v. 3, n. 6, 2023

Journal of
Engineering
Research

PERCPECTIVES OF PUBLISH-SUBSCRIBE MIDDLEWARES IN SMART CITIES: PRINCIPLES, ARCHITECTURE AND CHALLENGES FOR EMERGENCY SERVICES

Sediane C. L. Hernandes
UTFPR

Rosana Lachowski
UTFPR

Hermano Pereira
UTFPR

Abstract: Publish-subscribe middlewares are important communication models for smart devices in the Internet of Things environment. Publish-subscribe middlewares connect applications and service providers while hiding questions of connectivity, security and event handling. In this context, a publish-subscribe middleware manages events sent to and from smart city applications in order to offer common services and facilitate their deployment and development. Emergency services can adopt the Publish-Subscribe paradigm to notify mobile units as soon as possible with minimal human intervention. Factors such as proximity to the emergency event and traffic conditions can be considered when selecting the mobile units to handle the event. This paper presents a survey of publish-subscribe middlewares in smart cities, with an overview of the principles, architectures, and perspectives of challenges for emergency services.
Keywords: Publish-Subscribe Middlewares; Emergency Services; Internet of Things, Smart Cities.

INTRODUCTION

Smart Cities explore the Information and Communication Technologies (ICTs) aiming to improve the quality of services (QoS) offered to their citizens (QIAN et al., 2019). Fifty percent of the global population lived in cities in 2012 (ALKANDARI; ALNASHEET; ALSHAIKHLI, 2012) and according to the report from United Nations by 2050 it is expected a world population increase of seventy percent (UNITED, 2017). Consequently, new difficulties arise in the cities, as urban traffic management, noise pollution monitoring, scarce resources management and fast emergency response times.

The Internet of Things (IoT) is a powerful tool to face the challenges of modern cities (ALDELAIMI et al., 2020). The IoT connects devices (smart objects) such as computers, smartphones, sensors, actuators, appliances and vehicles (MOHANTY; CHOPPALI; KOUGIANOS, 2016), that exchange information, react to events and make decisions without human intervention. In an urban context, smart objects can offer services that greatly assist the public administration of cities and businesses (ZANELLA et al., 2014). For instance, IoT can be of great help in spreading the required information and collecting critical data in emergencies, like accidents, fires, flooding.

The publish-subscribe paradigm is often adopted by smart objects in smart cities (ANTONIC et al., 2015). In this case, smart objects (publishers) send data to the infrastructure independent of the application (middleware). Clients register as subscribers for certain events via the middleware and receive events of interest (BALDONI et al., 2005), (EUGSTER et al., 2003); exploiting the functional decomposition between the layers (BALDONI et al., 2005) and the decoupling of the communicating entities in time and space (EUGSTER et al., 2003). Publishers notify subscribers through the middleware (RAZZAQUE et al., 2016). Publish-subscribe middlewares carry out the management of events sent to subscribers (HERNANDES et al., 2020).

Therefore, a publish-subscribe middleware provides a mechanism for selecting event notification delivery (CARZANIGA, ROSENBLUM, 2001). The selection separates published notifications according to subscriptions. Concretely, the publish-subscribe middleware applies a filter to verify the match of event notification and subscriptions. So, only the event notifications that correspond to the subscriptions are delivered to subscribers (i.e., notification delivery). These middlewares facilitate the

development of applications and optimize the communication in the network (ANTONIC et al.m 2015).

However, the simple event matching allows all subscribers receive the events that realized the subscriptions. But, in many services like emergency services, this is not appropriate because only the subscribers who need to handle the emergency must be notified. So, an event double-filtering is necessary. In emergency services it is more efficient to notify subscribers who can respond to the event more quickly. Besides, not all subscribers must be notified, but only those required to attend the event (HERNANDES et al., 2020).

In (RAZZAQUE et al., 2016) a survey introducing the key characteristics of middlewares for IoT is presented and (BELLAVISTA, CORRADI, 2014) discusses a study about middleware publish-subscribe evaluating QoS. However, these studies do not consider requirements related to middlewares for emergency services, specially according to event filtering to meet the requirements of delay-sensitive applications. This paper goes in a different direction, presenting a survey of publish-subscribe middlewaes for smart cities including main concepts, subscription models, architectures, event routing, and a comparison of selected middlewares. In particular, we evaluate the performance of event filtering to modify the event notification for emergency services in smart cities from the perspective of event double-filtering. Besides that, we present enabling technologies for smart citiest. Urban IoT challenges in smart cities for emergency response services are also explored.

The paper is organized as follows. Concepts about smart cities and challenges related to IoT in smart cities for emergency response services are presented in Section 2. Publish-subscribe middlewares for smart cities are discussed in Section 3. Section 4 presents a comparation between middlewares publish-subscribe for Smart Cities. Finally, Section 5 concludes the paper.

SMART CITIES

Smart City is an urban environment that uses ICTs to increase the quality of the city processes and their effectiveness (KHAN et al., 2012). In (UBEDA, 2018), authors present a Smart Sustainable City concept "*to improve the quality of life, the efficiency of urban services and operations and competitiveness, while ensuring the needs of present and future generations in economic, social and environmental aspects*}". The IoT is the core of smart cities applications by supplying the connectivity between processes and smart objects.

Technological advances mainly drive IoT, but it moves towards the needs of users in a specific context, such as some services and applications for smart cities (VERMESAN et al., 2011). By connecting smart objects to the Internet, IoT has enabled the emergence of a large number of smart applications. Smart objects have limited communication, processing and sensing capabilities (KORTUEM et al., 2009). They can be providers and consumers of services to and from other objects or applications and can be remotely controlled. In addition, the produced data can be used for analysis and decision making. IoT provides the means for device integration and communication, while the Smart City can be seen as the ecosystem of applications and service providers.

Some examples of applications that offer possible services in smart cities using IoT in the urban environment are shown in Figure 1. These services meet the United Nation's sustainable development goals, especially Sustainable Cities and Communities (https://brasil.un.org/pt-br/sdgs/11). Smart Grid consists

of an electrical network, a communication network, as well as hardware and software for network monitoring and control (AL et al., 2015). Many cities around the world are investing to move their traditional power grid to a Smart Grid (COLAK, BAYINDIR, SAGIROGLU, 2020).

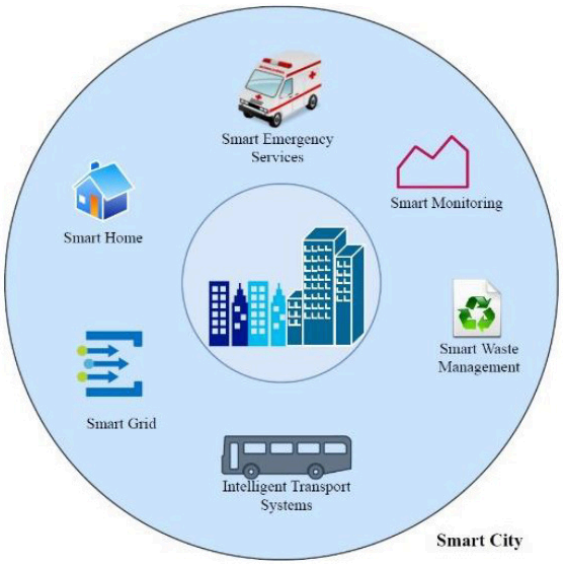

Figure 1. Examples of some smart city applications.

