Received February 14, 2017, accepted March 2, 2017, date of publication March 8, 2017, date of current version April 24, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2679338

CARED-SOA: A Context-Aware Event-Driven Service-Oriented Architecture

ALFONSO GARCÍA DE PRADO[1], GUADALUPE ORTIZ[2], AND JUAN BOUBETA-PUIG[2]
[1]UCASE Software Engineering Research Group, Department of Computer Technology and Architecture, School of Engineering, University of Cádiz, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain
[2]UCASE Software Engineering Research Group, Department of Computer Science and Engineering, School of Engineering, University of Cádiz, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain

Corresponding author: A. García de Prado (alfonso.garciadeprado@uca.es)

This work was supported in part by the Spanish Ministry of Science and Innovation and the European Union FEDER Funds under Project TIN2015-65845-C3-3-R and in part by the University of Cádiz under Project UCA PR2016-032.

ABSTRACT Currently, context awareness has become essential in software applications and services owing to the high demand by users, especially for mobile computing applications. This need to provide context awareness requires a software infrastructure not only to receive context information but also to make use of it so that it provides advantageous services that may be customized according to user needs. In this paper, we provide an event-driven service-oriented architecture supported by an enterprise service bus, which will facilitate the incorporation of Internet of Things data and provide real-time context-aware services. The result, which has been validated through a real-world case study, is a scalable context-aware architecture which can be applied in a wide spectrum of domains.

INDEX TERMS Service-oriented architecture, context awareness, complex event processing, Internet of Things.

I. INTRODUCTION

Context awareness has become a fundamental requirement for software engineering, since nowadays it takes part of citizens' day-to-day life. People leave their homes expecting the lights, which they left on, to be turned off automatically; also that their mobile device will warn them if there is traffic congestion on their way to work; that windscreen wipers automatically turn on if it is raining; that their office heater sets itself to their favorite temperature when they go in; that the mobile device will tell them if there is a friend close by, and a long list of additional expected context awareness issues that are taken for granted nowadays. However, there is a limited amount of context-aware services that users can benefit from; there are still many offered services which are not context-aware. Citizens expect to be able to make use of such services easily and intuitively; besides, they expect the service to be adapted to his particular context and that it will warn him about any relevant information concerning his particular circumstances. Furthermore, they expect service context awareness to be carried out *here and now*, when things happen, without delays which might be detrimental to him. This need for having the context under control, being aware of what happens at every instant, requires a software infrastructure, not only for receiving the context information but also to make use of it to provide advantageous services which can be customized according to user needs.

We have to bear in mind that currently the strategy for software development oriented to citizens, as well as to other agents, is mainly based on services [1], [2], since *Service Oriented Architectures* (SOAs) are platform-independent and loosely coupled, essential requirements when trying to reach a high number of users. Besides, *Representational State Transfer* (REST) services [3] have become very successful since they are light services which can be easily consumed by any third-party client [4].

On the other hand, the impressive evolution of the *Internet of Things* (IoT) over the last years has strongly favored the provision of information by multiple sensors and other devices connected to the Internet. IoT platforms capture these data and transform them to a light format that other software applications can easily consume. The amount of generated data is huge and the term *Big Data* is coined: Big Data refers to a large amount of heterogeneous data which flow along the information systems and are stored and analyzed with the aim of improving decision making in the domain in question. However, the amount of data generated in the scope of the IoT is so huge, and it is generated so fast, that a constant streaming processing is required so as to obtain real-time data

2169-3536 © 2017 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

for our business decision-making. New technologies, such as Complex Event Processing (CEP), rise in order to provide this constant data processing in *streaming*; with CEP, we can analyze and correlate huge amounts of data which flow through information systems in real time.

Therefore, we are facing a scenario in which we have a lot of information from diverse sources that could be offered to citizens and particularized to their context, we have the means to analyze the data, by using CEP, and we have the chance to reach the citizens easily, by using SOAs; but we do not have a wide spectrum architecture which facilitates the integration of all these elements so that they provide the required context-aware services in the scope of the IoT. Indeed, the European Union identifies, among the Horizon 2020 challenges, research and development for context-aware IoT computation. According to Xu *et al.* [5] the challenge should be faced by designing a SOA (1) which facilitates the incorporation of data coming from devices connected to the IoT, (2) which makes data transport among system agents easier and (3) which can process these data and (4) offers services to the user.

The main aim of this paper is to provide a software architecture to face this challenge: CARED-SOA. First of all, as it could not be otherwise in this scope, CARED-SOA is an Event-Driven SOA (ED-SOA) supported by an Enterprise Service Bus (ESB) which (1) will facilitate the incorporation of data coming from devices connected to the IoT through several connectors and (2) will facilitate communications among all involved agents (IoT devices, data analyzers, context broker, users, et cetera). Besides, the architecture (3) will also provide real-time stream data processing through the integration of CEP technology and (4) will offer REST services to the users, which will be context-aware.

Besides, the paper also provides the implementation of a real-world case study based on the proposal, which permits the evaluation of the architecture.

The rest of the paper is organized as follows: Section II presents the background required for understanding the technologies and paradigms used in the paper. Then, Section III explains the design of the proposed context-aware ED-SOA; afterwards, how the architecture is implemented by specific technologies is explained in Section IV. The case study description follows in Section V and its evaluation can be found in Section VI. Finally, related work is examined in Section VII and conclusions are summarized in section VIII.

II. BACKGROUND

In this section, we introduce the most relevant technologies and knowledge in order to understand the paper: SOA, ED-SOA, context awareness and IoT.

A. SERVICE ORIENTED ARCHITECTURE

SOA consists of a paradigm for the design and implementation of loosely coupled distributed systems which make use of services for their implementation. These architectures easily integrate 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 [1].

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. Decoupling is obtained thanks to technology independence.

One of the most common ways to implement SOAs are web services. Web services are self-descriptive software modules which can be accessed through a net and which develop a task facilitating machine to machine interoperability [1]. REST web services emerged as an alternative to more traditional SOAP web services. REST is an architectural style for distributed hypermedia systems where services provide resources identified by URLs [3]. Communications between REST services and their clients take place using HTTP main operations, mainly GET, POST, PUT and DELETE.

With the growth of service components and processes in service oriented applications, a new service infrastructure is required for maintaining applications in a flexible way. This infrastructure must support well-known web service standards and provide support for a message middleware [1]. These requirements are fulfilled by an ESB. An ESB provides services to complex architectures through a messaging system [1], supplying interoperability among diverse applications and components through standard interfaces; that allows applications to be offered as services in the ESB.

B. EVENT-DRIVEN SERVICE ORIENTED ARCHITECTURE

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 [6]. To facilitate this paradigm, a software abstraction layer is required to integrate diverse heterogeneous data sources and distributed invocations [7]. These functionalities are offered by an ESB, which permits interoperability among several communication protocols and heterogeneous data sources and targets.

Despite all the advantages provided by SOA 2.0, this type of architecture might not be ideal to analyze and correlate big amounts of data in terms of events. To meet this requirement, it is necessary to integrate CEP [6], which is a technology that allows the capturing, analyzing and correlating of a large amount of heterogeneous data with the aim of detecting relevant situations in a particular domain [8]. 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. 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.

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, as well as permitting storing the messages in the broker until they can be processed by the target element. These brokers may use standard message queues or be combined with a publish/subscribe mechanism.

C. CONTEXT AWARENESS

Dey *et al.*'s context definition in [9] is specially well-known– page 3, section 2.2: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*". One of the particular features of context information is that it is specific to each system, so that one specific type of information can be considered as part of the context in a given system but not in a different one.

The term context awareness supports the fact that the context information provided by the client, or taken from the 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 its necessities 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, adapting the system behavior to the particular needs of a specific user [10].

Context has been faced from several perspectives: classification, frameworks, middlewares, et cetera; but the main drawback is the difficulty of providing a wide spectrum proposal due to the particularities of context in every different scope. This also implies great difficulty to establish a standard definition or classification for contexts and their characteristics [11]; therefore, this leads to several classifications. For instance in COIVA [12], context is subdivided in *users*, *devices*, *environment* and *services*; in MoDAS [13], however, seven different types are defined. These are only two of multiple examples which can be given.

D. INTERNET OF THINGS

IoT is defined as a network formed by interconnected physical objects uniquely identified [14] and implies integration, transfer and analysis of the data coming from such objects. Currently, several algorithms, tools, technologies and best practices enable IoT applications and architectures for a variety of application domains [15]. IoT architectures must provide a set of essential elements such as sensors, offered services, communication networks and event context processing. One key requirement is interoperability, so that standard interfaces must be provided to facilitate information submission from the devices. Reliability and scalability are also essential [15].

There are multiple IoT platforms where we can obtain real-time data. It is important to be aware of the importance that IoT is gaining: in the near future, the economic impact expected from IoT applications is 11% of the worldwide economy [15].

III. DESIGN OF CONTEXT-AWARE EVENT-DRIVEN SERVICE ORIENTED ARCHITECTURE

When designing an architecture for the IoT, we have to bear in mind that any stream data processing architecture has to include a set of key elements to cover all the stages of data processing, from their capture to final user consumption [15]:

- A system to collect data from diverse sources and to add them into a unique channel. In the scope of IoT, we currently have distributed *stream* data which comes from diverse sources in heterogeneous formats.
- A messaging and storing system that behaves as a message broker and permits the combination of messages coming from different sources.
- A system for non-stop data processing to detect relevant information for the domain in question.
- A presentation system, through which the final user can benefit from the processed data.
- A storing system crosscutting all the remaining elements so that it maintains historic data and is able to query relevant domain data.

