proceedings

MDPI

Proceedings

A Microservice-Based Framework for Developing Internet of Things and People Applications †

Aurora Macías, Elena Navarro * and Pascual González

LoUISE Research Group, Computing Systems Department, University of Castilla-La Mancha, 02071 Albacete, Spain; Aurora.Macias@alu.uclm.es (A.M.); Pascual.Gonzalez@uclm.es (P.G.)

* Correspondence: elena.navarro@uclm.es

† Presented at the 13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019, Toledo, Spain, 2–5 December 2019.

Published: 21 November 2019

Abstract: The Internet of things (IoT) is characterized by billions of heterogeneous, distributed, and intelligent objects—both from the digital and the physical worlds—running applications and services. Objects are connected through heterogeneous platforms providing support for the collection and management of data that need to be understood. Since IoT systems are composed by a variety of objects and services, a key aspect for engineering them is their architecture. The new paradigm called Internet of people (IoP) is not unaware of this need. In IoP, humans play an important role so that design considering aspects as context becomes critical for making the most of these applications. This work presents a context-aware, serverless, microservice-based, and cloud-centric framework for the Internet of things and people (IoT-P) applications that extends the three-layer classic IoT reference architecture. It integrates most of the aspects considered by the architecture of IoT solutions emerging from different perspectives, being also domain independent. This work focuses on the application paradigm of IoT neglected by most proposals. This framework, combined with a previous work, offers a higher separation of concerns (SoC) degree than other proposals, by splitting the application layer into different sublayers or subsystems based on their responsibilities and tracing atomic components to serverless microservices, to facilitate the design, development, and deployment of IoT-P applications. An IoT-P application in the healthcare domain is presented to illustrate how this framework can be put into practice.

Keywords: software architecture; layer; microservice; serverless; cloud; context-aware; Internet of things; Internet of people

1. Introduction

Internet of things (IoT), also known as Internet of everything or cyber-physical systems [1], can be considered a new computing paradigm [2–4] that encompasses a great amount of technologies that promotes its vision [2] in a disruptive way [5]. IoT can lead to a modern society where people and things are virtually integrated with information systems via wireless sensors [3]. It offers a great potential for improving not only the efficiency of a wide diversity of industrial processes, but also the quality of human beings with application scenarios such as e-health or assisted living. Therefore, IoT could contribute invaluably to economic development having social implications as well [4]. A new paradigm, Internet of people (IoP) [6], emerges as an evolution of IoT and cyber-physical social systems (CPSSs), so that the social nature of the human beings is now considered for the development of these systems. In this IoP proposal, a human being has a myriad of connection possibilities through the Internet by self-organizing networks of users as well as of physical devices through IoT. Moreover, in IoP, devices could become representatives of their owner that can act on their behalf [7]. This new paradigm not only does not compete or substitute IoT infrastructures but it uses its

infrastructures and enlarge its capabilities. Thus, human and devices create a complex socio-technical ecosystem [8].

However, there are many issues and challenges that limit the effectiveness and performance of the IoT [1,4], such as scalability, amount of data to be managed, real-time processing, etc., which are related to the software architecture (SA) [1]. With the rapid increase in the number of smart devices, existing IoT architectures are not, anymore, fully applicable to provide ideal services [4]. Moreover, there is no a standard accepted architecture that guides their development, as stated in [4]. Most of the approaches propose a field- or domain-specific architecture. Therefore, it is necessary to define a set of design patterns that may be used to provide end-user applications with self-adaptive [9] and context-aware properties [2], and a reference IoT architecture fulfilling all the IoT needs in different domains [10].

The aim of this work is to identify requirements and open issues of IoT architectures that should be addressed by a suitable domain-independent IoT architectural solution. This study not only considers IoT needs but also consider the new features that IoP may introduce in the proposed architecture. In order to tackle this goal, this work summarizes the most important aspects regarding IoT architectures, identifies certain issues associated to IoT architectures that should be addressed, and presents a context-aware, microservice-based, and domain-independent architectural framework for IoT applications. The proposal addresses many of the identified problems focusing on the application paradigm of IoT neglected by most proposals. The remainder of this paper is organized as follows: Section 2 gives an overview of related work about IoT architectures. Section 3 presents the core of our architectural proposal highlighting the way in which the separation of concerns (SoC) degree is achieved by splitting the application layer into different sublayers or subsystems based on their responsibilities. Section 4 presents how our proposal is based on microservice and serverless aspects. Section 5 describes how our proposal has been put into practice within an IoT system prototype in the healthcare domain. Finally, Section 6 draws our conclusions and lays out our future work.

2. Related Work

In this section, a review about IoT architectures is presented highlighting which issues should be considered by an IoT architectural proposal. As claimed in [1,4], most of the architectural specifications used at the initial stages of IoT research were structured into three layers, each layer being related to one of the three main IoT paradigms or perspectives [5]:

- *Application (or presentation or semantic) layer:* It employs intelligent computing technologies (e.g., data mining, cloud computing) to extract valuable information for processing a huge amount of data [4]. Data received are then analyzed to provide users with services and make decisions [1], offering an interface between users and IoT [4].
- *Transport (or network) layer*: It deals with network operations [4]. This layer is responsible for the transmission of gathered information [1].
- *Sensing (or perception or hardware or physical) layer:* It is responsible for collecting information [4].

In these initial approaches, the user should be considered in a paramount way and the SA should enable the use of data and the infrastructure to develop new applications [10], connecting all perspectives. However, most proposals were focused just on the generic aspects of the network and, especially, on those aspects related to sensoring [5,10]. Therefore, such proposals *neglect the application domain and data presentation aspects that enable the creation of valuable knowledge for business or use cases* [5] *(Issue 1).*

Normally, a traditional application-based approach that connects sensors directly to applications becomes unfeasible and results as inefficient [2]. For this reason, existing IoT architectures need to be scaled up. Regarding flexibility, as stated in [3], autonomous services needed by IoT users can be supported by constructing an adaptive, context-aware, and reconfigurable service architecture able to handle applications according to its requirements. These features are especially

relevant to IoP applications, as the context and the needs of adaptation are a must for their development. As Lagerspetz et al. pointed out [7], for IoP applications, "devices also need to obtain information about the social context they are operating in, so they can share resources as their owners would".

