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

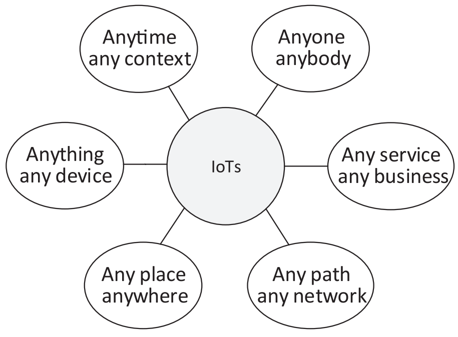

Fig. 1. Definition of IoT [52].

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

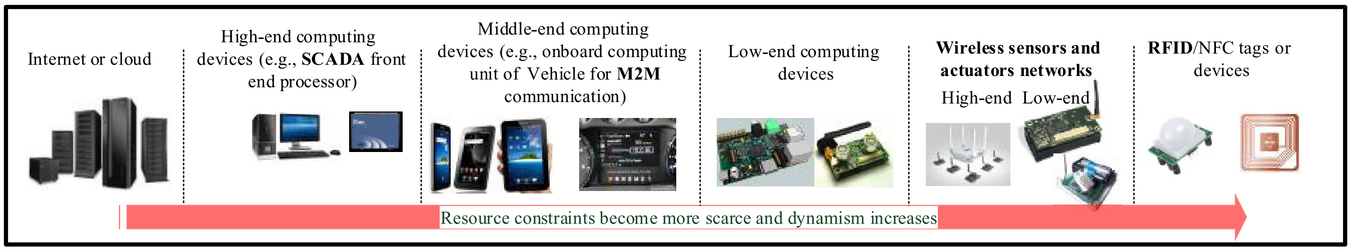
Internet or cloud
High-end computing devices (e.g., SCADA front end processor)
Middle-end computing devices (e.g., onboard computing unit of Vehicle for M2M communication)
Low-end computing devices
Wireless sensors and actuators networks
High-end Low-end
RFID/NFC tags or devices
Resource constraints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

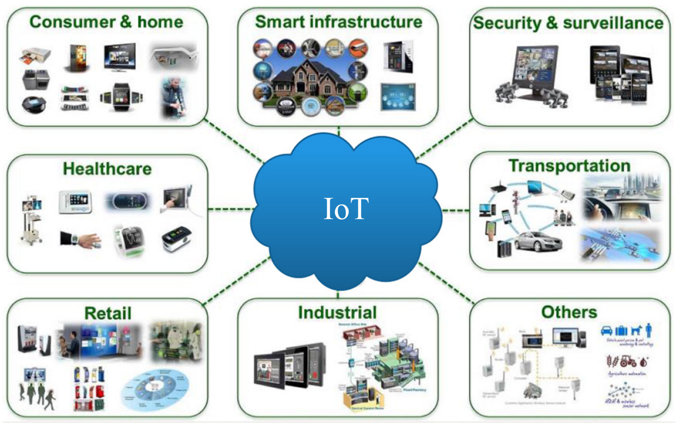

Fig. 3. Potential applications of IoT [66].

- *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

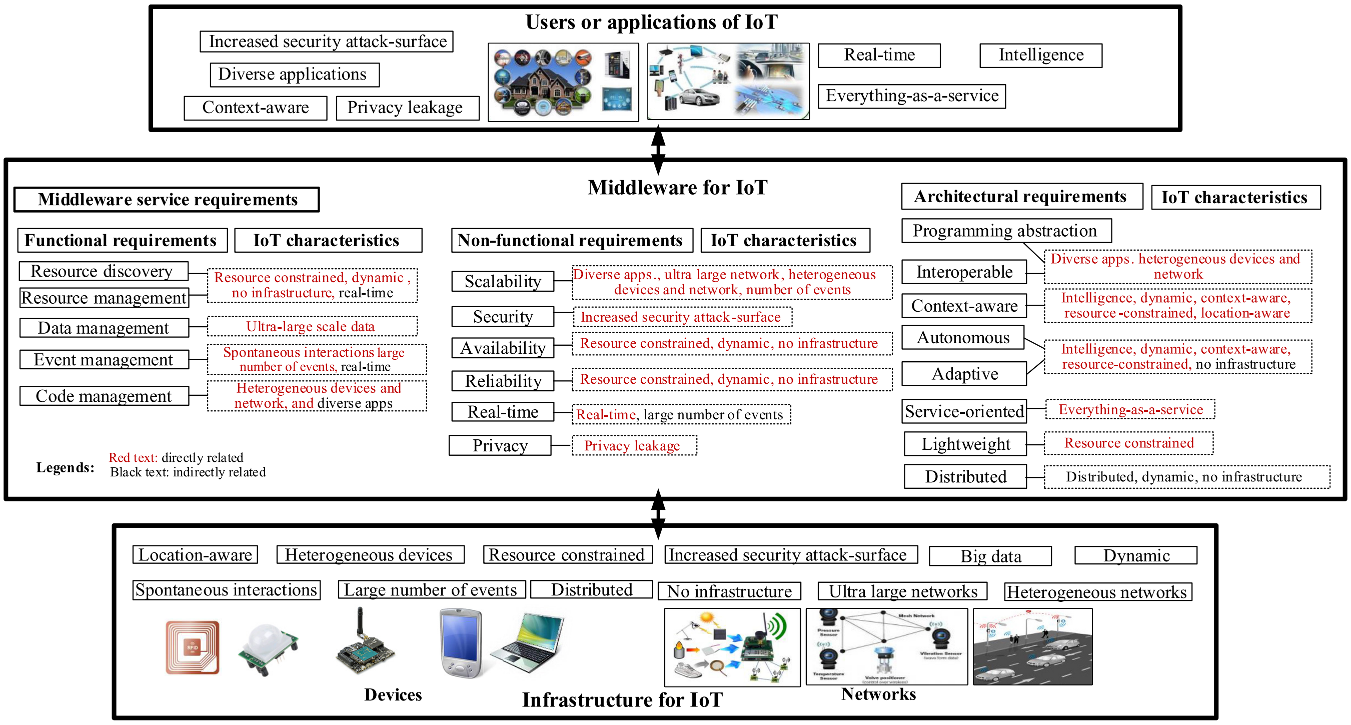
Users or applications of IoT
Increased security attack-surface
Diverse applications
Context-aware
Privacy leakage
Real-time
Intelligence
Everything-as-a-service
Middleware for IoT
Middleware service requirements
Functional requirements
IoT characteristics
Resource discovery
Resource management
Resource 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