Taking into account such requirements, in this section we explain the components which will compose CARED-SOA—the context-aware proposed architecture—, we describe how these components are integrated altogether and what the communication among them is like, and then we pay special attention to the proposed context broker.

A. CARED-SOA COMPONENTS

To comply with the established requirements, we designed and implemented the architecture in Fig. 1 which is composed of the following elements:

1) ENTERPRISE SERVICE BUS

An ESB is in charge of routing and facilitating communications in a SOA. In the proposed architecture, the bus channels the following communications: simple event detection, event transformation and routing to the CEP engine and database, complex event reception of the events notified by the CEP engine, complex event routing to the context broker, user database query and notification submission based on their subscriptions and contexts.

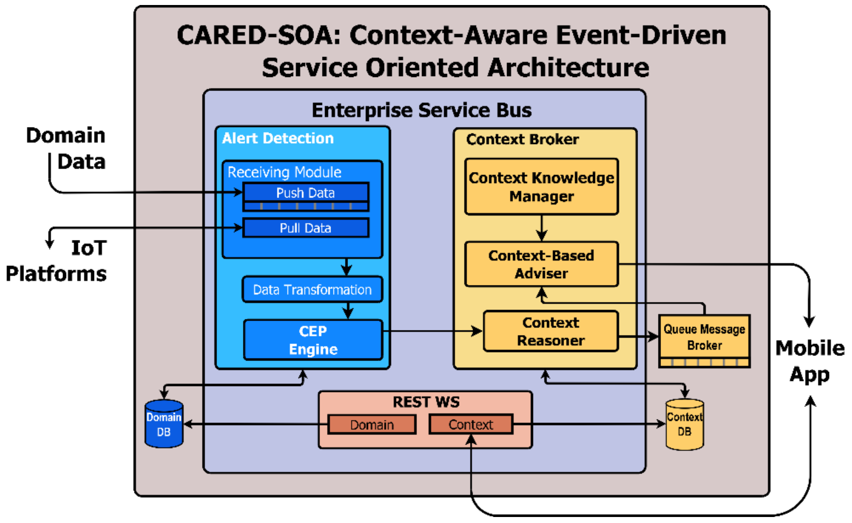


FIGURE 1. CARED-SOA architecture.

2) DATA TRANSFORMATION

Several data transformation procedures are followed in the bus; these allow us to obtain data from heterogeneous sources and to adapt them to a common format to be sent to the databases and the CEP engine, as appropriate.

3) RECEIVING MODULE

The receiving module is composed of two independent elements to be able to obtain events data both following the push and pull models. The ideal system is a push model, where sensor data are sent to the system. In this case, we have a message queue managed by the ESB which receives and manages the messages to ensure correct delivery into the system. However, we may wish to integrate sensor information which is published in another platforms into the system. In this case we have to follow a pull model and retrieve the data from the platforms. In particular, our system retrieves information from ThingSpeak.

4) DOMAIN DB

This database stores all domain-specific information, particularly all the data reaching the system and all the required information related to the sources from where data are obtained.

5) DOMAIN REST WEB SERVICE

This service provides the functionality for querying the information about the domain events. The REST service is provided in order to facilitate the use of the data stored in the domain database in a transparent manner for the final user.

6) CEP ENGINE

It is a specific software capable of analyzing the data coming in form of events in real time. The patterns to detect relevant alerts in the domain in question are deployed in this engine.

7) CONTEXT BROKER

The context broker will be in charge of managing the prospective clients which may receive notifications, as well as submitting such notifications in real time. The context broker is composed of three modules which are explained in Section III.C. These modules interact with other elements which are distributed along the architecture: the context and subscriptions database, the context and notification service and the message broker which manages the notification of relevant complex events.

8) CONTEXT DB

This database stores user personal data and their context as well as data about the alerts available in the system.

9) CONTEXT AND NOTIFICATION REST WEB SERVICE

This service permits checking available notifications and storing user context data as well as the notifications the user would like to subscribe to.

10) COMPLEX EVENT NOTIFICATION MESSAGE BROKER

Complex events which are associated to a notification (that is, events which are really relevant in the domain in question) are sent to one or more message queues whose messages will then be managed for the corresponding notification to be sent. Even though the queue message broker is outside the bus, it is part of our proposed architecture and it is where we are going to publish all the relevant complex events to which our system subscribes in order to send notifications. The purpose of sending these events to the context broker is to improve system performance, so that the notification module can be completely decoupled should it be necessary.

Fig. 1 also shows the external implemented elements interacting with the architecture:

11) DOMAIN DATA

Domain data is expected to be obtained, according to the push model, from a message topic in a message broker.

12) IoT

From the ESB we pursue regular queries to the IoT data provider to obtain sensor data of the domain in question. In particular, we query the API offered by ThingSpeak to obtain channels data.

13) MOBILE APP

A mobile application can be used to send context information to the system and to receive domain notifications.

B. 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*. in [16].
    a. On the one hand, 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 suitable 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 such complex event to the ESB, where it is managed as appropriate.
2. REST services have to be deployed in the ESB and are offered to prospective external clients through an HTTP *listener*. They can be invoked from a mobile application or from any other implemented client (own or from a third party). REST services communicate with the databases, which store and update contextual, notification and subscription data as well as events received and detected in the system.
3. Databases receive queries from the bus flows as well as from the REST services. We have used relational databases. Even though No-SQL databases are well considered in the scope of the IoT, we prioritized speeding up the database reading through the use of relational ones, so as to obtain event information and statistics with low latency.
4. The bus sends data requests to the IoT platform through HTTP. The domain database keeps information about the ThingSpeak channels to be queried and about the data from these channels of interest for the domain. With this data a periodic call is sent to the named channels.

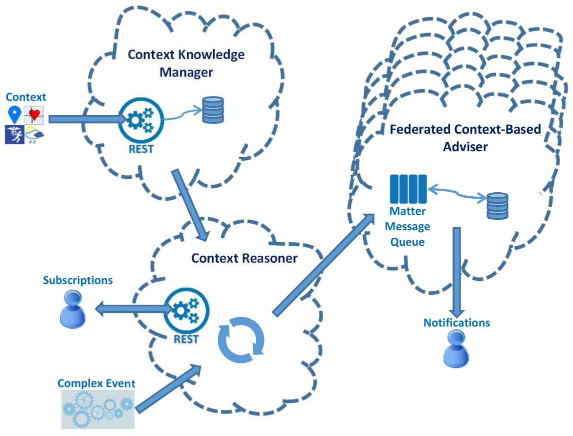


FIGURE 2. CARED-SOA context broker.

5. In order to receive additional domain events, the bus receives data from the topic in the message broker to which it is subscribed.
6. The bus sends the detected relevant complex events to the system message broker, which manages one or more message queues according to one or more types of complex events which implies notification.
7. The bus is subscribed to the complex events message queue to proceed with the notifications.

C. CARED-SOA CONTEXT BROKER

As previously explained, the context broker is in charge of managing everything related to the context: the users which may receive notifications and their context, as well as the notifications to be sent in real time.

The context broker is composed of three main modules: the context knowledge manager, the context reasoner and a federation of context-based advisers, as represented in Fig. 2.

1) CONTEXT KNOWLEDGE MANAGER

The context knowledge manager keeps the subscribed users' context up to date. Such manager is composed of the REST service which receives the user context data and stores them in the context database.

Context data may be static (users do not require a constant update in the system) or dynamic, which may require constant update in the system [15]. For example, static context for a user could be his age, a disease, a hobby, et cetera; dynamic contexts could be the user's location, the activity he is carrying out, weather conditions, et cetera. It is important to remember that the relevant context is particular for the domain in question. A user, when registering to the system, provides his static context data (which may be modified at any time); data from dynamic contexts are sent to the system whenever they change. It is worth mentioning that the extraction and submission of the context should be done by the software client (invoking the context REST service) and it is out of the scope of this paper.

2) CONTEXT REASONER

The context reasoner will match the notifications which a user is subscribed to with his context. It makes use of the context REST service, which allows him to check and subscribe to notifications of the domain in question and to access the database for storing the notifications to which every user subscribes.

When a user requests which type of notifications he may subscribe to, the context reasoner suggests relevant ones depending on his particular context, which would have been previously checked in the database.

On the other hand, the context reasoner, when receiving a relevant complex event, will resubmit it to the notification message queue according to the type of complex event and its subsequent notification.

3) CONTEXT-BASED ADVISER

Then, the context-based adviser receives such complex events from the notification message queue, accesses the database which keeps the notifications corresponding to such complex events and sends them to the users according to the type of notification and to their context and preferences.

In order to facilitate system scalability and agility for notifications, the architecture provides the opportunity of dividing the notification system depending on the type of complex event detected. That is, if a system can detect relevant A, B and C complex events, we could have a context-based adviser federation, with three advisers (for types A, B and C). Each adviser would have a distributed database with the users subscribed to the type of event in question, A, B or C, respectively.

Depending on domain application and system magnitude, the developer implementing the architecture must take a decision with regards to granularity, choosing the most appropriate adviser federation for the system in question. When having an adviser federation, only the information needed for user notifications is replicated in the local databases.

IV. CARED-SOA IMPLEMENTATION

In this section, we describe the implementation of our proposed architecture.

A. TECHNOLOGIES AND FLOWS

We have used the following technologies for our implementation, which are represented in a schematic way in Fig. 3 and explained below:

- Mule [17]. It is an ESB with a wide functionality and good support from the community.
- Esper and Esper Event Processing Language (EPL). Esper is a recognized CEP engine, which provides Esper EPL for event pattern implementation.
- MySQL [18]. Relational databases have been implemented using MySQL, since it is a popular open source database manager.