In order to address the problems exhibited by those approaches and the requirements of IoT, such as scalability, flexibility, interoperability, quality of service (QoS), and security among others [3], several middleware solutions were introduced as an abstraction layer between the transport and the application layers. However, those middleware solutions focus just on some of the mentioned aspects as stated by [2,11]. Therefore, *an ideal middleware solution or architectural framework that addresses all the aspects required by the IoT is yet to be designed* [2] *(Issue 2)*.

Most of the middleware solutions, such as [4,9], proposed for architecting IoT follow a service-oriented architecture (SOA) approach. Despite its advantages for IoT solutions, SOA may become too heavy for being deployed on resource-constrained devices when the system is finally developed as a monolith difficult to be scaled up and evolved. It should be considered that *an appropriate SoC degree can ease the traceability between middleware or framework abstractions and components or services (Issue 3)*. Furthermore, some of the existing IoT SOA-based middleware solutions rely on several layers that consider support to *object abstraction, service management,* and *service composition* [4]. However, the definition of specific data models and representations is not properly addressed by such proposals. Consequently, *new methods are necessary to adapt SOA concepts to IoT needs* [9] (*Issue 4*). All the mentioned aspects are paramount for context information management [1], as a proposal able to both *process the huge amount of data that are continuously generated and decide how to process them in order to obtain valuable information* [2] *(Issue 5)* is necessary, because gathering, modelling, reasoning, and distribution of context plays a critical aspect for IoT. Such context information management is also paramount in IoP, as pointed out in [12].

Context-aware computing allows context information linked to sensor data to be stored, so that the processing and interpretation can be done more easily and meaningfully. However, many IoT middleware and framework solutions do not provide context-awareness support or the context-aware support they offer do not satisfy other important requirements that IoT demands, such as self-adaptation aspects [2]. An example is the Context Toolkit [13] (CTK). It satisfies common features required by context-aware applications as well as the requirements to deal with context in a successful way, such as a proper SoC. The CTK defines a set of abstractions to support the design and can be applied to any application domain. A key aspect of those abstractions is that physical objects or software components, people, and places, as well as their interactions, are properly managed, offering a more complete template for architectural design and implementation [10]. The CTK also provides support for resource discovery, but it only provides partial support to self-adaptation [14].

Self-adaptive systems [15] are able to adapt themselves according to both external and internal changes of their execution environment in order to continue achieving their goals. They have been used for the development of context-aware systems [16]. Their main constituent parts are the following: (i) Managed subsystem, that comprises the application logic that provides the system domain functionality; and (ii) managing subsystem, that provides the adaptation logic that deals with one or more concerns for managing the managed subsystem. In some works, such as [15], the managed subsystem maps to the system layer and the managing subsystem to the architecture layer. Then, the managed subsystem can be decomposed into another managing and managed subsystems providing an additional SoC between the adaptation and the application logic. Similarly, the managing subsystem can get a higher SoC degree by following a MAPE-K (monitor–analyze–plan–execute plus knowledge) loop [17]. Normally, the self-adaptive approach requires the coding of an "intelligent" self-management logic in order to satisfy the adaptation requirements necessary for the system-to-be, which prevents developers from focusing on the application domain exclusively. In order to avoid that, it would be *desirable to minimize the coding efforts of the adaptation logic delegating those aspects to the architectural support (Issue 6)*. Furthermore, regarding IoP, devices and system components should act proactively without requiring users' actions, as mentioned in [11].

Another approach that is quite frequently used for the development of IoT applications is cloud computing [2], because of its promises of high reliability, autonomy to provide ubiquitous access, dynamic resource discovery, as well as the composability required for the next generation of IoT applications. Cloud resource management and scheduling systems are able to dynamically prioritize requests and provision resources so that critical requests may be served in real time [10]. Moreover, cloud brings added scalability to context management in the IoT, since it offers significant amounts of processing and storage capabilities. Furthermore, cloud computing allows all parties to share sensor data based on a financial model [2]. Adaptation offered by cloud computing is, therefore, powerful. Nevertheless, the *cloud computing platform to be adopted should support a wide variety of devices, environments, scenarios, processing patterns, and standards in a scalable and secured manner* [1] *(Issue 7).*

3. An Enhanced Integrated Proposal for IoT Architecting

A conclusion that can be drawn from Section 2 is that it is necessary to define an IoT design framework (integrating a reference architecture, a set of patterns, etc.), with built-in context-aware and self-adaptive capabilities among other aspects. This framework should be able to satisfy IoT-specific requirements and address challenges like scalability, flexibility, security, etc., as well as all the issues highlighted in the previous Section. This is the aim of this work, to describe such an IoT architectural framework.

3.1. Previous Results

Recently, in [18], some of the most well-known architectural approaches used in the design of context-aware and self-adaptive systems were analyzed. Their common aspects were also identified, such as the close relation between the MAPE-K loop and the CTK in terms of their elements' responsibilities, and a new proposal was presented based on such common aspects using the CTK as the foundation of the proposal. The resulting proposal offered a modified set of abstractions or components, namely (*i)* widgets, that exposed data received from sensors when the datum is atomic, or from other widgets when the datum is composed; (ii) aggregators, that receive data from widgets to collect all the data that define an entity context; (iii) interpreters, responsible of simple inference or derivation, and (iv) reasoners, responsible of complex inference or reasoning, that derive or generate new information from that provided by widgets, aggregators, or applications; (v) actions, that are carried out on the system or on the environment, triggered by widgets, aggregators, or applications; and (vi) discoverers, that receive notifications from other abstractions, as well as from their sub-discoverers in the hierarchy, and record information about the components available in the system. All these components make up what is called the context architecture [13]. This architecture supports the context life cycle, that is, the context information acquisition, its delivery to the interested application(s), and the execution of actions related to the context that must be supported directly by the architecture.

