a new service infrastructure is required for maintaining applications flexibly. This infrastructure must support well-known web service standards and provide support for a message middleware (Papazoglou, 2012). These requirements are fulfilled by an ESB. An ESB provides services to complex architectures through a messaging system, supplying interoperability among diverse applications and components through standard interfaces; that allows applications to be offered as services in the ESB. The bus also reinforces the reliability of the communication in the SOA as well as ensures system scalability.

ED-SOA, or SOA 2.0., evolves from traditional SOA. In SOA 2.0., communication between users, applications and services is carried out by events, rather than using remote procedure calls

(Luckham, 2012). To facilitate this paradigm, a software abstraction layer is required to integrate diverse heterogeneous data sources and distributed invocations (Papazoglou & Heuvel, 2006). These functionalities are offered by the previously explained ESB, which permits interoperability among several communication protocols and heterogeneous data sources and targets.

2.3. Complex Event Processing

Despite all the advantages provided by SOA 2.0, this type of architecture might not be ideal to analyze and correlate large amounts of data in terms of events in real time. To meet this requirement, it is necessary to integrate CEP (Luckham, 2012), which is a technology that allows the capturing, analyzing and correlating of a large amount of heterogeneous data — simple events — with the aim of detecting relevant situations in a particular domain (Inzinger, Hummer, Satzger, Leitner, & Dustdar, 2014). Event patterns specify the conditions to be met in order to detect such situations. These situations are named complex events and managed by a CEP engine, the software capable of analyzing the data in real time.

In this paper, we are going to use the terms “domain events” and “contextual events”. Martin Fowler defines domain events as an event which “*captures the memory of something interesting which affects the domain*” (Fowler, 2005); indeed we are going to use this term for referring to simple events related to something interesting for a particular domain that happens and is captured in the current system node. On the other hand, we will refer to “contextual events” as those complex events already detected in some other node in the architecture, which of course also contain information of interest for the domain in question, but these are not simple events obtained from external sources, but complex ones generated from within system.

2.4. Message Brokers

In these types of architecture where large amounts of events are received and have to be processed, a message broker can be of great use. Message brokers implement an asynchronous mechanism which allows source and target messages to be completely decoupled; brokers can as well store the messages locally until they can be processed by the target element. These brokers may use standard message queues or be combined with a publish/subscribe mechanism, where messages are published according to a set of topics and users subscribe to the topics of their interest.

Java message brokers are mostly based on Java Message Service (JMS). JMS allows the developer to focus on the application business logic and provides a versatile message API with diverse messaging models: point to point, load balance, publish/subscribe, et cetera. In particular, there are two most widely used models: queues and topics. Message queues implement a load balance algorithm so that only one consumer receives the message; such message is kept in the system until the consumer is ready to process it. In the case of message topics, a publish/subscribe standard mechanism is implemented, where every published message can be processed by all the consumers subscribed to that topic.

On the other hand, devices with limited capacity require low consumption of message services and brokers. In particular, MQ Telemetry Transport (MQTT) protocol was proposed as a light protocol implementing a publish/subscribe mechanism for Machine-to-Machine (M2M) communications; the broker Mosquitto implements MQTT and is currently being frequently used in the scope of IoT.

2.5. Internet of Things and Fog Computing

In the near future, the economic impact expected from IoT applications is 11% of the worldwide economy (Buyya & Vahid Dastjerdi, 2016). IoT is defined as a network formed by interconnected physical objects uniquely identified (Atzori, Iera, & Morabito, 2010) and implies integration, transfer and analysis of the data coming from such objects and currently, several algorithms, tools, technologies and best practices enable IoT applications and architectures in a variety of application domains (Buyya & Vahid Dastjerdi, 2016). However, as explained in the introduction, IoT definition has evolved towards concepts where service integration plays a crucial role and where IoT proposed architectures must fulfill essential requirements such as interoperability, reliability and scalability. Such architectures must also provide a set of essential elements such as sensors, offered services, communication networks and event context processing.

In this new IoT era, new additional terms arise. Now, the term smart device is used to refer to a device which can communicate with other devices through a network connection and that can proceed to some extent with autonomous computation. Such devices are also called edge or fog devices, since they are opposite to cloud computing nodes in the network connection. In this scope, the term fog computing stands for the processing of local data in the smart edge device rather than in the cloud (Dastjerdi et al., 2016). That is, these devices process all the relevant local events and save resources by only sending, to the cloud or to other fog nodes, the most relevant information that can be processed in the remote nodes together with their own one and that improves intelligent decision-making wherever it is required.

3. Related Work

There are multiple approaches for context adaptation in different computer science domains (Kapitsaki, Prezerakos, Tselikas, & Venieris, 2009): middleware and platform solutions, ontology-based solutions, rule-based reasoning, model driven approaches, et cetera. These techniques are not exclusive and a developer could opt for combining several of them in order to deal with context.

Some works integrate CEP and SOA or use ESBs to follow some adaptation or provide context awareness. For instance, Taher, Fauvet, Dumas and Benslimane (2008) propose the adaptation of interactions of web service messages between incompatible interfaces. In this regard, they develop an architecture that integrates a CEP engine and input/output adapters for SOAP messages. Input adapters receive messages sent by web services, transform them into the appropriate representation to be manipulated by the CEP engine and send them to the latter. Similarly, output adapters receive events from the engine, transform them into SOAP messages and then they are sent to web services. CA-ESB is presented by Chanda, Sengupta, Kanjilal, and India (2011) as a context-aware enterprise service bus; in fact, it is a bus which deals with service composition based on client location; that is, services register in the system with a location and the closest services are selected when pursuing a service orchestration.

Most of these proposals are focused on a unique aspect of context awareness: some of them on the modelling phase, others on context provisioning; others on adaptation code, et cetera; but none of them presents a holistic architecture which permits dealing with context awareness in SOAs for intelligent decision-making, providing the means for context dealing from data reception to delivery of context-aware services. Concerning context awareness approaches, the main restriction is that they focus on a particular domain and are not extensible to others, and they do not provide the means to allow collaboration among several heterogeneous nodes to share context, which would enrich the information each node has.

This is why we propose here a context-aware architecture that can be configured for multiple domains.

In the scope of expert and intelligent systems, there are several papers related to intelligent decision-making based on context. For instance, the paper from Uhm, Lee, Hwang, Kim, and Park (2011) where they support adaptive service reconstruction based on context and pattern analysis. Collaboration is focused from another perspective in (Papageorgiou, Verginadis, Apostolou, & Mentzas, 2011), where the events detected in the system and matching a particular pattern cause rearrangements in a particular collaboration. Bruns, Dunkel, Masbruch, and Stipkovic (2015) present an event-driven architecture for intelligent Machine-to-Machine where CEP is a key technology. Using their approach, M2M systems can respond in a flexible, adaptable and intelligent way. However, the proposed architecture is too general and therefore difficult to put into practice in any domain; at the same time, however, it is restricted to the M2M scope. Kim. Kim, Kim and Jung (2016) present an intelligent risk management framework for cold chain logistics, where food context allows users to evaluate how a certain food will be affected by the cold chain being broken. Even though the applicability of the approach is quite narrow, it shows an interesting particularized example of how relevant context is used in the IoT and how useful it can be to be aware of it.

In particular we can find multiple approaches related to ambient assisted living, such as (De Backere, Bonte, Verstichel, Ongenae, & De Turck, 2017), where the system obtains context information from heterogeneous sources, but all related with an unique local node in the architecture. Another relevant work on the topic is CoCaMAAL (Forkan, Khalil, & Tari, 2014), a cloud-oriented context-aware middleware in ambient assisted living. Other papers focuses on context-awareness in hospitals (Immanuel & Raj, 2015). The great advantage would be combining the context from several scopes, let's say the ambient assisted living house sensors, with the hospital context when the dependent people are there. This is what we provide with COLLECT: a multi-node and multi-domain architecture which makes use of context in a collaborative way.

In the scope of collaborative SOAs, most works are focused on mobile phone collaboration or team work (Rubinsztejn, Endler, Sacramento, Gonçalves, & Nascimento, 2004; Benítez-Guerrero, Mezura-Godoy, & Montané-Jiménez, 2012). In the scope of the IoT, there are two relevant surveys. Perera, Zaslavsky, Christen and Georgakopoulo (2014) present a survey about context-oriented computation for the IoT. Among the frameworks studied in the survey, two works can be highlighted. Firstly, the framework presented by Badii Crouch and Lallah (2010) is focused on context acquisition and processing and they provide a context ontology and a rule based reasoning engine. This work could be complementary to ours, where we could make use of the provided context and their reasoning. Secondly, Katasonov, Kaykova, Khriyenko, Nikitin, and Terziyan (2008) describe a middleware where communications among the different agents providing and consuming context are based on semantic descriptions. Even though the survey mostly revolves around on context acquisition, where high quality papers are described, those concerning context management are not useful for collaborative architectures. The authors of the survey also have their own proposal: they supply CA4IOT, a context-aware framework for IoT (Perera, Zaslavsky, Christen, & Georgakopoulos, 2012). Their architecture is centred on helping the user to choose, among the available sensors, which are more suitable to resolve their problems. Therefore, their main aim is automating, selecting, filtering and reasoning data coming from several sensors. Once more, this semantic proposal could be used in conjunction with ours. The survey presented by Gil, Ferrández, Mora-Mora and Peral (2016) is focused on several fields of IoT (general purpose surveys, data-oriented ones, data mining and data cloud integration); as we can see they are mainly focused on the acquisition and analysis of IoT data. The fact is that we do not find relevant works that, rather than focusing on IoT context-aware

data acquisitions, look into the matter of providing a collaborative architecture which would benefit from such context.

4. COLLECT Overview

COLLECT is a COLLaborativE ConText-aware Service Oriented Architecture for the Internet of Things, which is composed of several collaborative nodes. As explained in the introduction, we can have two types of node in the architecture: cloud nodes and fog nodes. Figure 1 shows a COLLECT architecture where only one cloud node and two fog nodes have been included for simplicity. However, the architecture could have additional nodes of both types; typically more fog nodes than cloud nodes, or even only fog nodes, always depending on the domain requirements.