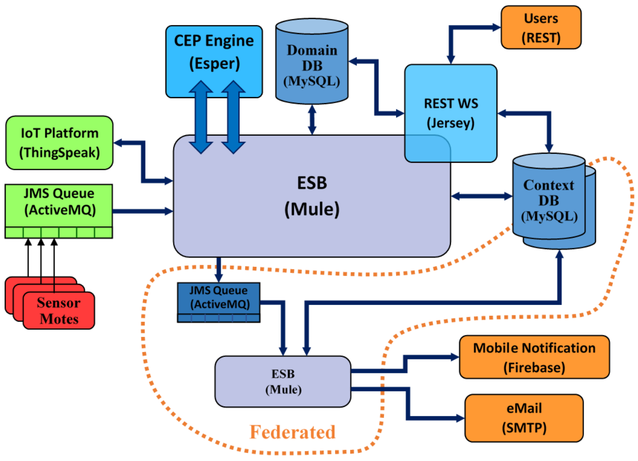


FIGURE 3. CARED-SOA technologies schematic view.

- ActiveMQ [19]. This Apache message broker facilitates queue and topic message dealing with diverse messaging models.
- ThingSpeak [20]. It is a free use highly scalable and well documented platform, which provides a RESTful API for manipulating data in JSON, CSV and XML formats.
- Jersey [21]. Jersey in an Oracle library that implements JAX-RS API and provides utilities for JAX-RS REST web service implementation.
- Simple Mail Transfer Protocol (SMTP) [22]. SMTP is the de facto standard message protocol for electronic mail.
- Firebase. Firebase is a platform recently presented by Google which improves Android applications significantly [23]. Among other utilities, it facilitates notifications to mobile devices through a manager in the cloud.

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 [24]. In order to provide the SOA, a Mule application with the flows explained below has been implemented. Flows have been enumerated for clarity purposes, but they all run in parallel, even though some of them are loosely coupled: Flow 3 and 4 outputs are sent to Flow 5; and Flow 6 receives the complex event detected by the CEP engine, that receives simple events from Flow 5. The flows are continuously running independently.

1) FLOW 1. PATTERN DEPLOYMENT IN THE CEP ENGINE

As we can see in Fig. 4, 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.* [16].

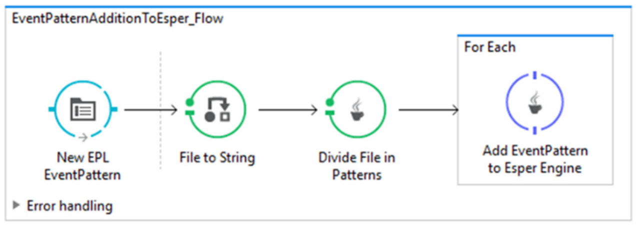


FIGURE 4. CARED-SOA Flow 1: Pattern deployment in the CEP engine.

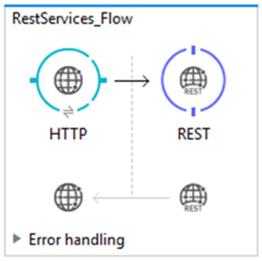


FIGURE 5. CARED-SOA Flow 2: Access to REST services.

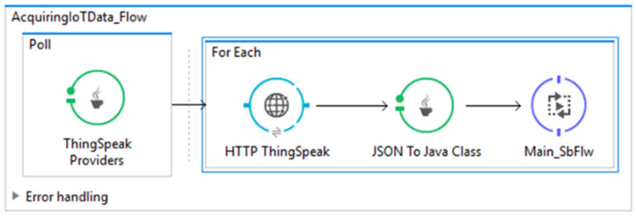


FIGURE 6. CARED-SOA Flow 3: Querying data through the pull model.

2) FLOW 2. ACCESS TO REST SERVICES

Through the invocation of an HTTP listener, the bus can invoke the REST services, as shown in Fig. 5.

3) FLOW 3. QUERYING DATA FROM THE INTERNET OF THINGS (PULL MODEL)

As shown in Fig. 6, first of all, this flow queries the database where we have the IoT channels of interest. Then, it makes a request to the IoT platform for each selected channel, it transforms the obtained JSON data into a class instance which will be later used for database storage and CEP submission and it submits such class instance to the main flow.

4) FLOW 4. MESSAGE BROKER DATA RECEPTION (PUSH MODEL)

As shown in Fig. 7, this flow receives the message broker data through a message topic. As done in the previous flow, it transforms the obtained JSON data into a class instance which will be later used for database storage and CEP submission and it submits such class instance to the main flow.

5) FLOW 5. MAIN FLOW

As shown in Fig. 8, this flow receives the class instances from flow 3 and 4 and transforms the data to store them in the database and submit them to the CEP engine, which is done in parallel.

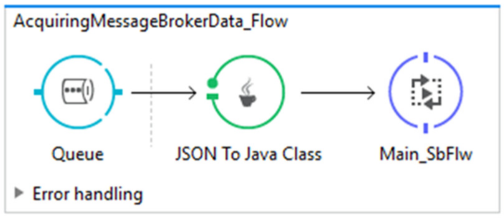


FIGURE 7. CARED-SOA Flow 4: Obtaining data through the push model.

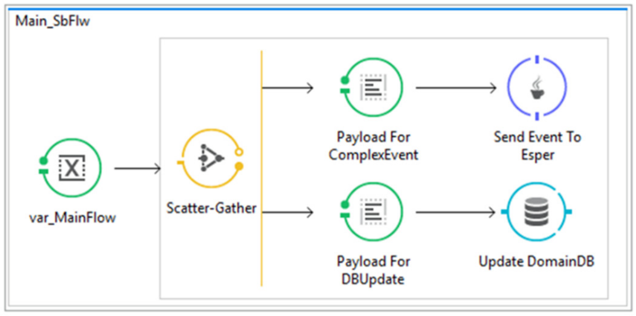


FIGURE 8. CARED-SOA Flow 5: Storing and processing events.

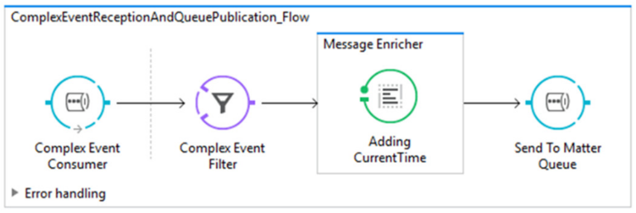


FIGURE 9. CARED-SOA Flow 6: Event filtering and notification.

6) FLOW 6. EVENT FILTERING AND ENRICHMENT

Fig. 9 shows that the complex events received from the CEP engine are filtered according to their relevance. Once those which do not require notification have been discarded, the remaining ones are enriched with the current timestamp (to be aware of when the event was detected) and sent to the message queue of the corresponding notification flow.

7) FLOW 7. NOTIFICATION

We receive the events which imply notification through the message broker. These are filtered to ensure that only relevant events are processed for notification, as shown in Fig. 10.

Then, data are formatted and the database is queried in order to obtain the users' information for their notification. Afterwards, in parallel, (1) we store complex events in the database and (2) we notify the user via email or mobile notification according to his preferences. We also store the notification timestamp in the database to be able to follow user preferences with regards to notification regularity. Please be reminded that, according to the federation of context-based advisers, we may have one or more notification flows depending of the requirements of the domain in question. All flows follow the same design that has just been explained.

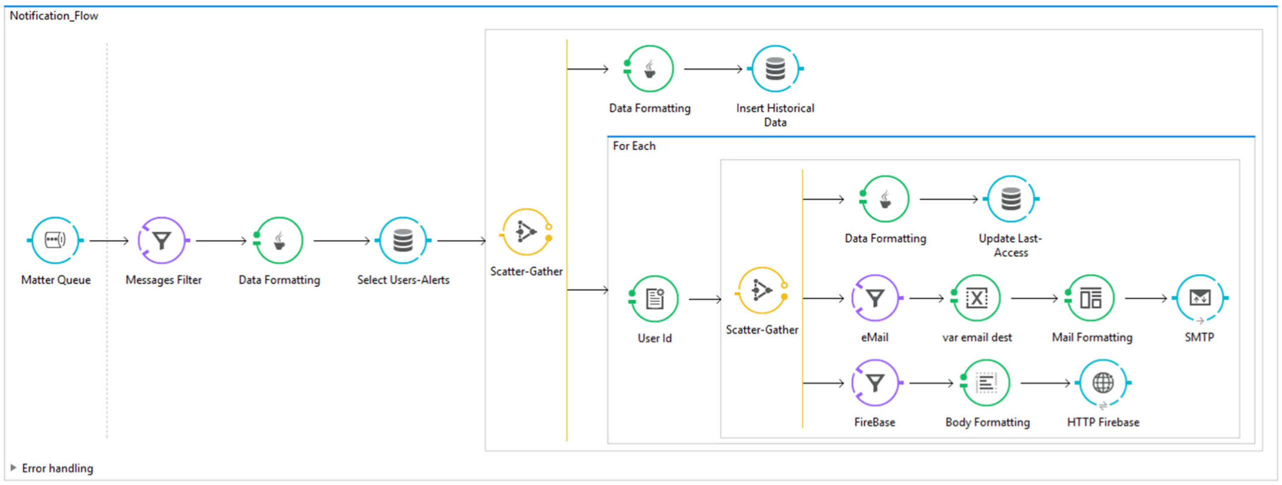


FIGURE 10. **CARED-SOA Flow 7: Notification.**

B. CARED-SOA FUNCTIONALITY

In a summarized way, this is the behavior of our proposed architecture:

- Initially, event patterns are deployed in the system through flow number 1 (Fig. 4). Additional patterns can be deployed at any time.
- At any time, users can invoke the context REST services in flow 2 in order to register in the system, to provide context data, update their subscriptions, et cetera (Fig. 5). The context broker is in charge of (1) reasoning about the user context and recommending which notifications to subscribe for and (2) updating the user context and his subscriptions as they change.
- Equally, users can access the system to check domain event information invoking the domain REST service.
- Simple events reach the system along its lifecycle through the implementation of both the pull and push model:
  - Through the pull model (Fig. 6): Flow 3 checks the channels from which the information has to be obtained, requests are sent to the channels in question and data are obtained.
  - Through the pull model (Fig. 7): data messages enter the system through the subscription to a message topic in the message broker, as shown in Flow 4.
- Then, data obtained from ThingSpeak and the message broker are filtered (according to the events of interest) in Flow 5. Besides, filtered data are transformed into the suitable data for their storage in the database and their submission to the CEP engine (Fig. 8).
- Afterwards, data are stored in the database and sent to the CEP engine. The latter processes all received events; when a pattern is matched it is sent to Flow 6 (Fig. 9). In this flow, relevant events are enriched with the timestamp (included for evaluation purposes) and sent to the relevant notification queue.
- When a new message reaches the notification flow, Flow 7 in Fig. 10, not relevant events are filtered and data is formatted to query the database in order to obtain the users to be notified; then, notifications are sent, and both the event and sent notification are stored in the database.