TABLE I
SUMMARY OF THE IoT MIDDLEWARES: SUPPORTED FUNCTIONAL REQUIREMENTS

	Functional requirements				
	Resource discovery	Resource management	Data management	Event management	Code management
Event-based approach					
Hermes [79]	CD-DD	RM	DPF	LS	NI
EMMA [27]	CD-DeD, CD-SD	RM	DS	SS	CA
GREEN [81]	DD-ND	RM	DS, DPF	LS	CA
RUNES [82]	CD-DeD, CD-SD	RM, RCA	DPA	SS	CA
PRISMA [29]	CD-DD	RM	DS, DPA	SS	CA
SensorBus [87]	CD-DeD	RM	NS	SS	NI
Mires [88]	CD-DeD, CD-SD	RM	DPA	SS	CA
Service-oriented approach					
Hydra [101]	DD-DeD, DD-SD	RA, RM, RCP	DS	SS	NS
Sensewrap [103]	DD-DeD, DD-SD	NI	NI	SS	NS
MUSIC [70]	DD-SD	RA, RM, RCA	NS	NS	NS
TinySOA [105]	DD-DeD, DD-SD	RA	DS	NS	NS
SOCRADES [93]	DD-DeD, DD-SD	RA, RM, RCP	NI	LS	NS
SENSEI [109]	DD-DeD	RA, RM, RCA	DS, DPA	NI	NS
ubiSOAP [94]	DD-DeD, DD-ND	RA, RM, RCA, RCL	NI	SS	NS
Servilla [95]	DD-SD	RA, RM, RCA	DPC	SS	CA, CM
KASOM [110]	DD-SD	RA, RM, RCA	NS	LS	NS
CHOReOS [112]	DD-DeD, DD-SD	RA, RM, RCA	DPA	SS	NS
MOSDEN [46]	DD-DeD, DD-SD	RA, RM, RCP	DS, DPA	NI	NS
Xively [99]	DD-DeD, DD-SD	RA, RM	DS, DPA	NI	NI
CarrIoT [98]	DD-DeD, DD-SD	RA, RM	DS, DPA	NI	NI
Echelon [118]	DD-DeD, DD-SD	RA, RM	DS, DPA	NI	NI
VM approach					
Maté [125]	DD-DeD	RA, RM	DS, DPA	LS	CA
VM* [128]	DD-DeD	RM	DS, DPA	LS	CA
Melete [130]	DD-DeD	RA, RM, RCA	DS, DPA	LS	CA, CM
MagnetOS [132]	DD-DeD	RA, RM, RCA	DS, DPA	LS	CA
Squawk [133]	DD-DeD	RA, RM	DS, DPA	NI	CA
Sensorware [129]	DD-DeD	RA, RM, RCA	DS, DPA	LS	CA, CM
Extended Maté [137]	DD-DeD	RA, RM	DS, DPA	LS	CA
DVM [138]	DD-DeD	RA, RM, RCL	DS, DPA	SS	CA
DAViM [139]	CD-DD	RA, RM	DS, DPA	SS	CA
SwissQM [140]	DD-DeD, DD-SD	RA, RM, RCA	DS, DPA, DPF	LS	CA
TinyVM [141]	NI	NI	DPA	NI	NI
TinyReef [123]	NI	NS	DS, DPA	SS	CA
Agent-based approach					
Impala [149]	DD-DeD	RA, RM	DPA	LS	CA, CM
Smart messages [150]	DD-ND	RA, RM	DPA	SS	CA, CM
ActorNet [151]	DD-DeD	RA	DPA	SS	CA, CM
Agilla [28]	DD-DeD	RA, RM, RCA	DPA	LS	CA, CM
Ubiware [152]	DD-DeD, DD-SD	RA, RM, RCA	DPA	LS	CA, CM
UbiROAD [153]	CD-SD	RM	DS	LS	CA, CM
AFME [154]	CD-DD	RA	DPA	SS	CM
MAPS [155]	DD-DeD	RA, RM	DPA	LS	CA, CM
MASPOT [147]	DD-DeD	RA, RM, RCA	DPA	LS	CA, CM
TinyMAPS [156]	DD-DeD	RCA	DPA, DPF	LS	CA, CM
Tuple-space approach					
LIME [160]	CD-DeD, CD-SD	RM	DS	SS	CM
TeenyLIME [162]	CD-DeD, CD-SD	RM	DS, DPA	SS	CM
TinyLIME [161]	CD-DeD, CD-SD	RM	DPA	SS	CM
TS-Mid [164]	CD-DeD, CD-SD	RM	DS, DPA	SS	NS
A3-TAG [165]	CD-DeD, CD-SD	RM	DS, DPA	SS	NS
Database approach					
SINA [166]	DD-DeD	RM	DS, DPA	SS	NS
COUGAR [168]	DD-ND	RM	DS, DPA	LS	NS
IrisNet [169]	DD-SD	RA, RM, RCL	DS, DPA, DPF	SS	CA, CM
Sensation [170]	CD-DeD	RM	DS, DPA	SS	NS
TinyDB [69], [171]	DD-DeD	RM	DS, DPA, DPF	SS	NI
GSN [172]	DD-Ded, DD-SD	RA, RM	DS, DPF	LS	CA
KSpot^{+} [173]	DD-SD	RM	DS, DPA	NS	NS
HyCache [174]	DD-DeD	RM	DS, DPA	NS	NS
Application-specific approach					
AutoSec [175]	CD-SD	RA, RM, RCA, RCL	DS, DPA, DPF	LS	CA, CM
Adaptive middleware [176]	CD-SD	RA, RM	DS, DPA	LS	CA
MiLAN [177]	CD-SD	RA, RM, RCA	DS, DPA	LS	CA, CM
TinyCubus [178]	CD-SD	RA, RM	DS, DPA	LS	CA
MidFusion [179]	CD-SD	RA, RM, RCA	DS, DPA, DPC, DPF	LS	CA
Legend Not supported (NS) No information (NI)	Centralised discovery (CD) Distributed discovery (DD) Device discovery (DeD) Network discovery (ND) Service discovery (SD)	Resource allocation (RA) Resource monitor (RM) Resource composition (RC) - Adaptive (A) - Predefined (P) Resource conflict (RCL)	Data storage (DS) Data preprocessing (DP) - Aggregation (A) - Compression(C) - Filtering (F)	Supported - Large scale (LS) - Small scale (SS)	Code allocation (CA) Code migration (CM)