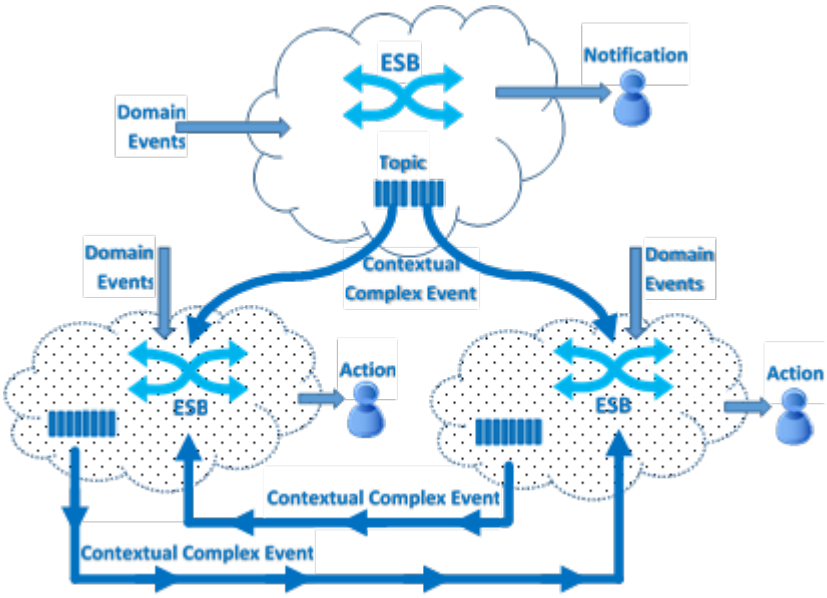

Figure 1. COLLECT: Collaborative Context-Aware Service Oriented Architecture

The cloud node is expected to have many resources and permits storing large amounts of data and which provides high processing capacity for events, patterns and notifications. The fog node is expected to have limited capacity, having been released from the need to store data and all irrelevant events being discarded.

For instance, imagine that we want to have a system (which will be later proposed as the case study in this paper) that lets us predict when there is going to be an increase in patients with respiratory problems attending hospital emergency services, for example with the aim of having enough lung specialists available at the hospital in such situations. We could have a cloud node processing huge information about air quality, pollen concentrations and weather forecasts; this cloud node could provide alerts when weather conditions, air quality detected and pollen concentrations are at levels which can be dangerous for people with respiratory problems. Several fog nodes could be located in a number of hospitals and would receive information about local patient admissions and alerts from the cloud node, and they could also be subscribed to external alerts (as explained below).

Communications between nodes would be as follows:

The cloud node sends all the detected complex events to a light message broker, so that any node in the fog (or additional nodes in the cloud, if interested) can subscribe to them. In the

example, that means that dangerous values of current air quality for people with respiratory problems would be submitted to the light message broker. We could have, for instance, a topic for all the alerts in every hospital's area of influence. Let us explain it: people who attend a particular hospital are usually from a specific geographic area; therefore, every topic would include the alerts from the air quality stations in the said area; so each hospital only receives those alerts which are relevant to it. From now on, we will call such geographic areas as the hospital's area of influence. This node, with higher capacity, could also send notifications to interested parties, outside the fog nodes, this being out of the scope of this paper.

A fog node can subscribe to the complex event topics provided by a cloud node (these will be contextual complex events of the domain). This means that nodes in the hospitals can subscribe to the air quality alerts offered by the cloud node in order to be aware of when air quality conditions are unsuitable for patients with respiratory problems.

A fog node can subscribe to domain event topics provided by the system (these will be domain events). Therefore, nodes in the hospitals subscribe to the local topic in which hospital patient admissions are registered.

A fog node sends contextual complex events detected in such node to a light message broker to which any node in the architecture can subscribe. This means that, for example, when a hospital is aware of unsuitable air quality conditions for people with respiratory problems within its area of influence and there has been an important increase on the number of patients with breathing conditions attending the hospital emergencies, it can submit a local alert to a message topic to which other hospitals can subscribe.

A fog node can subscribe to the contextual complex event topics provided by other fog nodes (these will be contextual complex events for this node). This means that a hospital can subscribe to the alerts submitted by the others, so as to predict situations that already happened in other hospitals and be prepared for them. For instance, imagine that one hospital, having level 3 air quality for particular pollutants within its area of influence, experiences a peak of patients with respiratory conditions by emergency admissions. Other hospitals, being aware of such situation, can take action when air quality for the same pollutants reaches level 3 in their area of influence and receive the first emergency admission of patient with breathing problems.

A fog node sends relevant complex events detected in such node to the system in charge of the named node actions. Once a "situation of danger" is predicted, it is submitted to another topic to which an actuator or notification system is subscribed in order to anticipate actions before the emergency is already there. Even though it is not shown in the illustration, it is assumed that for each fog node there will be a system pursuing required actions; the said system will be subscribed to the topic receiving the relevant domain complex events detected in the named node. Since the action system can be completely decoupled thanks to the use of a message broker, we have placed it outside the node to alleviate the node processing load and therefore such action system is out of the scope of this paper.

4.1. COLLECT Architecture Components

As previously explained, in COLLECT, we are going to have a SOA deployed in the cloud and one or more SOAs deployed in the fog. Both components and internal communications are explained in the following sub-sections.

4.1.1. Components Description

The Context-Aware SOA deployed in the cloud is not the main aim of this paper, since the collaboration is fostered in the fog nodes. In any case, such cloud context aware SOA would also integrate CEP; would receive data from IoT platforms, sensors or from a message broker and could be similar to the one described in (Boubeta-Puig, Ortiz, & Medina-Bulo, 2017), but with the additional requirements of sending relevant detected complex events to a light message broker that implements MQTT.

It is assumed that the context-aware architecture to be deployed in the fog will have to be deployed in a smart device with limited capacity to a greater or lesser degree and which does not require data storage, since irrelevant data are discarded and relevant ones are sent to a message broker to which other fog or cloud nodes or action system could subscribe. Such architecture, represented in Figure 2, is explained in the following paragraphs.

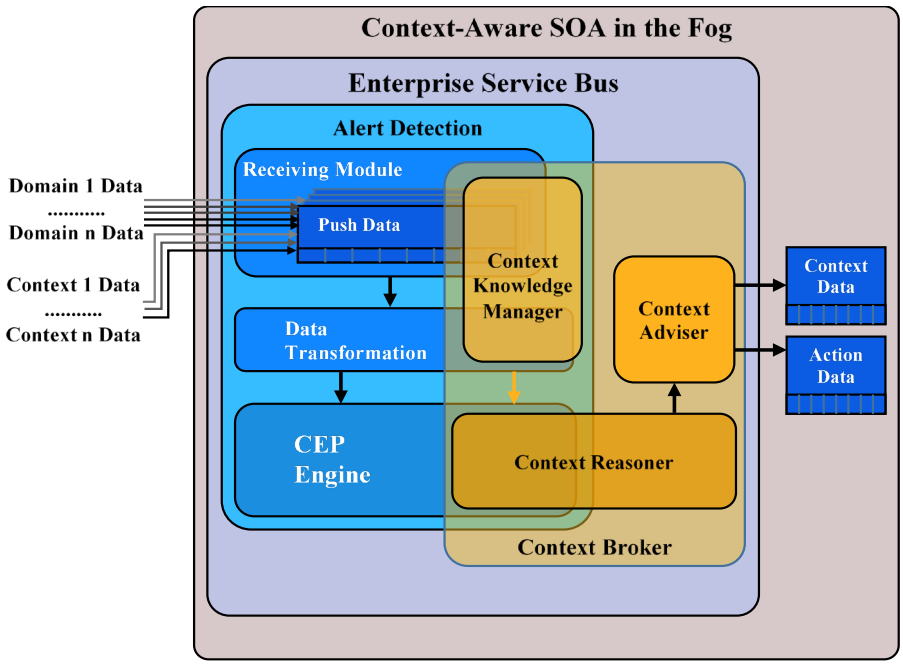

Figure 2. Context-aware SOA node in the fog

Enterprise Service Bus. An ESB is in charge of routing and facilitating communication in a SOA. In the proposed architecture, the bus channels the following communications: reception of simple events coming from heterogeneous domains and environments, reception of complex events providing contextual information to the system, event transformation and routing to the CEP engine, complex event reception notified by the CEP engine, complex event routing to the context broker and relevant complex event submission to (1) the topic to which other nodes in the architecture subscribe and (2) the topic to which the domain-specific action system subscribes.

Context Broker. The context broker will be in charge of managing the context knowledge (implicit in the contextual complex events to which the system is subscribed) as well as submitting the relevant complex events detected to the contextual event topic or to the action topic, depending on the patterns found. The context broker is composed of three modules which are explained in Section 4.2. The three modules interact with other elements which are distributed along the architecture: the CEP engine and the message broker which manages the relevant complex events detected in the node.

CEP engine. Domain-specific event patterns are deployed in the CEP engine for early intelligent detection of relevant alerts for the domain in question, including management of the context knowledge obtained from other nodes in the collaborative architecture.

Data suppliers. There are two types of data suppliers in this node; in both cases data enter the system with a *push* model so that events reach the system without the need of performing any query. In particular, the architecture subscribes to one or several message topics provided by a light message broker.

- On the one hand, we receive the domain-specific data coming from heterogeneous sources and domains. Please bear in mind that even though we speak about data coming from diverse domains, they are specific to a particular domain where we need to detect relevant events. For instance, for the previously explained example, we can obtain heterogeneous events from diverse domains such as the hospital admission or air quality station measurements, but the system application domain is unique: patients with respiratory conditions and illnesses exacerbation.
- On the other hand, we receive contextual data from other nodes in the architecture. They are also relevant data from the domain in question, but they are complex events with contextual knowledge that can be useful to detect other events in the current node and to prevent non-desired situations in the domain in question. For instance, one hospital may be interested in knowing when another hospital, with the same air quality level in its area of influence, experiences a peak in patients with breathing problems by emergency admissions.

Complex events topic-based message broker. Relevant complex events detected in the local node are sent to certain topics in the message broker to which other nodes subscribe or to which the domain-specific action system is subscribed. This message broker has been included inside the node but it could also be extracted from it, and it would be up to the software engineer to balance the processing requirements versus the data communications ones according to the particular system's available resources and requirements.

4.1.2. Components Integration and Communication

As previously explained, the ESB is in charge of routing all communications.

1. First of all, the bus communicates with the CEP engine through the component developed by Boubeta-Puig et al. (2014).
    a. Firstly, patterns are deployed in the CEP engine through the use of an initial load file. Additional patterns can be added to the engine with additional load files at any time. In both cases, the bus manages the load file and deploys the patterns in the CEP engine.
    b. Afterwards, when data reach the system they are adapted to the appropriate event format to be sent to the CEP engine and the bus routes them to the latter.
    c. Finally, when the CEP engine detects that a pattern has been matched, it returns the new complex event generated by the pattern detection to the ESB, where it is managed as appropriate.