V. CASE STUDY

IoT applications for health care are taking great relevance nowadays [25]; 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 [26]. This is why the whole society is becoming more interested in this topic, paying extra attention to air quality. Indeed, air quality monitoring is a fundamental issue to be tackled by the whole society in general, and a representative sample of citizens (such as elders and children, people doing physical exercise outside and those who have certain types of lung disease) in particular.

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; we need to guarantee that citizens can easily be made aware of this information and that this information is kept updated in real time. Furthermore, if we want the risk groups to be aware of their particular dangers and how they should act according to the different risk levels, we should provide them with the information and vehemently encourage users to act upon it. We should not forget that physical exercise being done also increases the risk of being affected by unsuitable air quality.

We firmly believe that our proposed context-aware architecture provides a solution to this scenario, as we will explain in the following subsections, where we examine the case study requirements, we explain how air quality is measured by organizations, define the specific context for the case study and explain how the architecture —named Air4People for this scope— is particularized for this case study.

A. AIR4PEOPLE REQUIREMENTS

For air monitoring and notification systems to be effective concerning the use citizens make of such information, these systems have to fulfill the following requirements: (1) air quality information and alerts have to be updated in real time; (2) the information has to be actively provided to citizens in a clear and user-friendly way; (3) the information provided to users, in particular to risks groups, needs to be adapted to their specific characteristics and (4) the system should also take into account the type of activity the user is going to be engaged in and adapt notifications accordingly.

On the other hand, for the system to be technologically efficient it should ensure it meets the needs that follow: (1) the system should be able to tackle a big amount of data coming from several sensors in order to be able to cover as much territory as possible; (2) it should be able to process data coming from sources which might belong to third parties so that we again can cover the largest possible area; (3) the system must be aware of citizens' particular characteristics and physical activities if it is to provide appropriate information and notifications, not relying on end-user devices for this task; and (4) the information and notifications have to be provided to end users in a way that is not hugely time or battery consuming.

To fulfil these goals we have analyzed the needs for the system to be effective and efficient and we have focused on ensuring that the architecture meets the following requirements:

1. First of all, the system must be able to read air pollutant concentrations from diverse heterogeneous data sources. The dataset should be extensible and user-adjustable. This is key for any IoT system to remain in time [27].
2. Secondly, it must have up to date information about which quality of air levels are to a greater or lesser extent harmful for the general public and for specific risk groups.
3. Thirdly, the system, based on the incoming pollutant concentration data, must detect the air quality levels according to a standard in real time, and has to generate and notify the resulting alerts immediately.
4. Besides, it must allow users interested in real-time air quality information to register in order to be notified in case of danger to their health.
5. Furthermore, the system must be aware of user context, based on location, type of physical activity and personal characteristics.
6. In addition, personalized notifications must be sent to users based on their context.
7. Finally, it also has to let users check historical data about pollutant concentration levels for any required purpose: statistics, medical information, environmental issues, et cetera.

B. AIR QUALITY MEASURING

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). Each air quality level belongs to a value range, for instance, the US AQI [28] establishes a 51-200 range for moderate air quality level. However, a different range for the same level is set by other indexes, such as a 4-6 range in the UK DAQI [29].

Although Air4People could be used in conjunction with any of these ranges, for illustration and comprehension purposes we have decided to use the one provided by the US Environmental Protection Agency (EPA) as our air quality level classification [28], [30]. 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. The AQI general air quality level will be determined by the highest level given by a single pollutant. That is, if O_3, CO, NO_2, $PM_{2,5}$ and PM_{10} show, for example, levels lower than 4 (we have enumerated them for convenience), but SO_2 level is 4; then the general air quality level is regarded as 4 (Unhealthy). For example, AQI level 4 information is shown in Table 1.

TABLE 1. AQI Categorization [28].

AQI Values	Levels of health concerns	Health Concerns	Action to Protect your health
Level 4 (151 to 200)	Unhealthy	Everyone may begin to experience health effects. Members of sensitive groups may experience more serious health effects.	The following groups should avoid prolonged or heavy exertion: • People with heart or lung disease. • Children and older adults. Everyone else should reduce prolonged or heavy exertion.

Nevertheless, restricting our approach to the general AQI index would be a poor approximation. We therefore propose to follow the control of every air pollutant whose concentration is relevant to citizen health. Depending on the concentration of each pollutant, citizen health might be affected in a different way. Table 2 shows, as an illustration,

TABLE 2. AQI reference for Ozone values in 8-hour periods [30].

AQI Values	Ozone (ppm) [8 hours]	Values and Health Concerns
Level 4 – Unhealthy (151-200)	Values	0.096-0.115
	Health Concerns	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.

level 4 Ozone values in 8-hour periods and how they affect several risk groups as well as the general public [30].

C. AIR4PEOPLE CONTEXT DEFINITION

As mentioned in the background section, depending on the scope the context may embrace different sets of characteristics [11]. The context for Air4People will be the particular *location*, *personal* characteristics and *physical* activity of the user in question. This context will be used with the aim of personalizing the alerts submitted to users. Let us explain the three types in detail.

1) LOCATION CONTEXT

When the user registers on the system, he does it for a particular location (for instance, the city where he lives and works, if this is the case). Besides, he may choose for his location to be monitored: using the GPS, the mobile device will know the user's location and will be continuously sending it to the system, which will update it in the database. The user will be made aware that this option consumes more battery (we plan to include low battery consuming alternatives in the future).

2) PHYSICAL CONTEXT

On the other hand, the user might be interested in adapting the notifications to his current activity. We have mentioned in Section 2.3 that poor air quality affects people doing physical exercise outside more seriously. Therefore, it would be important to know if the user is, for instance, running or making some sort of effort (for instance, a construction worker would be doing physical activity most of his workday). For this purpose, the system provides two ways of adapting notifications to physical behavior:

- One way would be detecting the speed of the user. Depending of the speed we can guess if (a) the user is motionless or walking calmly, (b) he is running or biking or (c) he is on a public transport. Only for option (b) would notifications reach a higher level.
- The second alternative would be to establish a fixed timetable: an hourly schedule is provided by the user for every day in the week, so that he can register his routines in it. For instance, from 8 to 9, running outside (effort); from 9 to 17, at the office (no effort); from 17 to 19, shopping (no effort); from 19 on, at home (no effort). In the case of the construction worker, he would mark effort during his work hours. This option is very convenient for people with fixed habits since it saves a lot of battery too.

TABLE 3. Types of context supported in Air4People.

Type of Context	Location Context	Personal Context	Physical Context
Static	Provided during registration or updated through the mobile app.	Age, lung and heart diseases, genetic variants and limited diet.	Schedule provided when registering or updated through the mobile app.
Dynamic	Obtained through the mobile GPS.	-	Speed obtained from the mobile GPS.

3) PERSONAL CONTEXT

This type of context would consist of the following personal characteristics:

- Lung disease: right now we consider all lung diseases to be the same, but we will categorize them in the future depending on their severity and exacerbation due to poor air quality levels. We are currently working with a pulmonologist on this issue.
- Heart disease: right now we place all heart conditions within the same category, but we will categorize them in the future depending on their severity and exacerbation due to poor air quality levels.
- Children (including teenagers): we will regard everybody younger than 19 years old as being included in this category [31].
- Older people: we will regard everybody older than 60 years old as being included in this category [32].
- Genetic variants: there are several articles which link genetic variants and poor air quality exposures to an increase in certain illnesses (such as [33]). AQI index provides information for genetic variants in general and so does Air4People.
- Diets limited in Omega-3 and vitamins. Even though AQI only specifies ''diets limited in certain nutrients'', according to several studies it seems significant that Omega-3 and certain vitamins are key for preventing health risks derived from air pollution [34]. We should bear in mind that diet specifications can be varied in the systems at any time according to current health recommendations.

Table 3 summarizes the current contexts supported by Air4People, classified according to whether they are static (it needs not be continuously monitored and therefore the user sends it to the system during registration or occasionally updates his data) or dynamic (they might require continuous monitoring).

D. PARTICULARIZATION OF CARED-SOA FOR THE CASE STUDY

CARED-SOA was particularized for the case study as follows:

1) DATABASES

Domain databases store information about air quality stations, their sensors and pollutant concentrations detected in the sensors, as well as detected air quality levels.

- With this database we can identify every air quality station, and for each station which type of sensors it has, as well as the data required to connect to the station data source.
- It also provides a mechanism to identify the same type of sensors with several synonyms; for instance, Temperature could be found with the following identifiers: *temperatura* (in Spanish), *temp*, *t*, or any other word chosen by the data provider. The system will manage all these identifiers as a unique one, transparently to the final user as well as to the data provider.
- Besides, it stores all the information about each reading including the timestamp, as well as detected air quality levels.