Intelligent Transportation Systems (ITSs) also offers several benefits: improvement of safety, efficiency and convenience of land transport, for people and for transport vehicles (DIMITRAKOPOULOS, DEMESTICHAS, 2010). Smart parking, based on sensors deployed on the streets and smart displays, reduces the searching time for a parking space, resulting in less CO_2 emissions, less traffic.

Smart Home solutions provide lighting, power and temperature control from everywhere. In addition, safety and protection are enhanced by alarms, temperature and motion sensors. Smart Monitoring can be applied in several areas: air quality, noise, traffic congestion, structural health of buildings, among others. Smart Waste Management uses smart bins to detect the load level. This simple information can optimize the routes of the trucks and reduce the costs of garbage collection. In (GUTIERREZ et al., 2015), authors present a smart garbage collection system. Sensors send collected data to a server over the Internet. Thus, the daily selection of bins to be collected is monitored and optimized by a system that calculates routes. The system makes decisions based on the level of daily trash and based on predictions of the future state and traffic congestion.

Smart Emergency Services allow mobile units to move to the location of the emergency in order to provide services as soon as possible. The provision of Smart Emergency Services occurs in at least two phases:

- Services triggering: After the notification, the service sets the urgency and intensity of the action in order to decide how many and which mobile units are needed.
- Services execution: Mobile units are transferred to the emergency location.

To provide Smart Emergency Services is a complex task. Therefore, smart cities should provide computational systems, as a middleware, to overcome the challenges. These computational systems should require minimal human intervention for achieving high efficiency. Drones could supervise the emergency care and smart devices could be deployed around the city in order to detect emergencies or to activate mobile units (HERNANDES et al., 2019). In addition, mobile units could have sensors deployed in mobile units could gather data from the environment.

CHALLENGES

Urban IoT in smart cities faces several challenges, especially for emergency response services:

- Naming -- Each smart object requires a unique identity. An identity management system is necessary to manage the identity of objects and an efficient naming mechanism.
- Interoperability -- Many devices use different technologies and services. The standardization of these technologies and services is critical to provide interoperability.
- Security and privacy -- IoT devices are commonly constrained. Soon, they are vulnerable to threats and attacks (KHAN et al., 2012). In addition, applications for smart cities use personal information. Strategies to overcome challenges related to security and privacy include authentication to verify the identity of devices and access control policies. Moreover, protocols must implement end-to-end secure communication mechanisms (ZANELLA et al., 2014).
- Geo-distribution and low latency -- Smart objects can be distributed in the city and low latency is necessary for many applications. Fog Computing, a distributed computational paradigm specially placed between IoT objects and Cloud Data Centers, that is at the edge of the network.
- (AKRIVOPOULOS ET AL.), 2017), may be exploited for minimizing latency. However, Fog Computing is not able to provide functionalities such as complex data analysis, data access to large numbers of users and historical data storage.
- Mobility support and location awareness - Smart objects move around the city and often need to connect to another object or network to provide data and event notifications. Mobility support is essential, as well as location awareness.
- Context-aware computing -- Smart applications employ context-aware computing to store context information associated with smart objects. It is important to send this information to a cloud computing platform for storage and knowledge discovery.
- Event model -- It is necessary to propose solutions for applications that do not require notifying all objects about a specific event.

The event model is the focus of this paper apply to emergency response services.

ENABLING TECHNOLOGIES

IoT technology as devices (i.e., smart objects) and wireless communication (MAHMOOD, ZUBAIRI, 2019), (COSTA, DURAN-FAUNDEZ, 2018) have driven the development of new services and applications for Smart Cities. Therefore, the characteristics of these smart objects and how they communicate are significant for the development of services and application for Smart Cities, as well as other contexts like a simple smartphone application.

The infrastructure of smart cities is constructed using various types of smart objects that work collaboratively to perform uncountable tasks (MAHMOOD, ZUBAIRI, 2019).

a) Devices

The features of smart objects and how they communicate directly impact on the development of new applications and services for smart cities (ZANELLA et al., 2014). Figure 2 shows the devices and the communication flow between ones. Normally in the smart city model, the information exchange is established between three main types of devices:

- Backend Servers: Collect, store and process data from smart objects to provide information for smart city services. Backend Servers are a fundamental component of an urban IoT and facilitate the access to the services.
- Gateways: Interconnect IoT devices to the main communication infrastructure of the system (backend servers). Gateways provide the interconnection between link-layer technologies in the core of the IoT network and IoT peripheral nodes.
- IoT Nodes: Usually called smart objects, IoT nodes are heterogeneous devices that generate the data delivered to the backend servers. Many times, IoT nodes perform some action in the environment like actuators.

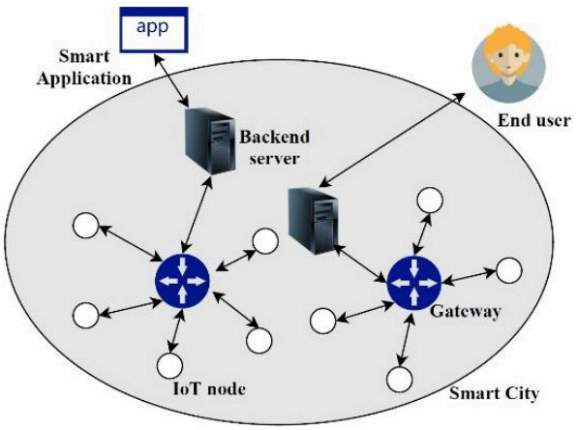

Figure 2. Smart city devices and the communication flow between ones.

The heterogeneity of smart objects, such as smartphones, laptops, sensors, actuators, and RFID tags, arises because of the different capabilities (i.e., processing power, memory, communication and energy) and characteristics of the applications and services. They can interact with each other and make decisions based on context, circumstances or environments (smartness). Interactions are highly dependent on their physical context, such as localization, surrounding object and presence of people (context-aware computing). Besides, interactions between smart objects can happen when an object enters the communication range of each other, leading to the spontaneous generation of events. Typically, an event is generated and pushed into the system without human intervention when an interaction with a smart object occurs (ROSE et al., 2015). The spontaneous generation of events among a large number of objects can produce an enormous amount of data. This large number of events reduces the event processing capacity of the objects and cause network congestion.

Smart objects can interact in a small environment e.g.}, office, home, store) or in a huge one in a distributed way (ZANELLA et al., 2014). Thus, IoT presents different scales, from global to local, depending on the application area. Often, the network is decentralized, dynamic, and unstructured (ZANELLA et al., 2014).

The communication between smart objects follows several communication models (TSCHOFENIG ET AL., 2015) and (ARA et al., 2016):

- **Device-to-Device (D2D):** Smart objects communicate directly with each other. An intermediate application server is not required.
- **Device-to-Cloud (D2C):** A cloud service is directly connected to smart objects.
- **Device-to-Gateway (D2G):** A cloud service is connected to smart objects through an application-level gateway service, see Figure 3 (a).
- **Back-End Data Sharing:** Smart objects send and receive data to and from many application service providers. This approach is an extension of the D2C communication model, see Figure 3.

Depending on the application or service, smart objects can use one or more communication technology at the same time (TSCHOFENIG ET AL., 2015). The development of applications for mobile devices can facilitate interaction with IoT objects and the entire system. Figure 3 shows examples of protocols that can be used in each communication model.