TABLE II
SUMMARY OF THE IoT MIDDLEWARES: SUPPORTED NONFUNCTIONAL REQUIREMENTS

	Non-functional requirements						
	Scalability	Security	Availability	Reliability	Real-Time	Privacy	Popularity
Event-based approach							
Hermes [79]	AL, NLIoTS	NI	S	CR, DR	HRT	NI	M
EMMA [27]	NLWSNS	NI	S	CR	SRT	NI	M
GREEN [81]	NLIoTS	NI	S	NI	SRT	NI	M
RUNES [82]	NLWSNS	NS	NI	NI	HRT	NS	H
PRISMA [29]	NLWSNS	NI	S	CR	SRT	NI	H
SensorBus [87]	NLWSNS	NI	NI	NI	HRT	NI	L
Mires [88]	AL, NLWSNS	NI	S	CR	SRT	NI	M
Service-oriented approach							
Hydra [101]	AL, NLWSNS	S	NI	NI	SRT	NS	H
Sensewrap [103]	AL, NLIoTS	C	NI	NI	SRT	NS	L
MUSIC [70]	AL, NLWSNS	NS	S	DR	NI	NS	M
TinySOA [105]	AL, NLWSNS	NS	NS	NS	HRT	NS	M
SOCRADES [93]	AL, NLIoTS	C	NI	NI	SRT	NS	H
SENSEI [109]	AL, NLWSNS	C	NI	NI	SRT	S	H
ubiSOAP [94]	AL, NLWSNS	NS	S	NS	NI	NS	M
Servilla [95]	AL, NLWSNS	NS	S	NI	NI	NS	L
KASOM [110]	AL, NLWSNS	C	S	CR	HRT	NI	M
CHOReOS [111]	AL, NLIoTS	NI	S	NI	NI	NS	H
MOSDEN [46]	AL, NLWSNS	NS	S	NS	NI	NS	H
Xively [99]	AL, NLIoTS	C	S	NS	SRT	NS	H
CarrIoT [98]	AL, NLIoTS	C, A	S	NI	HRT	NI	H
Echelon [118]	AL, NLIoTS	C, A	S	NI	HRT	NI	H
VM approach							
Maté [125]	AL, NLWSNS	NI	S	CR	SRT	NI	H
VM* [128]	AL, NLWSNS	NI	S	CR	SRT	NI	L
Melete [130]	AL, NLWSNS	C	NI	CR	SRT	S	L
MagnetOS [132]	AL, NLWSNS	NI	S	CR	SRT	NI	L
Squawk [133]	AL, NLWSNS	NI	NI	CR, DR	SRT	NI	M
Sensorware [129]	NLWSNS	NI	NI	NI	SRT	NI	M
Extended Maté [137]	NLWSNS	NI	S	CR, DR	SRT	NI	H
DVM [138]	NLWSNS	NI	S	DR	SRT	NI	L
DAViM [139]	AL, NLWSNS	NI	S	DR	SRT	NI	L
SwissQM [140]	AL, NLIoTS	A	S	CR, DR	NRT	NI	L
TinyVM [141]	NI	NI	NI	NI	NI	NI	L
TinyReef [123]	NLWSNS	NI	NI	CR	SRT	NI	L
Agent-based approach							
Impala [149]	AL, NLWSNS	I, A	NS	DR	SRT	S	H
Smart messages [150]	NLWSNS	A	S	NI	SRT	NS	M
ActorNet [151]	NLWSNS	NI	NI	DR	NRT	NI	L
Agilla [28]	NLWSNS	NS	S	DR	SRT	NS	M
Ubiware [152]	AL, NLIoTS	C, A	S	NI	SRT	NS	H
UbiROAD [153]	AL, NLIoTS	C	S	NS	NRT	S	L
AFME [154]	NLWSNS	NI	S	CR, DR	SRT	NI	L
MAPS [155]	AL, NLWSNS	NI	NI	CR	SRT	NI	M
MASPOT [147]	AL, NLWSNS	NI	S	CR	SRT	NI	M
TinyMAPS [156]	AL, NLWSNS	NI	S	CR, DR	SRT	NI	M
Tuple-space approach							
LIME [160]	NLWSNS	NS	NS	NS	SRT	NI	H
TeenyLIME [162]	NLWSNS	NS	NS	NS	SRT	NI	H
TinyLIME [161]	NLWSNS	A	NS	NS	SRT	S	H
TS-Mid [164]	NS	NI	S	S	NRT	NI	L
A3-TAG [165]	NLWSNS	NI	S	S	SRT	NI	L
Database approach							
SINA [166]	NLWSNS	NS	NS	NS	NRT	NS	M
COUGAR [168]	NLWSNS	I	S	NI	SRT	S	L
IrisNet [169]	NLWSNS	C	S	NS	SRT	S	L
Sensation [170]	NLWSNS	NS	S	NI	NRT	NS	L
TinyDB [69], [171]	NLWSNS	NS	NS	NS	NRT	NS	H
GSN [172]	AL	I	S	NI	SRT	NS	H
KSpot $^+$ [173]	NLWSNS	I	S	NI	NRT	NS	L
HyCache [174]	AL	NI	NS	DR	NRT	NI	M
Application-specific approach							
AutoSec [175]	NLIoTS	NI	S	CR, DR	HRT	NI	M
Adaptive middleware [176]	NLWSNS	NI	S	CR, DR	HRT	S	L
MiLAN [177]	NLWSNS	NI	S	CR	HRT	NI	M
TinyCubus [178]	NLWSNS	NS	S	DR	HRT	NS	L
MidFusion [179]	NLWSNS	NI	S	CR	HRT	NI	L
Legend Not supported (NS) No information (NI)	Application level (AL) Network level (NL) - IoT scale (IoTS) - WSN scale (WSNS)	Confidentiality(C) Integrity (I) Availability (A)	Supported (S)	Communication (CR) Data (DR)	Hard real time (HRT) Soft real time (SRT) Non-real time (NRT)	Supported (S)	Highly (H) Moderately (M) Less (L)