Context and notification information are also stored in a database. We store:

- The registration details (user, password, et cetera).
- The information related to personal context, physical context and location: static location and physical context can coexist with dynamic ones. The former are stored at registration and modified occasionally and the latter are updated continuously when the user has activated this option. In such cases, only dynamic information is taken into account for context awareness; otherwise the static one is.
- All types of notification which the users can register to, as well as the corresponding health warnings and suggestions.

2) REST WEB SERVICES

Context web service provides operations to read the information about available subscription alerts and to post and update information about users, their context and their subscription. In particular, personal context, fixed location and fixed physical schedule are provided during registration and monitored location and activity is provided constantly based on a predefined period of time.

Domain web service provides operations to be able to read all the available information about every air quality station, sensor and measurement (historical data). This is used by the system to check which channels have to be queried, which information we obtain from each air quality station and the last reading from the station. Besides, it is also used by the mobile application to obtain information about the stations and their sensors and about the current or historical air quality in a given location.

3) DATA SOURCES

In this case, we have used an emulator for the pull model; the emulator sends data to ThingSpeak and the system reads the data from it. For push models we have used the data coming from the Andalusian government which is obtained from a message queue as well as an emulator data for pursuing performance tests.

4) COMPLEX EVENT PATTERNS

As previously mentioned, the CEP engine analyzes the data coming from the air quality stations and detects poor air quality thanks to a set of predefined and pre-deployed patterns. For pattern definition, we have followed air quality indexes defined by the EPA [28], [30].

We have defined patterns for each level in which health might be affected. For instance, for Ozone, in eight-hour periods we have defined the patterns for levels 2, 3, 4, 5 and 6 with the ranges indicated for each level by the EPA (see [30]). Please note that level 1 depicts good air quality and therefore there is no need to send any alert for such a condition and thus no need for the corresponding pattern to be defined. For every pollutant, the patterns from levels 2 or 3 to level 6 have been defined (ranges and levels can be consulted in detail in the EPA documents).

For the sake of illustration, code listing 1 shows a pattern for Level 2 Ozone in eight-hour periods.

Code Listing 1. EPL code for a pattern detecting Ozone Level 2 in eight-hour periods.

```
(1) @Name("Ozone2")
(2) insert into AQILevels
(3) select
(4)    1 as alertKind, 2 as alertLevel,
(5)    e.ChannelId as ChannelId, e.ChannelName as ChannelName,
(6)    e.timestamp as timestamp,
(7)    e.ChannelLat as ChannelLat, e.ChannelLong as ChannelLong,
(8)    e.T as T, e.RH as RH, e.CO2 as CO2,
(9)    e.O3 as O3, e.PM25 as PM25, e.PM10 as PM10,
(10)   e.CO as CO, e.SO2 as SO2, e.NO2 as NO2
(11) from
(12)   pattern [every-distinct (e.ChannelId)e = AirEvent(
(13)   (O3 >= 0.060 and O3 <= 0.075))].win:time (8 hour)
```

Line 1 assigns a name to the pattern; line 2 inserts in a flow, called *AQILevels*, the new complex event created from the single events; lines 3 to 10 establish the values for the type and level of alert, and also save channel, timestamp and pollutant values from the single events in the complex event; lines 11 to 13 describe the condition to be met to match the event pattern. It is noteworthy that even though patterns can be manually coded by computer scientists, this can also be accomplished by air quality experts using the multi-domain graphical editor provided in [35].

Once a pattern is matched, the air quality level detected is stored in the domain database and notified to the context reasoner.

5) CONTEXT KNOWLEDGE MANAGER

As explained, the context database has been particularized with the required fields for the case study and the operations offered in the REST service allows us to introduce the context and notifications' information.

6) CONTEXT REASONER

We defined a Java class which establishes the relations between the types and values of context and suitable notifications based on the detected air quality level. For instance, for O_3 in level 4, outdoor workers will be advised to subscribe to this notification event if they do not have any lung condition.

7) CONTEXT-BASED ADVISER

As previously mentioned, whenever an unsuitable air quality level is detected in the CEP engine, it is managed by the context-based adviser. The latter will have to check:

- Which users are subscribed to this type of notification.
- Which users are in the location where the alert was detected.
- What type of notification the user subscribed to: e-mail or mobile notification.

For this case study, we will have a federation of context-based advisers where each level of each pollutant has its own adviser (and therefore we have a notification queue for each of them).

8) SOFTWARE CLIENTS

Regarding the software clients, we have developed a mobile app for testing purposes from which current and historical air quality data can be consulted and where mobile notifications can be selected and received.

Additional clients (apps, websites or other application types) can be built without restrictions; they would only have to invoke the REST service operations.

VI. EVALUATION

We have conducted different validation procedures in order to test all the relevant characteristics of the system. As explained in the following subsections, we have used an emulator to send data to ThingSpeak in order to test the pull model. In addition, we have tested the system with real data coming from the sensor motes deployed all over the Andalusian territory and we have also used the emulator to submit several simulated third parties through the pull model.

In all cases the architecture was deployed in a workstation with i3-540 (4M cache, 3.6 Ghz), 8GB of RAM memory and SATA2 hard drive. The notification queues were located in a virtual machine with 8GB of RAM memory and a hard drive of 22GB. Communications between both computers were fulfilled thorough a virtual private network belonging to the University of Cádiz. For testing purposes, the user database was populated with 150 users with different contexts, that is, different personal characteristics, locations and schedules and we tested physical and location monitoring with an average mobile device (Android 4.1.2). The alert database (to which users may subscribe) is populated with the previously explained alert types defined by the EPA. The mote database is populated with the characteristics and sensors of the motes we are going to simulate with the emulator.

A. TESTING THE PULL MODEL WITH AN EMULATOR

We tested the system with both a physical mote providing the sensors in question and third-party sensors, but due to the lack of possible scenarios using current public values of air quality we decided to create an emulator which allows us to test the system with several random situations as well as manually specified particular ones.

Some of the main characteristics of the emulator[1] are described as follows:

- It can generate pseudorandom values for several motes and sensors related to air quality and send it to different channels in ThingSpeak.
- When the emulator is launched, we can add as many motes as we wish.
- A new mote is created with a set of predefined default sensors, which can be modified as required.
- At any point in time, we can activate or deactivate data submission to ThingSpeak, as well as Save and Load a mote configuration.
- We can also manually introduce data at any time, in case we want to test a particular situation.

We tested several random situations as well as some particular ones manually using the emulator. As previously explained, we sent the data to ThingSpeak to emulate third-party data. A capture of some of these data in ThingSpeak is shown in Fig. 11.

The implemented event patterns for detecting air quality levels, whose syntax was previously checked using the Esper EPL Online tool [36], were correctly detected by Air4People and their corresponding notifications were received by e-mail and through our mobile application.

B. TESTING THE SYSTEM WITH REAL DATA FROM THIRD PARTIES

The system was also tested with real data coming from 95 sensor stations the Andalusian regional government has all over its territory. These stations provide a total of 708 sensors with 89600 data registers per day. On average, this is translated into receiving around 62 sensor measurement per minute in the message queue (stations submit data in 10 minute intervals approximately). The system perfectly supported this load of events and responded in less than a millisecond with corresponding complex events detected as well as their corresponding notifications.

Real and simulated data were checked through the ESB console and database updates, so that it was confirmed that the notifications were sent according to the events entering

[1]Additional information showing the potential of the emulator to generate multiple sensors, motes and situations can be found at http://hdl.handle.net/10498/18582. We have not included it in the paper since we did not consider it relevant for the paper objectives.

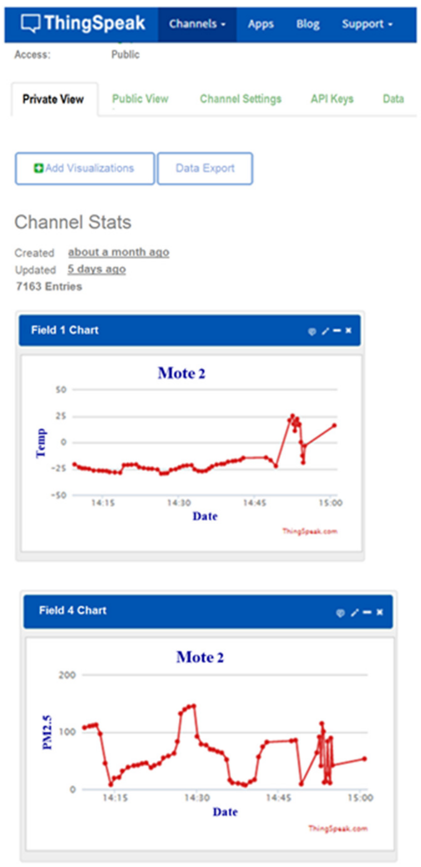


FIGURE 11. ThingSpeak emulated channels.

TABLE 4. Configurations tested in Air4People.

Configuration	DB	Patterns	Windows	Queues
Conf. 1	No	7	Sliding	1
Conf. 2	No	All	Sliding	All
Conf. 3	No	All	Sliding	1
Conf. 4	No	All	Batch	All
Conf. 5	Yes	All	Sliding	All

TABLE 5. Air4People performance test results.