2. The bus receives data through the message broker that manages the message topics of domain events and contextual events.
3. The bus sends the detected relevant complex events to the node message broker, which manages several message topics according to the contextual complex events to which other nodes can subscribe and those to which the action system subscribes.

4.2. The Context Broker

It is important to be aware that complex events provide context knowledge implicitly, and that bearing in mind the device's limited capacity, no context knowledge is stored in databases and its reasoning is implicit in the patterns deployed in the CEP engine.

Therefore, the context broker is composed of three modules, represented in Figure 3: the context knowledge manager, the context reasoner and the context-based adviser. In the following paragraphs we explain these three modules within the context broker.

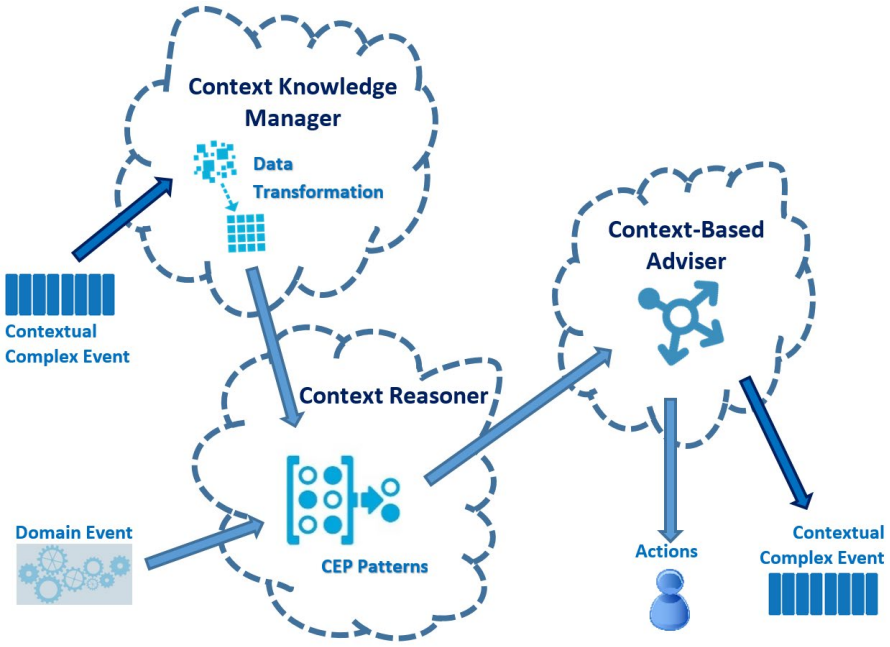

Figure 3. COLLECT Context Broker

Context Knowledge Manager. The context knowledge manager is in charge of receiving the contextual complex events that reach the system through the message topic and letting them reach the CEP engine. As previously explained and shown in Figure 2, this procedure takes part of the process of events transforming and routing to the ESB. For the presented example, this knowledge would be, for instance, the air quality alerts emitted by the cloud node or local alerts from other hospitals.

Context Reasoner. The context reasoner will be implicit in the contextual events processing patterns deployed in the CEP engine. It is the pattern designer who provides the reasoner with rules through the pattern definition and programming. Here, we would have patterns that take into account air quality alerts, other hospital alerts and patients who are currently arriving at hospital emergency services.

Context-Based Adviser. The context-based adviser, depending on whether the detected patterns require actions and/or are related to additional contextual complex events of interest, redirects them to the corresponding message topic. For instance, alerts relevant to

other hospitals would be sent to the topic in question, situations predicted based on what happened in other hospitals and the domain events would go to the action one.

4.3. Architecture Implementation

In this section, we describe the implementation and functionality of our proposed architecture. For the implementation we had to select an ESB, a CEP engine and an MQTT light message broker. We selected Mule open source ESB (MuleSoft, 2017) due to its ability to integrate itself with cloud platforms as well as multiple tools and domain scenarios; Esper, since it is a recognized CEP engine, which provides Esper Event Processing Language (EPL) (EsperTech, 2017a) for event pattern implementation; and Mosquitto (Eclipse, 2016) because it is a widely known light message broker for M2M that facilitates reducing communication load and smart devices can subscribe to them through the use of MQTT protocol.

4.3.1. Implementation Flows

The key element in our SOA is the ESB, in particular Mule ESB, and its integration with Esper CEP engine and Mosquitto broker. The Mule ESB uses flows as its main control structure in order to manage the messages and communications among the different elements connected to the bus. Currently, Mule application starts processing a message received by an inbound endpoint and a set of processing and routing actions are implemented in the flow (MuleSoft, 2016). In order to provide the collaborative context-aware SOA, a Mule application has been implemented for cloud and fog nodes. Flows have been enumerated for clarity purposes, but they all run in parallel, even though some of them are loosely coupled: Flow 2 and 3 outputs are sent to Esper and Flow 4 receives the complex event detected by Esper The flows, explained in the following paragraphs, are continuously running independently.

Required flow for the Context-Aware SOA deployed in the cloud:

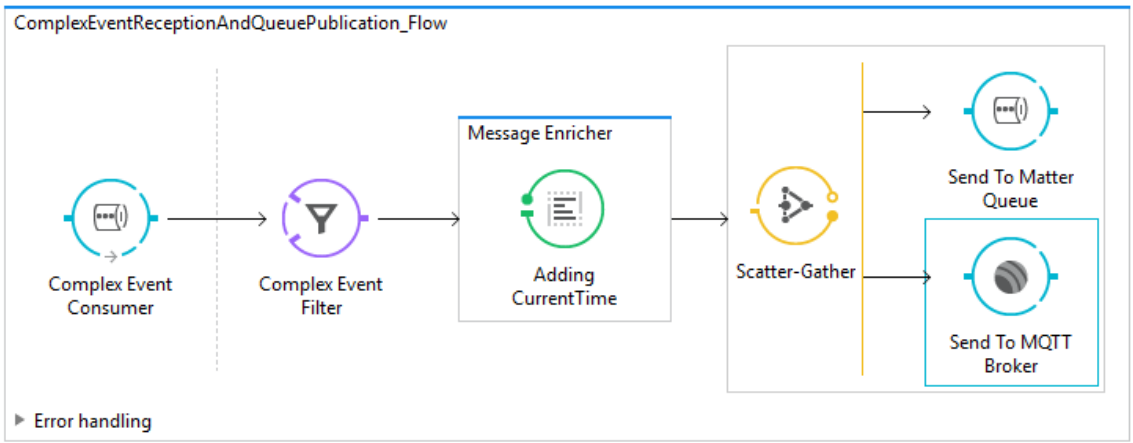

Figure 4. Flow extended with Mosquitto broker in the cloud SOA

For the cloud node, we would need to ensure that there is a flow that sends the relevant complex events detected to the MQTT message broker topic to which the remaining nodes can subscribe. The flow in Figure 4 would be an example of it, which extends the previously mentioned architecture presented in (Boubeta-Puig et al., 2017).

Implemented Flows for the Context-Aware SOA deployed in the fog:

FLOW 1. Pattern deployment in the CEP engine. As we can see in Figure 5, in this flow the EPL patterns are read from a file where they are separated by commas; then they are deployed in the CEP engine through the use of the connector implemented by Boubeta-Puig *et al*. (2014).

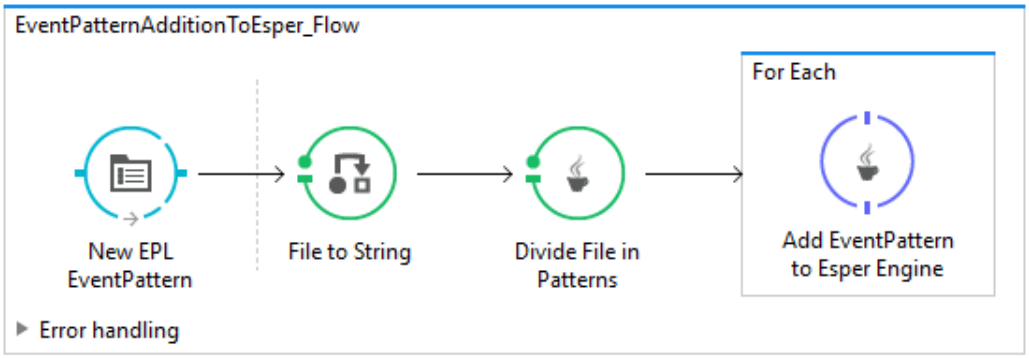

Figure 5. Fog Flow 1. Pattern deployment in the CEP engine

FLOW 2. Domain-specific data reception. As shown in Figure 6, this flow receives the Mosquitto broker data through a message topic to which it is subscribed. Data reach the system in JSON and are transformed into a Java Map and submitted to CEP engine. We can receive data from as many MQTT domain topics as the system requires. Following up from the previous example, these data could be patient admissions or, for instance, regional news on air quality alerts.

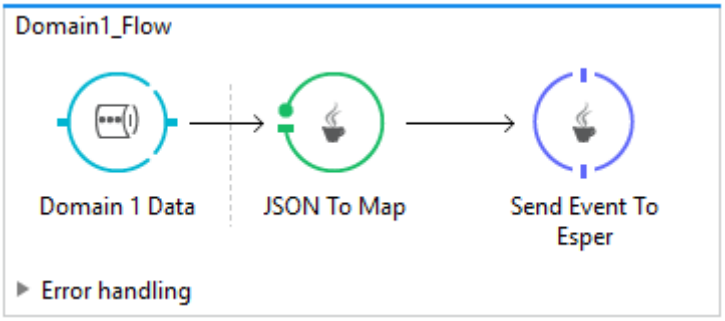

Figure 6. Fog Flow 2. Domain-specific data reception.

FLOW 3. Contextual data reception. Thorough the flow in Figure 7, data from a Mosquitto broker topic are received in the system. Data reach the system in JSON and are transformed into a Java Map and submitted to the CEP engine. We can receive data from as many MQTT domain topics as the system requires, either events coming from other fog nodes or coming from the cloud node.

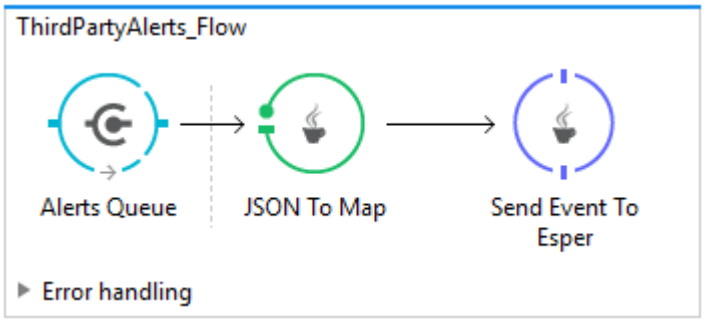

Figure 7. Fog Flow 3. Contextual data reception.