TABLE III
SUMMARY OF THE IoT MIDDLEWARES: SUPPORTED ARCHITECTURAL REQUIREMENTS

	Architectural requirements							
	Abstraction	Interoperable	Context-aware	Autonomous	Adaptive	Service-based	Lightweight	Distributed
				Event-based approach				
Hermes [79]	S	NeI, SeI	N	Y	DA	Y	M	Y
EMMA [27]	S	NeI, SeI	N	Y	SA	Y	E, M	Y
GREEN [81]	S	NeI	Y	N	DA	Y	M	Y
RUNES [82]	S	NeI	Y	Y	DA	Y	E,M	Y
PRISMA [29]	S	NeI	Y	N	SA	Y	E	Y
SensorBus [87]	S	NeI	Y	NI	NI	Y	E, M	Y
Mires [88]	S	NeI	N	N	NI	Y	E	Y
				Service-oriented approach				
Hydra [101]	S	NeI,SI,SeI	Y	NI	DA	Y	E	Y
Sensewrap [103]	S	NeI	NI	NI	SA	Y	M	Y
MUSIC [70]	S	NeI	Y	Y	DA	Y	N	Y
TinySOA [105]	S	NeI	NS	NS	NI	Y	E, M	Y
SOCRADES [93]	S	NeI	Y	Y	DA	Y	NI	Y
SENSEI [109]	S	NeI, SeI	Y	NI	DA	Y	NI	Y
ubiSOAP [94]	S	NeI	NS	NI	DA	Y	E	Y
Servilla [95]	S	NeI	NS	NI	DA	Y	E, M	Y
KASOM [110]	S	NeI, SeI	Y	NI	DA	Y	E,M	Y
CHOReOS [111]	S	NeI, SeI	Y	NI	DA	Y	N	Y
MOSDEN [46]	S	NeI, SeI	Y	NI	DA	Y	E, M	Y
Xively [99]	S	NeI	Y	NI	DA	Y	NI	Y
CarrIoT [98]	S	NeI	NS	NI	DA	Y	E	Y
Echelon [118]	S	NeI	NS	NI	DA	Y	E	Y
				VM approach				
Maté [125]	S	NeI	Y	Y	DA	Y	E, M	Y
VM* [128]	S	NeI	NI	Y	NS	Y	M	Y
Melete [130]	S	NeI	No	Y	DA	NI	M	Y
MagnetOS [132]	S	NeI, SeI	NI	Y	DA	Y	E, M	Y
Squawk [133]	S	NeI	NI	Y	NI	NI	M	Y
Sensorware [129]	S	NeI	Y	Y	DA	Y	N	Y
Extended Maté [137]	S	NeI, SI, SeI	Y	Y	DA	Y	M	Y
DVM [138]	S	NeI	N	Y	NS	Y	E, M	Y
DAViM [139]	S	NeI	Y	Y	DA	Y	M	Y
SwissQM [140]	S	SeI	N	Y	DA	Y	M	Y
TinyVM [141]	S	NI	NI	NI	NI	NI	E, M	Y
TinyReef [123]	S	NI	NI	NI	NI	NI	M	Y
				Agent-based approach				
Impala [149]	S	NeI	Y	Y	DA	Y	E	Y
Smart messages [150]	S	NeI	N	Y	DA	Y	NI	Y
ActorNet [151]	S	NeI	N	Y	DA	Y	E, M	Y
Agilla [28]	S	NeI, SI, SeI	Y	Y	DA	Y	M	Y
Ubiware [152]	S	NeI, SI, SeI	Y	Y	DA	Y	NI	Y
UbiROAD [153]	NI	SI	Y	Y	NI	Y	NI	Y
AFME [154]	S	NeI	Y	Y	DA	Y	M	Y
MAPS [155]	S	NeI, SeI	N	Y	DA	Y	M	Y
MASPOT [147]	S	NeI, SeI	Y	Y	DA	Y	M	Y
TinyMAPS [156]	S	NeI, SeI	Y	Y	DA	Y	E, M	Y
				Tuple-space approach				
LIME [160]	S	NeI	Y	N	NS	Y	M	Y
TeenyLIME [162]	S	NeI	Y	N	NI	Y	E,M	Y
TinyLIME [161]	S	NeI	Y	N	N	Y	E	Y
TS-Mid [164]	S	NeI	Y	Y	DA	Y	E	Y
A3-TAG [165]	S	NeI	Y	Y	DA	Y	E, M	Y
				Database approach				
SINA [166]	S	NS	N	Y	DA	N	E	Y
COUGAR [168]	S	NS	N	N	NI	N	M	Y
IrisNet [169]	S	NS	N	N	NS	Y	M	Y
Sensation [170]	S	SI	Y	Y	NS	NI	NI	Y
TinyDB [69], [171]	S	NI	NI	N	DA	N	E	Y
GSN [172]	S	NI	NI	NI	SA	Y	M	Y
KSpot $^{+}$ [173]	S	NeI	N	Y	NS	NI	E	Y
HyCache [174]	S	NI	NI	NI	NI	N	NS	Y
				Application-specific approach				
AutoSec [175]	NS	NeI	Y	Y	DA	Y	E, M	Y
Adaptive middleware [176]	NS	NeI	Y	Y	DA	Y	E, M	Y
MiLAN [177]	NS	NeI, SeI	Y	Y	DA	Y	E	Y
TinyCubus [178]	S	NeI	Y	Y	SA	Y	E	Y
MidFusion [179]	NS	NeI	Y	Y	DA	Y	N	Y
Legend Not supported (NS) No information (NI)	Supported (S) Not supported (NS)	Network (NeI) Syntactic (SI) Semantic (SeI)	Yes (Y) Not (N)	Yes (Y) Not (N)	Dynamically (DA) Statically (SA)	Yes (Y) Not (N)	Energy (E) Memory(M) Not (N)	Yes (Y) Not (N)

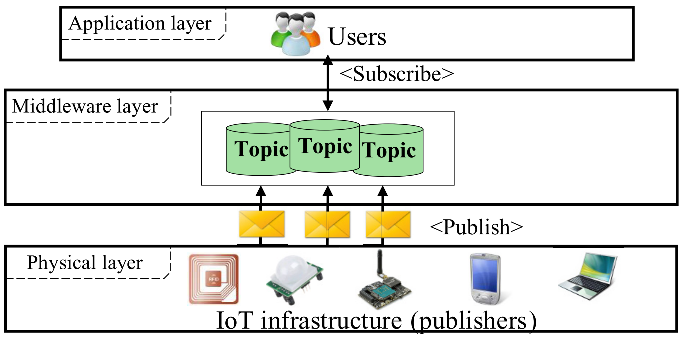

Fig. 5. General design model for an event-based middleware.

an event-based middleware service, designed for the mobile computing domain. It uses different types of events to address the problems related to the dynamic reconfiguration of the network, scalability of a system and the real-time delivery of events. *MiSense* is a cluster-based lightweight layered middleware that separates application semantics from the underlying hardware, operating system, and network infrastructure. In *MiSense*, each node owns a broker that manages the generated messages. This overloads the node, and makes the middleware resource-inefficient. *PSWare* is a real-time event-based middleware for WSN, developed to support composite events. It provides high-level abstractions, and achieves high expressiveness and availability. *TinyDDS* [86] middleware enables interoperability between WSNs and access networks. It provides programming language and protocol interoperability based on the standard data distribution service (DDS) specification. The *TinyDSS* framework allows WSN applications to have control over application-level and middleware-level nonfunctional properties. Simulation and empirical evaluation results showed that *TinyDDS* is lightweight and has small memory footprint. However, *TinyDDS* does not provide a holistic view of IoT requirements and does not address key IoT requirements such as adaptation. Also, it does not offer a topology control mechanism. *Steam*, *PSWare*, *MiSense*, and *TinyDDS* do not address the heterogeneity of an IoT infrastructure. These middleware solutions have been designed only for WSNs or mobile devices.