Configuration	Test Set	1	2	3
Conf. 1	Total events submitted	959023	1872412	3800751
	Events per minute submitted	13700	26749	54296
	Relevant events detected	115105	205170	509098
Conf. 2	Total events submitted	324519	636130	958991
	Events per minute submitted	5003	9087	13700
	Relevant events detected	277327	543624	814221
Conf. 3	Total events submitted	959000	1400001	1862474
	Events per minute submitted	13700	20000	26607
	Relevant events detected	112171	163753	229746
Conf. 4	Total events submitted	959017	1882588	3748064
	Events per minute submitted	13700	26894	53543
	Relevant events detected	2198	2448	3805
Conf. 5	Total events submitted	355019	626255	959035
	Events per minute submitted	5000	8820,7	13700
	Relevant events detected	312239	550791	806696

into the system and database user contexts. Thanks to the timestamp stores, we could also verify how fast the complex event was detected after the simple event was received in the system.

In addition, dynamic physical and location contexts were tested with one mobile device. The alerts were received according to the context detected. Static contexts were tested modifying the data in the database. In all cases, the alerts were received according to the context detected.

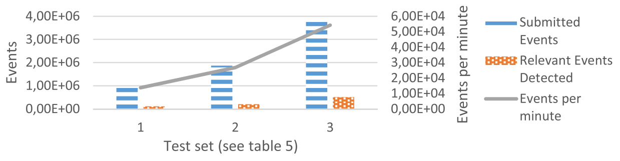
C. TESTING THE PULL MODEL WITH THE EMULATOR

If we plan to extend the system for all government air quality stations around Spain, we calculated an average number of 500 sensor measurements per minute. The load test was conducted with 1000 simulated measurements per minute. The system continued working perfectly.

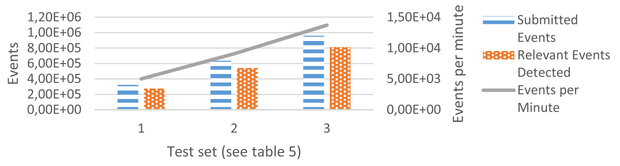
In order to see how the system responds depending on several variable parameters, we have made a set of performance and stress tests with the different configurations. The configurations tested are included in Table 4 and consist of (a) storing or not the simple events in the database, (b) using the defined patterns for all pollutants (47 patterns) or only for one (the 7 patterns defined for NO_2), (c) using sliding windows versus batch ones and (d) having a notification queue for every level in every pollutant or a single one for one particular level of a pollutant (discarding the remaining notifications). It is important to clarify that *batch* windows have been set with reduced times —15 minutes— to test the system with a high number of detected complex events (if we set them to 1 hour or more, the cadence for detected events would be very low).

Performance tests have been carried out according to these configurations; these have been performed over 70 minutes —enough time to see the effect of 1 hour windows— with an estimated number of events per minute to find the system's limit.

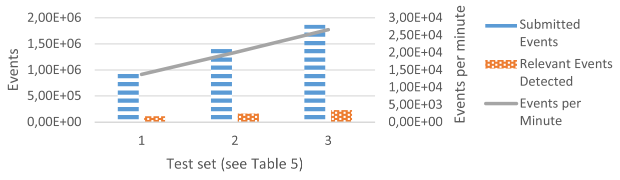
Each configuration has been tested with several rates of events per minute sent to the system and the latter's behavior was observed for each test. If the system responded properly, then the test was performed with a higher speed, searching the highest speed at which the system still responds appropriately. For illustration purposes, we have selected three different representative speeds of events per minute sent to the system (test sets 1, 2 and 3). Such selected tests represent two speeds at which the system responds properly (sets 1 and 2), to see the amount of events increase, and a third speed where the system still responds but close to the speed where the system starts becoming deteriorated. Table 5 and charts in Fig. 12 show the values of the total number of events submitted to the system in the 70 minutes the test lasted, the received events per minute and the number of complex events detected. Please remember that the events submitted to the system are the *simple* events that the emulator generates, which entry the system and are processed by the CEP engine; and the relevant events are complex events detected in the CEP engine which provide valuable information in the scope of the case study. This information is shown for each configuration for the three test sets.



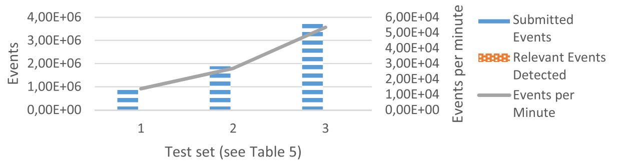
(a) Performance test for configuration 1



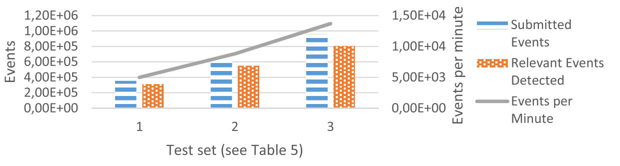
(b) Performance test for configuration 2



(c) Performance test for configuration 3



(d) Performance test for configuration 4



(e) Performance test for configuration 5

FIGURE 12. **Air4People performance tests results. (a) Performance test for configuration 1. (b) Performance test for configuration 2. (c) Performance test for configuration 3. (d) Performance test for configuration 4. (e) Performance test for configuration 5.**

In configuration 1, we can see how the system still responds at 54,296 events per minute, in configuration 2 at 13,700 events per minute, in configuration 3 at 26,607 events per minute, in configuration 4 at 53,543 events per minute and in configuration 5 at 13,700 events per minute. We can observe that the system admits a higher speed for configurations 1 and 4. The main reason, on the one hand, is that we have reduced the number of patterns to a sixth in configuration 1, so the system's resources needed to keep the pattern matching procedure by the Esper engine is considerably reduced. On the other hand, patterns in configuration 4 have been defined with batch windows with the objective of validating how this type of windows might improve the performance versus sliding ones. Configurations 2 and 5 stop working properly earlier since they are tested with all the patterns and notification queues, therefore having to deal with a higher number of complex events. Finally, configuration 3 has an intermediate limit since, even though all the patterns

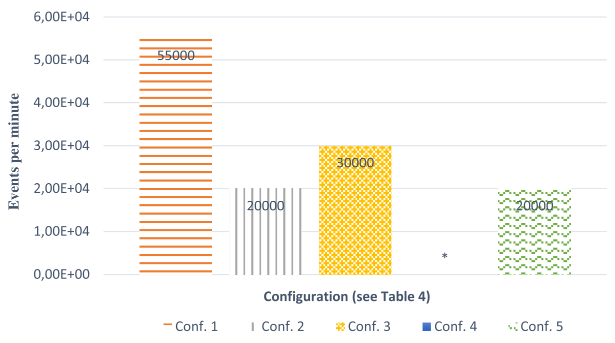


FIGURE 13. **Estimated limit for events per minute entering Air4People.**

are deployed in the CEP engine, only one queue is managed for notification (which implies one sixth of notifications approximately versus configurations 2 and 5).

We also performed stress tests: the chart in Fig. 13 represents the estimated limit load for the system for every configuration based on the stress tests results. We have called the *limit* of the system to refer to the entering speed of events per minute with which the system collapses and cannot attend such amount of entering data. With such an entrance of events per minute, we can estimate that the system will not respond appropriately, suffering delays and probably losing data due to bottlenecks. In configuration 4, we could not find a limit when using batch windows.

We have also analyzed the average response time and number of relevant events detected at one speed accepted by all configurations during 70 minutes: 13,700 events per minute; results are shown in Table 6.

TABLE 6. **Air4People response to an entrance of 13,700 events per minute during 70 minutes.**

Configuration	Average response time (seconds)	Relevant Events Detected
Conf. 1	0	112171
Conf. 2	0.002	814221
Conf. 3	0.001	112171
Conf. 4	57	2198
Conf. 5	186	806696

At this speed, we can see in Table 6 that configuration 1 detects events in less than 1 millisecond, configuration 2 in 2 milliseconds, configuration 3 in 1 millisecond, configuration 4 in 57 seconds and configuration 5 in 186 seconds. These response times are acceptable for the current case study; times for configurations 1, 2 and 3 are clearly acceptable; the response time for configuration 4 is not relevant since batch windows imply that even though the event takes place before the end of the window, such event is not notified until the window is closed; that is, if we have a 1-hour batch window and the event takes place at minute 15, it will not be notified until 1 hour has gone by. Finally, response time for configuration 5 is notably higher for this event speed, but still acceptable for this case study.

Comparing all the configurations, we can conclude that when we introduce all the patterns the system support a lower load of events per minute, since the sliding windows overhead the system. This is why the developer must choose carefully the patterns for the domain in question and follow Esper performance recommendations [37].

In Table 6 we can also see the total number of relevant events detected per configuration with an entrance of 13,700 events per minute and a test over 70 minutes: configurations 2 and 5 are those which detect a higher number of events (since they have all the patterns and queues); configurations 1 and 3 detect much fewer events (given that they only detect events related to one pollutant); configuration 4, since it is using *batch* windows, detects an insignificant number of events compared to the others.

D. RESULTS DISCUSSION

As a limitation of the proposed approach, we have seen that at a certain amount of events per minute entering the system, the architecture might collapse. The threshold for such a bottleneck will depend on the particular case study and, as we have seen, on the system's particular configuration. It is important to highlight that the emulator has been programmed to foster a high number of event detections, which most probably would not happen easily in the case study with real data: the objective was to search for the system's limit, and we would expect the system to perform better in real situations. Therefore, due to the results obtained in the tests we consider that CARED-SOA is highly scalable, moreover bearing in mind that we can follow a set of actions to improve its performance and scalability if the domain required it, such as:

- Distributing diverse components of the system in different machines (such as databases, notification modules, et cetera).
- We can also distribute patterns in different machines, either patterns working with different data, or through a hierarchy of events (where a machine can perform initial filtering).
- Using high-speed communication networks, depending on the response times required.
- Of course, if we think of scaling the system, for instance, to Europe more powerful machines could be used.
- Equally, it would be more appropriate to use Enterprise distributions of the software (ESB and CEP engine), rather than Community ones, since they guarantee better performance results.

