Digital Communications and Networks 4 (2018) 176–188

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/en/journals/digital-communications-and-networks/

Middleware technologies for cloud of things: a survey

Amirhossein Farahzadi [a,*], Pooyan Shams [a], Javad Rezazadeh [a,c], Reza Farahbakhsh [b]

[a] Dept. of Electrical and Computer Engineering, Islamic Azad University, North Tehran Branch, Tehran, Iran
[b] Institut Mines Telecom and Université Paris Saclay, Telecom SudParis, CNRS Lab UMR5157, 91011 Evry, France
[c] School of Computing and Communication, Faculty of Engineering and IT, University of Technology Sydney, Australia

ARTICLE INFO

Keywords:
CoT
IoT
Middleware
Fog computing
Cloud

ABSTRACT

The next wave of communication and applications will rely on new services provided by the Internet of Things which is becoming an important aspect in human and machines future. IoT services are a key solution for providing smart environments in homes, buildings, and cities. In the era of massive number of connected things and objects with high growth rate, several challenges have been raised, such as management, aggregation, and storage for big produced data. To address some of these issues, cloud computing emerged to the IoT as Cloud of Things (CoT), which provides virtually unlimited cloud services to enhance the large-scale IoT platforms. There are several factors to be considered in the design and implementation of a CoT platform. One of the most important and challenging problems is the heterogeneity of different objects. This problem can be addressed by deploying a suitable "middleware" which sits between things and applications as a reliable platform for communication among things with different interfaces, operating systems, and architectures. The main aim of this paper is to study the middleware technologies for CoT. Toward this end, we first present the main features and characteristics of middlewares. Next, we study different architecture styles and service domains. Then, we present several middlewares that are suitable for CoT-based platforms and finally, a list of current challenges and issues in the design of CoT-based middlewares is discussed.

1. Introduction

The appearance of the Internet of Things (IoT) concept is shaping and reshaping the definition of future services. The main idea behind this concept is to develop different types of communication network based on a group of physical objects or simply "things". The IoT objects are embedded with electronic chips, software, sensors, and internet connectivity to collect and process data from the environment or affect it by deploying actuators. IoT combines real-world data and computer processing to lower the costs and increase the efficiency and accuracy. Each thing can be recognized separately through its embedded computing system and can communicate with other things through the internet infrastructure. Recently, the number of connected and embedded smart devices grows rapidly. According to Cisco IBSG [1], the IoT world will include more than 50 billion objects in 2020.

IoT is translated into different concepts or approaches, such as "Network-Oriented" or "Object-Oriented" as discussed in [2], "Semantic Oriented". These visions emerged because of different stakeholder ideas and different vendors and IT experts have their own vision of this technology. IoT semantically means "a worldwide network of interconnected objects uniquely addressable based on standard communication protocols" [3]. The International Telecommunication Union (ITU) also defines IoT as a network that provides connectivity "anytime, anyplace for any connected smart devices".

Fig. 1 shows a high-level concept of IoT [4], including the main concept and its high-level functionalities. As shown in Fig.1, the main characteristics of the IoT are presented in the core circle of the figure, including anywhere, anything and anytime features that indicate the limitless IoT realm. It is noteworthy to know that the application of this technology with CPS (Cyber Physical Systems) and Cloud Computing created Industry 4.0. The middle circle includes the general application domains. Cross-system automation allows tasks to be performed more accurately, coordinately, and conveniently. In the outer circle, we illustrate a general cyber-physical learning process. Each cycle of this process boosts systems knowledge and performance. First, the system monitors the operations carried out by the sensors. Then, the system will measure and store the data. In the control phase, the system will check whether the measured data has touched or passed the pre-defined minimums,

* corresponding author.
E-mail addresses: a.farahzadi@iau-tnb.ac.ir (A. Farahzadi), pooyan.shams@gmail.com (P. Shams), rezazadeh@ieee.org (J. Rezazadeh), reza.farahbakhsh@it-sudparis.eu (R. Farahbakhsh).

https://doi.org/10.1016/j.dcan.2017.04.005
Received 10 January 2017; Received in revised form 17 March 2017; Accepted 15 April 2017
Available online 18 April 2017

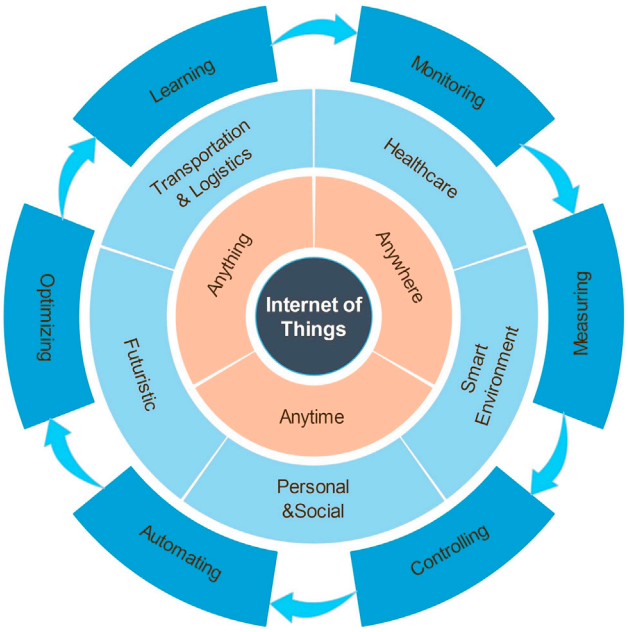

Fig. 1. Internet of Things - main concept and functionalities.

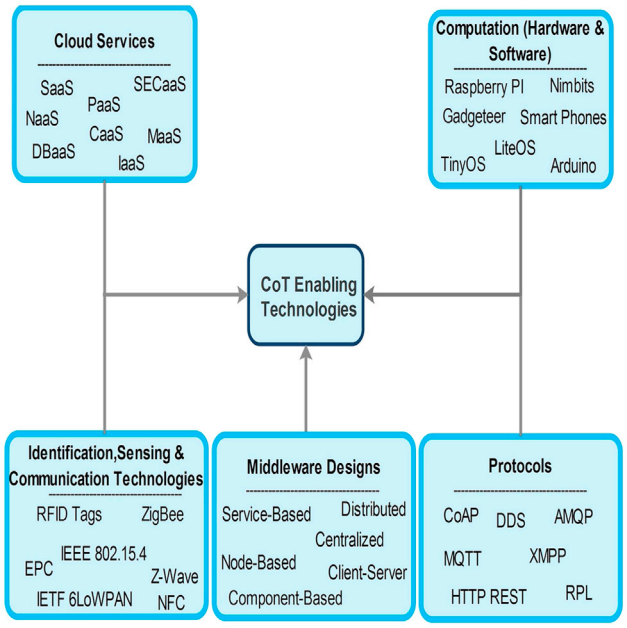

Fig. 2. CoT enabling technologies.

maximums or thresholds. Next, according to the controlled data, the system decides which automated tasks should be performed. These tasks can be a typical operation, error handling, or alerting. Optimization tries to fix the problems, defects or simply makes the system perform better. The last section is the learning phase which helps to improve system knowledge and documentation.

Considering the huge population and the fact that all connected objects actively produce or request data and require various services, the issue of limited available resources is a key factor to be considered in the design of large-scale IoT infrastructure. In addition, other issues, such as scalability, storage capacity, and maintainability are significant challenges as well. In this era, Cloud Computing became an efficient, accessible, and reasonable solution. IoT in combination with Cloud functionalities provide a new phenomenon called *Cloud of Things (CoT)* where many new possibilities, such as Big Data processing as well as covering security concerns, resource constraints, and scalability to some extend are enabled. CoT creates new revenue streams, improves customer services, and inspires product innovations. Fig. 2 shows that CoT comprises at least these five enabling technologies. Among these enablers, middleware a key role in CoT (similarly to IoT).

The paper focuses on middleware technologies and aims to study their roles and necessities in the CoT environment. Middleware is a software layer that sits between applications and objects. It aims to provide solutions to frequently encountered problems, such as heterogeneity, interoperability, security, and dependability [5]. We can consider middleware as a "network-oriented" vision according to [6].

We are witnessing the growth in middlewares development daily because this enabler makes it easier to combine new services and previous technologies to produce a novel and more capable service. Transparency is the main feature offered by middlewares. It provides an abstraction to the applications from different objects, and this feature will solve architecture mismatch problems. There are some other features that middlewares should present for desirable performance : flexibility, context management, interoperability, reusability, portability, maintainability, and few other properties that will be explained in this paper.

This paper is organized as follows. Section 2 provides a brief overview of the related studies. Section 3 discusses the features and characteristics of middlewares, Section 4 describes the different architecture designs for middlewares. Section 5 is devoted to middleware service domains and their applications. In Section 6, we review several middlewares and compare them based on different features. Finally, we discuss a few challenges and issues in Section 7 and conclude this paper in Section 8.