Another common concept in context-awareness, also considered in [18], is situation. A situation is defined as the state of the current and past context in a certain region of the space and a concrete interval of time that are relevant to identify what is happening in the environment [19]. Situations can be considered complex events that affect entities related to the environment [13]. In order to detect different situations, the context architecture can be decomposed in different situation context architectures or subsystems, each of them supporting a concrete situation. Situation subsystems, or its components in isolation, could be reused by several applications, as pointed out by [13]. All the abstractions, as well as the concepts presented in this Section, are integrated into our new proposal described in the following.

3.2. A Framework for Architecting the Application Layer in IoT Applications

With the goal of offering architectural support to adaptation aspects, in the new proposal, the application layer presented in the initial three-layer architecture is now split into two different main subsystems matching the self-adaptive architecture (see Figure 1). These subsystems are named

adaptation subsystem (equivalent to managing subsystem) and domain functionality subsystem (equivalent to managed subsystem). Moreover, the domain functionality subsystem is also structured into two subsystems to establish a clearer separation between the adaptation concerns of the system and the domain application logic: One equivalent to another managing subsystem and another equivalent to another managed subsystem. Bearing in mind the equivalence between MAPE-K and the CTK, in terms of responsibilities previously indicated, the new two subsystems are: global context subsystem, which is equivalent to the new managing subsystem; and domain application(s) logic, which is equivalent to the new managed subsystem. Furthermore, the global context subsystem is structured into different situation context subsystems that are specified using the abstractions and concepts previously indicated (see Figure 1). Finally, as in other solutions, the domain logic of the application(s) to be developed for every system-to-be is out of the architecture specification here defined. However, it is worth noting that our proposal facilitates the definition of different aspects of the application layer that other proposals neglect, as it will be explained in more detail in the following sections. The other elements of the three-layer architecture remain unchanged.

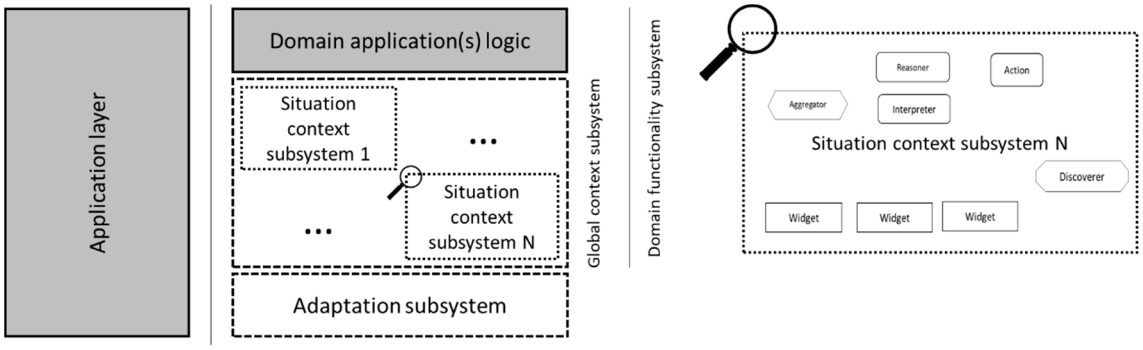

Figure 1. The proposed framework for developing Internet of Things (IoT) applications is made up of the adaptation subsystem and the global context subsystem. The latter is made up of different situation context subsystems, which are specified using the abstractions defined in [18].

In the two following subsections, the most relevant subsystems of the proposal, global context subsystem, and adaptation subsystem, that support the IoT required context-awareness and self-adaptation capabilities respectively, are described. It is also summarized how some of the issues identified in Section 2 are addressed and how certain elements like cloud computing [20] have been exploited in order to obtain an integrated architectural approach.

3.2.1. Global Context Subsystem

In order to increase the required SoC, the global context subsystem is specified as a hierarchical structure based on the domain-driven design notion of *bounded context* [20]. This notion is used to split a complex domain into multiple bounded contexts matching business or user goals that should be satisfied by the application. Specifically, bounded contexts establish different subsystems around the concept of situation: Each bonded context corresponds to a subsystem related to a situation, that is, a situation context subsystem. A situation context subsystem supports the management of a concrete situation by using the results of our previous work [18].

As introduced, each situation context subsystem is specified using the abstractions described in Section 3.1 and composed using the data flow of the context life cycle (see Figure 2 for an example). These abstractions are then traced to services, being their interactions carried out according to the relations described in Section 3.1. Therefore, the situation context subsystems trace to system goals or use cases of a situation. This facilitates the definition of different aspects of the application layer that other proposals neglect, addressing *Issue 1—neglection of the application domain and data presentation aspects that enable the creation of valuable knowledge for business or use cases*. In a similar way, that definition facilitates the system capability to take proactive steps according to a user's situation and expectations minimizing user explicit intervention, as it is desirable in IoP applications.

Situations management is required to understand the context. In order to enable the detection of situations and their associated subevents, it is necessary to previously define the required data as well as their related rules or patterns. Therefore, data drive the selection of sensors to be used. The abstractions included in this proposal, from widgets to actions, support not only the gathering of context information by means of sensors but the whole context life cycle including the reasoning needed to detect situations. Concretely, widgets and aggregators model and store simple data and entities data respectively, which is fundamental in data processing. These aspects address *Issue 5—gathered data need to be understood and it is required to decide what data need to be processed in order to obtain valuable information*.

Abstractions included in the proposal have well-defined responsibilities and their implementation as software components is highly cohesive around those concerns. This aspect facilitates abstractions traceability to the services that made up the global context subsystem. These aspects tackle *Issue 3—appropriate SoC degree to ease the matching between middleware abstractions and components or services*.