Of course, distributing the different components of the system in several machines will imply further communications with their consequent security and reliability issues. Besides, distributed databases involve the consequent consistency and partition tolerance issues. The software engineer will have to balance the processing requirements versus the distribution of the databases or other architecture components according to the particular system's available resources and requirements.

To conclude, we could test and validate that (1) the system responds in real time to the incoming sensor data; (2) third-party data (as those coming from the Andalusian regional government) were perfectly integrated into the system; (3) the mobile application was receiving the notifications in due time; (4) the push model is very appropriate for the architecture and was used to integrate the Andalusian regional government data; (5) alert levels were launched when the sensors data were in the specified interval during the corresponding time windows, as well as risk groups being alerted in those cases; (6) REST API is currently providing all the data received and detected in the system as well as providing the user the chance to subscribe to the alerts; (7) dynamic physical and location contexts were properly received in the system, though consuming more battery than expected (tackling this issue is part of our future work).

VII. RELATED WORK

There are multiple approaches for context adaptation in different computer science domains [38]: 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. In the following paragraphs, some outstanding approaches related to the most relevant techniques are presented.

Some prominent frameworks and middlewares for context awareness can be mentioned: CoWSAMI is a middleware which gives support to context, supplied by Athanasopoulos *et al.* [39]. It provides a Context Manager in order to deal with context sources. Services here are only used as interfaces to collect information. CAS-Mine is a framework presented by Baralis *et al.* [40], which focuses on discovering relevant relationships between user context data and invoked services. CAS-Mine extracts generalized association rules, which provide a high-level abstraction of both user habits and service characteristics, depending on the context. However, it does not provide an adaptation to context mechanism. The ESCAPE framework and the inContext project deserve a special mention. Both are proposed by Truong *et al.*, in [41] and [42], respectively, and deal with a service-based context-management system for team collaboration. Even though the concepts provided for exchanging and making use of context information in a service-based scope are high quality ones, the narrow scope of their main focus (collaboration teams) weaken the value of this proposal for the scope of this paper.

The paper from Gilman *et al.* [43] also deserves special attention. They provide a framework for context-aware pervasive services to context through a complex architecture composed of several components, among them a context reasoner, context discoverer and observers, handlers and managers. Their framework implies that context providers and actuators are designed specifically for their low level context ontology and suffer from communication delays. The paper from Li *et al.* [44] is also of interest: they provide a framework for context provisioning, which can include new context provisioning schemas dynamically. They have also defined a DSL to facilitate the definition of new context schemas for additional application domains. There are two aspect-oriented approaches which can also be mentioned: Yahyaoui *et al.* focus on adaptable web services, providing a new Web Service Policy Language for context specification and with the main aim of using it for BPEL compositions [45]; Yu *et. Al.* present PerCAS, a model-driven approach which enables web service adaptation to user personal context preferences or user personalization [46]. In any case, none of these works take advantage of the use of the ESB and CEP, which leverages the context-aware system's usability and maintenance.

Some works integrate CEP and SOA or use ESBs to follow some adaptation or provide context awareness. For instance, Taher *et al.* [47] 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 to SOAP messages and then they are sent to web services. CA-ESB is presented by Chanda *et al.* as a context-aware ESB [48]; 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 when pursuing a service orchestration the closest services are selected.

Most of these proposals are focused on a unique aspect of context awareness: some of them on the modeling phase, other 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, providing the means for context dealing from reception to delivery of personalized context-aware services.

If we pay special attention to the case study presented in the paper and therefore to particular approaches for air quality notification, there are a few that could be mentioned; we have selected three relevant ones. CITI-SENSE [49] is one of the most thorough EU projects for monitoring air quality in several cities (Barcelona, Belgrade, Edinburgh, Haifa, Ljubljana, Ostrava, Oslo, Vienna and Vitoria). Mainly, it consists of a solution that integrates different components as follows: (1) a web portal that provides an access point to all its sensors and apps, tools and questionnaires; (2) a personal air monitoring toolkit that allows us to assess air quality, monitoring only three gases (NO_2, NO and O_3); (3) a city air smartphone app that allows us to share our perception of air quality and (4) an on-line air quality perception questionnaire to be used in campaigns to assess citizens' perception of air quality so as to obtain feedback. OpenSense [50] is a research project with the purpose of monitoring air pollution by using wireless sensor network technology. Sensors have been deployed in public transportation (trams in Zürich and buses in Lausanne) to measure CO, CO_2, NO_2, O_3, fine particles, temperature and humidity. OpenSense II [51] is an improved version of the project OpenSense in which sensor nodes have been updated with GSM, GPS and UFP and deployed in electric cars and bicycles. In this new version, citizens can contribute data collected or generated from their mobile devices. SmartSantander [52] is a EU project that proposes a huge infrastructure with around 20,000 sensors deployed in public transportation vehicles to monitor air pollutants CO, CO_2 and NO, as well as temperature and humidity in different cities: Lübeck, Guildford, Belgrade and Santander. This platform includes a REST API that allows developers to query the last air-pollutant-related observation from any node in the infrastructure equipped with air pollutant sensors, all air-pollutant-related observations within the boundaries of selected dates, and within a particular range of kilometers from a given position between the boundaries of the selected dates.

In the context of air quality monitoring, we summarize the most relevant features required in the proposed systems; we will then examine if the mentioned proposals fulfill these features:

- Real Time: the system provides real-time monitoring of air quality.
- Third Party: the system allows the integration of third-party sensors easily.
- Push: the system provides a mechanism to receive notifications without the user having to check the system constantly.
- Alert Levels: the system distinguishes among several alert levels according to pollutant concentrations.
- Risk Groups: the system distinguishes among several alert levels according to the risk group the user belongs to.
- REST API: the system offers a REST API so that anybody can use it to obtain information about air quality in his own software client.
- Physical Context: the system takes into account the activity the user is carrying out when notifying the unsuitable air quality detected.

Most of the presented proposals provide real time information and even push notification, but they lack some other features. Firstly, in CITI-SENSE platform there is no possibility of easily integrating air quality data with third-party services since there is no REST API available for users; data have to be downloaded in CSV format or directly into an Excel spreadsheet. Another limitation of this project is that air quality alerts are displayed as an APIN (Air Pollution Indicator) value that, although related to EU CAQI (Common Air Quality Index), it cannot be directly compared to this EU index. In addition, this information is not particularized according to risk groups. Secondly, in OpenSense air quality notifications are not customized based on risk groups and the documentation and resources on the website are limited, and it is not clear how to integrate it with third-party applications and how to access/monitor air quality data. Finally, SmartSantander, as we have already mentioned, includes a REST API; however, air quality levels and air quality notifications are neither calculated nor sent to risk groups. Thus, personalized health recommendations are not provided to citizens.

Regardless of other differences with our presented approach, the most remarkable issue is that none of these approaches, which are relevant European projects in the domain in question, takes into account the context of the user which is key when notifying him; therefore, their notifications are not customized based on risk groups and personalized health recommendations are not given, what is key in this domain.

VIII. CONCLUSION

In this paper, we have proposed an event-driven SOA which provides context awareness in the scope of IoT. The architecture benefits from key elements to cover all stages of data processing [15]: it collects and adds data from heterogeneous sources; agent communications are facilitated thanks to the use of message queues and brokers; IoT received data are stored in the databases and processed by a CEP engine; users can benefit from the processed data through the invocation of REST services or through the subscription to notifications. Therefore, this is a holistic architecture which permits dealing with context awareness in SOAs, providing the means for context dealing from reception to delivery of personalized context-aware services.

In our future work, we plan to extend our architecture for collaborative scopes, where several nodes in the architecture can exchange relevant contextual information based on detected complex events.

ACKNOWLEDGMENT

The authors 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 C. Maza. They are also grateful to researchers W. Lamersdorf and S. Reiff-Marganiec for their interest in our ongoing research projects.

REFERENCES

[1] M. Papazoglou, *Web Services and SOA: Principles and Technology*, 2nd ed. New York, NY, USA: Pearson Education, 2012.

[2] T. Erl, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[3] F. R. Thomas, *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, CA, USA: Univ. California, 2000.

[4] L. Richardson and S. Ruby, *RESTful Web Services*. Farnham, U.K.: O'Reilly, 2007.

[5] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[6] D. C. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. Hoboken, NJ, USA: Wiley, 2012.

[7] M. P. Papazoglou and W. V. D. Heuvel, "Service-oriented design and development methodology," *Int. J. Web Eng. Technol.*, vol. 2, no. 4, pp. 412–442, Jul. 2006.

[8] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, and S. Dustdar, "Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems: Event-based monitoring and adptation for distributed systems," *Softw. Pract. Exp.*, vol. 44, no. 7, pp. 805–822, Jul. 2014.

[9] A. K. Dey, "Understanding and Using Context," *Pers. Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, Jan. 2001.

[10] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," presented at the 1st Int. Symp. Handheld Ubiquitous Comput., Karlsruhe, Germany, 1999, pp. 304–307.

[11] U. Alegre, J. C. Augusto, and T. Clark, "Engineering context-aware systems and applications: A survey," *J. Syst. Softw.*, vol. 117, pp. 55–83, Jul. 2016.

[12] R. Hervás and J. Bravo, "COIVA: Context-aware and ontology-powered information visualization architecture," *Softw. Pract. Exp.*, vol. 41, no. 4, pp. 403–426, Apr. 2011.

[13] S. Peinado, G. Ortiz, and J. M. Dodero, "A metamodel and taxonomy to facilitate context-aware service adaptation," *Comput. Electr. Eng.*, vol. 44, pp. 262–279, May 2015.