Besides that, many softwares (e.g., operating systems) and hardware platforms (e.g., Arduino and Raspberry PI) were developed to execute IoT applications in smart cities (AL-FUQAHA et al., 2015). Cloud Computing is another part of the IoT. Cloud computing is a shared pool of conFigureble resources (e.g., servers, storage, networks, applications and services) that can be quickly offered and released with minimal management or interaction effort with the service provider (MELL et al., 2011). Cloud Computing platforms refers to software systems, hardware, and the possibility of developing applications directly in the cloud offered as a service, found in data centers (i.e., service providers) that offer these functionalities (ARMBRUST ET AL, 2010). Facilities are provided by these platforms for smart objects to send their data to the cloud (AL-FUQAHA et al., 2015). Thus, collected data or big data can be processed in real-time, near real-time, or offline to generate knowledge to end users (AL-FUQAHA et al., 2015). Azure Cloud and Amazon Web Services are examples of cloud platforms (KOTAS et al., 2018).

PUBLISH-SUBSCRIBE MIDDLEWARES

An event notification service is defined as the event-based middleware which implements an event-based protocol (i.e., event model), offering event-based communication to an event-based application (i.e., event system) (MEIER, CAHILL, 2002). An event model consists of a set of rules describing how event-based communication occurs (CUGOLA, JACOBSEN, 2002). An application that uses a middleware is called event-based system. The components of the application interact through event notifications. Applications using an event-based middleware are organized as a collection of standalone components, the clients that emit the subscriptions for the event classes they are interested (i.e., subscribers)

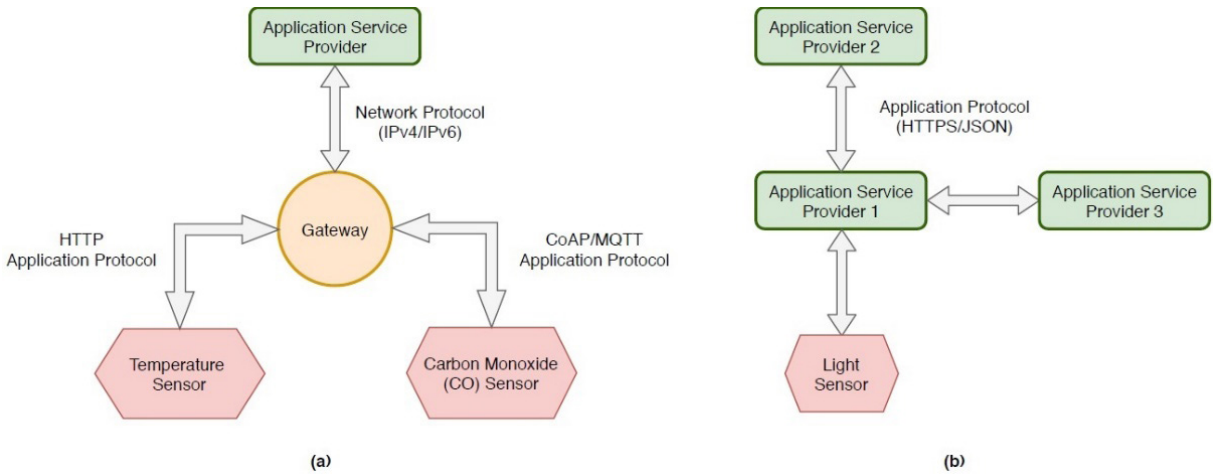

Figure 3. (a) Device-to-Gateway Communication Model; (b) Back-end Data Sharing Communication Model (TSCHOFENIG ET AL., 2015), (ARA et al., 2016).

and the publishers that send event notifications (CUGOLA, JACOBSEN, 2002). The subscriptions and the event notifications are sent to event-based middleware. It manages the subscriptions and the event delivery. Figure 4 shows a generic view of an event-based middleware.

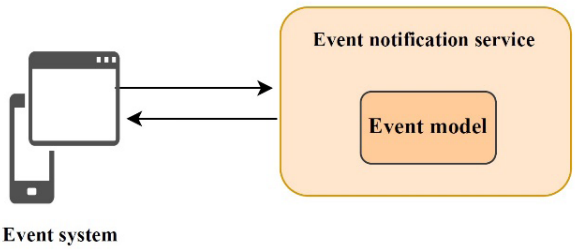

Figure 4. An event-based middleware.

The event notification service must implement (CARZANIGA, ROSENBLUM, 2001):

a) The selection of the event, which consists of determining which events or notifications correspond to the subscriptions, also known as matching;

b) The delivery or notification of the event, which forwards the corresponding events from the publishers to the subscribers.

For the matching in the notification selection, filters should be defined in subscriptions by subscribers. This means specifying a set of attributes and restrictions on the values of these event attributes. Matching is one of the processes performed by the event notification service and its function is to check if an event matches to a subscription, thus determining whether it will be triggered for the subscriber or not (Baldoni et al., 2005).

SUBSCRIPTION MODELS

Subscriptions can be made in different ways (EUGSTER et al., 2003) , based on topics, based on content, based on type, ontologies and location-adaptable. In the topic-based subscription, subscribers subscribe to a topic and all events related to this topic are received by subscribers. The topic is like a logical channel connecting the publisher with all subscribers. Therefore, subscribers are known a priori and topics can also be organized in hierarchical form. The advantage is the simplicity of implementation. However, the disadvantage of this subscription model is the limited expressiveness due to its static behaviour.

In the content-based subscription, a subscriber subscribes to an event by specifying filters in the subscriptions. Therefore, subscribers cannot be determined before publication. It is a more expressive and general model than the previous one. However, it requires protocols that will have greater overhead when publishing events because of the comparisons that need to be made.

In the type-based subscription model, the subscriber specifies in the subscription the type that wants to receive and event matching is performed based on the specified type. Therefore, events are filtered according to their type. In this model, events are objects with attributes and methods. Additional expressiveness can be acquired by applying content-based filters in the context of types to express restrictions on the values of objects.

Other subscription models can be found such as ontologies (i. e., concept-based publish-subscribe) and the location-adaptable publish-subscribe (Baldoni et al., 2005). Ontology subscription allows subscriptions based on a specific domain. The location-adaptive subscription allows you to subscribe based on your location. Figure 5 shows the subscription models presented. The advantage and disadvantage of subscription models are summarized in Tabela 1.

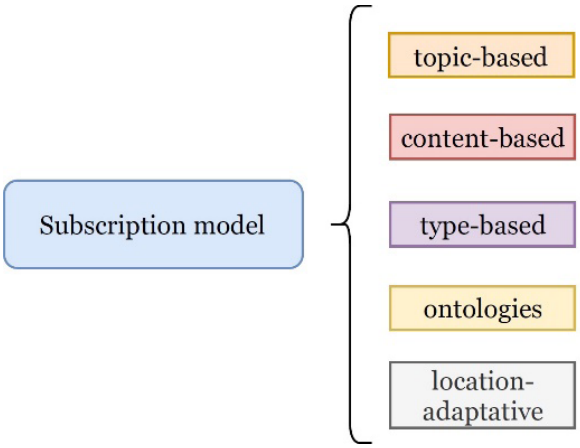

Figure 5. Publish-subscribe middleware subscription models.

Subscription model	Main Advantage	Main Disadvantage
Topic-based	Simplicity of implementation	Limited expressiveness
Content-based	Expressive and general	Sophisticated protocols for filtering (hard)
Type-based	Type is ensure at compile-time	Sophisticated protocols for filtering (easy)
Ontologies	Describe events at a higher level of abstraction	Sophisticated protocols for filtering (median)
Location-adaptative	Ability to monitor subscriber mobility	Off-location subscribers do not receive events

Tabela 1. Subscription Models.