3.2.2. Adaptation Subsystem

As mentioned, adaptation subsystem (which is equivalent to a managing subsystem) is one of the main subsystems of our proposal. As indicated in [18], the MAPE-K loop, the common approach to structure managing subsystems, can be considered equivalent to the CTK in terms of their elements' responsibilities. Consequently, the adaptation subsystem could also be designed using such abstractions. However, instead of designing and developing the adaptation logic for IoT systems in a traditional way, it is possible to adopt a cloud-based approach. In this sense, some companies offer different cloud-based architectural solutions integrated by certain services that are supposed to be the backend for IoT applications [1].

Cloud computing promises high reliability, scalability, and autonomy to provide ubiquitous access, dynamic resource discovery, and composability required for IoT applications [10]. Furthermore, the integration of cloud with IoT offers a viable approach to facilitate application development [3] as developers can focus on application domain aspects rather than on infrastructure aspects. Cloud-based architectures are also commonly adopted in IoP [12]. For these reasons, the architectural proposal is combined with the adoption of a public cloud platform, as will be detailed in the following. Thus, cloud computing will support adaptation capabilities, as it will be explained in Section 5. This adoption address *Issue 6—desirable to minimize the coding efforts of the adaptation logic delegating those aspects to the architectural support.*

4. A Microservice-Based Framework for Developing IoT-P Applications

In the previous section, the structure of a new context-aware framework for IoT applications, considering self-adaptive capabilities has been introduced. The architectural framework has different enhancements regarding the original three-layer architecture. It establishes different conceptual layers and subsystems that have different and well-defined responsibilities. Since the extension of the domain-independent conceptual framework CTK presented in [18] is used as its foundation, the proposal is also applicable to any domain. As aforementioned, the framework proposed has been combined with certain elements like cloud computing and also with microservices' architectural style [21] or serverless computing [22]. The purpose of such combination is to tackle *Issue 2—ideal middleware solution or architectural framework addressing all the aspects required by the IoT is yet to be designed*. The proposal could also be used in the development of IoP applications due to IoT and IoP common basis [12]. In the following, the integration of such elements is described in more detail, summarizing how most of the rest of the identified issues are addressed by the presented proposal. Moreover, the trace between the abstractions proposed in [18] and cloud computing services or elements that can be used to implement them is also presented.

The abstractions defined in [18] used in this work can facilitate not only the virtualization of objects but also the componentization of the global context architecture and, even, of the adaptation subsystem following the microservices architectural style. Microservices style refers to an approach for developing a single application as a suite of small services, each running in its own process, being

independently deployable, communicating with lightweight mechanisms, and minimizing the coupling [23]. Some of the main characteristics of microservice style is the organization around business capabilities or goals (following the *bounded context* notion), their decentralized governance and data management, their "intelligent" endpoints releasing the services of the corresponding management concerns, and their design for failure. Moreover, microservices style does not require a high level of resource discovering capabilities. It is worth highlighting that decentralized data management usual in microservices facilitates the definition of specific data models. Thus, as widgets and aggregators are traced to microservices, they will be responsible for the management of their own data and those data will be well-defined around the concerns of such abstractions (sense or gathered data will be stored along with some convenient metadata). This offers a standard method to derive data models (or context models [13]) from the architecture specification, an aspect especially neglected by the SOA approach [9]. Furthermore, microservices' decentralized governance facilitate the use of different technologies per microservice as appropriate [21].

In a similar way, the global context subsystem and, specifically, their situation context subsystems or architectures can be designed as a serverless architecture [24]. This means that microservices' code will run in managed, ephemeral containers offered by the platform, which is suitable because the abstractions are highly cohesive and microservices do not require a high level of resource discovering capabilities. This approach addresses also *Issue 6,* because it reduces the operational cost, complexity, and engineering lead time. Then, microservices and serverless address *Issue 4—new lightweight methods needed to adapt SOA concepts to IoT needs*. However, the adoption of a serverless microservices approach has some common drawbacks like the reliance on vendor dependencies and immature supporting services.

As aforementioned, there exist different cloud-based architectural solutions integrated by certain services that are supposed to serve as the backend for IoT applications. Considering the adoption of diverse cloud platforms, the traceability from the abstractions here used (see Figure 2) to cloud computing services that can be suitable to implement them is explained in the following. The most popular public cloud platforms are Amazon AWS, Microsoft Azure, IBM Cloud, and Google Cloud Platform. All of them offer a kind of platform as a service (PaaS) "polyglot" service able to run lightweight and highly-cohesive code, scale automatically on demand in response to events, as well as exploit a serverless architecture with a pay-per-use consumption model. A serverless service is called function in Azure Functions [25] or in Google Cloud Functions, action in IBM Cloud Functions, or lambda-function in AWS. No matter their name be functions, lambdas, or actions, serverless services are exposed by means of representational state transfer (REST) application programming interfaces (API) that define the way in which the different registered services can be invoked. Each defined API can group functions associated, so that, in this proposal, each situation context subsystem is described as an API that can expose all the situations' functions. Functions—or lambdas, or actions—are then used to implement every abstraction presented in Section 3.1. Table 1 summarizes the identified traceability from the architectural abstractions here proposed to the services or resources available in each cloud platform. However, there are certain exceptions.

The first exception is related to the implementation of discoverers. Functions do not seem to be the best approach since discoverer abstractions are responsible for certain adaptation aspects, and those aspects should be delegated to the architecture as much as possible, that is, to the adaptation subsystem. The adaptation subsystem has been traced to the cloud infrastructure. All cloud platforms considered offer, at least, a kind of service inspired in the API gateway pattern [26] that, in microservice architecture style, allows application clients to interact with more than one front-end service. API gateways offer information that applications may use to "know" what endpoints can be called based on an API definition. This facilitates the introduction of new services or the refactorization of existing ones, the management of authentication and security aspects, as well as other adaptation concerns. It is worth highlighting that some API gateway services allow services from other platforms to be integrated as well. Consequently, an API gateway service emerges as the best approach to implement the general discoverer of the system-to-be. API gateways manage the APIs defined for each group of functions for each situation. An API defined for a group of functions

or situation context subsystem would be equivalent to a situation discoverer. Both API gateway service and situation API, equivalent to discoverers, are integrated, establishing a hierarchy of different discoverers in the system.