FLOW 4. Complex event detection and submission to the message broker. In this flow, as shown in Figure 8, relevant complex events are received. Depending on the type of event, the context broker, after transforming them to JSON, submits them either to the topic to which other fog nodes subscribe or to the topic to which the domain action system subscribes.

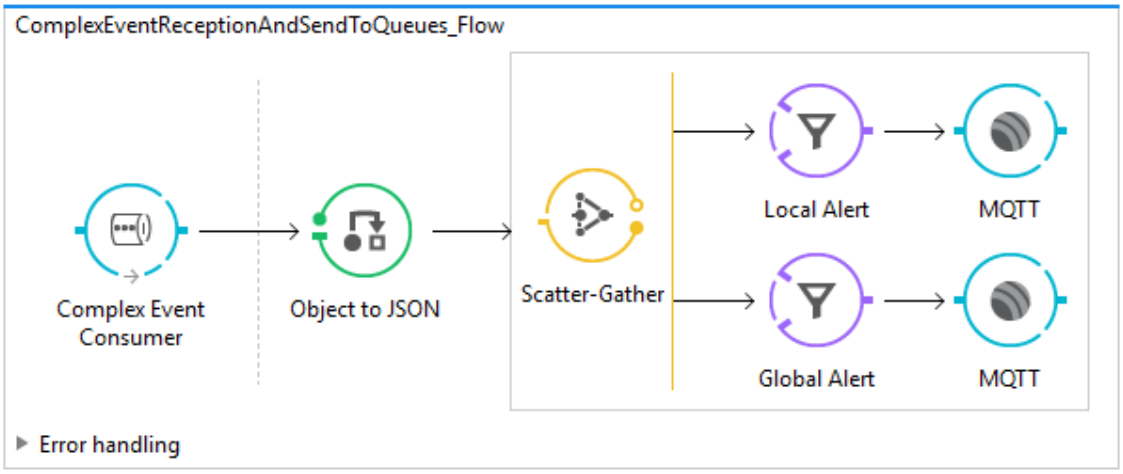

Figure 8. Fog Flow 4. Complex events detection and their submission to the Mosquitto Broker

4.3.2. Architecture Functionality

The SOA deployed in the cloud will follow the same functionality explained in (Boubeta-Puig et al., 2017) with the only difference of additionally publishing complex events of interest in a Mosquitto topic. That is, in a nutshell, the SOA receives data from several sources through a message queue or through querying an IoT platform, these data are saved and processed by the CEP engine and, when an alert is detected, a notification can be sent to subscribed users; additionally detected alerts are now sent to the Mosquitto broker.

The SOA deployed in the Fog will have the following functionality:

- Initially, event patterns are deployed in the system through flow number 1 (Figure 5). Additional patterns can be deployed at any time.
- The following messages reach the system throughout its lifecycle and are transformed into Java Maps for their submission to the CEP engine:
    - Domain-specific data reaching the system through the subscription to one or more message topics (Flow 2, Figure 6).
    - Contextual complex events reaching the system through the subscription to one or more message topics (Flow 3, Figure 7).
- The CEP engine processes all received events and, through the context reasoner implicit in the patterns, it detects new relevant contextual complex events as well as new action ones (Flow 4, Figure 8).
- The context-based adviser is responsible for routing the new complex events of interest to the corresponding topic: the topic to which other nodes in the collaborative architecture subscribe (contextual events) and the topic to which the action-taking system does (Flow 4, Figure 8).

5. Case Study

IoT applications for health care are taking great relevance nowadays (Islam et al., 2015); in this regard, air quality is one of the key topics in the focus of IoT applications. Indeed, air quality deserves special attention since it plays an essential role for citizens nowadays. Year after year, the world altogether is increasingly more conscious and worried about air pollution and how it can affect their daily lives. Among other consequences, air pollution can seriously affect citizens' health; particularly, it may worsen and favor certain illnesses or even cause death to specific risk groups (World Health Organization, 2013). This is why the whole society is

becoming more interested in this topic, paying extra attention to air quality. Moreover, health administration systems are paying special attention to it, since bad air quality implies a risk for citizens' welfare and the cost of a higher number of patients being admitted in emergency services.

The fact is that due to this worldwide concern, several IoT systems for air quality monitoring have been created over the last years. Nevertheless, the problem is that monitoring alone is not enough: correlation with hospital data and collaboration among them is required.

Since there is a lack of an internationally recognized standard for measuring air quality levels, several indexes have been created over the last years for reporting air quality. These provide us with information about how polluted or clean the air is in a particular area and which related effects on citizens' health might be a concern. In order to calculate the current air quality level for a particular location, each index requires the most relevant pollutants to be measured: Particulate Matter ($PM_{2,5}$ and PM_{10}), Carbon Monoxide (CO), Ozone (O_3), Nitrogen Dioxide (NO_2) and Sulphur Dioxide (SO_2). For testing purposes we have used the ranges and indexes provided by the US Environmental Protection Agency (EPA) as our air quality level classification (U.S. Environmental Protection Agency, 2014) (U.S. Environmental Protection Agency, 2016). In the referenced documents, we can find the categorization about general air quality based on a parameter calculated for the Air Quality Index (AQI), its influence on the public, as well as recommendations; we also present itemized information based on every air pollutant concentration which is relevant to citizen health. Depending on the concentration of each pollutant, citizen health might be affected in a different way. For instance, for level 4 of Ozone values in 8-hour periods the health concerns are: "*Greater likelihood of respiratory symptoms and breathing in people with lung disease (such as asthma), children, older adults, people who are active outdoors (including outdoor workers), people with certain genetic variants, and people with diets limited in certain nutrients; possible respiratory effects in general population*" (U.S. Environmental Protection Agency, 2016). Analogous information is provided for the 6 defined levels for all the relevant pollutants ($PM_{2,5}$, PM_{10}, CO, O_3, NO_2 and SO_2).

5.1. Air4HealthAdmin Description

Air4HealthAdmin is a collaborative context-aware SOA which provides a cloud node and several fog nodes for hospitals' intelligent decision-making regarding sudden rises in people with respiratory problems.

The cloud node consists of an architecture, which processes all the data from several air quality stations and sends notifications to registered users based on every station air quality. Equally, it submits all the relevant complex events detected to the message topic; that is, every time a pollutant changes level within the hospital's areas of influence, detected complex events are submitted to the message topics.

Fog nodes will be devices with limited capacity deployed physically in hospitals within a particular area. In particular, when implementing the case study, each of these nodes has been deployed in a Raspberry Pi 3 Model B, with 1GB of RAM memory and a 2GHz 64-bit quad-core ARMv8. These devices will receive (1) the data of emergency patient admissions, (2) the relevant complex events from the cloud node, (3) the relevant complex events from other fog nodes to which this node has subscribed.

The objective is that once a hospital receives a set of events coming from heterogeneous environments which trigger a complex event of interest, this can be used as an early warning for the remaining hospitals. This way, when there are evidences (simple events) that the same situation could be happening in another hospital, they can take special measures. For instance, let's assume than hospital 1 detects that when O_3 reaches level 4 they start receiving many asthmatic people with respiratory insufficiency in emergency admissions (many standing for when the number of patients exceed a particular threshold). Under these circumstances, a complex event could be submitted to the topic to which other hospitals are subscribed warning that they have level 4 of O_3 and that asthmatic patients are suffering from respiratory distress . Hospital 2 might have a good level of O_3 and not worry about the warning; they could even have a level 4 of O_3 and not worry because they are not receiving emergency patients with asthma, but it could also be the case that they also have a level 4 of O_3 and observe that the cadency for asthma patients through the emergency services is increasing. Then, without the need to reach the threshold of shortage patients in emergency, knowing that they have had this problem in other hospitals with such circumstances of air quality, they can start taking certain precautionary actions. Among them, for instance, they could send an alert to the doctor on duty, prepare rooms for this type of patients or require doctors who are off duty to come to hospital, warn patients who are registered in the hospital database such as people with asthma or simply warn the citizens subscribed to a notification system via their mobile or electronic mail.

5.2. Air4HealthAdmin Technical Requirements

As previously said, the cloud node, which is processing all the information about air pollutant values, should send the relevant detected air quality alerts per area of influence to the topics in a Mosquitto broker to which fog nodes can subscribe. We have used an emulator to send pollutant values for several geographic areas to the cloud node, so that relevant alerts can be detected (otherwise we would have to wait until such pollutant levels are reached locally in order to test the system).

For the fog node, the software engineer will have to do the following tasks:

- Decide the topics to which the local node will subscribe in the message broker to receive domain events. In the case study, making use of an emulator, we will send the messages of patient admissions in hospital emergency services with relevant symptoms. All fog nodes will subscribe to these topics.
- Decide the topics to which the local node subscribes in the message broker to receive contextual events. In the case study, both fog nodes subscribe to the cloud topic and to the other fog node topic.
- Configure the context broker: (1) creating patterns which establish a relation between context types and relevant complex event types (as we will see in the following subsection); (2) as well as setting which complex events detected in the local node are sent whether to the action topic or to the topic to which other nodes subscribe.

5.3. Air4HealthAdmin Context and Patterns

In this section, we are going to explain the type of contextual events available for the Air4HealthAdmin case study and the patterns defined in order to obtain added knowledge through the complex event processing of such contextual events and other domain events from the domain in question.

5.3.1. Contextual Event Types

The types of events that the CEP engine can receive are the following; their schemas are shown in Listing 1:

- Quality of air alerts. For this type of events, we will receive the air quality station in which the alert was detected; the alert description — the pollutant which raised the alert —, the kind of alert — using an identifier for the pollutant which raised the alert —, the alert level — according to EPA categories — and the value taken by the pollutant.
- Patient admission with certain symptoms: In this type of event, we will receive the air quality station to which influence area the patient belongs to, a patient identifier, the symptom — for instance, fever — and the value, when necessary — for instance, 39 degrees —.
- Local Hospital Alerts. These types of event are complex events detected in the local node, which contain the hospital identifier, the air quality station of the influence area the hospital belongs to, the alert description — the pollutant which raised the alert —, the kind of alert — using an identifier for the pollutant which raised the alert —, the alert level — according to EPA categories — and the total number of patients with relevant symptoms.
- External Hospital Alerts. These types of events are complex event detected in another node of the collaborative architecture, which contain the hospital identifier, the air quality station of the influence area that the hospital belongs to, the alert description — the pollutant which raised the alert —, the kind of alert — using an identifier for the pollutant which raised the alert —, the alert level — according to EPA categories — and the total number of patients with relevant symptoms.
- Global Alert. These types of events are global complex events detected in the local node, which contain the hospital identifier, the air quality station of the influence area that the hospital belongs to, the alert description — the pollutant which raised the alert —, the kind of alert — using an identifier for the pollutant which raised the alert —, the alert level — according to EPA categories — and the total number of patients with relevant symptoms.

