Middleware for Internet of Things: A Survey

Mohammad Abdur Razzaque, *Member, IEEE*, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke

***Abstract*—The Internet of Things (IoT) envisages a future in which digital and physical things or objects (e.g., smartphones, TVs, cars) can be connected by means of suitable information and communication technologies, to enable a range of applications and services. The IoT's characteristics, including an ultra-large-scale network of things, device and network level heterogeneity, and large numbers of events generated spontaneously by these things, will make development of the diverse applications and services a very challenging task. In general, middleware can ease a development process by integrating heterogeneous computing and communications devices, and supporting interoperability within the diverse applications and services. Recently, there have been a number of proposals for IoT middleware. These proposals mostly addressed wireless sensor networks (WSNs), a key component of IoT, but do not consider RF identification (RFID), machine-to-machine (M2M) communications, and supervisory control and data acquisition (SCADA), other three core elements in the IoT vision. In this paper, we outline a set of requirements for IoT middleware, and present a comprehensive review of the existing middleware solutions against those requirements. In addition, open research issues, challenges, and future research directions are highlighted.**

***Index Terms*—Internet of Things (IoT) characteristics, machine-to-machine (M2M) communication, middleware requirements, RF identification (RFID), supervisory control and data acquisition (SCADA), wireless sensor networks (WSNs).**

I. Introduction

WITH THE advance of numerous technologies including sensors, actuators, embedded computing and cloud computing, and the emergence of a new generation of cheaper, smaller wireless devices, many objects, or things in our daily lives are becoming wirelessly interoperable with attached miniature and low-powered or passive wireless devices (e.g., passive RF identification (RFID) tags). The Wireless World Research Forum predicts that by 2017, there will be 7 trillion wireless devices serving 7 billion people [1] (i.e., 1000 devices/person). This ultra large number of connected things or devices will form the IoT [2], [3].

By enabling easy access of, and interaction with, a wide variety of physical devices or things such as, home appliances, surveillance cameras, monitoring sensors, actuators, displays, vehicles, machines and so on, the IoT will foster the development of applications in many different domains, such as home automation, industrial automation, medical aids, mobile healthcare, elderly assistance, intelligent energy management and smart grids, automotive, traffic management, and many others [4]. These applications will make use of the potentially enormous amount and variety of data generated by such objects to provide new services to citizens, companies, and public administrations [3], [5]–[8].

In a ubiquitous computing environment like IoT, it is impractical to impose standards and make everyone comply. An ultra-large-scale network of things and the large number of events that can be generated spontaneously by these things, along with heterogeneous devices/technologies/applications of IoT bring new challenges in developing applications, and make the existing challenges in ubiquitous computing considerably more difficult [2], [3]. In this context, a middleware can offer common services for applications and ease application development by integrating heterogeneous computing and communications devices, and supporting interoperability within the diverse applications and services running on these devices. A number of operating systems have been developed [9]–[16] to support the development of IoT middleware solutions. In general, these reside on the physical devices, and provide the necessary functionalities to enable service deployment. Complementary to middleware are programming language approaches [17], [18]. These approaches tackle some of the challenges (such as discovery, network disconnections, and group communication) posed by the IoT, but are limited in their support for others such as context-awareness (e.g., context-aware service discovery) and scalability.

Wireless sensor networks (WSNs), RFID, machine-to-machine (M2M) communications, and supervisory control and data acquisition (SCADA) are the four essential components (Fig. 2) of IoT [19], [20]. A fully functional IoT middleware needs to integrate these technologies to support the envisioned diverse application domains [19]. To date, the majority of the existing IoT middleware proposals [21]–[29] are WSNs centric. Many surveys have been conducted on WSNs middlewares [30]–[36], these are either not comprehensive [34]–[36] or do not report more recent work [30]–[32]. From these surveys, it is evident that no single existing middleware can support all the necessary requirements for WSNs or IoT applications. For instance, Perera *et al.* [20] identified that most existing WSN middleware and IoT-focused solutions do not support context-awareness. In addition, unlike WSNs, the number of middleware proposals for RFID as well as M2M communications, and SCADA is limited [19], [37]–[41]. On the other hand, in the recent years, IoT-specific middlewares are emerging [19], [25], [42]–[46] as are some surveys [19], [24], [47]. Bandyopadhyay *et al.* [24], [47] have focused on highlighting the importance of a middleware system in IoT, and do not

Manuscript received April 8, 2015; revised July 9, 2015; accepted October 21, 2015. Date of publication November 09, 2015; date of current version January 20, 2016. This work was supported by the Science Foundation Ireland (SFI) under the Service-centric networking for urban-scale feedback systems (SURF) project.