The second exception is related to the fact that inference or reasoning needed for a specific reasoner could be too complex to be implemented as a function. Thus, the use of artificial intelligence (AI) services, such as Azure AI, Google Cloud AI, or Amazon AI, for example, could be more suitable. All these AI services offer different facilities that could be used to develop specific services according to the reasoning to be carried out (e.g., machine learning, deep learning, etc.).

Besides the indicated cloud platforms, there are other open-source platforms that offer most of the kinds of services aforementioned; Apache OpenWhisk [27] or StackStorm [28] being some of the most popular ones. Both open-source platforms offer serverless functions, equivalent to those indicated above, that are officially called actions. Furthermore, both platforms also offer support to implement REST APIs to expose such actions. OpenWhisk offers a kind of API gateway service while StackStorm allows the defined APIs to be managed by means of a self-service portal and other orchestration services. Regarding AI services, OpenWhisk allows a type of AI action to be created that can include some AI frameworks and libs, such as the Google ML Engine, etc. However, StackStorm does not offer this kind of service up to date. It is worth highlighting that Table 1 also shows the traceability to OpenWhisk and StackStorm. This highlights that the proposal here presented may be applied to the design of IoT applications independent of the cloud platform(s) selected. In order to show how the proposal may be put into practice, the design and development of an IoT system prototype in the healthcare domain is presented in the following Section.

Table 1. Traceability from the architectural abstractions to cloud platform services or resources.

Arch. Abstractions	Amazon AWS	Microsoft Azure	Google Cloud	IBM Cloud	OpenWhisk	StackStorm
Context Widget	Lambda	Function	Function	Action	Action	Action
Aggregator	Lambda	Function	Function	Action	Action	Action
Interpreter	Lambda	Function	Function	Action	Action	Action
Reasoner	Lambda AI Service	Function AI Service	Function AI Service	Action AI Service	Action AI Action	Action ---
Action	Lambda	Function	Function	Action	Action	Action
Situation Discoverer	REST API ---	Function App REST API	REST API ---	REST API ---	REST API ---	REST API ---
Global Discoverer	API Gateway	API Management	API Management	API Gateway	API Gateway	Self-service user portal
	---	Application Gateway	---	---	---	Other orchestrator services

REST: Representational State Transfer; API: Application Programming Interface.

5. Case Study

An important application of IoT is in the smart healthcare domain [4]. IoT offers a perfect approach to support ubiquitous healthcare using body area sensors, closely related to the IoP paradigm, as well as other devices for monitoring and IoT back-ends to upload data to servers [10].

The architectural framework proposed was applied to the development of an IoT system in the healthcare domain that detects, using contextual information gathered by sensors, different emergency or illness situations that may affect users. The system also generates alarm warnings and carries out other similar actions based on user preferences, their contacts, or their geographical proximity among others. Thus, this application considers relevant aspects related to human, their relationship with others, and the use of wearables to control some physiological signals, that is, it is a classic example of an IoP application. The system includes monitoring applications that allows relatives, caregivers, or general physicians, for example, to access health information. One of the situations or complex events this system should react to is whether a user has been under a high level of stress for a long time. Biomedical signals to be considered to determine such situation are heart

rate (HR), galvanic skin response (GSR), and body temperature (BT) [18]. Due to space constrains, only the stress situation was tackled in the following.

To detail the system prototype implementation, the extended three-layer architecture could be used as a guide following a top-down approach. In this case, we focused on the subsystems equivalent to the original application layer (see Figure 2), that is, subsystems that belong to the proposed framework and application(s)-specific logic. As indicated in Section 3.2, application(s)-specific logic was designed and implemented without using the architectural proposal specification. For that reason, details about users' applications, that correspond mainly to user interfaces, were not provided in the following.

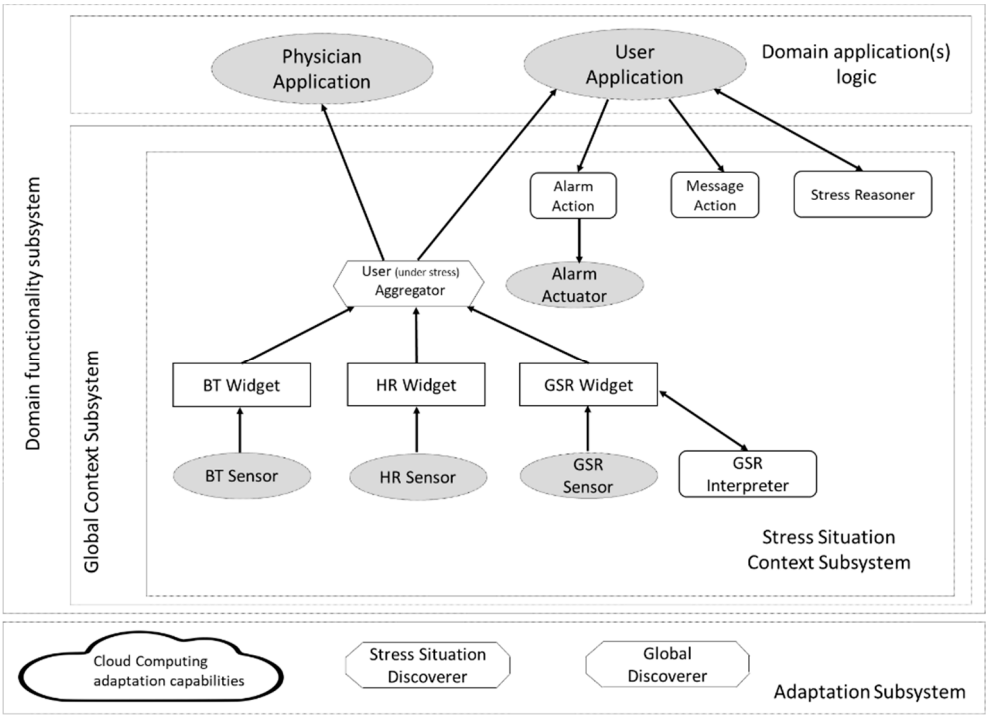

Figure 2. Stress situation context subsystem (adapted from [18]) of the global context subsystem as part of the redefined application layer by the proposed framework. BT: Body Temperature; HR: Heart Rate; GSR: Galvanic Skin Response.