Listing 1. Air4HealthAdmin Contextual Event Schema

```
AlertAirQuality(StationId integer, KndAlrtDscr string, KindAlert integer,
AlertLevel integer, Value double);

Admissions(StationId integer, PatientId integer, Symptom String, Value
double);

HospitalAlert(HospitalId string, StationId integer, KndAlrtDscr string,
KindAlert integer, AlertLevel integer, Total long);

ExtHospitalAlert(HospitalId string, StationId integer, KndAlrtDscr string,
KindAlert integer, AlertLevel integer, Total long);

GlobalAlert(HospitalId string, StationId integer, KndAlrtDscr string,
KindAlert integer, AlertLevel integer, Total long);
```

5.3.2. Contextual Events Pattern Definition

Based on the presented types of contextual complex events that we can have in the system, the following patterns have been defined.

The pattern in Listing 2 detects when patients with breathing shortage or chest tightness are reaching a hospital through emergency admissions; these are relevant respiratory symptoms which patients may have when air quality for several pollutants reaches level 3.

Listing 2. Pattern detecting patient admissions with relevant respiratory symptoms with level 3 air quality

```
@Name('Grp_SymptomAir_B01_3')
INSERT INTO Grp_SymptomAir_B01_3
SELECT adm1.StationId as StationId, COUNT (DISTINCT adm1.PatientId) as Total
FROM Admissions.win:time(1 hour) adm1
WHERE adm1.Symptom = "shortness of breath" OR
          adm1.Symptom = "chest tightness"
GROUP BY adm1.StationId;
```

Pattern in Listing 3 detects a local hospital alert when, having a level 3 air quality alert for O_3, $PM_{2.5}$ and PM_{10}, NO_2 or SO_2, at least 3 patients (we used a low threshold for testing purposes) coming from the area of influence of the station which detected the air quality alert with the symptoms in Listing 2 are received in the hospital over a period of one hour.

Listing 3. Pattern detecting patient admissions with relevant respiratory symptoms with level 3 air quality

```
@Name('HospitalAlert')
INSERT INTO HospitalAlert
SELECT "Hospital_002" as HospitalId,
            a1.StationId as StationId,
            a2.KndAlrtDscr as KndAlrtDscr,
            a2.KindAlert as KindAlert,
            a2.AlertLevel as AlertLevel,
            a1.Total as Total
FROM Grp_SymptomAir_B01_3.win:time(1 hour) as a1,
         AlertAirQuality.win:time(1 hour) as a2
WHERE a1.StationId  = a2.StationId AND
          ( a2.KndAlrtDscr = "O3" OR
            a2.KndAlrtDscr = "PM25" OR
            a2.KndAlrtDscr = "PM10" OR
            a2.KndAlrtDscr = "NO2" OR
            a2.KndAlrtDscr = "SO2"
          ) AND
          a2.AlertLevel = 3 AND
          Total > 10;
```

The pattern in Listing 4 detects a global precautionary alert — an alert which requires an action — in a hospital when, having received an alert of type *HospitalAlert* from another hospital, the local hospital receives 3 patients with the symptoms in Listing 2 (pattern *Grp_SymptomAir_B01_3*) coming from the area of influence of a station with level 3 air quality alert for the pollutants previously mentioned. It is important to be aware of the differentiation of local and external complex events *HospitalAlert*: when we receive a contextual complex event of type *HospitalAlert* from another hospital, Mule flow renames it as *ExtHospitalAlert*.

Listing 4. Pattern detecting a global precautionary alert with level 3 air quality

```
@Name('GlobalAlert')
INSERT INTO GlobalAlert
SELECT "Hospital_002" as HospitalId,
           a1.KndAlrtDscr as KndAlrtDscr,
           a1.KindAlert as KindAlert,
```

```
            a1.AlertLevel as AlertLevel,
            gs1.Total as Total
FROM Grp_SymptomAir_B01_3.win:time(1 hour) as gs1,
          AlertAirQuality.win:time(1 hour) as a1,
          ExtHospitalAlert.win:time(1 hour) as h1
WHERE  ( a1.KindAlert   = h1.KindAlert AND
a1.AlertLevel  = h1.AlertLevel AND
               gs1.Total > 3 ) AND
          ( h1.KndAlrtDscr = "O3" OR
            h1.KndAlrtDscr = "PM25" OR
            h1.KndAlrtDscr = "PM10" OR
            h1.KndAlrtDscr = "NO2" OR
            h1.KndAlrtDscr = "SO2"
          ) AND
          h1.AlertLevel = 3;
```

The three following patterns show how another global alert is detected. The pattern in Listing 5 detects when wheezing patients are admitted into hospital emergencies and group them by the station of their area of influence. The pattern in Listing 6 detects when, having a level 4 alert for NO2 or SO2, at least 2 patients (we used a low threshold for testing purposes) with wheezing symptoms and coming from the area of influence of the station with level 4 air quality alert have been admitted into hospital emergencies over a period of one hour. The pattern in Listing 7 detects a global precautionary alert in a hospital when, having received a contextual complex event of type *HospitalAlert* from another hospital, a patient belonging to the area of influence of the station with a level 4 air quality alert for NO_2 or SO_2 is admitted into emergencies with wheezing symptoms.

Listing 5. Pattern detecting patient admissions with relevant respiratory symptoms with level 4 air quality

```
@Name('Grp_SymptomAir_B03_4')
INSERT INTO Grp_SymptomAir_B03_4
     SELECT adm1.StationId as StationId, COUNT (DISTINCT adm1.PatientId) as
Total
     FROM Admissions.win:time(1 hour) adm1
     WHERE adm1.Symptom = "chest wheezing"
     GROUP BY adm1.StationId;
```

Listing 6. Pattern detecting patient admissions with relevant respiratory symptoms with level 4 air quality

```
@Name('HospitalAlert')
INSERT INTO HospitalAlert
     SELECT "Hospital_001" as HospitalId,
             a1.StationId as StationId,
             a2.KndAlrtDscr as KndAlrtDscr,
             a2.KindAlert as KindAlert,
             a2.AlertLevel as AlertLevel,
             a1.Total as Total
     FROM Grp_SymptomAir_B03_4.win:time(1 hour) as a1,
          AlertAirQuality.win:time(1 hour) as a2
     WHERE a1.StationId  = a2.StationId AND
           ( a2.KndAlrtDscr = "NO2" OR
```

```
            a2.KndAlrtDscr = "SO2"
          ) AND
          a2.AlertLevel = 4 AND
          Total > 2;
```

Listing 7. Pattern detecting a global precautionary alert with level 4 air quality

```
@Name('GlobalAlert')
INSERT INTO HospitalAlert
     SELECT "Hospital1" as HospitalId,
            a1.KndAlrtDscr as KndAlrtDscr,
            a1.KindAlert as KindAlert,
            a1.AlertLevel as AlertLevel,
            gs1.Total as Total
     FROM Grp_SymptomAir_B03_4.win:time(1 hour) as gs1,
          AlertAirQuality.win:time(1 hour) as a1,
          HospitalAlert.win:time(1 hour) as h1
     WHERE
             ( h1.HospitalId  != "Hospital_001" AND
               a1.KndAlrtDscr = h1.KndAlrtDscr  AND
  a1.KindAlert   = h1.KindAlert AND
  a1.AlertLevel  = h1.AlertLevel AND
               gs1.Total > 1 ) AND
          ( h1.KndAlrtDscr = "NO2" OR
            h1.KndAlrtDscr = "SO2"
          ) AND
          h1.AlertLevel = 4;
```

6. Evaluation

In this section, we explain firstly the performance tests we carried out for the architecture deployed in fog nodes in general, secondly the performance tests pursued for the Air4HealthAdmin case-study and finally we discuss our proposal strengths and weakness.

6.1. Fog Node Architecture Performance Evaluation

We have developed performance tests for the architecture deployed in the fog nodes. In particular, such a fog node architecture has been deployed in a Raspberry Pi 3 Model B ARM Cortex-A53 CPU 1.2GHz 64-Bit, with 1GB of RAM memory and with a SanDisk Ultra 16GB card. *Java Virtual Machine* (JVM) 8 on the Raspberry Pi has 512Mb of RAM memory assigned. Data enter and leave the node through a local network with a router switch 10/100.

As a guidance for the tests to be developed we have followed the statements used by Esper for their performance evaluation (EsperTech, 2017b), but characterizing the statements for the type of events of our proposed case study in Section 5 (see Listing 1); such statements are shown in Listing 8, where we have ordered the statements from lesser to higher complexity.

Listing 8. Pattern statements used for performance evaluation of the architecture in a fog node

```
@Name('Statement1')
INSERT INTO Statement1
SELECT *
FROM Admissions adm1
```

```
@Name('Statement2')
INSERT INTO Statement2
SELECT *
FROM Admissions adm1
WHERE adm1.Symptom = "shortness of breath"

@Name('Statement3')
INSERT INTO Statement3
SELECT *
FROM Admissions.win:length(1) adm1
WHERE adm1.Symptom = "shortness of breath"

@Name('Statement4')
INSERT INTO Statement4
SELECT *
FROM Admissions.win:time(10 min) adm1
WHERE adm1.Symptom = "shortness of breath"

@Name(' Statement5')
INSERT INTO Statement5
SELECT adm1.StationId as StationId, COUNT (DISTINCT adm1.PatientId) as Total
FROM Admissions.win:time(10 min) adm1
WHERE adm1.Symptom = "shortness of breath"

@Name(' Statement6')
INSERT INTO Statement6
SELECT adm1.StationId as StationId, COUNT (DISTINCT adm1.PatientId) as Total
FROM Admissions.win:time(10 min) adm1
WHERE adm1.Symptom = "shortness of breath"
GROUP BY adm1.StationId;
```

For every statement we performed four different test sets with several income events ratio; in particular for 50, 100, 200 and 300 events per second. The tests were prepared so that every simple event implies the generation of one complex event; therefore, in every test for each statement 50, 100, 200 and 300 complex events per second are detected and resubmitted, respectively. In every case, we measured the response time for every statement (the time passed from the instant a new event enters in the architecture until the corresponding generated complex event is notified), RAM memory used and percentage of CPU throughput. For every test, such values have been obtained as the average of the values obtained along the 10 minutes of execution after the JVM warm-up period, once the system was stable (JVM requires a period of execution — warm-up — to provide the best performance results; usually such performance is obtained after around 5 minutes of execution).