PRISMA [29] is a resource-oriented event-based middleware for WSN. By providing a high-level and standardized interface for data access, *PRISMA* supports interoperability of the heterogeneous network technologies. *PRISMA* is popular among developers, and encourages early adopters. The *PRISMA* design deploys a layered architecture, composed of three layers: 1) access; 2) service; and 3) application. The access layer manages communication, data acquisition, verification of QoS requirements, and reconfiguration. Reconfiguration is supported in several cases (e.g., device failure). The service layer provides a resource discovery component. The application layer offers support for programming abstraction and is responsible for receiving and managing applications messages. *PRISMA* assumes a heterogeneous and hierarchical WSN, with three levels: 1) gateway; 2) cluster head; and 3) sensor node. However, this centralized approach creates bottlenecks in the sink nodes. The current version does not support hard real time or dynamic adaptation. Also, the centralized service discovery approach will not scale well in the IoT. Moreover, it provides limited support for interoperability since it uses a custom service description template. Also, it was designed and implemented on an Arduino platform. Future work aims to redesign the architecture of *PRISMA* to enable support for dynamic reconfiguration at runtime.

SensorBus [87] is an MOM for WSNs. It allows free exchange of more than one communication mechanism among sensor nodes. To answer service request from applications in several contexts, *SensorBus* provides customisable services through metadata. Its architecture has three layers, developed for application, message, and context services. The application service layer provides an API simplifying application development. This layer also deploys application filters to aggregate internal data, which reduces data flow in the network, leading to the reduction of power consumption in sensor nodes. The message service layer is responsible for providing communication and coordination for the distributed components, abstracting the developer from these issues. The context service layer manages the heterogeneous sensors that collect information from the environment. It does not offer a holistic view of the IoT middleware requirements, and is not widely used.

Mires [88] is another MOM for WSNs. Using the publish/subscribe pattern, *Mires* lets subscribes to select the data streams they are interested in and receive data directly from the requested sensors. *Mires* allows sensors to conduct local data aggregation to reduce the number of message transmissions and power consumption. *Mires* has been designed to facilitate the development of applications over WSNs. However, *Mires* does not offer support for persistent messages. Also, it does not support a dynamic network topology and it is not fault tolerant. The work done with respect to an IoT middleware is limited and like *SensorsBus* focuses *Mires* only on certain requirements and does not provide a holistic view of the IoT middleware requirements.

Alongside MOMs, there are MQ brokers, which offer support for M2M communications between services, or service providers and service subscribers, conversion between different transport protocols and homogenization of message streams between subscribers and providers. *WebSphere MQ* [89] and *Mosquitto* [90] are examples of this approach. *WebSphere MQ*, currently known as *IBM MQ* [91], maintains the messages queues, the relationships between programs and queues, handling network restarts and moving messages around the network. The resource management is focused on queue management, which establishes communication between multiple queue managers. The events are treated as uninterpreted data, which imply that *WebSphere MQ* does not support context-awareness. Also, it does not support composite events or complex messages handling. *Mosquitto* is an MQTT broker that enables communication between subscribers and publishers through a topic subscription. Its main purpose is to create communication channels and does it not address to the IoT requirements. Recently, many MQ broker solutions have been proposed. However, these were not designed for an IoT environment.

Event-based middlewares are appropriate in systems where mobility and failures are common. A main advantage of this

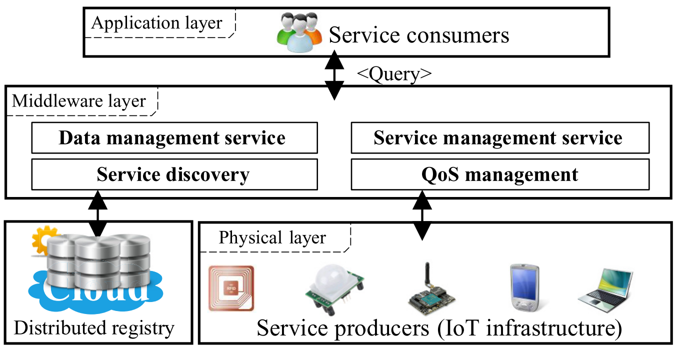

Fig. 6. General design model for an SOM.

approach is support for strong decoupling of producers and subscribers. Although many challenges are addressed by most of the event-based middlewares, their support is not totally satisfactory, in particular, interoperability, adaptability, timeliness, and context-awareness are not adequately addressed. Event-based middlewares are also rarely autonomous. The programming paradigm in event-based middlewares is not sufficiently flexible in many cases. Appropriate protocols and models for security and privacy need to be developed.

B. *Service-Oriented Middlewares*

The service-oriented design paradigm builds software or applications in the form of services. Service-oriented computing (SOC) is based on service-oriented architecture (SOA) approaches and has been traditionally used in corporate IT systems. The characteristics of SOC, such as technology neutrality, loose coupling, service reusability, service composability, and service discoverability [92], are also potentially beneficial to IoT applications. However, IoT's ultra-large-scale network, resource-constrained devices, and mobility characteristics make service discovery and composition challenging.

An SOM has the potential to alleviate these challenges through the provision of appropriate functionalities (as shown in Fig. 6) for deploying, publishing/discovering, and accessing services at runtime. SOM also provides support for adaptive service compositions when services are unavailable. A large number of service-oriented IoT middlewares are available. These middlewares can be categorized as standalone SOM for IoT [93]–[97] or middleware services provided by cloud computing platform as a service (PaaS) model [98]–[100].