2. Literature review

A large and growing body of literature has investigated IoT and Cloud Computing. Most of these studies focused on the combination of these domains to come up with a novel and mature technology [7–10]. As an example, Aazam et al., [11] introduced CoT and explained its necessity. In addition, several key issues on integration of IoT and Cloud Computing are discussed including data management, security and privacy, resource allocation, and identity management. In another work [12], the authors pointed that moving toward CoT is essential and can help to implement smart environments more efficiently. The main concerns are data protection, privacy, and consumer law. CoT can utilize system performance by taking advantage of cloud services, but exchanging massive control or data packets can be harmful to this system and make it less efficient. There are situations where exchanging data between IoT and Cloud is not reasonable (i.e., requesting simple services or storing temporary data in the cloud). Therefore, [13] presented a smart gateway to process and analyze requests and decides whether to answer them locally or sends them to the cloud.

There is a relatively small body of literature that is concerned with middlewares in CoT or even in IoT. The authors in [14] discussed the benefits that can be offered by Service Oriented Computing (SOC) to build a middleware for the Internet of Things. Some concepts have been applied in a Service Oriented Architecture (SOA) middleware that tries to leverage the existing IoT architectural concepts by using SOC features to provide more flexibility and dynamicity.

In [15], the authors presented a new application layer resolution for interoperability. The key concept is to utilize device semantics provided by available specifications and dynamically wrapped them into the middleware as semantic services. The paper in [16] presents CASSARAM, a Context-aware sensor probe, Selection and Ranking model for IoT to address the research challenges of choosing sensors when there are large number that are overlapping and sometimes redundant. In [17], the authors proposed a Mobile Sensor Data Processing Engine (MOSDEN), a

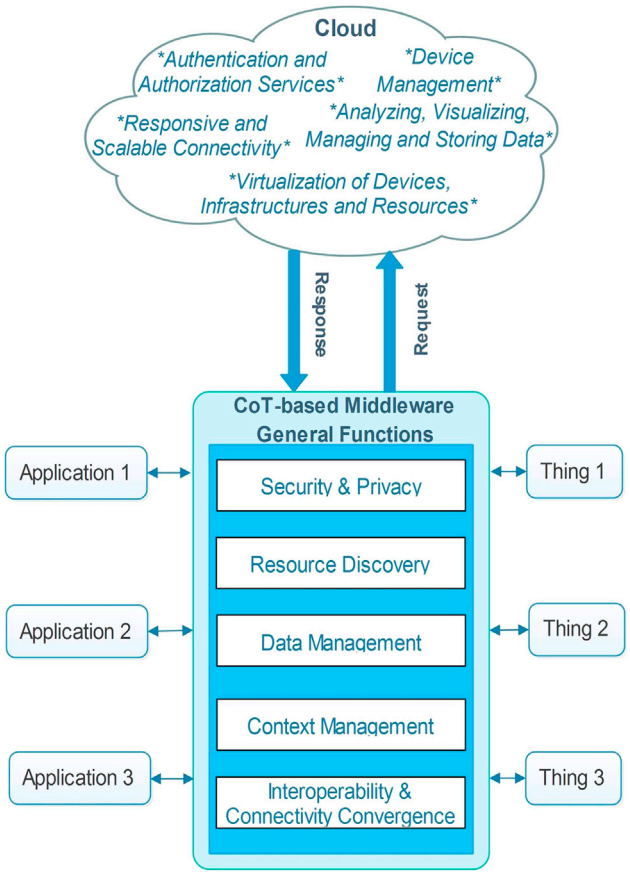

Fig. 3. CoT-based Middleware - overall concept, its position, and main functionalities.

plug-in-based IoT middleware for mobile devices that tries to collect and process sensor data without programming endeavors. This architecture also supports sensing as a service model. Another paper [18] has worked on e-health care domain. E-health features include tracking, identification, authentication, data collection, and sensing. VIRTUS is an Instant Messaging Protocol (XMPP)-based middleware that tries to provide a real-time, safe, and trustworthy communication channel among heterogeneous devices. CASP is another Context gathering framework that considers some essential requirements to act properly. A programming interface, active/passive sensor mode, simplicity, multi-transport support, separation of concerns, and sensor data model are some vital requirements in this framework [19].

Apart from the mentioned academic studies, several commercial CoT platforms have been developed including GroveStreams,[1] EVRYTHNG,[2] and Fusion Connect[3] projects. GroveStreams is a platform that can process Big Data from a wide range of devices. It can provide data analytics tools nearly in real-time. GroveStreams supports different industrial domains and by applying cloud services, it can convert the received raw data into meaningful information. EVRYTHNG platform is suitable for digital identification of products, so they can exchange data and information with authorized management applications. This platform guarantees the SLA by using end-to-end secure, reliable, and flexible management. Fusion Connect is a remarkable no-coding CoT platform. Every operation is done using the drag-and-drop approach. By deploying this platform, users can virtualize things, connect them to reporting devices and perform analysis to unlock their data. Fusion Connect can provide multiple services such as predicting products failure, automation of maintenance operations, producing performance statistics related to objects, optimizing supply chain, and calculate material replenishment costs.

Despite all the discussion, there is no dedicated study on middleware in CoT, which is the main aim of this study.

3. Features and characteristics of middlewares

As discussed previously, middleware is an important enabler that provides communication among heterogeneous things. It is a mid-layer between things and application services and provides an abstraction of the thing's functionality for application services. Fig. 3 shows a general vision of CoT-based middleware and includes its overall concept as well as its position in the design and its main functionalities.

The middleware can bring flexibility and several features and characteristics that we can see their various combinations according to the system's requirements. In the following section, we discuss number of features and characteristics of middlewares.

3.1. Flexibility

Flexibility is one of the most important capabilities that a middleware can offer to the IoT or CoT systems. By using middleware, part of the application developers' concerns are covered because it can handle conflicting issues due to communication between applications and things. There are different kinds of flexibility (e.g., response time or delay flexibility). As a result, flexibility moves from the application level to the middleware level to handle different forms of flexibility. It is essential to determine which software or hardware components needs more flexibility. The level of flexibility is important because high level of flexibility denotes more connectivity APIs and the processing workforce found it unreasonable for multiple domains, such as ultra-resource constrained IoT networks.

3.2. Transparency

Middleware hides many complexities and architectural information details from both application and object sides so that they can communicate with minimum knowledge of each other side's information. Transparency is a distinctive feature of middleware that can be very helpful for programmers, end users and applications. According to the application domain and services, developers need to decide which aforementioned part most requires transparency. Middleware transparency is in the form of platform and network. *(i) Platform transparency*: the middleware runs on a variety of platforms. It lets the organization use different hardware platforms according to their requirements. Clients and servers do not need previous knowledge to work with each other. *(ii) Network transparency*: the middleware provides transparency of the networks to the application users. This means that users do not need to know whether the resources are located locally or on remote devices.

3.3. Context management

This feature is a coordination management concept. It allows users to choose and configure a service or subject in one application, and then all other applications containing information about that specific subject will adjust themselves to the same user defined setting. This feature is the main foundation in context-aware systems. To achieve high level of flexibility, which is important for better overall performance, context management allows numerous programs' threads to work on the same task, a thread to work on a different task or each thread to perform a different task. CoT is a set of ubiquitous devices, therefore it is necessary to know the diversity of the objects technically, logically, and physically, which will lead to proper context management.

[1] https://grovestreams.com/.
[2] https://evrythng.com/.
[3] http://www.fusionconnect.com/.

3.4. Interoperability

This functionality allows two sets of applications on interconnected networks to be able to exchange data and services meaningfully with different assumption on protocols, data models, and configuration, without any problems and additional programming efforts by the developers. The nature of software and hardware heterogeneity of CoT requires interoperable components. Interoperability will contribute to standardization and has different degrees depending on the heterogeneity level of the environment or system components. Implementing interoperability requires the creation, management, reception and fulfillment of SMART standards.

3.5. Re-usability

On SOA-based middlewares, there is the possibility to reuse the software and hardware because there is no specific technology to impose its policies for service implementation [2]. The main purpose of reusability is to make designing and developments easier by modifying system components and assets for specific requirements, which results in cost efficiency. Cloud-based networks are crucial for obtaining this feature. These types of systems offer everything as a service that can be used multiple times.

3.6. Platform portability

Portability is a critical feature in CoT. There are many mobile hardware and software components that are constantly moving between different platforms. Therefore, a CoT platform should be able to communicate from everywhere, anytime with any device. Thus, middleware helps to boost portability by its flexibility. It helps move across the environment and avoids being restricted to one platform. Middleware runs in user side and can provide independence from network protocol, programming language, OS, and others. By considering platform variations in CoT, it is obvious that the applications and services need a kind of self-determining mechanism for cross-platform development which can be achieved by using platform independence or portability. The former run the applications on different platforms using virtual machines to execute the codes and the later is expected to adapt the applications to the new environment with reasonable user's efforts.

3.7. Maintainability

This is the ability of systems, applications or devices to respond to failures properly and rapidly return to normal functionality without any problem. This can be achieved by isolating, correcting, repairing, preventing, and other acts of resolving defects. Maintainability has a fault tolerance approximation. To perform maintainability efficiently, well-defined procedures and infrastructures are required. This feature is needed in hard-real time applications. The growth of deploying CoT in different contexts is significant. Without proper maintainability in middlewares, extending the network or fixing the bugs would be overwhelming. Designing a maintainable middleware from the outset is important. Providing relevant documentations and feedback checklists can be helpful in designing stable and extendable middleware.