The authors are with the Distributed Systems Group, School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland (e-mail: razzaqum@scss.tcd.ie; milojavm@scss.tcd.ie; paladea@scss.tcd.ie; Siobhan.Clarke@scss.tcd.ie).

Digital Object Identifier 10.1109/JIOT.2015.2498900

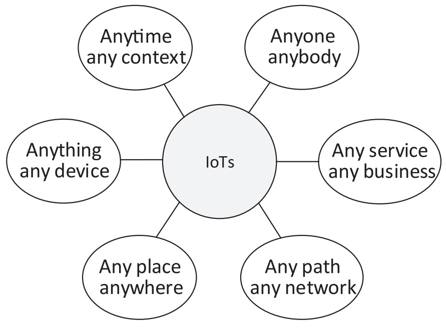

Fig. 1. Definition of IoT [52].

include most IoT-specific middlewares [19], [25], [44]–[46]. Zhou has presented only a conceptual view of a unified framework for IoT middleware based on service orientation [19]. Moreover, this work does not include recent, and IoT-specific middlewares [25], [46].

Considering the importance of IoT in various domains, this paper takes a holistic view of middleware for IoT and: 1) identifies the key characteristics of IoT, and the requirements of IoT's middleware (Section II); 2) based on the identified requirements, presents a comprehensive review of the existing middleware systems focusing on current, state-of-the-art research (Section III); and (3) outlines open research challenges, recommending future research directions (Section IV).

II. Background

A. IoT and Its Characteristics

Research into the IoT is still in its early stage, and a standard definition of the IoT is not yet available. IoT can be viewed from three perspectives: 1) Internet-oriented; 2) things-oriented (sensors or smart things); and 3) semantic-oriented (knowledge) [6]. Also, the IoT can be viewed as either supporting consumers (human) or industrial applications and indeed could be named as the human Internet of Things (HIoT) or the industrial Internet of Things (IIoT) [19], [48]–[50]. Even though these different views have evolved because of the interdisciplinary nature of the subject, they are likely to intersect in an application domain to achieve the IoT's goals.

The first definition of the IoT was from a "things-oriented" perspective, where RFID tags were considered as things [6]. According to the RFID community, IoT can be defined as, "The worldwide network of interconnected objects uniquely addressable based on standard communication protocols" [51]. Fig. 1 illustrates the European research cluster of IoT (IERC) definition, where "The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network, and any service" [52], [53]. The International Telecommunication Union (ITU) views IoT very similarly: "From anytime, anyplace connectivity for anyone, we will now have connectivity for anything" [54]. Semantically, IoT means "A world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" [51].

Most definitions of IoT do not explicitly highlight the industrial view of IoT (IIoT). World leading companies are giving special attention and making significant investments in the IoT for their industrial solutions (IIoT). Even though they use different terms such as "Smarter Planet" by IBM, "Internet of Everything" by Cisco and "Industrial Internet" by GE, their main objective is to use IoT to improve industrial production by reducing unplanned machine downtime and significantly reducing energy costs along with number of other potential benefits [19], [48]–[50], [55]. The IIoT refers to industrial objects, or "things," instrumented with sensors, automatically communicating over a network, without human-to-human or human-to-computer interaction, to exchange information and take intelligent decisions with the support of advanced analytics [50].

The definition of "things" in the IoT vision is very wide and includes a variety of physical elements. These include personal objects we carry around such as smart phones, tablets, and digital cameras. It also includes elements in our environments (e.g. home, vehicle, or work), industries (e.g., machines, motor, robot) as well as things fitted with tags (e.g., RFID), which become connected via a gateway device (e.g., a smart phone). Based on this view of "things," an enormous number of devices will be connected to the Internet, each providing data and information, and some, even services.

Sensor networks (SNs) including WSNs and wireless sensor and actuator networks (WSANs), RFID, M2M communications, and SCADA are the essential components of IoT. As described in more detail in this section, a number of the IoT's characteristics are inherited from one or more of these components. For instance, "resource-constrained" is inherited from RFID and SNs, and "intelligence" is inherited from WSNs and M2M. Other characteristics (e.g., ultra-large-scale network, spontaneous interactions) are specific to the IoT. The main characteristics of the IoT are presented from infrastructure and application perspectives.