The global context subsystem was structured into two situation context subsystems. One of them is the stress situation context subsystem that was designed following the proposal, as shown in Figure 2. Its design was carried out applying the composition rules, which are based on the abstractions' responsibilities as well as their possible interrelations following the context life cycle, as depicted in Section 3.1. As can be seen, those data needed to process a stress situation, BT, HR, and GSR are specified.

The implementation of the global context subsystem for this example was carried out using Microsoft Azure. As stated in [1], Azure offers important characteristics and possibilities for IoT, with respect to other platforms, and its services have proved to be reliable and more mature. Azure's multi-technology supports microservices' decentralized governance and facilitates the use of different technologies per microservice as appropriate. The selection of Azure facilitates tackling *Issue 7—support for a wide variety of devices, environments, scenarios, processing patterns, and standards*.

All the abstractions that make up the stress situation context subsystem, except for the discoverers, were implemented as serverless microservices using Azure Functions, as shown in Figure 3. As indicated, these functions facilitate the exploitation of the microservice architecture style. Concretely, every function was well-defined and cohesive around the matching abstraction concerns. Every function was also linked to a Cosmos DB [29] (an Azure Not only SQL -NoSQL- database) scheme based on the needed data. Concretely, the resulting data models associated to the specified widgets and aggregator satisfied the statements made in the previous Section (i.e., the BT Widget

stores BT values plus the sensing time, the sensor id, and the sensor precision; and the User (under stress) Aggregator stores tuples of BT, HR, and GSR values plus the gathering time and the associated user id). Each data scheme was managed and could be only accessed by the corresponding function. Every function was also serverless since Azure Functions act as containers offering automatic resource provisioning, as well as other adaptive capabilities supported by the cloud platform previously indicated.

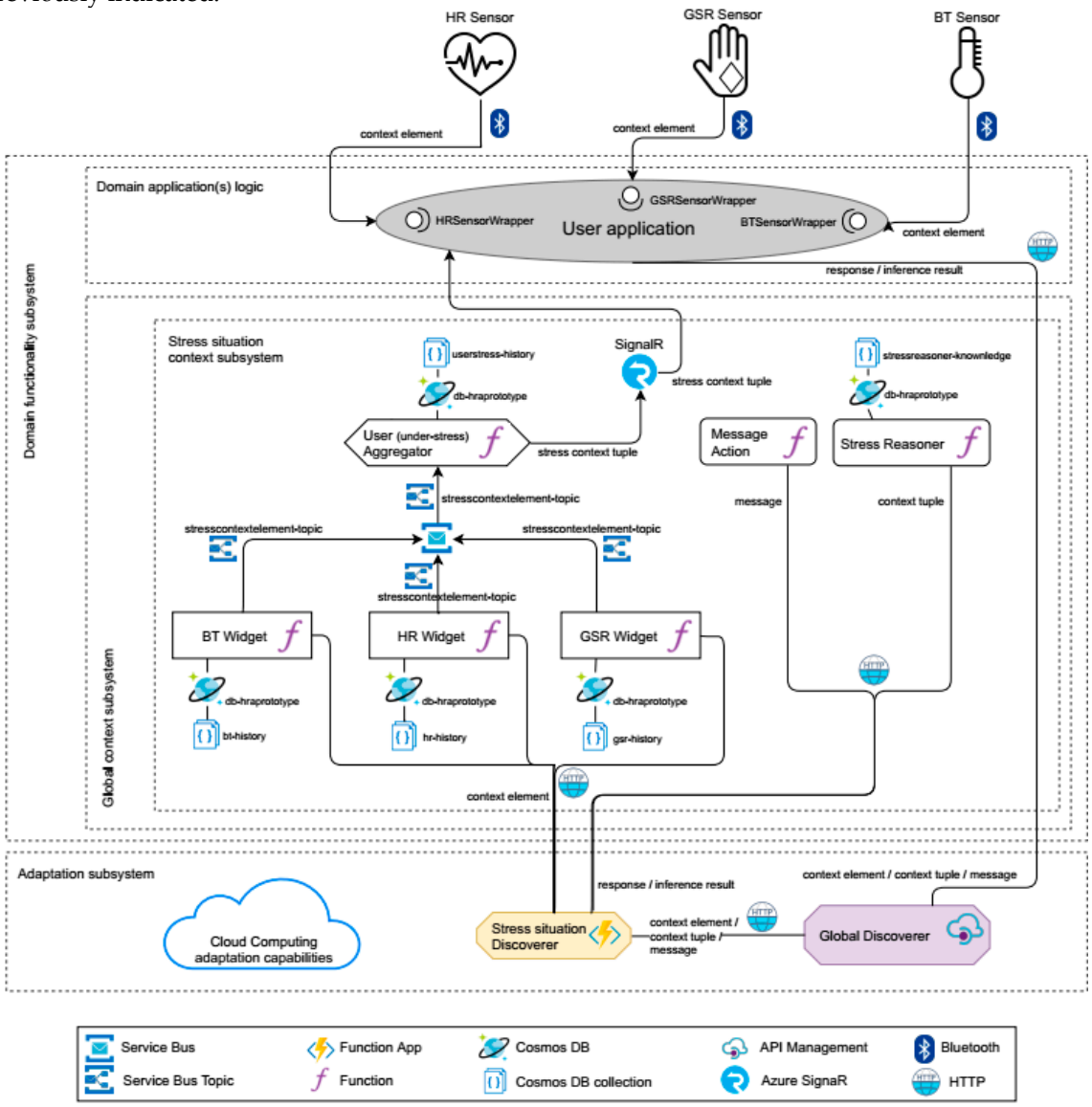

Figure 3. Detail of the prototype's deployment in Microsoft Azure using its serverless functions and services (only the stress situation context subsystem implementation is detailed).