3.8. Resource discovery

CoT system includes multiple heterogeneous devices. There is no reliable and global knowledge on the availability of the mentioned devices. Resource discovery is actually a process used by a node to search and probe for intended resources, such as services or data types, in the entire nodes of a network. After sending a search query, the resource discovery protocol will automatically choose the best resource that offers the most effective services and information [2]. For example, the process of choosing the best sensor in the environment that gives us the most reliable information. One of the key essentials of a Resource Discovery mechanism for middlewares is to cope with frequent failures. Developers must use self-stabilizing algorithms to bring the system into an ordinary state despite transient failures.

3.9. Trustworthiness management

This capability is required in social IoT applications. By this feature, we can develop trust and credit mechanisms (e.g., authentication, hashing, encryption and others) to ensure that services and information that have been prepared for us by other applications came from trustworthy systems [20]. Capability plays a key role in establishing trust relationships among different applications which will lead to a trustable, robust, and secure system.

3.10. Adaptability

In CoT systems, environment and networks infrastructure often changes due to different reasons including nodes mobility, power drain, topology shifts and others. Many middlewares usually have fixed capabilities and cannot be customized to dynamic and unpredictable situations. This feature shows how middleware should behave against environmental changes. Middlewares can react both statically and dynamically. Deploying optimized dynamic methods result better adaptability. Higher adaptability is critical in hard real-time systems. In contrast to flexibility, being adaptable denotes durability against long-term changes in the systems. To reach a reasonable amount of adaptability, it is necessary to provide it as services (such as security and identification protocol adaptation services, communication protocol adaptation services, and semantic protocol adaptation services).

3.11. Security and privacy

Security in middleware is a vital issue because most data transmission and operation occurs through it. To have a secure system, we must consider confidentially, integrity, and availability. Therefore, middleware should provide different security measures for ubiquitous applications, and pervasive environments, such as authentication of identification and credentials, authorized modifications, and access control policy, are needed for verification and accountability management [21]. Privacy requires that all IoT system components that access personal user information must guarantee protection of the mentioned information from unauthorized access.

3.12. Connectivity convergence

IoT contains and supports various types of hardware and software components that interact with each other through heterogeneous communication platforms. For example, software applications may send their request or queries to middleware in different kinds of communication methods such as WiFi Signals, Wired-based data, Fiber-optic lights, and Bluetooth. Meanwhile, objects may use communication technologies, such as Zigbee and RFID. It is even possible for sensors to send directly analog signals or send digital signals after processing them. Therefore, connectivity APIs and management are essentially needed which are convergent with interoperability.

4. Architectural comparison of middlewares

Middlewares are designed based on different architectures. One of the most important aspects in the design of a middleware is determining a well-defined framework or architecture. Each architecture has unique attributes; therefore, they can be categorized based on different parameters contrary to an explicit and fixed categorization. Perera et al., [22] proposed a general category for middle-ware's architectures as follows: Distributed architecture, Component-based architecture, Service-based

Table 1
Comparison of different possible architecture designs in middlewares.

Architecture	Benefits	Challenges
Component-based	Reusability, Abstraction support and Independency	Maintenance, Migration, Complexity and Compatibility
Distributed	Resource sharing, Openness, Scalability, Concurrency, Consistency and Fault tolerance	Interoperability, Security, Manageability and Maintainability
Service-based	Reusability, Scalability, Availability and Platform independence	Service discovery, Complex service management and Service identification
Node-based	Availability and Mobility	Security and Manageability
Centralized	Simplicity, Security and Manageability	Scalability, Availability and Portability
Client-server	Servers separation, Resource accessibility, Security, Back-up and Recovery	Congestion, Limited scalability and Single Point of failure

architecture, Node-based architecture, Centralized architecture, and Client-server architecture. The main features of these architectures are summarized in Table 1, which include benefits and challenges for each architecture design in middlewares. To provide more detail, Table 2 categorized a list of the well-known middlewares based on their application domains and architecture styles. The detail of each middleware is provided in Section 6.

4.1. Component-based

First, we explain the component-based architecture. In this type of architecture, there are some specific and mainly loosely couple independent components that are semantically are working together to perform tasks. However, technically, each one is responsible for solving a specific part of the problem. Therefore, it is often said that components are cohesive and modular. It decomposes the design based on logical or functional components which provide a higher level of abstraction. The principal objective of this architecture is the encapsulation of functionality and behaviors of system elements for minimal dependency, more reusability, and easier trouble-shooting.

4.2. Distributed

Distributed architectures or distributed systems consist of different networked software and hardware components that coordinate their operations to execute tasks. The main characteristics of this architecture are concurrency of components, lack of global clock, and the fact that failure of a component does not affect the whole system. The main advantages are fault tolerance, scalability, concurrency, flexibility, and hardware and software sharing. Components with different platforms and on several machines can cooperate on the specific goal. There are several frameworks that support this architecture, such as CORBA,. NET, and J2EE.

4.3. Service-based

This architecture is one of the most efficient designing styles. There are two methods used to implement the SOA-based middleware. The first is deploying it in a stand-alone manner and the second is using Cloud Computing services (PaaS). The components of this architecture provide services to each other over a communication protocol [23]. Each device offers its functionality as standard services, while the detection and invocation of new functionalities from other services could be performed simultaneously. This architecture is not recommended for Homogeneous system application because it is not cost effective and practical for a single vendor. Real-Time: Given that SOA requires synchronous communication between the service consumer and producer; it is not a suitable option for devices that need strictly-enforced response time such as embedded equipment. In this case, the tighten coupled architectures are preferred. GUI-based: SOA is not a desirable option for GUI functional applications, like maps, which require heavy data traffic exchange.

4.4. Node-based

In this architecture, there are many software components with the same or different functionalities that work on mobile and sensor networks to communicate and process data that are collected from the sensors [22]. This architecture is composed of streams and nodes. Nodes operate the data through streams with other nodes and makes them suitable for mobile devices.

4.5. Centralized

In this architecture, all services are gathered in a specific location and applications or devices can make a request to use the resources and services. Users are usually simple thin devices that hand over their request to the central server which is a resource-rich device. The network between the devices and central server can be implemented by connectionless or connection-oriented protocols, although there is no direct communication between application instances. This architecture is the exact opposite of distributed architecture. The failure of the central server will cause disable the whole network if a backup server does not take over immediately.

Table 2
Architecture and application domain of top 20 middlewares.

Middleware	Architecture	Main Application	Commercialized	Cloud-based
Aura	Distributed	Pervasive computing environment	✗	✗
ABC&S	Service-based	Car Parking automation	✗	✓
Capnet	Distributed, Node-based	Mobile multimedia applications	✗	✗
Carriots	Service-based	Smart city, Smart energy	✓	✓
CARISMA	Distributed	Mobile computing	✗	✗
CHOReOS	Service-based and component-based	Enabling large-scale, QoS-aware adaptive choreographies	✓	✓
C-MOSDEN	Distributed, Component based	Resource constrained mobile devices	✗	✓
COPAL	Centralized, Component based	Context provisioning	✗	✗
CoMiHoC	Centralized	Context management in MANET environment	✗	✗
DropLock	Service-based	Smart Home deployment	✗	✓
Gaia	Distributed, Service-based	Managing ubiquitous computing habitats and living spaces	✗	✗
GSN	Distributed	Deployment and interconnection of sensor network	✗	✗
Link smart	Service-based	Intelligent networked embedded systems	✓	✗
OpenIoT	Service-based	Smart cities and mobile crowd sensing	✗	✓
Rimware	Service-based	Heart Rate Monitor (HRM) and Smart lighting (SL)	✗	✓
SOCAM	Service-based	Building context-aware mobile services	✗	✗
ThingWorx	Service-based	Agriculture, Smart cities and Smart buildings	✓	✓
UPnP	Node-based	Ubiquitous mesh home networks	✗	✗
VIRTUS	Distributed	E-health caring	✗	✗
Xively	Service-based	Home appliances connectivity and management	✓	✓

Table 3
Summery of top 20 middlewares comparison (NS stands for Not-Specified and S&P for Security & Privacy).

Middleware	Event Detection	Service Discovery	Adaptability	Platform Portability	Interoperability	Context Awareness	S&P	Real-time
Aura	✓	✓	✓	✓	✗	✓	✗	✗
ABC&S	✓	NS	✓	✓	NS	✓	✓	✓
Capnet	✓	✓	✓	✓	✓	✓	NS	NS
Carriots	NS	✓	✓	✓	✓	NS	✓	✓
CARISMA	✗	✗	✓	✓	✓	✓	✗	✗
CHOReOS	✓	✓	✓	✓	✓	✓	✓	✓
C-MOSDEN	✓	✓	✓	✓	✓	✓	NS	NS
CoMiHoC	✓	NS	✓	✓	✓	✓	✗	✗
Copal	✓	✓	✓	✓	NS	✓	✓	✓
DropLock	NS	NS	✓	✓	NS	NS	✓	✗
Gaia	✓	✓	✓	✓	✓	✓	✓	✗
GSN	✓	✓	✓	✓	✗	✗	✓	✓
Link smart	✓	✓	✓	✓	✓	✓	✓	✓
OpenIoT	NS	✓	✓	✓	✓	✓	✓	NS
Rimware	✓	✗	✓	✓	✓	✗	✓	NS
SOCAM	✓	✓	✗	✓	✗	✓	✗	✗
ThingWorx	✓	✓	✓	✓	✓	✓	✓	✓
UPnP	✓	✓	✓	✓	NS	✓	✓	NS
VIRTUS	✓	✓	✓	✓	✓	✗	✓	✓
Xively	✓	✓	✓	✓	✓	✓	✓	✓

4.6. Client-server

The Client-server architecture is the most classical model, in which there is always a request from one side and a reply from the other side. Determining the device's role is challenging because sometimes, we encounter devices that can be considered as both client and server. CaSP used this model to separate processing and sensing from each other [22]. This architecture can be classified into two models based on the functionality of clients.