1) Characteristics of IoT Infrastructure:

- *Heterogeneous devices:* The embedded and sensor computing nature of many IoT devices means that low-cost computing platforms are likely to be used. In fact, to minimize the impact of such devices on the environment and energy consumption, low-power radios are likely to be used for connection to the Internet. Such low-power radios do not use WiFi, or well-established cellular network technologies. However, the IoT will not be composed only of embedded devices and sensors, it will also need higher-order computing devices to perform heavier duty tasks (routing, switching, data processing, etc.). Device heterogeneity emerges not only from differences in capacity and features, but also for other reasons including multivendor products and application requirements. [4], [54]. Fig. 2 illustrates six different types of IoT devices.
- *Resource-constrained:* Embedded computing and sensors need a small device form factor, which limits their processing, memory, and communication capacity. As shown in Fig. 2, resource capacity (e.g., computational, connectivity capabilities, and memory requirements) decreases moving from left to right. For example, RFID devices or

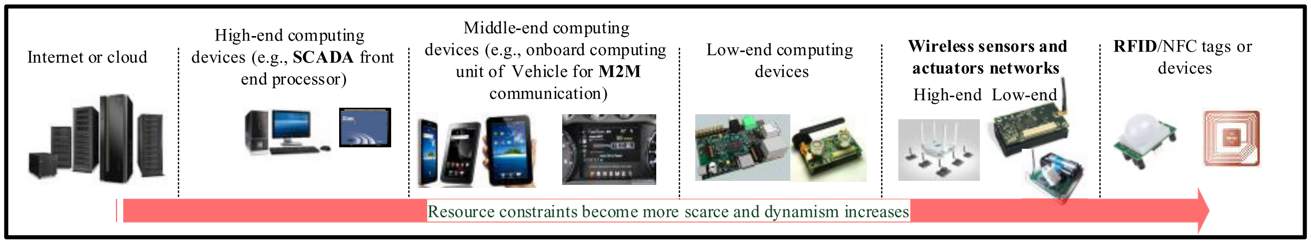
Internet or cloud
High-end computing devices (e.g., SCADA front end processor)
Middle-end computing devices (e.g., onboard computing unit of Vehicle for M2M communication)
Low-end computing devices
Wireless sensors and actuators networks
High-end Low-end
RFID/NFC tags or devices
Resource constraints become more scarce and dynamism increases

Fig. 2. Examples of device heterogeneity in IoT.

tags (in the right-most side of this figure) may not have any processing capacity or even battery to power them. On the other hand, in Fig. 2, devices become expensive and larger in form-factor when moving to the left.

- *Spontaneous interaction:* In IoT applications, sudden interactions can take place as objects move around, and come into other objects' communication range, leading to the spontaneous generation of events. For instance, a smartphone user can come in close contact with a TV/fridge/washing machine at home and that can generate events without the user's involvement. Typically, in IoT, an interaction with an object means that an event is generated and is pushed to the system without much human attention.
- *Ultra-large-scale network and large number of events:* In an IoT environment, thousands of devices or things may interact with each other even in one local place (e.g., in a building, supermarket, and university), which is much larger scale than most conventional networking systems. Globally, the IoT will be an ultra-large-scale network containing nodes in the scale of billions and even in trillions. Gartner has predicted [56] that there will be nearly 26 billion devices on the IoT by 2020. Similarly, ABI research [57] estimated that more than 30 billion devices will be wirelessly connected by 2020. In the IoT, spontaneous interactions among an ultra large number of things or devices will produce an enormous number of events as normal behavior. This uncontrolled number of events may cause problems such as event congestion and reduced event processing capability.
- *Dynamic network and no infrastructure:* As shown in Fig. 2, IoT will integrate devices, many of which will be mobile, wirelessly connected, and resource constrained. Mobile nodes within the network leave or join anytime they want. Also, nodes can be disconnected due to poor wireless links or battery shortage. These factors will make the network in IoT highly dynamic. Within such an *ad hoc* environment, where there is limited or no connection to a fixed infrastructure, it will be difficult to maintain a stable network to support many application scenarios that depend on the IoT. Nodes will need to cooperate to keep the network connected and active.
- *Context-aware:* Context is key in the IoT and its applications. A large number of sensors will generate large amounts of data, which will not have any value unless it is analyzed, interpreted, and understood. Context-aware computing stores context information related to sensor data, easing its interpretation. Context-awareness (especially in temporal and spatial context) plays a vital role in the adaptive and autonomous behavior of the things in the IoT [20], [58]. Such behavior will help to eliminate human-centric mediation in the IoT, which ultimately makes it easier to perform M2M communication, a core element of the IoT's vision.
- *Intelligence:* According to Intel's IoT vision, intelligent devices or things and intelligent systems of systems are the two key elements of IoT [59]. In IoT's dynamic and open network, these intelligent entities along with other entities such as Web services (WSs), SOA components, and virtual objects will be interoperable and able to act independently based on the context, circumstances, or environments [60], [61].
- *Location-aware:* Location or spatial information about things (objects) or sensors in IoT is critical, as location plays a vital role in context-aware computing. In a large-scale network of things, interactions are highly dependent on their locations, their surroundings, and presence of other entities (e.g., things and people).
- *Distributed:* The traditional Internet itself is a globally distributed network, and so also is the IoT. The strong spatial dimension within the IoT makes the network IoT distributed at different scales (i.e., both globally like the Internet, and also locally within an application area).