As we can see in Table 1, the response times are reasonable, as well as the memory and CPU usage. Please bear in mind that the event rate supported by the system differs from the one in Esper benchmarks due to several factors, such as (1) the fact of the system been deployed in a limited capacity device, rather than in a powerful station; (2) the use of events of 223 bytes of

size, rather those used in Esper performance tests of maximum 28 bytes; (3) the fact of measuring the performance for the whole architecture (through the ESB and input and output Mosquitto brokers), rather than directly with the CEP engine and (4) the fact of having to manipulate all the events received (a JSON element with an average of 5 fields) to obtain the relevant data and to format it according to the Esper engine requirements (Java Maps, in our case).

Table 1. COLLECT fog node performance tests results

	Test Set (Events/s)	Response Time (s)	Memory Usage (Mb)	CPU Throughput (%)
Statement 1	50	0.018	303	8.58
	100	0.010	307	8.30
	200	0.017	302	23.77
	300	0.011	300	30.78
Statement 2	50	0.015	303	7.55
	100	0.011	303	7.87
	200	0.011	301	24.01
	300	0.020	300	28.83
Statement 3	50	0.020	302	7.93
	100	0.015	305	7.85
	200	0.025	302	12.97
	300	0.011	304	20.39
Statement 4	50	0.009	303	10.24
	100	0.018	306	12.45
	200	0.010	303	23.53
	300	0.023	298	31.81
Statement 5	50	0.015	306	7.61
	100	0.020	305	7.78
	200	0.014	303	27.16
	300	0.024	302	31.07
Statement 6	50	0.010	305	7.95
	100	0.019	304	7.71
	200	0.011	305	23.65
	300	0.012	302	31.19

6.2. Air4healthAdmin Performance Evaluation

We have developed performance and stress tests for Air4HealthAdmin. We have tested the system with two Raspberries Pi — with the previously explained characteristics — which play the role of fog collaborative nodes. Both Raspberries have the same characteristics, except for the memory card, in the first case it is a Kingston (C10) 8GB card and in the second a SanDisk Ultra 16GB one, writing accesses being somehow slower in the first than in the second card. The node in the cloud was deployed in a workstation with i3-540 (4M cache memory, 3.6 GHz), 8GB of RAM memory and SATA2 hard drive. Communication between nodes has been carried out through a local network with a router switch 10/100.

We made the tests with an emulator which sends data for several air quality stations to the cloud node and data for patient admissions to the fog nodes. The emulator can generate random values as well as manually specified particular ones, and consists of an extension of the one explained in (Garcia de Prado, 2016). Every Raspberry Pi represents a different hospital. Therefore, each Raspberry Pi receives the same data about air quality alerts and their own patient admissions data, in both cases through topics coming from a Mosquitto broker to

which they are subscribed. Equally, both Raspberries Pi will receive contextual events of interest from the other node in the fog through another Mosquitto broker topic.

Performance tests have been pursued using temporal windows of 10 minutes (rather than 1 hour) in order to force the system. Besides, as previously mentioned, thresholds for number of patients with relevant symptoms have been set with low values to force the system to create and detect a higher number of complex events. Every test set has been performed for 20 minutes — enough time to check the effect of the time windows —, checking that system behavior did not deteriorate over time. Performance test results are shown on Table 2 and Figure 9 and Figure 10.

Table 2. Air4HealthAdmin performance tests results

TEST SET	1	2	3	4
Local events received per minute	100	175	250	400
External events received per minute	130	220	334	464
Total of events received per minute	230	395	584	864
Total of local events received	2000	3500	5000	8000
Total of external events received	2600	4400	6680	9280
Relevant detected events to be sent to the topic to which other nodes subscribe	2580	4400	6600	9680
Relevant detected events to be sent to the topic to which actions system subscribe	5140	8720	13260	12400
Total of relevant detected events	7720	13120	19860	22080
Local response time (RT) Rasp. 1 and Rasp. 2 (seconds)	0	0	0	0
External response time (RT) Rasp. 1 (seconds)	0.292	0.569	1.58	35.448
External response time (RT) Rasp. 2 (seconds)	0.04	0.42	1.861	30.835

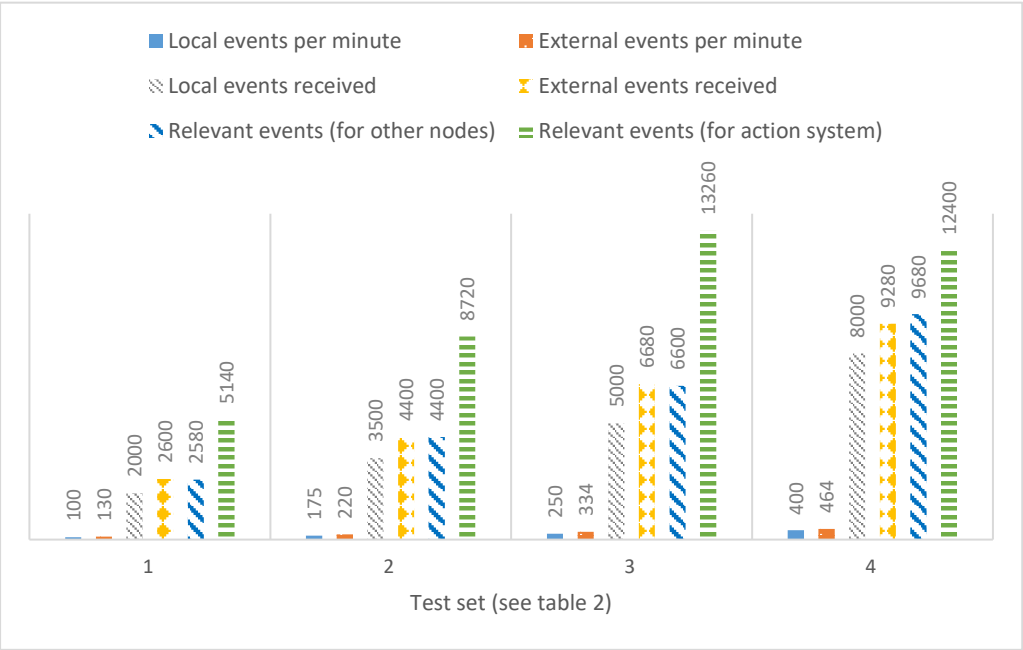

Figure 9. Received and detected events for Air4HealthAdmin fog node in performance tests

As can be seen in Figure 9, every node responds appropriately to an entry of around 864 events per minute, out of which 400 events consist of air quality alerts and patient admissions (in equal parts, approximately) and 464 are contextual events detected by another node in the architecture, to which the local node is subscribed. The chart shows, for the 4 test sets in Table 2 (230, 395, 584 and 864 events per minute received in the node, respectively), how many of

them are domain events (Local events per minute), how many are contextual events coming from another node (External events per minute), the total of local and remote events received (Received local events and Received external events) and the total of relevant complex events detected for their submission to a topic to which other nodes can subscribe (Relevant events for other nodes), and to the topic to which the action system is subscribed (Relevant events for actions).

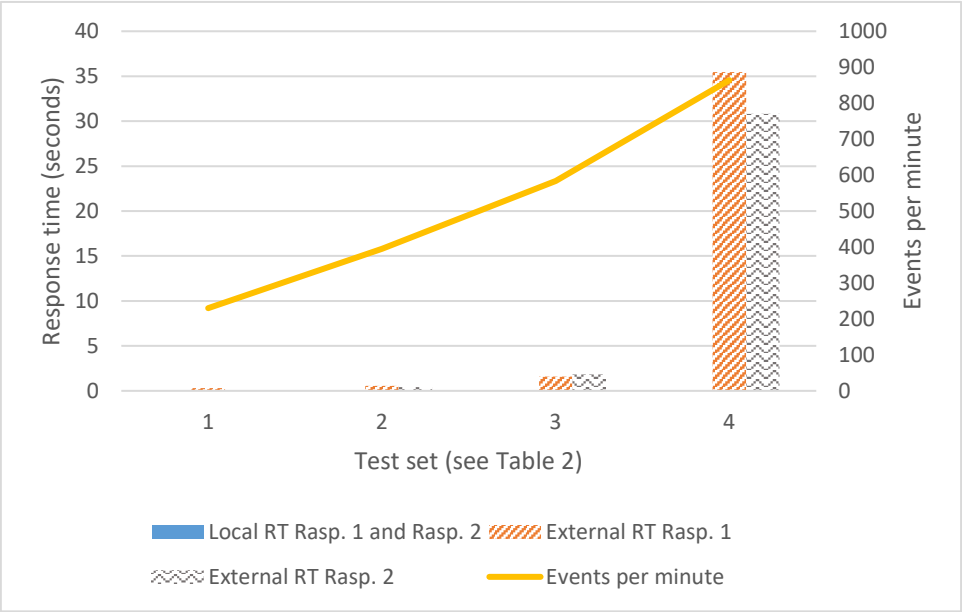

Figure 10. Collaborative architecture fog nodes response times

In the chart in Figure 10, we can see the response times for tests with 230, 395, 584 and 864 incoming events per minute (test sets in Table 2). We have observed response times depending on local events (Local RT) and depending on remote events (External RT); that is, how long the system takes to detect a complex event from the time in which a local event is created in the system, and how long it takes to detect a complex event from the time an external event is created in another node, respectively. We have grouped the local response times for both Raspberries since both of them respond in less than a millisecond. However, we can see that external response times are separate for both Raspberries since there is a small difference in their response times, even though it is not relevant. What can be clearly observed is that, as long as we increase the speed of events per minute submitted to the node, and therefore the number of submitted and detected events increases, the remote response time worsens. This fact can be due to the communication management between both nodes, since they have to place in the topic a higher number of relevant contextual events to be communicated between nodes through the message broker.

Finally, through the stress tests we have forced the system, in order to see at which speed of incoming events per minute the system would stop working properly. Table 3 shows the results for the stress tests for the first (R.1) and second (R.2) Raspberry. The table shows the events entering the system per minute (Events per min.), the incoming local events (Local events), events coming from another node which are entering the system (External events), and total relevant events detected in the node (Relevant events).