ARCHITECTURE OF THE MIDDLEWARE

The architecture of middleware can be centralized, distributed or even peer-to-peer (CUGOLA, JACOBSEN, 2002) as illustrated in Figure 6. In the centralized architecture, there is a central entity responsible for managing subscriptions, including storage, and forwarding events acting like as a dispatcher of events (Baldoni et al., 2005). This central entity is known as a broker or event broker. Asynchronism is implemented by having producers sending messages to the broker, which stores and then forwards messages to subscribers on-demand (EUGSTER et al., 2003). However, it depends on the method of receiving messages. If push, broker forwards messages to subscribers, otherwise, if pull, the subscriber retrieves messages from the broker. In this architecture, this single central broker introduces a point of failure, as well as decreases scalability when the rate of events or the number of subscribers and publishers grows significantly. However, it allows easy management of available resources and simple implementation.

In the *peer-to-peer architecture* (BELLAVISTA, CORRADI, 2014)) the flow of events goes from publishers to subscribers without intermediate nodes. The functions of collecting subscriptions, matching and routing are performed by the participants. In a peer-to-peer architecture participants may be publishers and subscribers. Asynchronism is implemented by the use of smart communication primitives which employ routing and storage mechanisms in both publisher and subscriber processes (EUGSTER et al., 2003). The gain of this architecture is the suitability for the dissemination of events with small-scale geographic deployment due to latency and high throughput of events among a limited number of participants (HERNANDES et al., 2020).

In the *distributed architecture*, there is no central entity (EUGSTER et al., 2003) in the event notification service, which reduces the network load and increases scalability. The distributed architecture is known as *Broker Overlay (BELLAVISTA, CORRADI, 2014)* in which the interactions between the participants follow the client-server model. Participants may be brokers, subscribers or publishers. The middleware is organized as a network of distributed servers (application-level routers), also known as

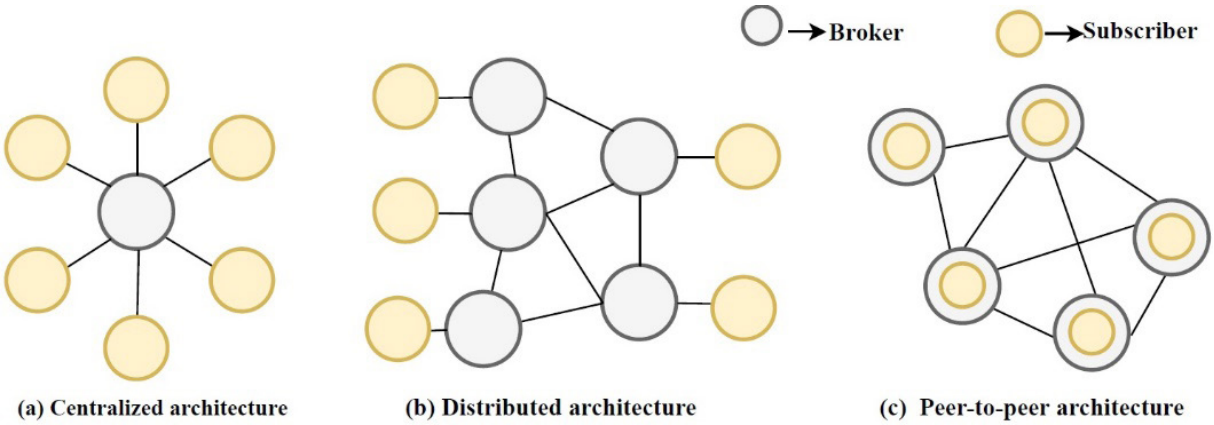

Figure 6. Broker architecture in a publish-subscribe middleware (EUGSTER et al., 2003).

dispatching servers, which collaborate in collecting subscriptions from subscribers and routing events (MOTTOLA, CUGOLA, PICCO, 2008). This network of distributed servers can be viewed as an overlay network over a physical network. Consequently, the event matching and routing are performed by algorithms distributed over the network. The event notification service ends up being a set of distributed nodes that dispatch published events only to interested subscribers. Communication is asynchronous and anonymous without the need for an intermediary entity, as is the case with centralized architecture. This type of architecture makes it possible to manage a large number of broker, subscribers, publishers and events because the responsibility and complexity of the matching functions and the routing decisions are divided among the overlay network of brokers. The topology of the brokers and the strategies to manage subscriptions and to deliver events change from middleware to middleware. Tabela 2 summarizes the middleware architectures.

Architecture	Main Advantage	Main Disadvantage
Centralized	A centralized component introducing a central point of failure decreasing scalability according to the number of subscribers, publishers, or the rate of events grows significantly	Easy administration of available resources and simple implementation
Distributed or Broker Overlay	Allow manage a large number of participants and events	Complex administration of available resources and hard implementation
Peer-to-peer	Allows the dissemination of events with small-scale geographic	Unsuited for the dissemination of events with high-scale geographic

Tabela 2 . Broker architecture in middlewares publish-subscribe.

BROKER TOPOLOGIES

The **broker topology** represents the logical organization of the brokers in a distributed middleware architecture. Many topologies can be created depending on the environment in which the brokers are located. Examples shown in Figure 7 include the following topologies: hierarchical, acyclic peer-to-peer and cyclic peer-to-peer (i.e.,

general peer-to-peer). The dotted arrows represent subscriptions.

The **hierarchical topology** is structured in levels of broker in which the higher level is the root. Peers of connected brokers communicate like in the client-server model, asymmetrically. A broker can have only a connection with their up broker and many connections of input from other brokers, thus forming a hierarchy. This topology is represented by an oriented graph and has a root broker.

In the **acyclic peer-to-peer topology**, peers are brokers that communicate symmetrically. A protocol that allows a bidirectional flow of subscriptions, announcements and notifications is adopted. An acyclic non-directed graph is used to represent this topology. In the **general peer-to-peer topology**, bidirectional communication between two brokers also is allowed. Hybrid topologies (CARZANIGA, ROSENBLUM, 2001) are also possible.

In the acyclic peer-to-peer and hierarchical topologies, the disadvantage is the lack of redundancy. However, the main disadvantage of the hierarchical topology is the overhead of the highest brokers in the hierarchy. If a broker fails, the brokers connected to it are isolated from the rest of the network. Routing algorithms that should handle failures if they happen. General peer-to-peer topology is more advantageous than others, since it offers redundancy due to the various paths between brokers. However, specifics routing algorithms must be developed to avoid cycles. Tabela 3shows a summary about the above discussed topologies.

Topology	Main Advantage	Main Disadvantage
Hierarchical	Events are not broadcast to the entire hierarchy	Overhead of the highest brokers and lack of brokers redundancy
Acyclic peer-to-peer	There is an unique path between brokers	Lack of brokers redundancy
General peer-to-peer	Offers redundancy due to the various paths between brokers	Specific routing algorithms must prevent cycles

Tabela 3. Topologies Comparison of publish-subscribe middlewares.