2) Characteristics of IoT Applications:

- *Diverse applications:* The IoT can offer its services to a large number of applications in numerous domains and environments. These domains and environments can be grouped into (nonexhaustive) domain categories such as: 1) transportation and logistics; 2) healthcare; 3) smart environment (home, office, and plant); 4) industrial; and 5) personal and social domain. Fig. 3 highlights some key application domains for the IoT. Different applications are likely to need different deployment architectures (e.g., event-driven and time-driven) and have different requirements. However, since the IoT is connected to the Internet, most of the devices comprising IoT services will need to operate within an environment that supports their mutual understanding.
- *Real time:* Applications using the IoT can be broadly classified as real time and non-real time. For instance, IoT for healthcare, transportation will need on-time delivery of their data or service. Delayed delivery of data can make the application or service useless and even dangerous in mission critical applications.

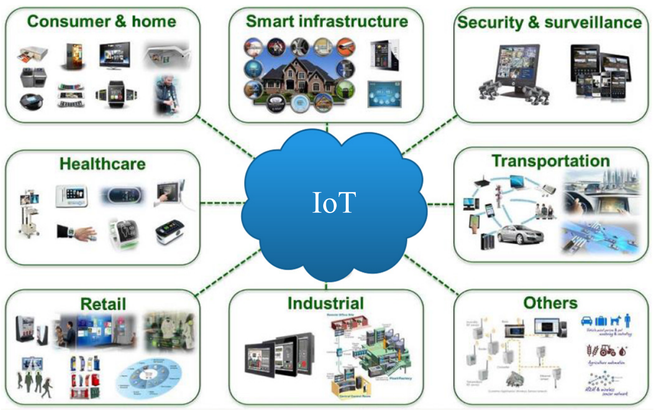

Fig. 3. Potential applications of IoT [66].

- *Everything-as-a-service (XaaS):* An everything-as-a-service model is very efficient, scalable, and easy to use [62]. The XaaS model has inspired the sensing as a service approach in WSNs [63], [64], and this may inevitably lead IoT toward an XaaS model. As more things get connected, the collection of services is also likely to grow, and as they become accessible online, they will be available for use and reuse.
- *Increased security attack-surface:* While there is huge potential for the IoT in different domains, there are also concerns for the security of applications and networks. The IoT needs global connectivity and accessibility, which means that anyone can access it anytime and anyway. This tremendously increases the attack surfaces for the IoT's applications and networks. The inherent complexity of the IoT further complicates the design and deployment of efficient, interoperable, and scalable security mechanisms.
- *Privacy leakage:* Using the IoT, applications may collect information about people's daily activities. As information reflecting such activities (e.g., travel routes, buying habits, and daily energy usage) is considered by many individuals as private, exposure of this information could impact the privacy of those individuals. The use of cloud computing makes the problem of privacy leakage even worse. Any IoT application not compliant with privacy requirements could be prohibited by law (e.g., in the EU [65]) because they violate citizens' privacy.

B. Middleware in IoT and Its Requirements

Generally, a middleware abstracts the complexities of the system or hardware, allowing the application developer to focus all his effort on the task to be solved, without the distraction of orthogonal concerns at the system or hardware level [67]. Such complexities may be related to communication concerns or to more general computation. A middleware provides a software layer between applications, the operating system and the network communications layers, which facilitates and coordinates some aspect of cooperative processing. From the computing perspective, a middleware provides a layer between application software and system software. In the IoT, there is likely to be considerable heterogeneity in both the communication technologies in use, and also the system level technologies, and a middleware should support both perspectives as necessary. Based on previously described characteristics of the IoT's infrastructure and the applications that depend on it, a set of requirements for a middleware to support the IoT is outlined. As follows, these requirements are grouped into two sets: 1) the services such a middleware should provide and 2) the system architecture should support.