Thin client model: The server oversees processing applications and managing data. The clients just provide the GUI.

Thick/ Fat client model: The server is responsible for data management. The implementation of applications and providing GUI is done by the client.

Another type of Client-server architecture is multi-tier, in which main functions, such as presentation, application, processing, and data management, are physically separated.

5. Middleware service domain

Middleware is a solution for the implementation of different services in a heterogeneous environment. First, we must identify the variety of these services. There are some researchers and organizations that try to detect all potential services that middleware can present and define an approach to implement them. We will discuss some of the services in the following section. Each of these domains may consists of multiple subdomains.

5.1. Information exchange and storing

Systems that benefit from this domain (i.e., transaction systems) should provide the ability for users to pass their request to middleware to ex-change it with other nodes or saving their information on a database without any problem. For instance, this service allows a group of operators to use a context-aware middleware to manage a smart environment. The IoT is primarily based on ubiquitous and pervasive computing; therefore, communicating and storing data through this distributed environment require appropriate considerations.

5.2. Data management and analytics

This is absolutely a huge and complex domain. Users should be able to manage databases, data security, data quality, reference and master data, metadata, and other topics. By using these services, there are no longer any concerns on managing and processing data, especially Big Data. Each user can implement personal policies without any concerns. Furthermore, services, such as data processing and data acquisition should be presented by the IoT middleware. Data analytics is another important service provided by the middleware by extracting statistical data or visualizing the obtained data and sending them to the user application.

5.3. Object middleware

This type of middleware is also known as object request broker and allows applications to transmit objects and demand services via an object-oriented system. In summary, these middlewares manage and control communication among objects. The Remote Procedure Call (RPC) in combination with these middleware makes Distributed Object Middleware (DOM). This added feature calls objects and procedures on remote systems and can implement synchronous or asynchronous interactions among objects, applications, and systems.

5.4. Communication

This domain is a baseline for many other domains. A communication middleware provides a framework or environment that enables two applications to negotiate and exchange data in a distributed system. Communication middleware provides an abstraction of the network protocol for software application and reduces the complexity of designing low-level communication mechanisms. DDSS is an example for communication middleware [24].

In addition to the above-mentioned domains and by considering system services and operations, there are some other special-purpose applications that need various features of middleware domains to perform their task properly. For example, WebCrawler [25] and GRank [26] are two search middlewares that use a combination of aforementioned domains capabilities.

6. Comparison of sample middlewares

In this section, we describe in detail the top 20 middlewares (mentioned in Table 2) and provide a comparison between their main functionalities and applications from different aspects (Table 3). These platforms are designed for specific applications but there are middlewares, such as LinkSmart, that offer a wide range of services for different needs. These middlewares have been selected for various reasons including being state-of-the-art, covering all architecture styles, working on different application domains, being commercialized and using different infrastructure as some of the selection criteria.

6.1. C-MOSDEN

C-MOSDEN (Context-aware Mobile Sensor Date Engine) [27] is a novel location and activity aware mobile sensing platform that can collect and process data without programming efforts. This is a plug-in-based middleware for mobile devices. To avoid cost in the process and storing data, save time and energy consumption (battery drainage), and reduce network traffic, this platform proposes an on-demand distributed crowd sensing platform that captures the required data based on user request and location. It consists of sensing as a service cloud platform (to supervise sensing task) and worker nodes (to perform the sensing task). Cloud middleware evaluates the availability of worker nodes and sends a request to the selected node. It can also impose specific condition on the data acquisition or transfer, such as a sense when certain activity occurs. It includes three main modules of context-aware, activity-aware and location-aware that work with GSN (Global Sensor Network) as cloud-based companion platform in an integrated system. Context-aware data streaming engine called mobile sensor data engine is based on previous MOSDEN platform. It is client-side tools that can be installed on any device.

C-MOSDEN also have context-aware functionality and supports push/pull data streaming. The activity aware module can recognize six activities, such as (I) moving in a vehicle, (II) cycling, (III) walking, (IV) running, (V) still (not moving), (VI) tilting (falling), and a combination of them. Location aware module recognizes when a device enters or exits from an area that is defined by longitude, latitude, and radius. GSN aims to provide flexible middleware to address sensor data integration and distributed query processing. It is based on four basics principles: (A) simplicity, (B) adaptability, (C) scalability, and (D) lightweight implementation. It integrates, discovers, combines, queries, and filters through a XML-based language. The key element in GSN is the virtual sensors that can be any data producer or a combination of them. It can have multiple input data streams, but only have one output. First, the sensor data consumer (city, researcher, doctors) submits the requirement. Then, the problem is analyzed and decided which sensor collects the relevant data. The global task scheduler provides a strategic plan on how to delegate the tasks to multiple worker nodes. This platform provides a high level of interoperability, scalability, usability and management of resources and costs by collecting the relevant data only.

6.2. Xively

The Xively[4] platform is a capable and enterprise solution that provides a middleware for creating, managing, and engaging ideal CoT. According to its functional specifications, there are three main properties specified as (I) scalable and flexible connectivity features, (II) data engagement features, and (III) management features that cover the product's provisioning, monitoring, updating, and user management. Presently, users want operations in real-time and quickly; therefore, Xively answered these demands by deploying MQTT, which is a messaging broker in CoT. The speed and scalability of Xively are made possible through Blueprint Template that is a flexible model for structuring devices and user information and mapping relationships between them. Each template also provides contexts for managing real-time products and user's data fields, processing both Xively time series and third party data. It also introduces data logging devices that offer a full view of remote products down to specific sensors. Another feature is cross business automation that offers an opportunity to integrate products, customers, and other business systems. Xively offers a Connected Product Management (CPM) platform that helps to capitalize the CoT. This makes modeling and connecting products easier and immediately allows them to start gathering data. With this platform, it is possible to find new features, proactively manage users, and integrate the obtained data with existing business systems.

[4] https://xively.com/.

6.3. ThingWorx

ThingWorx[5] is popular for its business management, big data, and analytics optimization. It is commonly used in agriculture, smart cities and smart buildings. It supports MQTT, XMPP, CoAP and DDS for communication, and TLS and AES as security protocols and includes Foundation, Utilities, Studio and kepware sections and related components. The utilities section includes tools and Mashup Builder for defining, monitoring, management, and optimization. The Mashup Builder allows the users to create their interactive applications, real-time dashboards, and mobile interfaces without any need for coding which reduces the developer time and increases scalability. As an analytical solution, analytics enables developers to find the real-time pattern easily, detect an anomaly, predictive outcomes, and improve performance by using the Worx analytic server. The thing watcher automatically observes and learns the normal state and alerts the end-user if it detects any anomaly.

The things predictor system can predict relevant outcome. The things optimizer can improve future performance and results with automated prescription and simulation. The unique frame work allows integration with technologies, such as industrial connectivity (Kepware) for IIoT (Industrial IoT). Kepware's communications platform is a complete solution for device-to-cloud interoperability. KEPServerEX provides a single source of industrial automation data to multiple applications, allowing users to connect, manage, monitor, and control diverse automation devices and software applications through one intuitive user interface. KEPServerEX provides two options for interoperability with the ThingWorx IoT Platform. The IoT Gateway advanced plug-in features enable real-time and read-only communication with the Platform as an agent. It allows users to model industrial things within the ThingWorx IoT Platform. The ThingWorx native client interface enables real-time and bi-directional communication with the Platform. SQUEAL (Search, Query, and Analysis) is an intelligent interactive search engine that allows a user to quickly search, query, and analyze through cloud data repository. It supports connectivity via 3rd party device clouds, direct network connection, open APIs, and using ThingWorx edge micro server (EMS). Composer models environments, which make it easy to model things, business logic, visualization, data storage, collaboration and security for application. The modeling for developer is based on making entities (things) that can produce events. To scale models and avoid repetition, the developer can use the thing templates (properties of things with the same nature) and things shapes (properties of things or templates).

6.4. Carriots

Carriot[6] is a cloud-based IoT platform (as a PaaS) mainly designed for M2M projects, such as smart city and smart energy. As an agent, this platform provides some modules for common M2M projects, such as date collection and storage, security, and device management. The seven-layer architecture of the platform can provide all the requirements of diverse projects and M2M applications. Carriots projects can run it in five steps, including connecting the devices, collecting data, management of devices and data, building the app and running the project. It collects and stores raw streams of data from any devices by sensors through a web connectivity (gateway or embedded 3G or GPRS modems) by using MQTT protocols and sending to an HTTP/ HTTPS RESTful API in XML or JSON format.