EVENT ROUTING

The delivery of an event to subscribers that did a corresponding subscription to the event before publication is event routing (Baldoni et al., 2005). Normally, in a distributed architecture, once the organization and the topology of the brokers

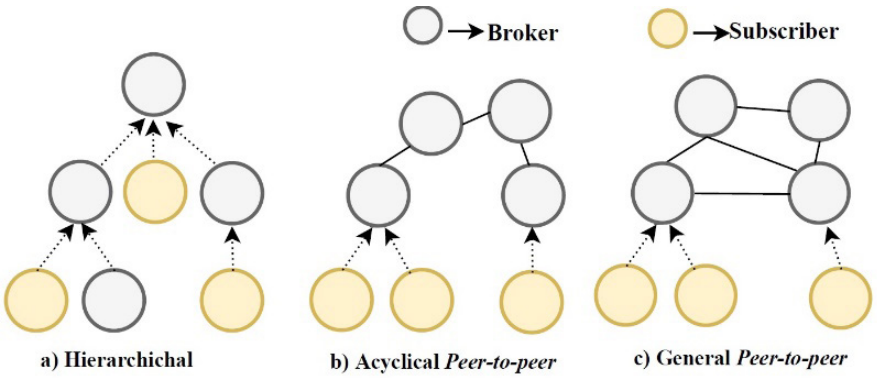

Figure 7. Topologies for the brokers in a distributed middleware architecture.

are defined, appropriate routing paths are established to ensure that published notifications will be correctly delivered to all subscribers (CARZANIGA, ROSENBLUM, 2001). There are basically three event-routing categories (RAZZAQUE et al., 2016): (1) Flooding algorithms: event and subscription flooding; (2) Selective algorithms: filter and rendezvous-based; (3) Event gossiping algorithms.

In the **event flooding algorithm**, each event is propagated from the publisher to all participants in the event notification service. In the **selective algorithms** (filter-based and rendezvous-based), events are compared with subscriptions to delivery for interested subscribers. In the *filter-based algorithm*, when an event reaches a participant it is compared with the stored subscriptions *match-first* (CHAND, FELBER, 2003) and forwarded only to interested subscribers. Rendezvous-based algorithm all subscriptions for the same event can be stored in the same participant and the delivery of events is simplified, consisting of the creation of a diffusion tree with a single root starting from brokers and spreading to all subscribers. In **gossip-based algorithms**, each participant exchanges information with one or some participants (COSTA et al., 2003). They do not require maintaining the event routing data structure at each node. The bottleneck of this approach is a message overhead due to the redundancy of messages.

MIDDLEWARES FOR SMART CITIES

In a smart city, users can produce or consume information for the provision of smart services aimed at improving the quality of life of citizens. A user can be an event producer (publisher) or an event consumer (subscriber). A middleware is a mediator (HERNANDES et al., 2020). It must provide storage and management of subscriptions and efficient event delivery (EUGSTER et al., 2003). In this paper, we analyze several event based middlewares regarding their suitability to urban IoT, specially for emergency services response. We study subscription models, architecture, event routing and, specially, event filtering.

Siena (CARZANIGA, ROSENBLUM, 2001) presents a distributed architecture that implements several topologies. The Siena middleware adopts a content-based subscription model and employs advertisements to construct paths for subscriptions and to send events. It is designed to work on wireless environments. The Hermes middleware (PIETZUCH, BACON, 2002) presents a distributed architecture and a cyclic peer-to-peer topology. It uses an adaptation of the subscription model based on type and content. Besides, Hermes middleware uses selective event routing such as rendezvous and is designed for wired networks. Siena and Hermes do not allow to choose only a few subscribers among those in which the matching is affirmative.

Steam (MEIER, CAHILL, 2002) is an event service to distribute events between publishers and subscribers that resides on mobile devices. It was specially designed for the domain of traffic management applications. In this scenarios, subscribers (cars and ambulances) and publishers (e.g., traffic lights, cars) share the same physical area. Events are related to the current traffic situation. Publishers are in charge of defining the geographical area of the events and the middleware is in charge of delivering events to subscribers physically located in the region. For this, Steam considers that all entities (publishers and subscribers) estimate their location. Steam also adopts a peer-to-peer architecture. Therefore, interactions between publishers and subscribers do not involve a

central infrastructure. Selecting subscribers considering only the distance from the event may not be efficient. This is because the subscribers closest to the event may not be available or may not be sufficient. Therefore, other factors must be considered.

SensorBus (RIBEIRO et al., 2005) has a centralized architecture in which a bus for communication between subscribers and publishers is used. The content-based subscription model is adopted. The middleware was implemented for use with sensor networks. Subscribers run in user computers and publishers run in sensor nodes. The method of receiving messages is pull, that is, the subscriber retrieves messages from the bus. The EMMA middleware (MUSOLESI, MASCOLO, HAILES, 2006) is an adaptation of the Java Message Service (JMS). It has a distributed architecture with epidemic routing and the subscription model is topic-based. EMMA works on ad-hoc networks. Publish-subscribe communication model was implemented and the point-to-point communication model. Mires (SOUTO et al., 2006) has a distributed architecture with a cyclic peer-to-peer topology. The subscription model of Mires is topic-based and for event routing it uses multi-hop. It was developed to be used along with sensor networks especially for environmental monitoring. The RUNES middleware (COSTA et al., 2005) was designed for a disaster scenario at a road tunnel. It has a distributed architecture with a hierarchical topology. Its subscription model is content-based and it was designed for wireless sensor networks. Always, events are sent to firefighters who have expressed interest, but without the possibility of sending events to the most suitable subscribers.

In the Publish-Subscribe Notification Middleware for Vehicular Networks - PSN (LEONTIADIS, 2007) vehicles are mobile sensors. They are publishers when informing about events (i.e., traffic conditions, accidents) and subscribers whenever receiving events. Subscriptions are locally stored at subscribers to perform matching. A navigation system evaluates a received event and recalculates a route according to the interest. Vehicles report the collected event to the nearest base station or to a WiFi hotspot using vehicle-to-vehicle routing or connecting directly to the a base station. PSN works with subscriber-side filters that do not help the selection of only the most suitable subscribers.

PSWare middleware (LAI, CAO, ZHENG, 2009) adopts a distributed architecture with hierarchical topology. The subscription model is content-based. It was designed for wireless sensor networks. TinyDDS (Tiny) (BOONMA, SUZUKI, 2012) is a publish-subscribe middleware that implements an OMG (Object Management Group) publish-subscribe protocol specification. The middleware offers interoperability between access networks and wireless sensor networks, data aggregation and event routing. It was designed for wireless networks. Three strategies can be applied for event routing: routing based on spanning trees, distributed hash table and Moonson. A gateway called DDS (*Data Distributed Service*) is used for communication between wireless sensor networks and the Internet. The gateway interacts with TinyDDS middleware that runs on sensor nodes. When a DDS subscriber subscribes to a topic, the subscription is sent over the DDS access network. The DDS gateway store the subscription into a subscription list. Thus, when a sensor node publishes an event, the event is distributed on the sensor network and get by the DDS gateway. If the topic of the published event matches a topic in the subscription list, it is sent to interested subscribers according to the

configured protocol. The routing algorithm does not send an event to only the most suitable subscribers.

In PRISMA middleware (SILVA et al., 2014), subscriptions are made by topics and the architecture is centralized. It is a resource-oriented middleware for wireless sensor networks in which each node can provide a resource or a set of resources. For this, PRISMA adopts REST (Representational State Transfer) to support interoperability between other networks and the middleware. Besides that, REST facilitate access to network data and their interfaces interact with client applications. The communication is realized by a WebService and a broker is used for asynchronous communication. The middleware adopts the publish-subscribe paradigm to notify subscribers about events. However, it is not possible to selectively send events.