To implement the stress situation context subsystem, a Function App service [25] that groups several functions was used. A template of the API that exposed the functions of the situation, that is, the situation discoverer, was generated using an option of the Function App service. This facility as well as some other managing options and capabilities are separated from functions code or logic. The configuration and adaptation aspects were supported by the cloud platform and belong to the adaptation subsystem identified in Figures 2 and 3, since they allow service operation information to be collected, to configure some security parameters, as well as to change and add triggers and outputs to the corresponding function. As Figure 3 shows, the global discover was implemented by using the Azure API Management service [30] as part of the adaptation subsystem. In this way, responsibilities of the discoverer abstraction were delegated from the context architecture to the system layer underneath. Therefore, the adaptation subsystem, along with the other self-adaptive capabilities, may

be offered by Azure. This facilitates the coding effort needed to implement the discoverers to be significantly reduced, since it is only needed to configure certain parameters of Azure.

Data needed to detect situations drove the selection of the sensors or devices to be used, as previously stated. In this example, Microsoft Band 2 [31] was used: a popular smart wristband supported by all the main smartphone operating systems available, includes sensors that can measure all the above-mentioned signals, as well as others. This wristband can be connected to users' smartphones. It is worth highlighting that details about protocols, communication mechanisms, etc., are outside of the focus of this work.

It is worth considering that all frameworks, patterns, architectural styles, etc., integrated into the architectural framework here proposed, have been validated and are widely accepted. For this reason, an in-depth evaluation of the proposed integrated framework will be carried out in a future work. As a preliminary sort of validation of the proposal, certain aspects of the performance of the system designed and developed using the framework were analyzed. Concretely, the system response time values offered by the Azure Analytics service regarding the global discoverer, which was implemented as an API Management service, were checked. During the monitored period, all the performed requests (one per second and device) were successful and the response time of each one varied between 0.25 and 4.5 s, 1.61 s being the average response time.

As the preliminary validation data suggested, the proposed cloud-centric approach could result inefficient if the transmission of great amounts of data to the cloud platform is required. The reason is that the cloud platform services are responsible for processing the data received and for analyzing them. That analysis is needed to executing actions on the system itself, or on the environment as part of the situation detection process. That requirement involves a great server-side bandwidth and increases client latency (and/or response time). That is why this approach could result inefficient when devices used are smart enough, but they are not used to analyze the data, relying on the cloud infrastructure instead [32]. For this reason, the *cloud computing approach should be extended in order to improve data management and processing efficiency (Issue 8).*

6. Conclusions and Future Work

The computing paradigm Internet of things and people (IoT-P) facilitates the connection of both virtual and physical generic everyday things or objects, invisibly embedded in our environment, and people by means of existing and new Internet aspects and network enhancements. IoT-P is related to the growth of the ubiquitous infrastructure in which those objects, some of them on behalf of people, flood the Internet with a high amount of new data that need to be understood. As has been stated, there are lot of issues and challenges that limit the effectiveness and performance of the IoT-P, particularly those related to the IoT reference architectures used that usually focus on sensors and network aspects, neglecting the application domain, information presentation aspects, and other relevant features of IoP.

This work presents a context-aware serverless microservice-based and cloud-centric architectural framework for IoT-P applications in order to fulfil their demands. The proposal extends the IoT three-layer classic architecture, focusing on the neglected application paradigm of IoT, and integrates most of the aspects considered by the existing IoT-P solutions (Issue 2). Concretely, the proposal allows a more fine-grained architectural definition that makes the design and development of IoT-P applications straightforward, by splitting the application layer into different sublayers or subsystems based on their responsibilities. These subsystems can, in turn, be componentized matching the abstractions defined in our previous work. This facilitates the definition of specific models and representations usually neglected by other proposals (Issue 1). Abstractions, implemented as serverless functions, enable a loosely-coupled plugin architecture in which each service or component is independently deployable. As a summary, the proposal tackles some other open issues remarked in this work such as the need to decide which data should be processed among the enormous amounts of data generated (Issue 5), an appropriate SoC degree that matches the middleware abstractions and the components or services (Issue 3), new lightweight methods to adapt SOA concepts to IoT and IoP peculiarities (Issue 4), the reduction of the development effort of the

adaptation logic by means of the architectural support (Issue 6), and the support for a wide variety of devices, social environments, scenarios, processing patterns, and standards in a secure manner (Issue 7). Moreover, the proposal is technology independent and can be used in any IoT-P domain.

The proposal was applied to the design and development of an IoT-P system, in the smart healthcare domain, able to detect certain health risk situations affecting users who wear a smart wristband linked to their smartphones, exemplifying a common IoP problem. As seen, the design and development were easily carried out since the components are created as serverless, cohesive, containerized cloud services and the required adaptive aspects are managed autonomously by them or by the cloud platform. The specification shows the division between adaptation and context management concerns and provides a high detail of the context architecture, while the specific application(s) logic design remains out of the proposal.

Despite the architectural benefits, that impact the system quality, the proposed cloud-centric approach could result inefficient in some cases and should be extended (Issue 8). As next steps, we plan to analyze the performance, throughput, costs, etc., of the alternative deployments (e.g., containers compared to functions, incorporating fog and edge computing aspects, etc.), in order validate and to extract conclusions useful to improve and extend the proposal. Moreover, we are developing a tool for the design of IoT-P systems using this approach as well as for the automatic generation of IoT-P systems.

Acknowledgments: This work was partially supported by the Spanish Ministerio de Economía, Industria y Competitividad, Agencia Estatal de Investigación (AEI)/European Regional Development Fund (FEDER, UE) under Vi-SMARt (TIN2016-79100-R).