It can also integrate with other systems and pull or push data from CRM and ERP or any other APIs, such as Zoho and Dropbox. The platform introduces Apikeys (define privileges and visibility), HTTPS, and HMAC hash solutions for security issues. Carriots include a NoSQL big database

[5] https://www.thingworx.com/.
[6] https://www.carriots.com/.

to store data in two replication sites with high transactions. The data can be accessible through PUSH and PULL strategies of the API. The powerful feature in Carriot is the capability of writing and executing code by an arbitrary code. By writing Groovy script and combining it with SDK, it can execute any action in Apps and run in the cloud. Carriots device management module allows the user to check the status and manage the configuration and firmware remotely. Carriots gives customers the freedom and flexibility in matching with the wide range of hardware and some platforms, such Microsoft Azure, that is suitable for public cloud services. The main benefits of Carriots platform include saving development time, lower costs (in development and operation), reliability, and scale up from tiny prototypes to thousands of devices (free account for up to 10 devices). It can scale up to millions of devices and customers.

6.5. CHOReOS

This middleware is a set of software components that implement and execute large-scale web service compositions. CHOReOS tried to converge the Internet of Things, Cloud Computing and the Internet of services. By using higher-level abstraction and services, this middleware tries to make scalable, complex and adaptive service structures. CHOReOS is a mixture of four main modules. (I) eXecutable Service Composition (XSC), which is responsible for coordinating the proper and required services for things. (II) eXtensible Service Access (XSA) that is used for accessing and selecting services and things. (III) eXtensible Service Discovery (XSD), which is a management framework for protocols and processes to find requested or suitable services and things. (IV) Cloud and Grid Middleware that is responsible for computational components and control implementation of choreographies. XSC have two main components: (1) Composition and Estimation (C&E) and (2) Reconfiguration Management for Service Substitution. The first component is responsible for the composition of thing-based services, and the second component is a composition of functional abstraction services and impact analyzers. XSA is the next module that includes (A) XSB that backs seamless integration on heterogeneous environment, (B) EasyESB presents a scalable and proper environment for business services to get into large choreographies, (C) LSB provide services related to things and addresses IoT challenges. XSD includes (I) AoSBM Discovery for resource discovery process by using service abstraction, (II) Things Discovery for the probabilistic method to implement it, (III) Plug-in Manager, which is a flexible framework for supporting all discovery solutions. Cloud and Grid Middlewares have three components: (1) The Storage Service that deploy a database server on cloud and define access control policies. (2) The Grid Service, which provides Grid Computing as a service; this means offering high-performance computing applications with enough needed resources. (3) The Enactment Engine is the main section of the Cloud Grid and Middleware and is a composition of the CHOReOS middleware, business services, and coordination delegates. Cloud and Grid components export RESTful services [28].

6.6. Rimware

Rimware is a middleware proposed by Chengjia Huo et al. [29] and the main concept behind that is positioning the middleware on rim that covers cloud on the inside and Network of things (NoT) on the outside. The middleware proposes a layer that includes gateways, clouds, and components that run on the both sides. The Rimware provides scalability, interoperability, security, and saving energy via gateway adapter and knowledge-based and access controller in cloud. It makes use of profile-based protocols for interoperability feature that allow devices from different vendors to discover each other in terms of their implemented profile. It uses BLE (Bluetooth 4.0 low energy) as a communication protocol that can help to lower energy consumption. First, the node establishes authentication and secure connection through a trusted gateway to the cloud via an adapter on the gateway (when smartphone and tablets are not available), which can substitute the role of application on smart phones. Then, the cloud side establishes mapping among the device profiles, web APIs and cloud operations.

6.7. DropLock

T. L. Vinh [30] presents a middleware architecture to integrate mobile devices, sensors, and cloud computing. This middleware uses cloud services and provides scalability, sufficient data storage, data processing capabilities and energy saving management. The specific goal here is the convergence of the Mobile Cloud Computing (MCC) and IoT. The authors presented a basic requirements framework for designing CoT middleware which included Network management services, Data management services, System management services, and Security and authentication services domains. For security concerns, before communicating with other devices, it is essential to perform the authentication process. For resource constraint devices, authentication could be performed by lightweight cryptography algorithms. Data must be encrypted before transferring on communication channels. For assurance of privacy, this middleware uses cloud-based privacy management services. The authors used a framework named DropLock to demonstrate their general service architecture on a smart city scenario [30].

6.8. ABC&S -based car parking middleware

This middleware aimed to deploy the Always Best Connected and best Served (ABC&S) paradigm to design a worthy cloud-based car parking middleware for Internet of Things. The authors used smart mobile devices, low-powered processing chips, cloud computing, and future network and communication environments (i.e., UCWW) as main enabling technologies to form their idea. This article proposed a three-layer architecture that consists of the sensor layer, communication layer, and application layer. The sensor layer consists of multiple sensor technologies, such as RFID and CCTV. The communication layer is a composition of different wireless technologies (i.e., WIFI, ZigBee, VANET, and WSN). The application layer is responsible for providing some cloud-based services for actions, such as finding the best parking lot available, vehicle license plate patrolling, car tracking and others. This layer is a three-tier model, which includes Cloud tier, Web server tier, and Mobile app tier. In the cloud tier, web applications arrange the data into an open-source software framework, such as Hadoop, for storing them and running applications on distributed hardware. In web server tier, it uses OSGi (a java framework) to dynamically publish, discover, and bind services to Bundles. The mobile app tier makes mobile devices to access web applications to gain best services. In cloud tier, multiple servers may send recommendations on the best available parking lots for users; therefore, a cloud-based middleware is developed with three clusters, (A) Kafka, which is a messaging platform for load balancing, (B) Storm that consumes topics produced by Message Queue and process data (i.e., filtering, mining, clustering), (C) HDFS that contains useful dataset, which web applications can access in real-time by putting, scanning, adding, and obtaining operations [31].

6.9. OpenIoT

OpenIoT (open source Internet of Things) [32] is an open source IoT platform with semantic interoperability in the Cloud. As a standard-based model for physical and virtual sensors, this platform applies the Semantic Sensor Networks (SSN) ontology for semantic unification of IoT systems. Discovering and collecting data from mobile sensors are achievable through a publish/ subscribe middleware called Cloud-base Publish/ Subscribe (CUPUS). The CUPUS interacts to cloud database via X-GSN (eXtended Global Sensor Network). The architecture of OpenIoT has seven elements, including a sensor middleware, cloud data storage, Scheduler, Service Delivery and Utility manager (SD&UM), Request definition, Request presentation, and Configuration and monitoring. For security and privacy issues, the platform adopts a flexible and generic

approach for authentication, authorization, and User management through its specific privacy and security module and CAS (Central Authentication Service) service. To ease the development of applications (zero-programming) and manage IoT applications, it offers an integrated development environment (IDE) comprising a range of visual tools. OpenIoT is recommended for application in areas where semantic interoperability is required. The ability to handle mobile sensors and provide QoS parameters makes it a suitable choice for smart cities and mobile crowd sensing.

6.10. Aura

This middleware is suitable for pervasive computing. It applies two broad concepts. First is proactively, which means that the system's layers are able to answer the request from a higher level. Second is self-tuning, which means that layers must adjust their performance and resource usage according to demands made on them. The personal aura acts as a proxy and by mobility, it provides adaptation to the new environment. In Aura, the context observer helps to do context managing. One of the main challenges in Aura is seamless integration rather than building blocks of pervasive computing. To achieve maximum capabilities of a resource-limited mobile client and thus improve user experience, Aura uses cyber foraging. Surrogates are nearby computing servers or data-staging servers that provide this amplification. The reliability and quality of connections between Surrogates and nodes are important [33].

6.11. Capnet

Capnet is mainly developed for mobile multi-media applications. This middleware is context-aware and with a common interface for context sensors, which makes it easier to embed a new sensor in the system. Capnet tries to address the requirements, such as mobility of the software components, multimedia applications, and adaptation. Context-aware pervasive networking focuses on adapting service application according to the user's environmental position and his personal preference. Service discovery and component management also provide the required level of transparency for applications making them able to perform operations in the mobile environment. This middleware is capable of asynchronous messaging, context management, service discovery, and resource management. Capnet can perform a proper task according to component mobility; it can decide whether to run a component on the client side (mobile device) or server side (fixed PC) depending on the resource requirements of the starting application or component [34].

6.12. CARISMA

CARISMA use the principle of reflection to achieve a dynamic behavior to context changes. It provides primitives for developers, which describe how context changes should be handled using policies. For example, for message exchange in this middleware, there are four policies: charMsg, which sent one character each time, plain Msg, to exchange messages explicitly, compressed Msg to exchange compressed messages, and encrypted Msg to deliver encrypted messages. It also enhances the construction of adaptive and context-aware mobile applications. The authors believe that middleware platforms must support both deployment time configurability and run-time re-configurability. CARISMA presents an approach to provide applications the highest economic quality of service as possible, which they can offer to consumers [35].