Hydra [101], which is currently known as *LinkSmart* [102], is a middleware for ambient intelligence (AmI) services and systems. It is built on an SoA and model-driven architecture. Its architecture consists of a number of management components, including a service manager, event manager, device manager, storage manager, context manager, and security manager. These components are grouped into application and device elements, each of which has a semantic layer, service layer, network layer, and security layer. *Hydra* provides syntactical and semantic level interoperability using semantic WB. In addition to a number of functional requirements (e.g., data management, event management, and resource management), it supports dynamic reconfiguration and self-configuration. *Hydra*'s resource, device, and policy managers make it lightweight by optimizing energy consumption in resource-constrained devices. Distributed security and social trust components offer secure and trustworthy communication within devices. Its security and privacy solution uses virtualization and an implementation of WS-based mechanisms enriched by semantic resolution [101]. However, its virtualization may introduce security concerns (e.g., side channel attacks). Also, ontology-based semantic security and interoperability solutions are likely to be unsuitable in IoT because, currently, there are no standard ontologies for ultra-large-scale IoT.

The *SenseWrap* [103] middleware combines the Zeroconf [104] protocols with hardware abstraction using virtual sensors. A virtual sensor provides transparent discovery of resources, mainly sensors, through the use of Zeroconf protocols, which applications can use to discover sensor-hosted services. *SenseWrap* also provides a standardized communication interface to hide the sensor-specific details from the applications. This interface depends on sensor modeling and custom wrappers (drivers) for each sensor model. Also, virtualization is applied only to sensors, not to actuators or computing resources. These issues make it unsuitable for IoT environments, which are ultra large scale, with heterogeneous networks and diverse applications.

The *MUSIC* [70] middleware provides a self-adaptive component-based architecture to support the building of systems in ubiquitous and SoA environments, where dynamic changes may occur in service providers and service consumers contexts. In particular, *MUSIC* focuses on changes in a service provider site, to interchange components and services providing the functionalities defined by the component framework. To support QoS-aware and context-based dynamic adaptation, its architecture contains a context manager, QoS manager, adaptation manager, plan repository, SLA negotiator and monitoring, and service discovery; these components provide different functionalities for the middleware. For instance, in planning-based adaptation, planning (available in a plan repository) is typically triggered by context changes detected by the context manager. With the support of these components, the dynamic adaptations work automatically to optimize the application's utility in a given context. Context may contain a lot of private and sensitive data (e.g., location or interests of a user) and thereby increases the risk of privacy leakage.

TinySOA [105] is an SOM that offers a high-level abstraction of the infrastructure for the development of WSN applications. It provides a simple service-oriented API through which application developers can access WSN resources from their applications. It handles WSN device and communication level heterogeneity, and offers easy integration of Internet applications with WSNs allowing them to collect information from the sensors. *TinySOA* employs simple and deterministic mechanisms for WSN resource (e.g., sensor node) registration and discovery. It supports only a few basic functional requirements (e.g., abstraction, resource discovery, and management).

The *SOCRADES* [93] middleware abstracts physical things as services using devices profile for WS (DPWS). It has extended two earlier works [106], [107]. *SOCRADES* simplifies the management of underlying devices or things for enterprise applications (e.g., industrial automation). Its architecture consists of a layer for application services (e.g., event storage) and a layer for device services (e.g., device manager and monitor, service discovery, service lifecycle management). Different components in the two layers fulfil different requirements of SOM. For instance, the device services layer's service discovery component, a key contribution of *SOCRADES* middleware, discovers the services provided by real-world devices or things, while its device manager handles resource management (e.g., device access). It also offers device and service discovery. The application services' layer provides event management and storage. The *SOCRADES* middleware's cross-layer service catalogue, which sits between the device and applications layers, supports service composition, which may not be fully dynamic, as composition relies on predefined building blocks. Role-based access control of devices communication to middleware and back end services, and vice versa, works as a security solution, but it is limited to only authentication. Moreover, direct access to devices or their offered services through this middleware raises the risk of privacy violations.

SensorsMW [108] is an adaptable and flexible SOM for QoS configuration and management of WSNs. It abstracts WSNs as a collection of services for seamless integration into an enterprise information system. This allows easy and efficient configuration of WSNs for information gathering using WS. Resources in a WSN are managed to comply with certain QoS requirements, according to SLAs. Importantly, it offers an abstract way to access these resources for high-level applications to reconfigure and maintain the network during their lifetime. Thus, applications can control and make tradeoffs between conflicting issues (e.g., lifetime and sampling rate). Resource reconfiguration and management need resource discovery, especially in mobile IoT, where resources are dynamic, and these are not addressed here. Also, in critical applications, this reconfiguration may fail as their strict QoS requirements may not allow any tradeoff between the necessary resources.

The *SENSEI* [109] middleware develops an architecture for the future and real-world Internet including IoT. It is one of the earliest proposals that included a context model, context services, actuation tasks, and dynamic service composition of both primitive and advanced services for the real-world Internet. The main component of this middleware is the resource layer, which sits between the application layer and communication services layer. Resources in *SENSEI* use ontologies for their semantic modeling. Currently, there are no standard ontologies for ultra-large-scale IoT, which is likely to make *SENSEI* inadequate for IoT.

ubiSOAP [94] is an SOM that provides seamless networking of WS. The architecture's resource layer contains the necessary functions, including unified abstraction for simple devices (e.g., sensors, actuators, processors, or software components) to facilitate the interaction of applications and services with the resources. A support services component enables discovery and dynamic composition of resources (e.g., services). Dynamic composition and instantiation of new services are facilitated by the semantically rich models and descriptions of sensors, actuators, and processing elements. The resource layer also contains functions for privacy and security (e.g., authentication). Its multiradio networking layer manages heterogeneous network resources using a network-agnostic addressing scheme and offers network-agnostic connectivity to services. This layer also offers the functionality for QoS-aware (e.g., energy consumption, availability) network selection. In general, *ubiSOAP* is a lightweight SOM that offers resource management and network level interoperability by supporting heterogeneous networking devices and technologies. The lack of context-awareness in *ubiSOAP* could be an issue, as this is key in adaptive and autonomous behavior of the things. Also, its focus only on authentication for security and privacy is a concern for many IoT applications.

Servilla [95] facilitates application development in heterogeneous WSNs. It uses SOC to decouple platform-specific code from platform-independent applications. It structures applications as platform-independent tasks that are dynamically bound to platform-specific services. *Servilla*'s architecture consists of a VM and a service provisioning framework (SPF) and runs on individual sensor nodes in a WSN. The VM executes application tasks while the SPF-consumer discovers and accesses services, and the SPF-provider advertises and executes services. It exploits dynamic service binding and binding semantics to support dynamic task deployment and task mobility. Dynamic service binding provides energy efficient in-network collaboration among heterogeneous devices. A specialized service description language facilitates flexible matching between applications and services residing on the same or different devices, but this specialized language requirements could limit the wider adoption of this middleware. Moreover, individual sensor level access could introduce privacy violations and security threats. *Servilla* is not widely used.