Apache Kafka (APACHE, 2018) enables to send messages between publishers and subscribers through topics. It is a distributed streaming platform. When a producer generates a message on the topic, consumers of that topic are notified with a copy of the published message. Topics are distributed on brokers, but before the topics are divided into partitions. However, routing does not allow selective event delivery.

RabbitMQ (RabbitMQ, 2019) is an open-source messaging broker originally developed to implement the AMQP protocol, but also implements other protocols such as MQTT. It supports multiple communication models such as peer-to-peer, publish-subscribe, and request/reply. It uses a simple consumer model and a smart broker delivers messages to subscribers according to rate in which they can receive messages. Selective event delivery is not allowed.

In (ZHOU et al., 2019), the authors propose NISU (New Index Structure on Uncertain Data), an event matching solution for uncertain data (inaccurate data). The Probabilistic Skyline (P-Skyline) model is used to filter events based on subscription adopting constraint satisfaction criteria for the matching. The subscription is realized by content and it is sent to the nearest broker. After receiving the event, the broker checks and process the content. The broker uses the P-Skyline to filter messages from the subscribers. So, it finds a matching subscription list for each subscriber and delivers the event to them. In (OZTURK, OZDEMIR, 2019), a content-based (CB) routing protocol is proposed for a point-to-point communication model. Event subscription and forwarding includes a filter for sending events. In this way, node transmission with its neighboring nodes uses the same filter. The node interested in the event is reached by passing through some nodes.

MiddSS middleware implements an event-based communication model called SmartMid-Event in order to send events only to the most suitable participants. The middleware was proposed by the authors of this paper in (HERNANDES et al., 2019). and (HERNANDES et al., 2020) . MiddSS applies a double filtering: (i) topic-based and (ii) the most suitable subscriber (most suitable filtering). The second filter uses a *fitness value* calculated on subscriptions. SmartMid-Event was proposed to provide a communication model for middlewares designed for emergency services. Subsequently, SmartMid-Event was extended to other smart services, such as smart transportation (HERNANDES et al., 2020). The aim is to notify the necessary and more appropriate mobile units (i. e., subscribers) in the shortest possible time in order to save lives (HERNANDES et al., 2020). The SmartMid-Event uses a *fitness value* to choose the more appropriate available

subscribers. The subscriber goes to the event to handle it. For *fitness* calculation, the following assumptions were considered:

- The subscriber is unavailable while handling the event.
- The highest the subscriber ranking, the greatest is its fitness value.
- Intense traffic means a longer time for a subscriber to reach a reference point of the event.
- The longer the work journey of a subscriber or the team associated with it, the greater is the level of fatigue associated with that subscriber.

The **fitness metric** was defined based on three main parameters, the *traffic level*, the *fatigue level* and the *ranking level*. The traffic level between the subscriber and the reference point associated with the event. It takes into account the traffic condition as being low, moderate, or intense. The fatigue level considers the accumulated time period of attendance of events. The *ranking level* measures the degree of satisfaction that users of the service confer on the subscriber. The better the ranking, the greater the value of fitness.

In this way, SmartMid-Event communication model allows a differentiated notification of subscribers. A middleware can implement SmartMid like MiddSS. MiddSS is composed of subscribers, publishers and brokers. It presents a distributed organization in which distributed brokers cooperate with each other to deliver events to the most suitable subscribers (HERNANDES et al., 2020). The brokers are part of the fixed infrastructure and subscribers can be mobile or fixed nodes, acting as distributed actuators that perform some activity when notified. Device-to-Gateway communication model is adopted. The broker is a gateway that communicates with subscribers and other brokers, makes event forwarding decisions and transmits the collected data to a cloud service. It focuses on the activating mobile units phase and supports subscribers mobility.

Moreover, MiddSS uses overlay networks to notify the subset of most suitable subscribers. Events happen at a reference point and brokers construct an overlay network for each service and reference point. So, when an event occurs, the overlay network is located and subscribers with the highest *fitness values* are activated. In each connection of the root node with another node, the *fitness value* is calculated. Each overlay network follows a hierarchical tree topology dynamically updated according to the movement of subscribers in the city. This logical organization is used to minimize the delay to forward messages between the middleware entities. Figure 8 shows the example of MiddSS event notification. The *publisher broker* is the root node and the mobile units are the leaf nodes that are notified. The broker b_1 receives the event and processes it. For the event attendance, two mobile units are required. The *fitness values* allows that broker b_1 chooses the subscribers. Therefore, it sends a publish message to broker b_2, which in turn notifies the mobile unit v_{13} and the subscriber v_{12}. Each subscriber is activated according to the fitness value. Red lines in Figure 8 represent the flow of messages related to the event.

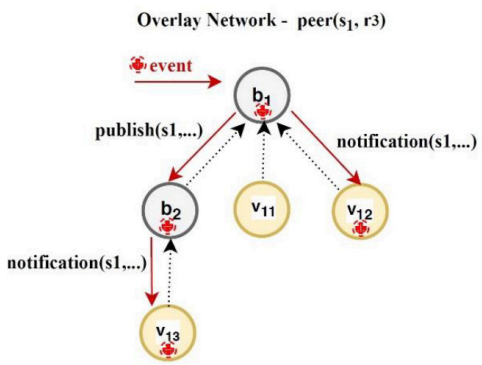

Figure 8. Example of MiddSS's Event Notification (HERNANDES et al., 2020) .

Therefore, brokers take coordinate decisions (i.e., the system state is shared) about which subscribers must handle the event. This process is initiated by the root broker of the overlay network where the event took place. Updates of *fitness values* are performed to ensure the computational effectiveness of the event system that adopts MiddSS. *Fitness values* could be updated periodically or in reaction an event.

COMPARISON OF MIDDLEWARES FOR SMART CITIES

The Figure 9 presents a synthesis of the main middlewares for smart cities. SensorBus, PRISMA and Apache Kafka use the pull method for retrieving messages, while the other middlewares use the push method. All these middlewares implement the publish-subscribe paradigm, except EMMA. EMMA middleware implements the point-to-point and publish-subscribe paradigms. The programming style provided by the evaluated middlewares is not flexible enough. The lack of flexibility found in the programming style of event based middlewares occurs due to the adopted models for sending events, that is the point-to-point and publish-subscribe models. In the point-to-point model, subscribers might register to specific queues, retrieve asynchronous messages and confirm the delivery. Publishers send messages to queues. This communication model is also called *one-to-one* communication, as each message is produced and consumed only once through a message queue. In one-to-one communication model, only one subscriber consumes the message retrieved. However, it allows that several subscribers to connect to the queue. Moreover, the messages are stored inside the queue until a subscriber is ready for retrieving, and they are always delivered.

In the publish-subscribe model, subscribers or event consumers subscribe to events of interest and publishers or event producers produce events, which are asynchronously sent to all registered subscribers (EUGSTER et al., 2003). That is, the middleware receives the events by publishers and delivers the events to the subscribers who expressed their interest to receive them by subscriptions (OZALP, TEKIN, 2014). It is the role of the middleware to route subscription messages to publishers and event messages to subscribers. The publish-subscribe model is used as a synonym for the one-to-many communication paradigm, guarantees message delivery for all subscribers (EL et al., 2014). This type of middleware has the advantage the strong decoupling of time, space, and synchronism between publishers and event subscribers (EUGSTER et al., 2003). Time decoupling occurs because subscribers and publishers may not participate in the interaction at the same time. Space decoupling happens because publishers and subscribers do not need to know each other. In addition, synchronization decoupled publishers and subscribers can perform simultaneous activities while events are produced and consumed. As a consequence, producers and subscribers are independent and the global middleware architecture becomes more complex (MAGNONI, 2015).