1) Middleware Service Requirements: Middleware service requirements for the IoT can be categorized as both functional and nonfunctional. Functional requirements capture the services or functions (e.g., abstractions, resource management) a middleware provides and nonfunctional requirements (e.g., reliability, security, and availability) capture QoS support or performance issues.

The view of a middleware in this paper is one which provides common or generic services to multiple different application domains. In this section, no attempt is made to capture domain or application-specific requirements, as the focus is on generic or common *functional* ones, as follows.

- *Resource discovery:* IoT resources include heterogeneous hardware devices (e.g., RFID tags, sensors, sensor mote, and smartphones), devices' power and memory, analogue to digital converter devices (A/D), the communications module available on those devices, and infrastructural or network level information (e.g., network topology and protocols), and the services provided by these devices. Assumptions related to global and deterministic knowledge of these resources' availability are invalid, as the IoT's infrastructure and environment is dynamic. By necessity, human intervention for resource discovery is infeasible, and therefore, an important requirement for resource discovery is that it be automated. Importantly, when there is no infrastructure network, every device must announce its presence and the resources it offers. This is a different model to centralized distributed systems, where resource publication, discovery, and communication are generally managed by a dedicated server. Discovery mechanisms also need to scale well, and there should be efficient distribution of discovery load, given the IoT's composition of resource-constrained devices.
- *Resource management:* An acceptable QoS is expected for all applications, and in an environment where resources that impact on QoS are constrained, such as the IoT, it is important that applications are provided with a service that manages those resources. This means that resource usage should be monitored, resources allocated or provisioned in a fair manner, and resource conflicts resolved. In IoT architectures, especially in service-oriented or virtual machine (VM)-based architectures, middleware needs to facilitate potentially spontaneous resource (service) (re)composition, to satisfy application needs.
- *Data management:* Data are key in IoT applications. In the IoT, data refer mainly to sensed data or any network infrastructure information of interest to applications. An IoT middleware needs to provide data management

services to applications, including data acquisition, data processing (including preprocessing), and data storage. Preprocessing may include data filtering, data compression, and data aggregation.

- *Event management:* There are potentially a massive number of events generated in IoT applications, which should be managed as an integral part of an IoT middleware. Event management transforms simple observed events into meaningful events. It should provide real-time analysis of high-velocity data so that downstream applications are driven by accurate, real-time information, and intelligence.
- *Code management:* Deploying code in an IoT environment is challenging, and should be directly supported by the middleware. In particular, code allocation and code migration services are required. Code allocation selects the set of devices or sensor nodes to be used to accomplish a user or application level task. Code migration transfers one node/device's code to another one, potentially reprogramming nodes in the network. Using code migration services, code is portable, which enables data computation to be relocated.

Key *nonfunctional* requirements of IoT middleware are as follows.

- *Scalability:* An IoT middleware needs to be scalable to accommodate growth in the IoT's network and applications/services. Considering the size of the IoT's network, IPv6 is a very scalable solution for addressability, as it can deal with a huge number of things that need to be included in the IoT [68]. Loose coupling and/or virtualization in middleware is useful in improving scalability, especially application and service level scalability, by hiding the complexity of the underlying hardware or service logic and implementation.
- *Real time or timeliness:* A middleware must provide real-time services when the correctness of an operation that supports depends not only on its logical correctness but also on the time in which it is performed. As the IoT will deal with many real-time applications (e.g., transportation, healthcare), on-time delivery of information or services in those applications is critical. Delayed information or services in such applications can make the system useless and even dangerous.
- *Reliability:* A middleware should remain operational for the duration of a mission, even in the presence of failures. The middleware's reliability ultimately helps in achieving system level reliability. Every component or service in a middleware needs to be reliable to achieve overall reliability, which includes communication, data, technologies, and devices from all layers.
- *Availability:* A middleware supporting an IoT's applications, especially mission critical ones, must be available, or appear available, at all times. Even if there is a failure somewhere in the system, its recovery time and failure frequency must be small enough to achieve the desired availability. The reliability and availability requirements should work together to ensure the highest fault tolerance required from an application.
- *Security and privacy:* Security is critical to the operation of IoT. In IoT middleware, security needs to be considered in all the functional and nonfunctional blocks including the user level application. Context-awareness in middleware may disclose personal information (e.g., the location of an object or a person). Like security, every block of middleware, which uses personal information, needs to preserve the owner's privacy.
- *Ease-of-deployment:* Since an IoT middleware (or more likely, updates to the middleware) is typically deployed by the user (or owner of the device), deployment should not require expert knowledge or support. Complicated installation and setup procedures must be avoided.
- *Popularity:* An IoT middleware (like any other software solution) should be continuously supported and extended. Usually, this facility is provided within a community of developers and researchers. While this is not necessarily a requirement, a large number of users who adopt a particular technology motivates future testing and development.