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

References

1. Al-Qaseemi, S.A.; Almulhim, H.A.; Almulhim, M.F.; Chaudhry, S.R. IoT architecture challenges and issues: Lack of standardization. In Proceedings of the 2016 Future Technologies Conference (FTC), San Francisco, CA, USA, 6–7 December 2016; pp. 731–738.
2. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454.
3. Yaqoob, I.; Ahmed, E.; Hashem, I.A.T.; Ahmed, A.I.A.; Gani, A.; Imran, M.; Guizani, M. Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IEEE Wirel. Commun.* **2017**, *24*, 10–16.
4. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A Survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
5. Weyrich, M.; Ebert, C. Reference Architectures for the Internet of Things. *IEEE Softw.* **2016**, *33*, 112–116.
6. Conti, M.; Passarella, A.; Das, S. The Internet of People (IoP): A new wave in pervasive mobile computing. *Pervasive Mob. Comput.* **2017**, *41*, 1–27.
7. Lagerspetz, E.; Flores, H.; Mäkitalo, N.; Hui, P.; Nurmi, P.; Tarkoma, S.; Passarella, A.; Ott, J.; Reichl, P.; Conti, M.; et al. Pervasive Communities in the Internet of People. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Athens, Greece, 19–23 March 2018; pp. 40–45.
8. Conti, M.; Passarella, A. The Internet of People: A human and data-centric paradigm for the Next Generation Internet. *Comput. Commun.* **2018**, *131*, 51–65.
9. Miorandi, D.; Sicari, S.; De Pellegrini, F.; Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* **2012**, *10*, 1497–1516.
10. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660.
11. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. CA4IOT: Context Awareness for Internet of Things. In Proceedings of the 2012 IEEE International Conference on Green Computing and Communications, Besancon, France, 20–23 November 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 775–782.
12. Miranda Carpintero, J.; Mäkitalo, N.; Garcia-Alonso, J.; Berrocal, J.; Mikkonen, T.; Canal, C.; Murillo, J. From the Internet of Things to the Internet of People. *IEEE Internet Comput.* **2015**, *19*, 40–47.

13. Dey, A.; Abowd, G.D.; Salber, D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Hum.-Comput. Interact.* **2001**, *16*, 97–166.
14. Kephart, J.O.; Chess, D.M. The Vision of Autonomic Computing. *Computer* **2003**, *36*, 41–50.
15. Weyns, D.; Schmerl, B.; Grassi, V.; Malek, S.; Mirandola, R.; Prehofer, C.; Wuttke, J.; Andersson, J.; Giese, H.; Göschka, K.M. On Patterns for Decentralized Control in Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems II*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7475, pp. 76–107, ISBN 9783642358128.
16. Arcaini, P.; Riccobene, E.; Scandurra, P. Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation. In Proceedings of the 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Florence, Italy, 18–19 May 2015; pp. 13–23.
17. La Iglesia, D.G.D.; Weyns, D. MAPE-K Formal Templates to Rigorously Design Behaviors for Self-Adaptive Systems. *ACM Trans. Auton. Adapt. Syst.* **2015**, *10*, 1–31.
18. Macías, A.; Navarro, E. An Integrated Approach for Context-Aware Development. In Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings (ECSA '18), Madrid, Spain, 24–28 September 2018; p. 7.
19. Martín, D.; de Ipiña, D.L.; Lamsfus, C.; Alzua, A. Situation-Driven Development: A Methodology for the Development of Context-Aware Systems. In *Ubiquitous Computing and Ambient Intelligence*; Bravo, J., López-de-Ipiña, D., Moya, F., Eds.; Springe: Berlin/Heidelberg, Germany, 2012; pp. 241–248.
20. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; et al. A View of Cloud Computing. *Commun. ACM* **2010**, *53*, 50–58.
21. Lewis, J.; Fowler, M. Microservices. *martinfowler.com*, 2014. Available online: https://martinfowler.com/articles/microservices.html (accessed on 20 August 2018).
22. Baldini, I.; Castro, P.; Chang, K.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Slominski, A.; et al. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*; Chaudhary, S., Somani, G., Buyya, R., Eds.; Springer: Singapore, 2017; pp. 1–20, ISBN 978-981-10-5026-8.
23. Pfleeger, S.L. *Ingeniería del Software. Teoría y Práctica*; Quiroga, E., Translator; Prentice Hall/Pearson Educación: Buenos Aires, Argentina, 2002.
24. Roberts, M. Serverless Architectures. *martinfowler.com*, 2017. Available online: https://martinfowler.com/articles/serverless.html (accessed on 20 August 2018).
25. Microsoft Corp. Azure Functions. *azure.microsoft.com*, 2018. Available online: https://azure.microsoft.com/en-ca/services/functions/ (accessed on 20 August 2018).
26. Wasson, M. Using API Gateways in Microservices. 2018. Available online: https://docs.microsoft.com/en-us/azure/architecture/microservices/design/gateway (accessed on 11 January 2019).
27. The Apache Foundation. Apache OpenWhisk Is a Serverless, Open Source Cloud Platform. 2018. Available online: https://openwhisk.apache.org/ (accessed on 10 January 2019).
28. StackStorm. 2018. Available online: https://stackstorm.com/ (accessed on 11 January 2019).
29. Microsoft Corp. Cosmos DB. *azure.microsoft.com*, 2018. Available online: https://azure.microsoft.com/es-es/services/cosmos-db/ (accessed on 20 October 2018).
30. Microsoft Corp. API Management. Publique, Administre, Proteja y Analice las API en Minutos. *azure.microsoft.com*, 2018. Available online: https://azure.microsoft.com/es-es/services/api-management/ (accessed on 20 August 2018).
31. Microsoft Corp. Microsoft Band 2. *www.microsoft.com*, 2017. Available online: https://www.microsoft.com/en-us/band (accessed on 13 January 2019).
32. Mass, J.; Chang, C.; Srirama, S.N. Context-aware Edge Process Management for Mobile Thing-to-fog Environment. In Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, Madrid, Spain, 24–28 September 2018; ACM: New York, NY, USA, 2018; pp. 44:1–44:7.