Therefore, most middlewares, except MiddSS, do not allow a subset of subscribers with high priority (or most suitable according to some specific event based system or application criteria) to be notified about an event. Notifying a subset of high priority subscribers is a key requirement for many urban IoT applications, such as smart garbage collection system and emergency service calls, making MiddSS very relevant in this context.

MIDDLEWARE		Siena	Hermes	Steam	SensorBus	EMMA	Mires	Runes	PSN
Subscription	Topic					X	X		X
	Content	X	X	X	X			X	
	Type		X						
Architecture	Centralized	X			X				
	Distributed	X	X				X	X	X
	Peer-to-peer			X		X			
Routing	Event flooding								X
	Selective	X	X						
	Multi-hop						X	X	
	Pull				X				
	Epidemic			X		X			
Event filtering	Publishers			X		X	X	X	
	Subscribers			X	X	X			X
	Set of participants	X	X						
	MSF*								

MIDDLEWARE		PSWare	Tiny	PRISMA	Kafka	RabbitMQ	NISU	CB	MiddSS
Subscription	Topic		X	X	X	X			
	Content	X					X	X	X
	Type								
Architecture	Centralized			X					
	Distributed	X	X		X	X	X	X	X
	Peer-to-peer								
Routing	Event flooding								
	Selective		X		X	X	X		X
	Multi-hop	X						X	
	Pull			X					
	Epidemic								
Event filtering	Publishers	X	X						
	Subscribers			X					
	Set of participants				X	X	X	X	
	MSF*								X

*MSF:Most suitable filtering.

Figure 9. Comparation of middlewares for Smart Cities.

FINAL CONSIDERATIONS

This paper presents a survey about publish-subscribe middlewares for emergency response services in smart cities. We evaluated in each middleware the event filtering to verify their suitability for attendance emergencies. So, event filtering was evaluated in each middleware. MiddSS middleware allows differentiated notification in emergencies. MiddSS middleware allows differentiated event notification in emergencies. It notifies the most suitable subscribers. The MiddSS mechanism based on a fitness function allows select specific subscribers to receive an event, showing suitable for emergency services response. Publish-subscribe middlewares are suitable for managing time-sensitive events between smart city applications.

Emergency management and response is a critical service that must be provided by cities. Nowadays, ICTs have helped in this regard. As challenges and future research directions for middlewares for emergency services in smart cities, we highlight the following:

- Geo-distribution and low latency: emergencies happen in any place in the city and geo-distribution is important for attending them as soon as possible. Infrastructure for this must be provided for the smart objects to communicate with low latency. To save lives is the main goal.

- Mobility support and location awareness: in an emergency attendance, smart objects move around the city and have to connect to another object or to a network in order to provide data to emergency control centers or to notificate other smart objects. Mobility support must be offered. Location awareness is important for supporting emergencies. Frequently it is important to know the location of the mobile units; for instance, hospitals can be located in the emergency proximity and the mobile units can be directed to them.

- Security and privacy: authentication to verify the identity of devices and users and access control policies must be provided, especially in the back-end servers because they can have data about emergency situations that help decision-making. In many cases, protocols must implement end-to-end secure communication mechanisms.

- Interoperability: smart objects from different manufacturers communicate with each other in a collaborative network to respond to emergency situations, thus they must interoperate.

- Context-aware computing: the store of context information associated with smart objects can help enhance the quality of emergency response services provided in the city.

Finally, publish-subscribe middlewares are very relevant in smart cities. They can be developed for specific applications will be optimized for an alone application or a specific group of applications.

ACKNOWLEDGMENTS

The authors would like to thank CAPES and CNPq for the financial support and UTFPR-Campus Guarapuava for the financial aid with the publication fee.

REFERENCES

(QIAN ET AL., 2019) Y. Qian et al. "The internet of things for Smart Cities: Technologies and applications". IEEE Network, v. 33, n. 2, 2019, p. 4-5.

(ALKANDARI et al., 2012) ALKANDARI, A; ALNASHEET, M; ALSHAIKHLI, IFT. Smart cities: survey. Journal of Advanced Computer Science and Technology Research, v. 2, n. 2, p. 79–90, 2012.

(UNITED, 2017) United Nations. "Population Division". Available in: www.un.org/en/development/desa/population/, Dec. 2017.

(ALDELAIMI et al., 2020) M. N. Aldelaimi et al. "Building Dynamic Communities of Interest for Internet of Things in Smart Cities". Sensors, v. 20, n. 10, 2020, p. 2986.

(MOHANTY; CHOPPALI; KOUGIANOS, 2016) S. P. Mohanty, U. Choppali and E. Kougianos. "Everything you wanted to know about Smart Cities: The internet of things is the backbone". IEEE Consumer Electronics Magazine, v. 5, n. 3, 2016, p. 60-70.

(ZANELLA et al., 2014) A. Zanella et al. "Internet of things for Smart Cities". IEEE Internet of Things journal, v. 1, n. 1, 2014, p. 22-32.

(ANTONIC et al.m 2015) A. Antonic et al. "Comparison of the CUPUS middleware and MQTT protocol for Smart City services". International Conference on Telecommunications (ConTEL). IEEE, 2015. p. 1-8.

(BALDONI ET AL., 2005) R. Baldoni and A. Virgillito. "Distributed event routing in publish/subscribe communication systems: a survey". DIS, Universita di Roma La Sapienza, Tech. Rep, 2005.

(EUGSTER et al., 2003) P. T. Eugster et al. "The many faces of publish/subscribe". ACM computing surveys, v. 35, n. 2, 2003, p. 114-131.

(RAZZAQUE et al., 2015) M. A. Razzaque et al. "Middleware for internet of things: a survey". IEEE Internet of things journal, v. 3, n. 1, 2015, p. 70-95.

(HERNANDES et al., 2020) S. C. L. Hernandes et al. "A New Event Model for Event Notification Services Applied to Transport Services in Smart Cities". International Conference on Information Networking (ICOIN), 2020, p. 202-207.

(CARZANIGA, ROSENBLUM, 2001) A. Carzaniga, D. S. Rosenblum, A. L. Wolf. "Design and evaluation of a wide-area event notification service". ACM Transactions on Computer Systems (TOCS), v. 19, n, 3, 2001, p. 332-383.

(AL et al., 2015) AL NUAIMI, Eiman et al. Applications of big data to smart cities. Journal of Internet Services and Applications, v. 6, n. 1, p. 1-15, 2015.

(UBEDA, 2018) ÚBEDA, Reyna. ITU transforming cities in smarter and more sustainable. In: Third Meeting of the United for Smart Sustainable Cities Initiative, 26 Apr, Malaga, Spain, by ITU and UNECE. 2018.

(VERMESAN et al., 2011). O. Vermesan et al. "Internet of things strategic research roadmap. Internet of things-global technological and societal trends". 1, 2011, p. 9-52.

(KORTUEM et al., 2009) G. Kortuem et al. "Smart objects as building blocks for the internet of things". IEEE Internet Computing, v. 14, n. 1, 2009, p. 44-51.

(COLAK, BAYINDIR, SAGIROGLU, 2020) COLAK, Ilhami; BAYINDIR, Ramazan; SAGIROGLU, Seref. The effects of the smart grid system on the national grids. In: 2020 8th International Conference on Smart Grid (icSmartGrid). IEEE, 2020. p. 122-126.