KASOM [110] is a knowledge-aware and SOM (KASOM) for pervasive embedded networks, especially for WSANs. Its architecture consists of three major subsystems: 1) framework services (e.g., security and runtime manager); 2) communication services (e.g., resource monitor); and 3) knowledge management services (e.g., service composition rules and context resources). These services offer an SOA for pervasive environments through registration, discovery, composition, and orchestration of services. Most of these services are established on complex reasoning mechanisms and protocols based on the WSAN's contextual model, which represents a semantic description of low- and high-level resources of the WSANs. Real-life implementations in hospital and health management show its potential in terms of response time, efficiency, and reliability. However, *KASOM* does not provide dynamic service composition in mobile and resource constrained IoT infrastructures because of predefined service composition rules provided by in-network agents. Moreover, the proposed security solution by access control is limited to authentication only.

CHOReOS [111], [112] enables large scale choreographies or compositions of adaptable, QoS-aware, and heterogeneous services in IoT. It addresses scalability, interoperability, mobility, and adaptability issues through approaches like scalable probabilistic thing-based service registration and discovery [45], [97]. *CHOReOS* is composed of four components: 1) eXecutable service composition (XSC) to coordinate the composition of services and things; 2) eXtensible service access (XSA) to access services and things; 3) eXtensible service discovery (XSD) to manage protocols and processes for discovery of services and things; and 4) the cloud and grid middleware to manage computational resources and drives the deployment of choreographies. *MobIoT*, a key component of CHOReOS [45], [97], is a thing-based SOM for the mobile IoT. Unlike most existing SOMs [94], [110], its thing-based probabilistic service discovery, registration and look-up protocols, and algorithms scale well in dynamic mobile IoT. Moreover, semantic thing-based service compositions are transparently and automatically executable by *MobIoT* and *CHOReOS*, with no involvement from end-users, which is highly desirable in IoT, especially in M2M communications. However, their probabilistic service registries and discovery mechanisms may fail to provide hard real-time support because of insufficient redundancies in the service registries and discovered services. For the same reason, reliability in these middlewares could be an issue. Moreover, ontology-based semantic support will be very challenging in heterogeneous IoT environments.

Mobile sensor data processing engine (*MOSDEN*) [46] supports a sensing as a service model [113], built on top of *GSN* [21]. The use of a plugin architecture improves the scalability and user friendliness of the middleware, as plugins for heterogeneous devices are easier to build and available in easily accessible places (e.g., Google play). *MOSDEN* added a plugin manager and a plugin layer to *GSN* to support and manipulate plugins. It also replaced sensor-dependent individual wrappers from *GSN* with a single generic wrapper to handle communications. *GSN* employs a decentralized P2P architecture [114] and predefined composition rules available in the virtual sensors, which may not work well in IoT's dynamic and ultra large networks. Like *GSN*, *MOSDEN* will suffer in an IoT environment because of its predefined resource/service discovery and service composition mechanisms.

Many cloud-based IoT platforms are available [115], [116]. To provide an impression of the field, a few of these are summarized in the following, and for the others, readers are referred to [115] and references therein.

Xively [99] is a PaaS that provides middleware services to create products and solutions for IoT. Public cloud-based *Xively* offers developers a standards-based directory, data, and business services. Directory services help to find appropriate objects with appropriate permission. Data management services, using a high-performance and time-series database, store and retrieve data reliably. Its Web-based tools simplify data and control other application complexities of IoT development. Business services include a device lifecycle management service including device provisioning. *Xively*'s device lifecycle management and real-time message bus supports large-scale and real-time deployments in IoT. Importantly, it offers support for end-to-end security over the entire platform to ensure IoT solutions' integrity. The lack of storage security [117] can be an issue in many IoT applications. It supports multiple data formats, however, it does not homogenise the incoming data so data processing must be done individually for each source or it needs a prior mapping process to standardize it. This creates an overhead in the system. Also, it supports a list of software and hardware combinations needed to develop IoT applications, but its support for interoperability is limited.

CarrIoTs [98] is a cloud-based SOM for IoT, especially for M2M communications, and focuses on: cost effective M2M application development, scalability, and ease of use. The main advantage of *CarrIoTs* is that it supports network level scalability. Users can put triggers on various stages of the data processing cycle to push data to an external system. Like *Xively*, *CarrIoTs* does not standardize the incoming data. It also does not guarantee storage security, and offers limited support for interoperability [117]. However, *CarrIoTs* and *Xively* benefit from active online communities, and are popular among developers and new adopters.

Echelon [118] is an IIoT platform with a full suite of chips, stacks, modules, interfaces, and management software for developing devices, and P2P communities. Unlike consumer IoT platforms, it addresses the core requirements for the IIoT, including autonomous control, industrial-strength reliability, support for legacy evolution, and exceptional security. Similar to *Xively*, *CarrIoTs* and other cloud platforms, its interoperability is limited within *Echelon*'s and a specific list of other hardware. Being a private cloud, its security is better than *Xively*, but trust is still an issue for sensitive IIoT applications.

The middleware presented by Faghih *et al.* [119] is designed for multimedia sensor networks, and supports scalability and network level heterogeneity. *WhereX* [120] is designed for RFID and mainly supports data management. This section does not cover an exhaustive set of the available SOMs for IoT. A number of recent (since 2009) representative works have been covered to present the state-of-the-art of service-oriented IoT middlewares. A survey of the WSN-specific SOMs (dated mostly pre 2009) is available in [121].

As SOC by nature supports abstraction and does not explicitly deal with code, existing SOMs do not explicitly consider abstraction and code management. Most existing SOMs are WSNs-centric and their scale is limited to WSNs, which is typically in the range of thousands, much less than the ultra large-scale (billions) of IoT. Most of these middlewares' resource discovery and management, and their predefined and deterministic composition mechanisms, will not scale well in ultra large and dynamic IoT environments. A global, scalable, understanding of IoT services' syntax, and semantics is required. Most existing standalone SOMs offer only limited security through authentication. Also, cloud platform storage security and trust could be a concern for many IoT applications.