6.13. LinkSmart

LinkSmart is an ambient intelligent (AmI) middleware that acts and reacts based-on object presence. In general, this technology has five main characteristics as follows: Context-aware, Embedded, Adaptive, Personalized, and Anticipatory. LinkSmart uses a lower-level Data Acquisition Component to collect accurate data from Context Providers. The Data Acquisition Component performs two main protocol, push and pull. Push handles data that needs to be sent, and pull retrieves data from sources. This middleware aims to support both distributed and centralized architectures. It will provide reflective properties, security and trust, and model-driven development of applications. LinkSmart also allows for secure, trustworthy, and fault-tolerant applications with the use of distributed security and social trust components. The main domains that end-users can use are facility management, smart homes, and health care [36].

6.14. GSN

Global Sensor Networks make a rapid and simple deployment of different sensor network technologies. The four main design goals are simplicity, adaptively, scalability, and lightweight implementation. It helps to make a flexible integration and sensor networks discovery. GSN will provide dynamic adaption of the system configuration during operation. It uses a container-based approach, which allows different sensors to be identified easily and most of the system complexities hide in that container. These containers communicate to each other in a peer-to-peer style. The main abstraction in GSN is the virtual sensor which can be any device that produces data, i.e., a real sensor, wireless camera, PC, and cell phone. VSM is responsible for multiple tasks, such as accessing virtual sensors, controlling the delivery of sensor data, and providing the necessary administrative infrastructure. VSM has two components, LCM and ISM. LCM manages the usage and communication of resources and virtual sensors. ISM is responsible for managing the quality streams; therefore, it ensures the QoS of streams [37].

6.15. COPAL

Copal is an adaptive context provisioning approach. Context provisioning refers to the approach of collecting, transferring, and processing context to setup context-awareness of ubiquitous services. Copal is designed to provide a runtime middleware. This will lead to loose-coupling between context and its processing, which is for integrating new context sources, creating new information models, and supporting different information processing requirements of context-aware services. This middleware has two key concepts. The first concept is Publishers, which are device services that indicate all sensors and devices in the environment. The complexity and heterogeneity of devices and communications will be solved by using wrappers. The second concept is Listeners, which refer to Context-aware services that COPAL must notify when corresponding inquiries are met. Each service may have numerous listeners [38].

6.16. Gaia

Gaia is a distributed middleware architecture that coordinates software services and heterogeneous networked physical devices. It provides a template to develop user-centric, multi-device, resource-aware, context-sensitive, and mobile applications. The Gaia main components are the kernel, the application framework, and the applications. Kernel component management core and application framework allows users to build, execute, or adapt existing applications to active spaces. Gaia offers 5 basic services: Presence service, Event manager, Context service, Space repository, and Context File System. It focuses on the interaction among users and active spaces. Active space is a programmable pervasive computing environment, which boosts mobile users ability to interact and configure multiple devices and services simultaneously. Active spaces are usually determined for specific tasks. Applications can use task context to detect meaningful information from incoherent data [39].

6.17. UPnP

The main goal here is designing a middleware for mesh home network. The wireless mesh home network consists of two kinds of devices: Mesh-Controller and Mesh client. MeshController sets a ubiquitous heterogeneous network within the house, which can be interconnected with IP net-work and limits the access to the wireless mesh home network and provides secure communications by deploying various mechanisms. A home gateway connects these Mesh-Controllers together and provides an easy and centralized users' management. UPnP offers peer-to-peer ubiquitous network connectivity between various devices. Service discovery and context-awareness are two critical capabilities of UPnP. The typical UPnP protocol does not support sufficient context-awareness. When the mesh client moves from one point to another, the user context must be reconfigured to keep the context-aware service discovery. The authors introduced an approach to utilize UPnP to make a middleware that considers user profile and context in an intermediate node. Context aggregators provide consistent and uniform contextual information to the middleware Context Management Module is responsible for creating the user related context information [40].

6.18. CoMiHoC

CoMiHoC is suitable for context management and situation reasoning in MANET environment. Context-awareness has to be established in a peer-to-peer manner to work in MANET environment. CoMiHoC builds location models and estimates the connection of contexts used for situation reasoning. This middleware defines a concept of context relevancy that is specified as the rating to which particular context information is suitable to the current condition of a context-aware application. The CoMiHoC architecture framework is divided into three different groups of component: Context provisioner, Request manager, and Situation reasoner. The context provisioner group provision relevant contexts to the submitted situation spaces and manages context buffer, which replaces less relevant context with more relevant context if they are available in the current position. The request manager is responsible for incoming context queries and will reply to them if they exist in the buffer. The situation reasoner consists of a previous component group and manages a set of situation spaces. The authors considered challenges, such as temporal relevancy, context uncertainty, distributed control and fault-tolerance, which need to be addressed in managing context in MANET environment [41].

6.19. SOCAM

In SOCAM, context is presented as an ontology markup language. The benefit of this approach is that the context information can be shared among devices or entities in a pervasive computing domain and context reasoning becomes possible. By considering the mobile device limitations and reducing the burden of context processing from them, the context model is designed in a two-level hierarchy. The pervasive computing domain is divided into multiple sub-domains and each domain consists of individual low-level ontology. To link all sub-domains, there is also a generalized ontology. Each component in this middleware represents as an autonomous service. Its main goal is to work on mobile and pervasive computing system. It will nearly respond to every context-based need. Service-oriented context-aware middleware is capable of doing tasks, such as acquiring, discovering, interpreting, and accessing various contexts and interoperability between various Context-aware systems [42].

6.20. Virtus

VIRTUS middleware is aiming to benefit the IoT paradigm for e-health solutions. E-health features include tracking, identification, authentication, data collection and sensing. The main goals of this system are collecting patient data, monitoring his/her health status and act as prevention. Instead of deploying SOA, it uses an Instant Messaging Protocol (XMPP)-based middleware, which tries to provide a real-time, safe, and trustworthy communication channel among heterogeneous devices. VIRTUS is a publishing/subscribing middleware. With XMPP on the side, it will allow the users of the system to be acknowledged if a message arrived at the destination or not. In addition, it can inform entities if the destination user is offline. VIRTUS can support three types of devices: resource rich devices (such as PC), resource-poor devices (such as tablets) and simple devices (such as sensors) [18].

7. Challenges and issues

This section discusses the main challenges that exist on using middlewares technologies in CoT applications. All challenges mentioned are interesting problems and open issues that need further investigation and have great potential to be considered as future directions in this study.

7.1. Near real-time prioritizing

CoT is a massive network of intelligent objects, which all of them provide or receive resources to fulfill their tasks. There are situations that resource/service allocation should be prioritized in real-time. Most of the CoT-based middlewares are not capable of prioritizing unforeseen tasks in real-time. For example in a CoT-based hospital, the middleware should be able to prioritize the patients' care based on their sickness in real-time manner and retrieve patient data that is in worse condition to be notified on doctors' platform. Therefore, the middlewares need to apply mechanisms to measure unforeseen situation in real-time and provide cloud services according to the importance of the requests.

7.2. Proper resource discovery implementation

CoT is a heterogeneous environment with the most dynamic topology changes. There is no guarantee for continuity of services/resources. There are many factors that make Resource Discovery challenging in this type of network (i.e., Nodes are added to the network or leave it suddenly, the bursting nature of network, Asynchronous interface protocols, Mobility). Recent research attempted to answer some of the requirements, but they were not comprehensive methods or solutions. For this purpose, moving toward a more capable resource/service discovery mechanisms that provide the expected performance is strictly necessary. Presently, the most widely used discovery protocols are XMPP, UPnP, and WSDiscovery.

7.3. Users security and privacy enhancement

Implementing trustable security and privacy methods is one of the major challenges in pervasive networks. Some middlewares use their local security capabilities to evaluate the system safety but, when we use cloud services, there would be some concerns. All CoT-based middlewares that we mentioned rely on security measures that cloud providers have guaranteed. A security management platform on user-side that provides the monitoring and controlling measures on cloud-side operations would be essential. Privacy is equally important because CoT is a composition of ubiquitous devices and is exposed to malicious devices. Common authentication and identification methods are not efficient due to frequent context changes, devices resource limitations, spontaneous device acquisition and others.