[14] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[15] R. Buyya and A. V. Dastjerdi, *Internet of Things: Principles and Paradigms*. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[16] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "Approaching the Internet of Things through integrating SOA and complex event processing," in *Handbook of Research on Demand-Driven Web Services: Theory, Technologies, and Applications*, Z. Sun, J. Yearwood, Eds. Hershey, PA, USA: IGI Global, 2014, pp. 304–323.

[17] MuleSoft. *What is Mule ESB?* accessed on Jan. 3, 2017. [Online]. Available: https://www.mulesoft.com/resources/esb/what-mule-esb

[18] Oracle Corporation. *MySQL*, accessed on Jan. 3, 2017. [Online]. Available: https://www.mysql.com/

[19] Apache Software Foundation. *ActiveMQ*, accessed on Jan. 3, 2017. [Online]. Available: http://activemq.apache.org/

[20] (2016). *ThingSpeak*, accessed on Jan. 3, 2017. [Online]. Available: https://thingspeak.com/

[21] Oracle Corporation. *JERSEY. RESTful Web Services in Java*, accessed on Jan. 3, 2017. [Online]. Available: https://jersey.java.net/

[22] IETF Network Working Group. *Simple Mail Transfer Protocol*, accessed on Jan. 3, 2017. [Online]. Available: https://tools.ietf.org/html/rfc5321

[23] Google. *Firebase*, accessed on Jan. 3, 2017. [Online]. Available: https://firebase.google.com/

[24] MuleSoft. *Flows and Sub-Flows*, accessed on Jan. 3, 2017. [Online]. Available: https://docs.mulesoft.com/mule-user-guide/v/3.7/flows-and-subflows

[25] S. M. Riazul Islam, D. Kwak, M. Humaun Kabir, M. Hossain, and K.-S. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, Jun. 2015.

[26] "Review of evidence on health aspects of air pollution—REVIHAAP project," World Health Organization, Copenhagen, Denmark, Tech. Rep., 2013. [Online]. Available: http://www.euro.who.int/__data/assets/pdf_file/0004/193108/REVIHAAP-Final-technical-report.pdf

[27] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "AHAB: A cloud-based distributed big data analytics framework for the Internet of Things," *Softw. Pract. Exp.*, vol. 47, no. 3, pp. 443–454, 2016.

[28] U.S. Environmental Protection Agency. (2014). "AQI Air Quality Index. A Guide to Air Quality and Your Health," accessed on Jan. 17, 2017. [Online]. Available: https://www3.epa.gov/airnow/aqi_brochure_02_14.pdf

[29] UK Department for Environment Food and Rural Affairs. *Daily Air Quality Index*, accessed on Jan. 17, 2017. [Online]. Available: https://uk-air.defra.gov.uk/air-pollution/daqi

[30] U.S. Environmental Protection Agency. *Technical Assistance Document for the Reporting of Daily Air Quality—the Air Quality Index (AQI)*, accessed on Jan. 17, 2017. [Online]. Available: https://www3.epa.gov/airnow/aqi-technical-assistance-document-may2016.pdf

[31] World Health Organization. *Definition of key Terms*, accessed on Jan. 17, 2017. [Online]. Available: http://www.who.int/hiv/pub/guidelines/arv2013/intro/keyterms/en/

[32] World Health Organization. *Ageing and Health*, accessed on Jan. 17, 2017. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs404/en/

[33] A. Levinsson *et al.*, "Interaction effects of long-term air pollution exposure and variants in the GSTP1, GSTT1 and GSTCD genes on risk of acute myocardial infarction and hypertension: A case-control study," *PLoS ONE*, vol. 9, no. 6, p. e99043, Jun. 2014.

[34] S. Péter *et al.*, "Nutritional solutions to reduce risks of negative health impacts of air pollution," *Nutrients*, vol. 7, no. 12, pp. 10398–10416, Dec. 2015.

[35] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowl.-Based Syst.*, vol. 89, pp. 97–112, Nov. 2015.

[36] EsperTech. *Esper EPL Online*, accessed on Jan. 17, 2017. [Online]. Available: http://www.esper-epl-tryout.appspot.com/epltryout/index.html

[37] EsperTech Inc. *Esper Reference Documentation. Performance*, accessed on Jan. 17, 2017. [Online]. Available: http://www.espertech.com/esper/release-5.3.0/esper-reference/html/performance.html

[38] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, and I. S. Venieris, "Context-aware service engineering: A survey," *J. Syst. Softw.*, vol. 82, no. 8, pp. 1285–1297, 2009.

[39] D. Athanasopoulos, A. V. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis, "CoWSAMI: Interface-aware context gathering in ambient intelligence environments," *Pervas. Mobile Comput.*, vol. 4, no. 3, pp. 360–389, Jun. 2008.

[40] E. Baralis, L. Cagliero, T. Cerquitelli, P. Garza, and M. Marchetti, "CAS-Mine: Providing personalized services in context-aware applications by means of generalized rules," *Knowl. Inf. Syst.*, vol. 28, no. 2, pp. 283–310, Nov. 2010.

[41] H. Truong, L. Juszczyk, A. Manzoor, and S. Dustdar, "Escape—An adaptive framework for managing and providing context information in emergency situations," presented at the 2nd Eur. Conf., (EuroSSC), Kendal, U.K., 2007, pp. 207–222.

[42] H.-L. Truong *et al.*, "inContext: A pervasive and collaborative working environment for emerging team forms," in *Proc. Int. Symp. Appl. Internet*, Turku, Finland, Aug. 2008, pp. 118–125.

[43] E. Gilman, X. Su, O. Davidyuk, J. Zhou, and J. Riekki, "Perception framework for supporting development of context-aware web services," *Int. J. Pervas. Comput. Commun.*, vol. 7, no. 4, pp. 339–364, Nov. 2011.

[44] F. Li, S. Sehic, and S. Dustdar, "COPAL: An adaptive approach to context provisioning," presented at the IEEE 6th Int. Conf. Wireless Mobile Comput., Netw. Commun., (WiMob), Niagara Falls, ON, Canada, Mar. 2010, pp. 286–293.

[45] H. Yahyaoui, A. Mourad, M. Almulla, L. Yao, and Q. Z. Sheng, "A synergy between context-aware policies and AOP to achieve highly adaptable Web services," *Service Oriented Comput. Appl.*, vol. 6, no. 4, pp. 379–392, Jun. 2012.

[46] J. Yu, J. Han, Q. Z. Sheng, and S. O. Gunarso, "PerCAS: An approach to enabling dynamic and personalized adaptation for context-aware services," in *Service-Oriented Computing* (Lecture Notes in Computer Science), vol. 7636, C. Liu, H. Ludwig, F. Toumani, and Q. Yu, Eds. Berlin, Germany: Springer, 2012, pp. 173–190.

[47] Y. Taher, M.-C. Fauvet, M. Dumas, and D. Benslimane, "Using CEP technology to adapt messages exchanged by web services," presented at the Proc. 17th Int. Conf. World Wide Web, Beijing, China, Jun. 2008, pp. 1231–1232.

[48] J. Chanda, S. Sengupta, A. Kanjilal, and K. India, "CA-ESB: Context aware enterprise service bus," *Int. J. Comput. Appl.*, vol. 30, no. 3, pp. 1–8, 2011.

[49] (2016). *CITI-SENSE*, accessed on Jan. 17, 2017. [Online]. Available: http://www.citi-sense.eu/

[50] (2016). *OpenSense*, accessed on Jan. 17, 2017. [Online]. Available: http://www.opensense.ethz.ch/trac/

[51] (2016). *OpenSense II*, accessed on Jan. 17, 2017. [Online]. Available: http://opensense.epfl.ch/wiki/index.php/OpenSense_2

[52] (2014). *SmartSantander*, accessed on Jan. 17, 2017.[Online]. Available: http://www.smartsantander.eu/

ALFONSO GARCÍA DE PRADO received the Ph.D. degree in computer science from the University of Cádiz, Spain, in 2017. For several years, he has been a Programmer, an Analyst, and a Consultant for various international industry partners, focusing on software development, evolution, and management for large sport events. Over the years, he has paid special attention to research within the scope of mobile devices and web services for which he has analyzed the advantages of using model-driven and aspect-oriented techniques for service context awareness and published his findings in several journal papers. His research focuses on trending topics, such as the complex event processing integration in service-oriented architectures and context awareness in the Internet of Things.

GUADALUPE ORTIZ received the Ph.D. degree in computer science from the University of Extremadura, Spain, in 2007. From 2001 to 2009, she was an Assistant Professor and a Research Engineer with the Computer Science Department, University of Extremadura. In 2009, she joined the Department of Computer Science and Engineering, University of Cádiz, as an Associate Professor. She has authored or co-authored numerous peer-reviewed papers in international journals, workshops, and conferences. Her research interests embrace aspect-oriented techniques as a way to improve Web service development, with an emphasis on model-driven extra-functional properties and quality of service, service context awareness and their adaptation to mobile devices, and complex event processing integration in service-oriented architectures. She has been a member of various programs and organization committees of scientific workshops and conferences over the years and is a Reviewer for several journals.

JUAN BOUBETA-PUIG received the degree in computer systems management and the B.Sc. and Ph.D. degrees in computer science from the University of Cádiz (UCA), Spain, in 2007, 2010, and 2014, respectively. Since 2009, he has been an Assistant Professor with the Department of Computer Science and Engineering, UCA. His research focuses on the integration of complex event processing in event-driven service-oriented architectures, Internet of Things, and model-driven development of advanced user interfaces. He received the Extraordinary Ph.D. Award from UCA and the Best Ph.D. Thesis Award from the Spanish Society of Software Engineering and Software Development Technologies.