2) Architectural Requirements: The architectural requirements included in this section are designed to support application developers. They include requirements for programming abstractions, and other implementation-level concerns.

- *Programming abstraction:* Providing an API for application developers is an important functional requirement for any middleware. For the application or service developer, high-level programming interfaces need to isolate the development of the applications or services from the operations provided by the underlying, heterogeneous IoT infrastructures. The level of abstraction, the programming paradigm, and the interface type all need to be considered when defining an API. The level of abstraction refers to how the application developer views the system (e.g., individual node/device level, system level). The programming paradigm (e.g., publish/subscribe) deals with the model for developing or programming the applications or services. The interface type defines the style of the programming interface. For instance, descriptive interfaces offer SQL-like languages for data query [69], XML-based specification files for context configuration [70].
- *Interoperable:* A middleware should work with heterogeneous devices/technologies/applications, without additional effort from the application or service developer. Heterogeneous components must be able to exchange data and services. Interoperability in a middleware can be viewed from network, syntactic, and semantic perspectives, each of which must be catered for in an IoT. A network should exchange information across different networks, potentially using different communication technologies. Syntactic interoperation should allow for heterogeneous formatting and encoding structures of any exchanged information or service. Semantic interoperability refers to the meaning of information or a service, and should allow for interchange between the ever-growing and changing set of devices and services in IoT. Meaningful information about services will be useful for the users in composing multiple services as semantic

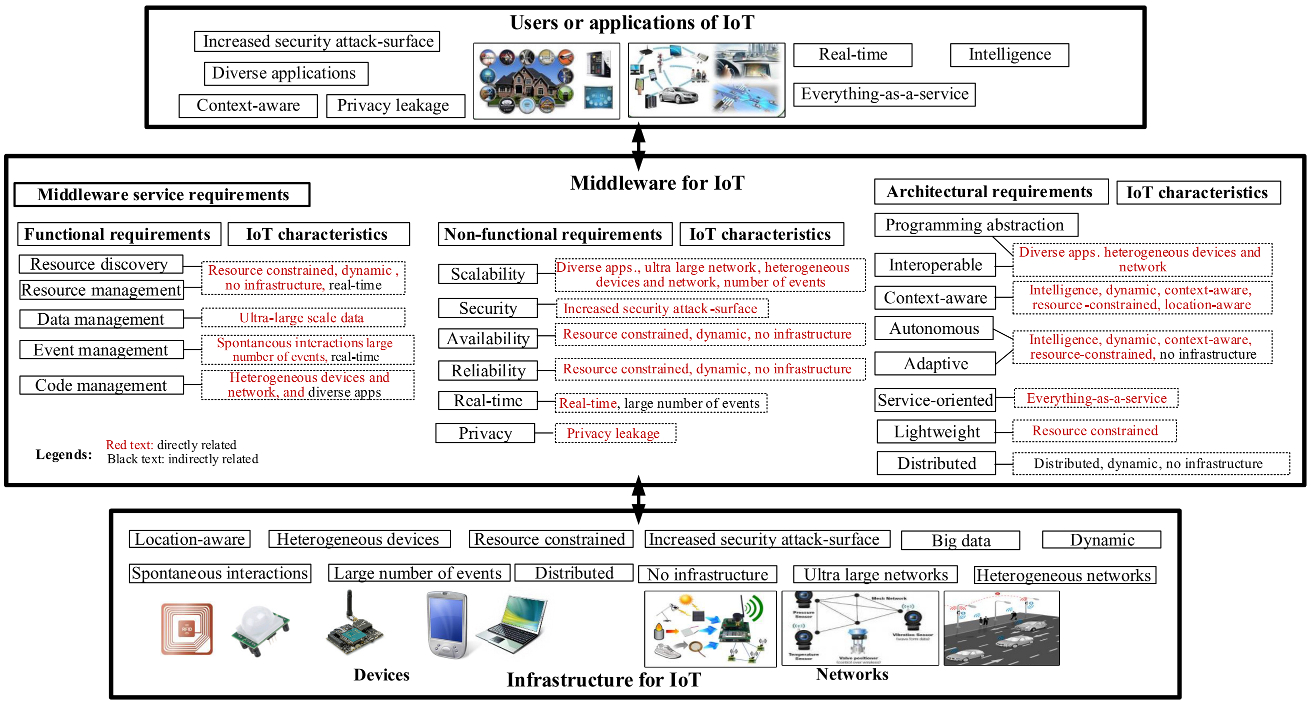
Users or applications of IoT
Increased security attack-surface
Diverse applications
Context-aware
Privacy leakage
Real-time
Intelligence
Everything-as-a-service
Middleware for IoT
Middleware service requirements
Functional requirements
IoT characteristics
Resource discovery
Resource management
Resource constrained, dynamic, no infrastructure, real-time
Data management
Ultra-large scale data
Event management
Spontaneous interactions large number of events, real-time
Code management
Heterogeneous devices and network, and diverse apps
Non-functional requirements
IoT characteristics
Scalability
Diverse apps., ultra large network, heterogeneous devices and network, number of events
Security
Increased security attack-surface
Availability
Resource constrained, dynamic, no infrastructure
Reliability
Resource constrained, dynamic, no infrastructure
Real-time
Real-time, large number of events
Privacy
Privacy leakage
Architectural requirements
IoT characteristics
Programming abstraction
Interoperable
Diverse apps. heterogeneous devices and network
Context-aware
Intelligence, dynamic, context-aware, resource-constrained, location-aware
Autonomous
Adaptive
Intelligence, dynamic, context-aware, resource-constrained, no infrastructure
Service-oriented
Everything-as-a-service
Lightweight
Resource constrained
Distributed
Distributed, dynamic, no infrastructure
Legends:
Red text: directly related
Black text: indirectly related
Location-aware
Heterogeneous devices
Resource constrained
Increased security attack-surface
Big data
Dynamic
Spontaneous interactions
Large number of events
Distributed
No infrastructure
Ultra large networks
Heterogeneous networks
Devices
Infrastructure for IoT
Networks