7.4. Supporting various interface protocol for E-healthcare domain

E-health can utilize CoT to provide various services, such as patients monitoring. However, designing a suitable middleware on this sensitive domain demands more efforts. Several CoT sensors, actuators, hubs or in general, things may be implanted into the patients body or carried by them as a wearable device. There has been inadequate attention on the

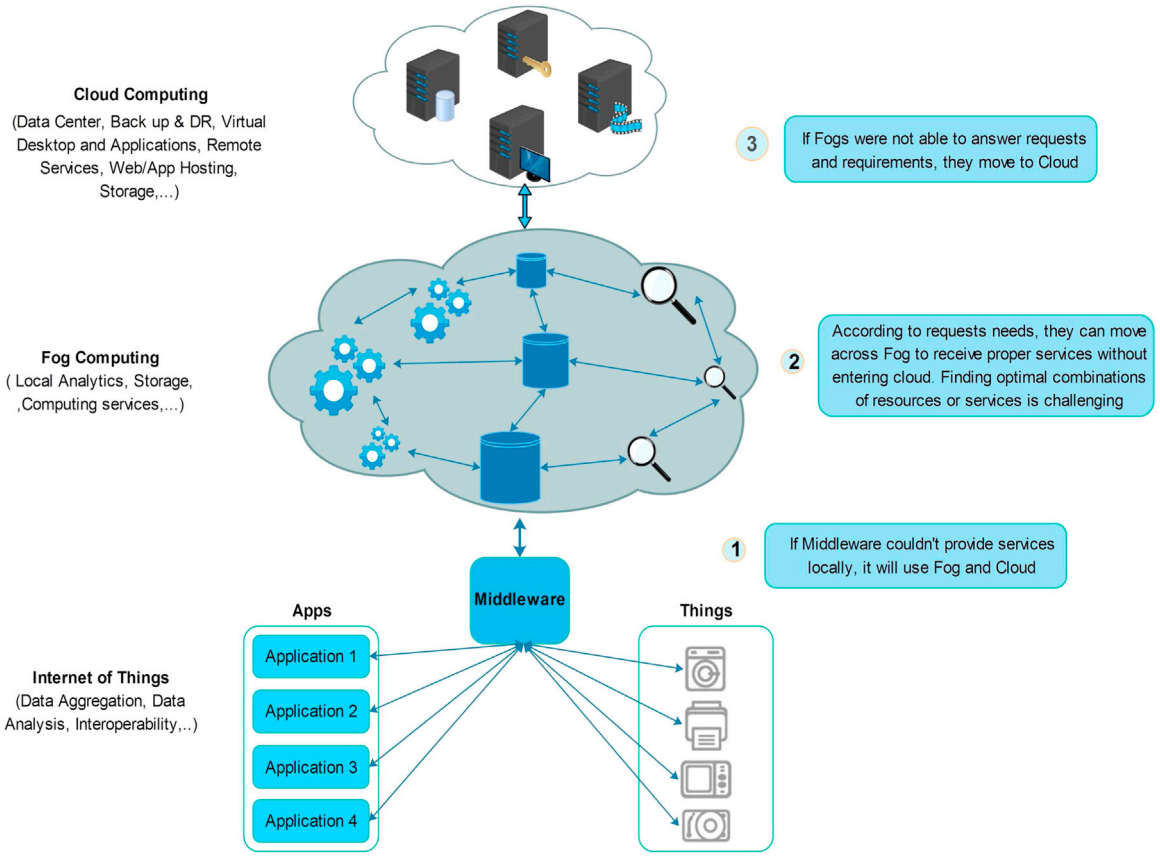

Fig. 4. Implementing CoT-based Middleware using Fog Computing.

harmful effects of communication signals to the humans body. Developers need to put additional efforts to design a middleware that can support maximum flexibility in deploying various interface protocols. For example, the authors in [43] investigated millimeter-waves frequencies side effects and threats on the humans body.

7.5. A light weight CoT-based middleware for ultra resources constrained devices

Internet of Nano Things (IoNT) is another IoT paradigm that is growing fast. It can be forecasted that this technology shapes a considerable proportion of IoT in the near future. There are several reasons that IoNT has attracted the attention of researchers and investors, but the main reasons are cost efficiency and high deployment density, which increase system accuracy. As we can expect, to implement this paradigm, it needs ultra resource-constrained devices that have limitations in processing capability, stored energy, storage capacity, communication range and others. Therefore, designing a light-weight dedicated CoT-based middleware can resolve many implementations issues.

7.6. Finding optimal place for data analysis by Fog Computing

Fog-Computing [44] has the potential to fulfill some of the requirements of CoT, such as lower latency, mobility support, and location-awareness. In this regard, the authors in [45] reviewed Fog Computing in CoT and underlined Fog as a middleware in Cloud Computing. Fog provides computation, storage, and networking services and stands between end nodes of IoT and traditional Clouds. One of the key functions is finding the optimal place for data analysis by Fog Computing. As shown in Fig. 4, the Fog layer is between IoT and Cloud and tries to be as close as possible to the underlying system for better real-time performance. On the other hand, the variation of the requested task from the middleware is significant. For instance, in real-time traffic classification, middleware may receive elastic, semi-elastic, soft real-time or hard real-time requests. Determining to what extent requests can move into Fog or Cloud is challenging. Passing them to the nearest or furthest distance is not always the most effective option. Given that the variation of services in Fog is considerable, it is necessary to perform some accurate calculations and assessments to determine the optimal selection or combination of resources for requested operations. Except for real-time classification, many other factors need Fog to find the most efficient place for processing them. The recent works were not really successful to implement this feature. The realization of this feature significantly has an effect on cloud computing position in industrial market.

7.7. Providing quality of service

In CoT, Providing SLA and guaranteeing QoS is quite challenging. Here, the dynamics of the network is high and predicting system behavior is a big problem. Furthermore, Cloud limitations even make the situation worse. As the main requirements of QoS, we can include Bandwidth, Delay, Jitter and Packet Drop. The nature of the Cloud prevents to address those requirements. To implement a desired level of QoS, Fog Computing can help by filling the gaps with better time service (Table 5). However, these are just basic requirements of providing QoS and researchers need to design proper mechanisms to deploy them efficiently.

7.8. Standardization needs

Based on the conducted literature studies, we found that one of the main and obvious problems is the lack of standardization in designing approaches. As a suggestion for future work, a proposal of layered-based model can be helpful and provides significant stability. However, by

Table 4
CoT-based Middleware vs. non CoT-based Middleware ("Tie" shows the feature is equally available in both CoT and non-CoT scenario and there is no competitive advantage in each one).

Functions	CoT	non-CoT	Description
Adaptability	Tie		In CoT-based Middleware, dynamics for adaptability is high but non CoT-based Middleware is quicker for critical situations.
Connectivity Convergence	✓	–	Cloud can provide more variety in interface protocol.
Context Management	✓	–	In Context-aware platforms, Context Management needs continuous processing that resides it in the cloud allows Middleware to save processing power.
Energy Efficiency	✓	–	It could be a tie because CoT-based Middleware consumes more energy for exchanging messages but non CoT-based Middleware use more processing power and in general the later one is more considerable.
Flexibility	–	✓	In non CoT-based Middleware, developers can provide services as they want and it is third-party platform- independent.
Interoperability	✓	–	Cloud inherently is interoperable therefore can provide better range of heterogeneous devices.
Maintainability	–	✓	In crisis time, it is essential for systems to perform appropriate operations in a fraction of time. non CoT-based Middleware is the better choice to fulfill this task.
Platform Portability	✓	–	IoT is the composition of heterogeneous devices. Encountering unknown platforms is possible in non CoT-based Middleware due to weak forecasting or resource limitation to cover all platforms. Cloud can address these type of issues.
Quality of Service	–	✓	Providing QoS in non CoT-based Middleware is more reasonable because there is more control on infrastructure.
Real-Time Tasks	–	✓	non CoT-based Middleware performs this feature better because there is less latency on performing tasks.
Resource Discovery	–	✓	Resources change at any moment. We can refer situations that resources lifetime in network is less than cloud-based Resource Discovery process time.
Reusability	✓	–	SoA is the main architecture style in CoT-based Middleware. This Architecture is known for reusability. CoT uses everything as a service scheme that is a heaven for reusability feature.
Security and Privacy	Tie		Cloud services implement security much safer (i.e., the possibility of DDOS attack reduces significantly). In contrary, Non CoT-based Middleware is reasonable for privacy. So a tie is a fair result.
Transparency	✓	–	Transparency and abstraction is more sensible in Cloud environment.
Trustworthiness Management	✓	–	This feature is crucial in Social Internet of Things. So we expect an enormous environment and a high volume of data exchanged. Therefore using cloud is much better.

considering the heterogeneity nature of Internet of Things it may seem quite challenging and add more complexity to the already complex scheme.

In summary and as concluding remarks of this study, we present a head to head table (Table 4) that compares different features of CoT-based and non-CoT-based Middleware. We found that CoT-based Middleware are suitable in several use cases, including social, smart home, smart city, smart agriculture, smart animal farming, smart grids, and smart retail applications, which produce massive data and donot need hard real-time processing. on the contrary, non-CoT-based middlewares can be more utilizable in industrial control, E-healthcare, and smart logistics domains, which need hard real-time processing.

Table 5
Cloud and Fog Computing QoS Requirements.

Requirements	Cloud	Fog
Latency	High	Low
Jitter	High	Very Low
Distance	Through Internet	Limited Hop Count
Hard Real-time	No or Limited	Yes
Mobility	Limited	Supported
Bandwidth	More Demand	Less Demand

8. Conclusion

Cloud of Things (CoT) platforms will play a key role in the near future for enabling service provision in large internet of things environments. Toward that goal, understanding the role and necessity of middlewares in CoT is important. This study focuses on middleware technologies in CoT in three main directions. First, it presents the major features and characteristics of middlewares in CoT domains and compares them from different architectural design possibilities. Next, several middlewares are introduced and studied based on their architectures, service domain, and application. Finally, a number of open challenges and issues are presented that are interesting problems to be tackled as future direction of this study.