Table 3. Air4HealthAdmin stress tests results for both Raspberry Pi

RASPBERRY PI	R. 1	R. 2	R. 1	R. 2	R. 1	R. 2
Test set	1		2		3	
Local events received per minute	400		650		1000	
External events received per minute	464	484	724	399	396	347
Total of events received per minute	864	884	1374	1049	1396	1347
Total of local events received	8000		13000		20000	
Total of external events received	9280	9680	14480	7980	7920	6940
Total of relevant events detected	22080	24780	19500	33180	17640	22560

The chart in Figure 11 shows the first Raspberry's behavior for the three test sets presented in Table 3. As can be observed for Raspberry 1, the system behaves properly when having an entry of almost 900 events per minute, out of which 400 events consist of local events; however, when the entry is higher, for instance with an entry of 650 local events per minute (total of 1374 events per minute), the number of detected relevant events does not increase but drops, as compared to the previous speed; when we increase the speed to up to 1000 local events per minute (total of 1396 events per minute), both external events and relevant detected events drop, which evidences that the system has collapsed.

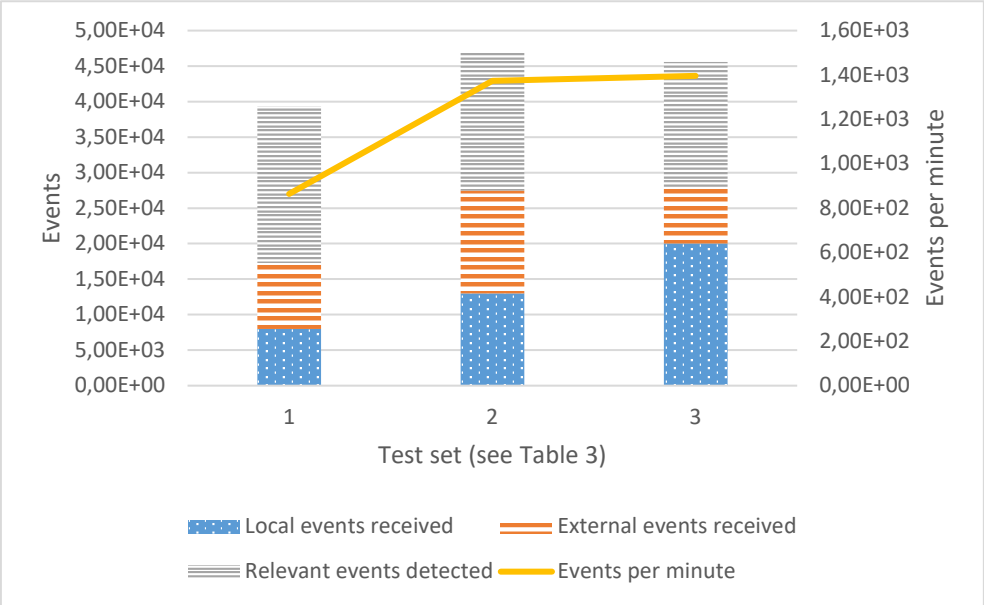

Figure 11. Stress test results for Raspberry 1

The chart in Figure 12 shows analogous results for the second Raspberry. As we can see, this one responds better than the previous one: at 650 local events per minute (total of 1049) events per minute) it still respond properly; however at 1000 local events per minute (total of 1347 events per minute) it also collapses. Even though this can be a collateral effect of the poor performance of the first one at such speed of events per minute, we estimate that the system is not responding properly to an entry of 1000 events per minute in total.

According to the tests performed, we can see that the collaborative architecture responds properly, since it is an architecture designed for smart devices with limited resources. We have to bear in mind that we have forced the number of events entering the system (it is not

realistic to have 200 air quality alerts per minute in a local area and 200 emergency hospital admissions per minute due to respiratory problems); besides, we would never receive contextual complex events from a remote node with a cadency of 464 events per minute). This fact allows us to scale the system for additional nodes, where the system designer must bear in mind the limitations in the number of topics of interest to which a node can subscribe. It is also important to remember that in these tests we have used Community versions of software, which have lower quality performance features than Enterprise ones, and also that communications have been done through a local network. We could also place the local message broker outside the smart device so as to alleviate the processing load in the node.

Figure 12. Stress test results for Raspberry 2

6.3. Discussion

In this section, we discuss the strengths and weaknesses of the proposal through the results of the evaluation and the comparison to other related works.

Through the case study evaluation, we can highlight the following strengths: we have confirmed that (1) the node in the cloud processes the data coming from the emulator and sends the detected quality of air alerts to the Mosquitto broker; (2) fog nodes receive the data from the cloud Mosquitto topic to which they are subscribed and they process these data together with the domain events (patient admissions) and contextual events (other hospital local alerts) entering the system through additional message topics; and (3) fog nodes obtain a greater knowledge of the domain in real time, thanks to contextual events being exchanged among nodes, and they can make intelligent decisions foreseeing situations of interest for the domain in question.

Besides, it has also been proved that smart devices with limited capacities, such as a Raspberry Pi, can receive and process events coming from heterogeneous sources through the proposed architecture in real time, as well as being able to send complex events of interest to a message broker to which other nodes in the collaborative architecture can subscribe. Performance tests have shown satisfactory results regarding the amount of messages expected to receive in the case study. Stress tests have shown the limit of the system concerning the number of events which can be received per minute.

Of course, COLLECT architecture, out of the particular case study, has limitations: we cannot process a huge number of events per second in the fog nodes; our fog nodes will be able to deal with a reduced number of events per second, as shown in Sections 6.1 and 6.2, and this is the main weakness of the system. However, precisely what we expect from a fog node is having to deal with a reduced number of events, since we have the cloud node to deal with higher number of events and only send to the fog ones those which are relevant for them. If we have a fog node in a hospital; how many patients can the emergency reception desk employee attend per minute? Obviously much less than the number supported by the fog node. If we are interested on relevant alerts from other hospitals, how many alerts will we receive per minute? For sure quite less than those supported by the system. Likewise, not all the events detected in our local node are relevant for other nodes in the architecture and we only need to send them those which are relevant for them and which will be easily handle by COLLECT. As a result, using COLLECT has the implication that the fog nodes can process a limited rate of events per second. However, let us highlight that replacing the proposed fog nodes by other more expensive ones with highest computing capacity, according to the requirements of the particular scenario, is not an issue; still keeping the essential goal of the proposed collaborative architecture.

Concerning the other approaches in the literature, we want to emphasize once more that most context-aware related approaches focuses on separate aspects of context-awareness, such as context modelling, provisioning, et cetera. The main strength of our work is the provision of a holistic architecture to deal with the context from the instant it enters the system until it is used to improve decision making in any node of the collaborative architecture. Of course, we require from other approaches in order to deal with initial context modeling, for instance using an ontological approach and we plan to work on this in the future. On the other hand, in the scope of IoT, even though some current proposals cover the fact of integrating heterogeneous context data for its processing, we could not find an architecture where third party relevant context could be shared among the different nodes participating in the architecture. Finally, context-aware expert systems mainly focus on particular fields or areas of application, again with the handicap of not benefitting from the chance of sharing context from multiple parties.

7. Conclusions

In this paper, we have presented an unprecedented collaborative context-aware service oriented architecture for the Internet of Things, the main contribution of the paper, which permits improving intelligent decision-making in IoT scopes facilitating real-time context spreading along the nodes in the architecture. Thanks to this novel approach, composed of both fog and cloud nodes, we process local IoT data with no need for submission to the cloud; therefore two additional contributions being (1) avoiding additional resource consumption to edge devices and (2) saving costs in cloud hosting. That way, cloud nodes focus on performing higher computational tasks which can be useful for several fog nodes. Besides, this architecture can also be useful to anonymize and preserve confidential data in the fog, so that such type of information does not have to travel towards the cloud through the Internet.

To conclude, COLLECT, unlike other traditional IoT architectures, lets us benefit from the great advantage of being able to infer more meaningful knowledge from real-time correlation of several heterogeneous domains and context data through distributed CEP, paving the way to collaborative intelligent decision-making.

COLLECT complements other approaches in the scope of expert and intelligent systems. Most of the existing approaches focus on information clustering and recommendation algorithms based on the use of privileged and intelligent systems, while COLLECT covers the open issue of how to obtain third-party contextual information by providing a software architecture to easy context information processing and sharing, as well as supplying real-time processing and delivery of such information to the interested parties in order to facilitate and speed up intelligent decision making. In general, other works of expert and intelligent systems, as explained in the related work section, focus on obtaining the context for a particular domain in an isolated mode (they can obtain heterogeneous data but all from domain-specific sources); we go one step further fostering collaboration among several nodes and therefore enriching the context information and improving the decision-making process in expert and intelligent systems.

In our near future work, we plan to extend COLLECT with real-time prediction which will let us improve intelligent decision-making in the domain in question; for whose aim contextual information is essential (Y. Xu, Yin, Deng, N. Xiong, & Huang, 2016). For this purpose, well-known expert system algorithms could be integrated in the context reasoner. We also plan to carry out research on how to incorporate a user profile in order to improve final user experience in the IoT scopes where it is required as similarly done, for instance, in the scope of the world wide web (Hawalah & Fasli, 2014); the user profile would be a key part of the expert knowledge base. Last but not least, we also plan, on the one hand, to extend our previous work on an ontological taxonomy for context-awareness (Peinado, Ortiz, & Dodero, 2015) to integrate it with COLLECT; on the other, to integrate our model-driven editor for real-time decision-making (MEdit4CEP) to facilitate the definition of such contextual domain and decision-making in a graphical way (Boubeta-Puig et al., 2015).

Acknowledgements

This work was supported by Spanish Ministry of Science and Innovation and the European Union FEDER Funds [grant number TIN2015-65845-C3-3-R] and the University of Cádiz [grant number PR2016-032]. We would like to thank companies 4gotas.com, Novayre Solutions and Homeria Open Solutions for their interest and support, as well as the personal support offered by Puerto Real Hospital pulmonologist Carmen Maza. We are also grateful to researchers Winfried Lamersdorf and Stephan Reiff-Marganiec for their interest in our ongoing research projects.

References

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999). Towards a Better Understanding of Context and Context-Awareness (pp. 304–307). Presented at the 1st International Symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany: Springer-Verlag. https://doi.org/10.1007/3-540-48157-5_29

Ashton, K. (2009). That “Internet of Things” Thing. *RFID Journal*, *22*(7), 97–114.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

Badii, A., Crouch, M., & Lallah, C. (2010). A Context-Awareness Framework for Intelligent Networked Embedded Systems (pp. 105–110). Presented at the 2010 Third International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services, Nice, France: IEEE. https://doi.org/10.1109/CENTRIC.2010.29

Behmann, F., & Wu, K. (2015). *Collaborative Internet of Things (C-IoT): for Future Smart Connected Life and Business*. Hoboken: John Wiley and Sons, Inc.