Fig. 4. Relationships between the IoT applications and infrastructure and its middleware requirements.

data can be better understood by "things" and humans compared to traditional protocol descriptions [71], [72].

- *Service-based:* A middleware architecture should be service-based to offer high flexibility when new and advanced functions need to be added to an IoT's middleware. A service-based middleware provides abstractions for the complex underlying hardware through a set of services (e.g., data management, reliability, security) needed by applications. All these and other advanced services can be designed, implemented, and integrated in a service-based framework to deliver a flexible and easy environment for application development.
- *Adaptive:* A middleware needs to be adaptive so that it can evolve to fit itself into changes in its environment or circumstances. In the IoT, the network and its environment are likely to change frequently. In addition, application-level demands or context are also likely to change frequently. To ensure user satisfaction and effectiveness of the IoT, a middleware needs to dynamically adapt or adjust itself to fit all such variations.
- *Context-aware:* Context-awareness is a key requirement in building adaptive systems and also in establishing value from sensed data. The IoT's middleware architecture needs to be aware of the context of users, devices, and the environment and use these for effective and essential services' offerings to users.
- *Autonomous:* It means self-governed. Devices/technologies/applications are active participants in the IoT's processes and they should be enabled to interact and communicate among themselves without direct human intervention [5], [73]. Use of intelligence including autonomous agents, embedded intelligence [74], predictive, and proactive approaches (e.g., a prediction engine) in middleware can fulfil this requirement [75].
- *Distributed:* A large-scale IoT system's applications/devices/users (e.g., WSNs and vehicular ad hoc networks) exchange information and collaborate with each other. Such applications/devices/users are likely to be geographically distributed, and so a centralized view or middleware implementation will not be sufficient to support many distributed services or applications. A middleware implementation needs to support functions that are distributed across the physical infrastructure of the IoT.

Fig. 4 presents the relationships between the IoT's middleware requirements and its infrastructural and application characteristics. As shown in this figure, most of the requirements are directly related (red colour text) to one or more characteristics of the IoT. A few of them are also indirectly linked (black text) to one or more characteristics of the IoT. For instance, the real-time behavior requirement is directly related to the application's real-time characteristics and indirectly to the large number of events. Also, a few of the middleware requirements (e.g., resource discovery and resource management) jointly capture the same set of IoT characteristics.

III. Overview of Existing Work

Middleware in IoT is a very active research area. Many solutions have been proposed and implemented, especially in the last couple of years. These solutions are highly diverse in their design approaches (e.g., event-based, database), level of programming abstractions (e.g., local or node level, global or network level), and implementation domains (e.g., WSNs, RFID, M2M, and SCADA).

In this survey, the existing middleware solutions are grouped for discussion based on their design approaches, as follows:

1) event-based;
2) service-oriented;
3) VM-based;
4) agent-based;
5) tuple-spaces;
6) database-oriented;
7) application-specific.