(Dimitrakopoulos, Demestichas, 2010) G. Dimitrakopoulos and P. Demestichas. "Intelligent transportation systems". IEEE Vehicular Technology Magazine, v. 5, n. 1, 2010, p. 77-84.

(GUTIERREZ et al., 2015) J. M. Gutierrez et al. "Smart waste collection system based on location intelligence". Procedia Computer Science, v. 61, 2015, p. 120-127.

(MAHMOOD, ZUBAIRI, 2019) I. Mahmood and A. J. Zubairi. "Efficient Waste Transportation and Recycling: Enabling technologies for smart cities using the Internet of Things". IEEE Electrification Magazine, v. 7, n. 3, 2019, p. 33-43.

(COSTA, DURAN-FAUNDEZ, 2018) D. G. Costa and C. Duran-Faundez. "Open-source electronics platforms as enabling technologies for smart cities: Recent developments and perspectives". Electronics, v. 7, n. 12, 2018, p. 404.

(ROSE et al., 2015) Rose et al. "The internet of things: An overview". The Internet Society (ISOC), v. 80, 2015, p. 1-50.

(TSCHOFENIG ET AL., 2015) H. Tschofenig et al. "Architectural considerations in smart object networking". IETF RFC 7452, 2015.

(ARA et al., 2016) T. Ara et al. "Internet of Things architecture and applications: a survey". Indian Journal of Science and Technology, v. 9, 2016, n. 45, p. 1-7.

(AL-FUQAHA et al., 2015) A. AL-FUQAHA et al. "Internet of things: A survey on enabling technologies, protocols, and applications". IEEE communications surveys \&tutorials, v. 17, n. 4, 2015, p. 2347-2376.

(ARMBRUST ET AL, 2010) M. Armbrust et al. "A View of Cloud Computing". Communications of the ACM, vol. 53, n. 4, 2010, p. 50-58.

(KOTAS et al., 2018) C. Kotas et al. "A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing". IEEE International Conference on Consumer Electronics (ICCE), 2018. p. 1-4.

(KHAN et al., 2012) R. Khan et al. "Future internet: the internet of things architecture, possible applications and key challenges". International conference on frontiers of information technology. IEEE, 2012. p. 257-260.

(AKRIVOPOULOS ET AL.), 2017) O. Akrivopoulos et al. "On the deployment of healthcare applications over fog computing infrastructure". Computer software and applications conference (COMPSAC), v. 2, 2017, p. 288-293. IEEE.

(MEIER, CAHILL, 2002) R. Meier and V. Cahill. "Steam: Event-based middleware for wireless ad hoc networks". International Conference on Distributed Computing Systems Workshops, 2002, p. 639-644.

(CUGOLA, JACOBSEN, 2002) G. Cugola and H. A. Jacobsen. "Using publish/subscribe middleware for mobile systems". ACM SIGMOBILE Mobile Computing and Communications Review, v. 6, n. 4, 2002, p. 25-33.

(BELLAVISTA, CORRADI, 2014) P. Bellavista, A. Corradi, A. Reale. "Quality of service in wide scale publish—Subscribe system". IEEE Communications Surveys \& Tutorials, v. 16, n. 3, 2014, p. 1591-1616.

(MOTTOLA, CUGOLA, PICCO, 2008) L. Mottola, G. Cugola and G. P. Picco, G. P. "A self-repairing tree topology enabling content-based routing in mobile ad hoc networks". IEEE Transactions on Mobile Computing, v. 7, n. 8, 2008, p. 946-960.

(COSTA et al., 2005) P. Costa et al. "The RUNES middleware: A reconFigureble component-based approach to networked embedded systems". International Symposium on Personal, Indoor and Mobile Radio Communications, v. 2, 2005, p. 806-810.

(LAI, CAO, ZHENG, 2009) S. Lai, J. Cao and Y. Zheng. "PSWare: A publish/subscribe middleware supporting composite event in wireless sensor network". International Conference on Pervasive Computing and Communications, 2009, p. 1-6.

(SOUTO et al., 2006) E. Souto et al. "Mires: a publish/subscribe middleware for sensor networks". Personal and Ubiquitous Computing, v. 10, n. 1, 2006, p. 37-44.

(OZALP, TEKIN, 2014) N. Ozalp and Y. Tekin. "Content-based routing for wireless sensor network using intelligent agents". International Symposium on Computational Intelligence and Informatics (CINTI), 2014, p. 345-350.

(COSTA et al., 2003) P. Costa et al. "Introducing reliability in content-based publish-subscribe through epidemic algorithms". International workshop on Distributed event-based systems, 2003, p. 1-8.

(PIETZUCH, BACON, 2002) P. R. Pietzuch and J. M. Bacon. "Hermes: A distributed event-based middleware architecture". International Conference on Distributed Computing Systems, 2002, p. 611-618.

(RIBEIRO et al., 2005) A. R. Ribeiro et al. “SensorBus: a middleware model for wireless sensor network”. International IFIP/ ACM Latin American conference on Networking, 2005, p. 1-9.%b74

(MUSOLESI, MASCOLO, HAILES, 2006) Musolesi, C. Mascolo, S. Hailes. “Emma: Epidemic messaging middleware for ad hoc networks”. Personal and Ubiquitous Computing, v. 10, n. 1, 2006, p. 28-36.

(LEONTIADIS, 2007) I. Leontiadis. “Publish/subscribe notification middleware for vehicular networks”. Middleware doctoral symposium. ACM, 2007, p. 1-12.

(BOONMA, SUZUKI, 2012) P. Boonma and J. Suzuki. “TinyDDS: An interoperable and conFigureble publish/subscribe middleware for wireless sensor networks”. Wireless Technologies: Concepts, Methodologies, Tools and Applicatio, IGI Global, 2012, p. 819-846.

(SILVA et al., 2014) J. R. Silva et al. “PRISMA: A publish-subscribe and resource-oriented middleware for wireless sensor networks”. Advanced International Conference on Telecommunications, v. 2024, 2014, p. 87-97.

(APACHE, 2019) Apache. “Apache Kafka: A distributed streaming plataform”. Available in kafka.apache.org/. 2019.

(RabbitMQ, 2019) RabbitMQ. Available in www.rabbitmq.com. 2019.

(ZHOU et al., 2019) H. Zhou et al. “NISU: A Novel Index Structure on Uncertain Data in Large-Scale Publish/Subscribe Systems”.IEEE International Conference on Smart City, 2019, p. 1205-1211.

(OZTURK, OZDEMIR, 2019) F. Ozturk and A. M. Ozdemir. “Content-based publish/subscribe communication model between IoT devices in Smart City environment”. International Istanbul Smart Grids and Cities Congress and Fair (ICSG), 2019, p. 189-193.

(EL et al., 2014) A. El Rheddane et al. “Elastic message queues”. International Conference on Cloud Computing, 2014, p. 17-23.

(MAGNONI, 2015) L. Magnoni. “Modern messaging for distributed systems”. Journal of Physics: Conference Series, v. 608, n. 1, 2015.

(CHAND, FELBER, 2003) CHAND, Raphaël; FELBER, P. A. A scalable protocol for content-based routing in overlay networks. In: Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003. IEEE, 2003. p. 123-130.

(MELL et al., 2011) MELL, Peter et al. The NIST definition of cloud computing. 2011.