References

[1] D. Evans, The internet of things-how the next evolution of the internet is changing everything, Tech. Rep., Cisco (04 2011).
[2] L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Netw. 54 (15) (2010) 2787–2805.
[3] Technical report, infso d.4 networked enterprise & rfid infso g.2 micro & nanosystems, in: Co-operation with the working group rfid of the etp eposs, internet of things in 2020, roadmap for the future, version 1.1; 27 May 2008.
[4] Itu Internet Reports, the Internet of Things, ⟨http://www-cs-faculty.stanford.edu/~uno/abcde.html⟩.
[5] V. Issarny, M. Caporuscio, N. Georgantas, A perspective on the future of middleware-based software engineering, in: Future of Software Engineering, FOSE '07, 2007, pp. 244–258.
[6] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.
[7] J. Zhou, T. Lepp?Nen, E. Harjula, M. Ylianttila, T. Ojala, C. Yu, H. Jin, Cloudthings: A common architecture for integrating the internet of things with cloud computing, in: IEEE Proceedings of the 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2013, pp. 651–657.
[8] A. Botta, W. de Donato, V. Persico, A. Pescapé, Integration of cloud computing and internet of things: a survey, Future Gener. Comput. Syst. 56 (2016) 684–700.
[9] B.B.P. Rao, P. Saluia, N. Sharma, A. Mittal, S.V. Sharma, Cloud computing for internet of things amp; sensing based applications, in: Proceedings of the Sixth International Conference on Sensing Technology (ICST), 2012, pp. 374–380.
[10] F. Tao, Y. Cheng, L.D. Xu, L. Zhang, B.H. Li, Cciot-cmfg: cloud computing and internet of things-based cloud manufacturing service system, IEEE Trans. Ind. Inform. 10 (2) (2014) 1435–1442.
[11] M. Aazam, I. Khan, A.A. Alsaffar, E.N. Huh, Cloud of things: Integrating internet of things and cloud computing and the issues involved, in: Proceedings of the 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 2014, pp. 414–419.
[12] G. Noto La Diega, Clouds of things: Data Protection and Consumer Law at the Intersection of Cloud Computing and the Internet of Things in the United Kingdom.
[13] M. Aazam, P.P. Hung, E.N. Huh, Smart gateway based communication for cloud of things, in: IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014, pp. 1–6. https://doi.org/10.1109/ISSNIP.2014.6827673.
[14] K. Gama, L. Touseau, D. Donsez, Combining heterogeneous service technologies for building an internet of things middleware, Comput. Commun. 35 (4) (2012) 405–417.
[15] Z. Song, A.A. Cardenas, R. Masuoka, Semantic middleware for the internet of things, Internet Things (IOT) (2010) 1–8.
[16] C. Perera, A. Zaslavsky, P. Christen, M. Compton, D. Georgakopoulos, Context-aware sensor search, selection and ranking model for internet of things middleware,

in: IEEE Proceedings of the 14th International Conference on Mobile Data Management, 1, 2013, pp. 314–322.
[17] C. Perera, P.P. Jayaraman, A. Zaslavsky, P. Christen, D. Georgakopoulos, Mosden: An internet of things middleware for resource constrained mobile devices, in: Proceedings of the 47th Hawaii International Conference on System Sciences, 2014, pp. 1053–1062.
[18] M. Bazzani, D. Conzon, A. Scalera, M.A. Spirito, C.I. Trainito, Enabling the iot paradigm in e-health solutions through the virtus middleware, in: IEEE Proceedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 1954–1959.
[19] A. Devaraju, S. Hoh, M. Hartley, A context gathering framework for context-aware mobile solutions, in: Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology, Mobility'07, ACM, 2007, pp. 39–46.
[20] L.D. Xu, W. He, S. Li, Internet of things in industries: a survey, IEEE Trans. Ind. Inform. 10 (4) (2014) 2233–2243.
[21] J. Al-Jaroodi, I. Jawhar, A. Al-Dhaheri, F. Al-Abdouli, N. Mohamed, Security middleware approaches and issues for ubiquitous applications, Comput. Math. Appl. 60 (2) (2010) 187–197 (advances in Cryptography, Security and Applications for Future Computer Science).
[22] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Context aware computing for the internet of things: A survey, IEEE Communications Surveys Tutorials 16 (1).
[23] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L.M.S.d. Souza, V. Trifa, Soa-based integration of the internet of things in enterprise services, in: IEEE International Conference on Web Services, 2009, pp. 968–975.
[24] K.J. Kwon, C.B. Park, H. Choi, Ddss: A communication middleware based on the dds for mobile and pervasive systems, in: Proceedings of the 10th International Conference on Advanced Communication Technology, , vol. 2, 2008, pp. 1364–1369. https://doi.org/10.1109/ICACT.2008.4494018.
[25] J. Wu, P. Teregowda, M. Khabsa, S. Carman, D. Jordan, J. San Pedro Wandelmer, X. Lu, P. Mitra, C.L. Giles, Web crawler middleware for search engine digital libraries: A case study for citeseerx, in: Proceedings of the Twelfth International Workshop on Web Information and Data Management, WIDM'12, ACM, 2012, pp. 57–64.
[26] K. Taha, D. Homouz, H. Al Muhairi, Z. Al Mahmoud, Grank: a middleware search engine for ranking genes by relevance to given genes, BMC Bioinform. 14 (1) (2013) 251.
[27] C. Perera, D.S. Talagala, C.H. Liu, J.C. Estrella, Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in iot clouds, IEEE Trans. Comput. Social. Syst. 2 (4) (2015) 171–181.
[28] A. Ben Hamida, F. Kon, N. Lago, A. Zarras, D. Athanasopoulos, D. Pilios, P. Vassiliadis, N. Georgantas, V. Issarny, G. Mathioudakis, G. Bouloukakis, Y. Jarma, S. Hachem, A. Pathak, Integrated CHOReOS middleware - Enabling large-scale, QoS-aware adaptive choreographies, working paper or preprint (Sep. 2013). ⟨https://hal.inria.fr/hal-00912882⟩.
[29] C. Huo, T.-C. Chien, P.H. Chou, Middleware for iot-cloud integration across application domains, IEEE Design and Test 31 (3).
[30] T.L. Vinh, S. Bouzefrane, J.-M. Farinone, A. Attar, B.P. Kennedy, Middleware to integrate mobile devices, sensors and cloud computing, Procedia Comput. Sci. 52 (2015) 234–243.
[31] Z. Ji, I. Ganchev, M. Droma, L. Zhao, X. Zhang, A cloud-based car parking middleware for iot-based smart cities: design and implementation, Sensors 14 (12) (2014) 22372.
[32] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P.Calbimonte, M. Riahi, K. Aberer, P.P. Jayaraman, A. Zaslavsky, I.P. ˇZarko, L. Skorin-Kapov, R. Herzog, OpenIoT:Open Source Internet-of-Things in the Cloud, 2015, pp. 13–25.
[33] S.W. Han, Y.B. Yoon, H.Y. Youn, W.-D. Cho, A new middleware architecture for ubiquitous computing environment, 2nd IEEE Workshop Softw. Technol. Future Embed. Ubiquitous Syst. (2004) 117–121.
[34] O. Davidyuk, J. Riekki, V.-M. Rautio, J. Sun, Context-aware middleware for mobile multimedia applications, in: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, MUM'04, ACM, 2004, pp. 213–220.
[35] L. Capra, W. Emmerich, C. Mascolo, Carisma: context-aware reflective middleware system for mobile applications, IEEE Trans. Softw. Eng. 29 (10) (2003) 929–945.
[36] A. Badii, M. Crouch, C. Lallah, A context-awareness framework for intelligent networked embedded systems, in: Proceedings of the Third International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services, 2010, pp. 105–110.
[37] K. Aberer, M. Hauswirth, A. Salehi, A middleware for fast and flexible sensor network deployment, in: Proceedings of the 32Nd International Conference on Very Large Data Bases, VLDB '06, 2006, pp. 1199–1202.
[38] F. Li, S. Sehic, S. Dustdar, Copal: An adaptive approach to context provisioning, in: IEEE Proceedings of the 6th International Conference on Wireless and Mobile Computing, Networking and Communications, 2010, pp. 286–293.
[39] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, K. Nahrstedt, A middleware infrastructure for active spaces, IEEE Pervasive Comput. 1 (4) (2002) 74–83.
[40] S. Gashti, G. Pujolle, J. Rotrou, An upnp-based context-aware framework for ubiquitous mesh home networks, in: IEEE Proceedings of the 20th International Symposium on Personal, Indoor and Mobile Radio Communications, 2009, pp. 400–404.
[41] W. Wibisono, A. Zaslavsky, S. Ling, Comihoc: A middleware framework for context management in manet environment, in: Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, 2010, pp. 620–627.
[42] T. Gu, H. K. Pung, D. Q. Zhang, A middleware for building context-aware mobile services, in: IEEE Proceedings of the 59th Vehicular Technology Conference. VTC 2004-Spring, 5, 2004, pp. 2656–2660.
[43] T. Wu, T.S. Rappaport, C.M. Collins, Safe for generations to come: considerations of safety for millimeter waves in wireless communications, IEEE Microw. Mag. 16 (2) (2015) 65–84.
[44] I. Stojmenovic, S. Wen, X. Huang, H. Luan, An overview of fog computing and its security issues, Concurr. Comput.: Pract. Exp. 28 (10) (2016) 2991–3005.
[45] M. Aazam, E.N. Huh, Fog computing: the cloud-iot/ioe middleware paradigm, IEEE Potentials 35 (3) (2016) 40–44.