Benítez-Guerrero, E., Mezura-Godoy, C., & Montané-Jiménez, L. G. (2012). Context-Aware Mobile Collaborative Systems: Conceptual Modeling and Case Study. *Sensors*, *12*(12), 13491–13507. https://doi.org/10.3390/s121013491

Bhargavi, R., Pathak, R., & Vaidehi, V. (2013). Dynamic complex event processing — Adaptive rule engine (pp. 189–194). Presented at the International Conference on Recent Trends in Information Technology (ICRTIT), IEEE. https://doi.org/10.1109/ICRTIT.2013.6844203

Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2014). Approaching the Internet of Things through Integrating SOA and Complex Event Processing. In Z. Sun & J. Yearwood (Eds.), *Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications* (pp. 304–323). IGI Global. Retrieved from http://dx.doi.org/10.4018/978-1-4666-5884-4.ch014

Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2015). MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0. *Knowledge-Based Systems*, *89*, 97–112. https://doi.org/10.1016/j.knosys.2015.06.021

Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2017). Preventing Health Risks Caused by Unhealthy Air Quality Using a CEP-Based SOA 2.0. In *Internet of Things and Advanced Application in Healthcare* (pp. 170–196). Hershey, PA, USA: IGI Global.

Bruns, R., Dunkel, J., Masbruch, H., & Stipkovic, S. (2015). Intelligent M2M: Complex event processing for machine-to-machine communication. *Expert Systems with Applications*, *42*(3), 1235–1246. https://doi.org/10.1016/j.eswa.2014.09.005

Burstein, F., Brézillon, P., & Zaslavsky, A. (Eds.). (2011). *Supporting Real Time Decision-Making* (Vol. 13). Boston, MA: Springer US. Retrieved from http://link.springer.com/10.1007/978-1-4419-7406-8

Buyya, R., & Vahid Dastjerdi, A. (2016). *Internet of things: principles and paradigms*. Morgan Kaufmann. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1158785

Chanda, J., Sengupta, S., Kanjilal, A., & India, K. (2011). CA-ESB: Context Aware Enterprise Service Bus. *International Journal of Computer Applications*, *30*, 1–8.

CISCO. (2015). *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are* (White Paper). Retrieved from https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf

Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., & Buyya, R. (2016). Fog Computing: principles, architectures, and applications. In *Internet of Things* (pp. 61–75). Elsevier.

De Backere, F., Bonte, P., Verstichel, S., Ongenae, F., & De Turck, F. (2017). The OCarePlatform: A context-aware system to support independent living. *Computer Methods and Programs in Biomedicine*, *140*, 111–120. https://doi.org/10.1016/j.cmpb.2016.11.008

Dey, A. K. (2001). Understanding and Using Context. *Personal Ubiquitous Comput.*, *5*(1), 4–7. https://doi.org/10.1007/s007790170019

Eclipse. (2016). Mosquitto. Retrieved January 3, 2017, from https://mosquitto.org/

EsperTech. (2017a). Esper - Complex Event Processing. Retrieved April 17, 2017, from http://www.espertech.com/esper/

EsperTech. (2017b). Performance-Related Information. Retrieved May 10, 2017, from http://www.espertech.com/esper/performance.php

European Research Group in the Internet of Things. (2012). The Internet of Things 2012 New Horizons. Retrieved January 3, 2017, from http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2012_WEB.pdf

Forkan, A., Khalil, I., & Tari, Z. (2014). CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living. *Future Generation Computer Systems*, *35*, 114–127. https://doi.org/10.1016/j.future.2013.07.009

Fowler, M. (2005). Domain Event. Retrieved January 24, 2017, from https://martinfowler.com/eaaDev/DomainEvent.html

Garcia de Prado, A. (2016). Quality of Air Sensors Emulator. Retrieved from http://hdl.handle.net/10498/18582

Gil, D., Ferrández, A., Mora-Mora, H., & Peral, J. (2016). Internet of Things: A Review of Surveys Based on Context Aware Intelligent Services. *Sensors*, *16*(7), E1069. https://doi.org/10.3390/s16071069

Hawalah, A., & Fasli, M. (2014). Utilizing contextual ontological user profiles for personalized recommendations. *Expert Systems with Applications*, *41*(10), 4777–4797. https://doi.org/10.1016/j.eswa.2014.01.039

Immanuel, V. A., & Raj, P. (2015). Enabling context-awareness: A service oriented architecture implementation for a hospital use case (pp. 224–228). Presented at the International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Davangere, India: IEEE. https://doi.org/10.1109/ICATCCT.2015.7456886

Inzinger, C., Hummer, W., Satzger, B., Leitner, P., & Dustdar, S. (2014). Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems: event-based monitoring and adptation for distributed systems. *Software: Practice and Experience*, *44*(7), 805–822. https://doi.org/10.1002/spe.2254

Islam, S. M. R., Kwak, D., Kabir, M. D. H., Hossain, M., & Kwak, K.-S. (2015). The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access*, *3*, 678–708. https://doi.org/10.1109/ACCESS.2015.2437951

Kapitsaki, G. M., Prezerakos, G. N., Tselikas, N. D., & Venieris, I. S. (2009). Context-aware service engineering: A survey. *J. Syst. Softw.*, *82*(8), 1285–1297. https://doi.org/10.1016/j.jss.2009.02.026

Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., & Terziyan, V. (2008). Smart Semantic Middleware for the Internet of Things. In *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics* (pp. 11–15). Funchal, Madeira.

Khodadadi, F., Dastjerdi, A. V., & Buyya, R. (2016). Internet of Things: an overview. In *Internet of Things* (pp. 3–27). Elsevier. Retrieved from http://linkinghub.elsevier.com/retrieve/pii/B9780128053959000010

Kim, K., Kim, H., Kim, S.-K., & Jung, J.-Y. (2016). i-RM: An intelligent risk management framework for context-aware ubiquitous cold chain logistics. *Expert Systems with Applications*, *46*, 463–473. https://doi.org/10.1016/j.eswa.2015.11.005

Kwon, O. (2006). The potential roles of context-aware computing technology in optimization-based intelligent decision-making. *Expert Systems with Applications*, *31*(3), 629–642. https://doi.org/10.1016/j.eswa.2005.09.075

Luckham, D. C. (2012). *Event processing for business: organizing the real-time enterprise*. Hoboken, N.J: John Wiley & Sons.

MuleSoft. (2016). Flows and Sub-flows. Retrieved January 3, 2017, from https://docs.mulesoft.com/mule-user-guide/v/3.7/flows-and-subflows

MuleSoft. (2017). What is Mule ESB? Retrieved April 17, 2017, from https://www.mulesoft.com/resources/esb/what-mule-esb

Papageorgiou, N., Verginadis, Y., Apostolou, D., & Mentzas, G. (2011). Event-driven adaptive collaboration using semantically-enriched patterns. *Expert Systems with Applications*, *38*(12), 15409–15424. https://doi.org/10.1016/j.eswa.2011.06.020

Papazoglou, M. (2012). *Web services and SOA: principles and technology* (2nd ed). Essex, England ; New York: Pearson Education.

Papazoglou, M., & Heuvel, W. V. D. (2006). Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, *2*(4), 412–442. https://doi.org/10.1504/IJWET.2006.010423

Peinado, S., Ortiz, G., & Dodero, J. M. (2015). A metamodel and taxonomy to facilitate context-aware service adaptation. *Computers & Electrical Engineering*, *44*, 262–279. https://doi.org/10.1016/j.compeleceng.2015.02.004

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2012). CA4IOT: Context Awareness for Internet of Things (pp. 775–782). Presented at the Proceedings of the 2012 IEEE International Conference on Green Computing and Communications, Besançon, France: IEEE. https://doi.org/10.1109/GreenCom.2012.128

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, *16*(1), 414–454. https://doi.org/10.1109/SURV.2013.042313.00197

Rubinsztejn, H. K., Endler, M., Sacramento, V., Gonçalves, K., & Nascimento, F. (2004). Support for Context-Aware Collaboration. In A. Karmouch, L. Korba, & E. R. M. Madeira (Eds.) (Vol. 3284, pp. 37–47). Presented at the First International Workshop on Mobility Aware Technologies and Applications, Florianópolis, Brazil: Springer Berlin Heidelberg.

Sundermann, C. V., Domingues, M. A., Conrado, M. da S., & Rezende, S. O. (2016). Privileged contextual information for context-aware recommender systems. *Expert Systems with Applications*, *57*, 139–158. https://doi.org/10.1016/j.eswa.2016.03.036

Taher, Y., Fauvet, M.-C., Dumas, M., & Benslimane, D. (2008). Using CEP technology to adapt messages exchanged by web services (pp. 1231–1232). New York, NY, USA: ACM. https://doi.org/10.1145/1367497.1367741

The Alliance for Internet of Things Innovation. (2015). Internet of Things Applications. Retrieved from https://ec.europa.eu/digital-single-market/en/news/aioti-recommendations-future-collaborative-work-context-internet-things-focus-area-horizon-2020

Uhm, Y., Lee, M., Hwang, Z., Kim, Y., & Park, S. (2011). A multi-resolution agent for service-oriented situations in ubiquitous domains. *Expert Systems with Applications*, *38*(10), 13291–13300. https://doi.org/10.1016/j.eswa.2011.04.150

U.S. Environmental Protection Agency. (2014). AQI Air Quality Index. A Guide to Air Quality and Your Health. Retrieved January 17, 2017, from https://www3.epa.gov/airnow/aqi_brochure_02_14.pdf

U.S. Environmental Protection Agency. (2016). Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI). Retrieved January 17, 2017, from https://www3.epa.gov/airnow/aqi-technical-assistance-document-may2016.pdf

World Health Organization. (2013). *Review of evidence on health aspects of air pollution – REVIHAAP project* (Technical Report). Retrieved from http://www.euro.who.int/__data/assets/pdf_file/0004/193108/REVIHAAP-Final-technical-report-final-version.pdf?ua=1

Xu, L. D., He, W., & Li, S. (2014). Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, *10*(4), 2233–2243. https://doi.org/10.1109/TII.2014.2300753

Xu, Y., Yin, J., Deng, S., N. Xiong, N., & Huang, J. (2016). Context-aware QoS prediction for web service recommendation and selection. *Expert Systems with Applications*, *53*, 75–86. https://doi.org/10.1016/j.eswa.2016.01.010