Some middleware use a combination of different design approaches. For instance, many service-oriented middlewares (SOMs) (e.g., *SOCRADES* and *Servilla*) also employ VMs in their design and development. Typically, hybrid approaches perform better than their individual design categories by taking the advantages of multiple approaches.

In the interest of space, the discussion of each work highlights only key points, without exhaustively capturing its performance against all requirements. For each group, the corresponding works are presented chronologically. See Tables I–III for a comprehensive summary.

A. Event-Based Middlewares

In event-based middleware, components, applications, and all the other participants interact through events. Each event has a type, as well as a set of typed parameters whose specific values describe the specific change to the producer's state. Events are propagated from the sending application components (producers), to the receiving application components (consumers). An event system (event service) may consist of a potentially large number of application components (entities) that produce and consume events [76]. Message-oriented middleware (MOM) is a type of event-based middleware. In this model, the communication relies on messages, which include extra-metadata compared to events. Generally, messages carry sender and receiver addresses and they are delivered by a particular subset of participants, whereas events are broadcast to all participants.

Typically, event-based middleware uses the publish/subscribe pattern. This model contains a set of subscribers and a set of publishers (as shown in Fig. 5). Subscribers are provided with access to publishers' data streams through a common database and they are registered for particular events. The notifications about the events are subsequently and asynchronously sent to the subscribers [77], [22]. This design approach addresses nonfunctional requirements, such as reliability, availability, real-time performance, scalability, and security [78].

Hermes [79] is an event-based middleware created for large-scale distributed applications. *Hermes* events can be either type-based or attribute-based. It uses a scalable routing algorithm and fault-tolerance mechanisms that can tolerate different kinds of failures in the middleware. Apart from scalability, it addresses interoperability and reliability requirements. *Hermes* has two components, event clients and event brokers. In its architecture, *Hermes* has the following layers: the middleware layer, event-based layer, type-based and attribute-based pub/sub layer, overlay routing network layer, and network layer. The event-based middleware layer provides an API that programmers use to implement applications. The middleware layer consists of several modules that implement functionalities such as fault-tolerance, reliable event delivery, event-type discovery, security, and transactions. The mobility is limited in terms of a dynamic network topology. *Hermes* does not support composite events or persistent storage for events. Moreover, its support for adaptation is limited to the network level.

EMMA [27] is an adaptation of Java message service (JMS) for mobile *ad hoc* environments. It is designed for multiparty video communication systems such as video chatting, where multiple video streams are distributed simultaneously on overlay networks [80]. *EMMA* is available, reliable, and autonomous through a quick recovery mechanism, which also makes it fault-tolerant. Moreover, *EMMA* offers multiple styles of messaging. In order to implement different levels of reliability, *EMMA* treats persistent and nonpersistent messages differently. *EMMA* provides very good performance in terms of delivery ratio and latency. However, the tradeoff between application-level routing and resource usage is not taken into consideration. Also, because of its design approach, *EMMA* is not energy efficient. Moreover, support for reliability is limited.

GREEN [81] is a runtime, highly configurable and reconfigurable event-based middleware developed to support pervasive computing applications that use heterogeneous networks and heterogeneous devices. *GREEN* is developed to operate in diverse network types (i.e., MANETs and WANs). It also supports pluggable pub/sub interaction types such as topic-based, content-based, context, and composite events. The high-event flow in the system is provided by replacing content-based interaction with a topic-based interaction. *GREEN* follows Lancaster's approach to building reconfigurable middleware platforms. It is built using nondistributed lightweight component model. *GREEN*'s strengths are support for reprogrammability and it can operate over heterogeneous network types. Its component structure is lightweight and enables dynamic behavior. However, *GREEN* is not autonomous and has limited support for interoperability. Moreover, its support for adaptation is limited to the network level. Also, the use of overlay networks brings serious challenges in meeting the IoT middleware requirements.

RUNES [82] is a component-based middleware for large-scale and widely distributed heterogeneous network of embedded systems. It introduces a standardized architecture capable of self-organization and dynamic adaptation to a changing environment. Like *GREEN*, *RUNES*'s architecture follows Lancaster's approach, which reduces coupling of middleware components and supports adding new components at runtime. *RUNES* this project focuses on design of a framework where new components can be installed at runtime. However, it is unclear if this can be done remotely, once the middleware is deployed. Also, *RUNES* does not provide a holistic view of IoT middleware requirements and does not consider resource-rich devices in heterogeneous environments.

Steam [76], *MiSense* [83], [84], *PSWare* [85], and *TinyDDS* [86] are other examples of event-based middlewares. *Steam* is