C. VM-Based Middlewares

VM-oriented middleware design provides programming support for a safe execution environment for user applications by virtualizing the infrastructure. The applications are divided into small separate modules, which are injected and distributed throughout the network. Each node in the network holds a VM, which interprets the modules (as shown in Fig. 7). This approach addresses architectural requirements such as high-level programming abstractions, self-management, and adaptivity, while supporting transparency in distributed heterogeneous IoT infrastructures [122], [123]. VMs can be divided into two

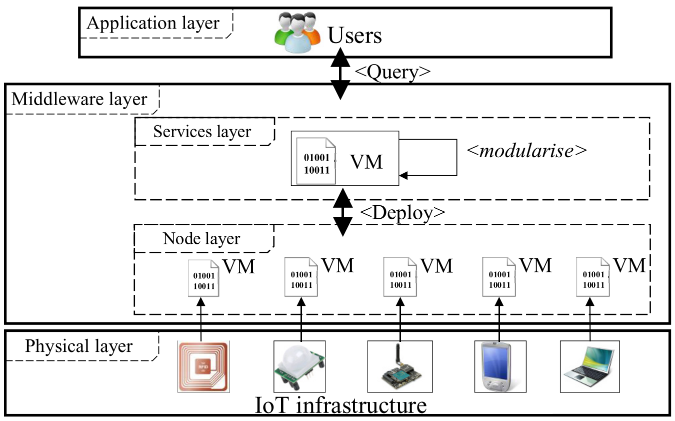

Fig. 7. General design model for a VM-based middleware.

categories: 1) middleware-level VMs (VMs are placed between the OS and applications) and 2) system-level VMs (substitute or replace the entire OS) [22], [122]. Middleware level VMs add capabilities (e.g., concurrency) to the underlying OSs [124]. System level VMs free up resources that would otherwise be consumed by the OS.

Maté [125] is a VM-based middleware for resource-constrained sensor nodes. *Maté* has received extra attention from researchers and developers because it addresses limitations in previous projects (e.g., *Scylla*), which have been focused only on bytecode verification and on-the-fly compilation, and introduces a byte code interpreter that runs on TinyOS. *Maté* effectively handles resource management for sensor network (e.g., bandwidth or energy) and provides support for adaptability [35]. Another key goal of *Maté* is code management, achieved by allowing updates to VM applications. *Maté*'s execution model inherits from the TinyOS synchronous event-based model. According to *Maté*'s developers, it simplifies the development at the application layer by making it less prone to bugs than dealing with asynchronous eventing notification. However, this makes *Maté* not suitable for event-based WSN applications [127], which require a nonblocking approach. Also, the VM itself does not support reprogrammability after deployment [128]. Moreover, *Maté* cannot run multiple applications concurrently in one node [129].

*VM** [128] and *Melete* [130] are based on *Maté* and extend its code management capabilities by enabling fine-grained updates to both the VM applications and the system software. *VM** adds a service layer, which improves resource management and eases application deployment. However, *VM** does not offer support for adaptability. *Melete* enhances the support for concurrent applications. Furthermore, *Melete* adds a code dissemination mechanism to distribute code selectively and reactively [131]. However, it assumes that the network topology is a connected graph, which means it cannot handle a dynamic network topology.

MagnetOS [132], *Squawk* [133], and *Sensorware* [129] are other examples of traditional VM solutions. *MagnetOS* is a distributed OS for sensor networks that abstracts the entire network as a single, unified Java VM, which makes the applications written for *MagnetOS* portable. The main goal of this solution is to reduce energy consumption and increase network longevity. Similar to *MagnetOS*, *Squawk* is a small Java VM that supports multiple applications, provides point-to-point connection types, and uses optimized code to reduce the memory footprint. *Sensorware* is another solution that implements a script interpreter to provide a way to program WSNs based on mobile scripts. However, *MagnetOS*, *Squawk*, and *Sensorware* are unsuitable for resource-constrained devices (i.e., they have a large code base and use RMI, which is a Java-based, heavyweight mechanism [122] for inter-component communication).

The resource-constrained characteristics of WSNs raise an important limitation: VMs require significant memory and processing power resources, which makes virtualization feasible only on resource-rich devices [22]. Code interpretation introduces a significant runtime overhead compared to native binary code [134]. Moreover, the new languages and tools that need to be adopted create a steep learning curve for users and developers [135].

Application-specific virtual machines (ASVMs) [124] solve the problems imposed by traditional VM solutions by limiting the generality of the VMs to subsets relevant to application domain(s) [136]. This type of VM minimizes overhead by reducing the size of the interpreted code and by using an on-the-fly compiler to native code. On the hardware side, the interpretation overhead is minimized using CPU-specific bytecode.

Maté has been extended into a framework for building ASVMs. The new version addresses code management requirements and improves code execution and code propagation by reducing the size of the interpreted code [137]. Also, a security system component was added to avoid propagation of malicious programs through the network [122].

DVM [138] and *DAViM* [139] are based on the concepts introduced by *Maté*. Both take a similar approach for dynamically updating sensor VMs. Like *VM**, *DVM* does not offer support for adaptability. Compared to *DVM*, *DAViM* is designed as a lightweight adaptable service platform for sensor networks [123]. Also, *DAViM* enables concurrent execution of multiple applications. However, *DAViM* is aimed at resource-rich devices. Also, reprogrammability introduces an extra overhead since it requires updates to all the nodes in the network. *DAViM* uses a coordinator to perform the necessary code management tasks, and this component can become a bottleneck in the system.

SwissQM [140] is another ASVM and simplifies WSN programming by increasing the programming abstraction level through a gateway system that accepts programs and queries written in a high-level language. The main design concern of *SwissQM* is to offer better support for data management compared to previous middleware solutions. The other design considerations include support for adaptability, resource management (by providing a dynamic, multiuser, multiprogramming environment through execution of concurrent queries) and support for code management (by offering the ability to dynamically reprogram *SwissQM*). However, only a subset of Java VM bytecode is available. Functionalities like arrays or multiple data types are missing.

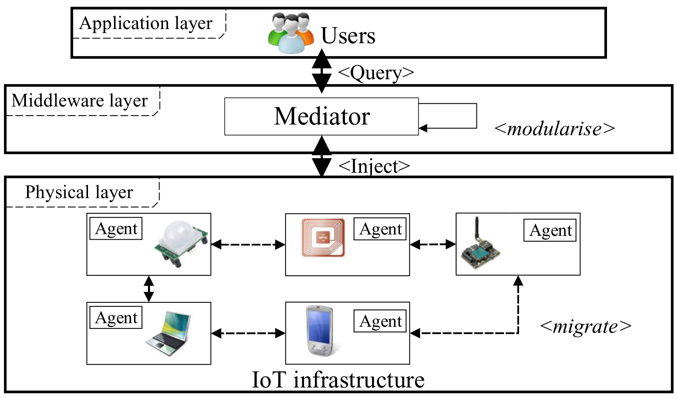

Fig. 8. General design model for an agent-based middleware.

TinyReef [123] and *TinyVM* [141] are other examples of ASVMs, which reduce interpretation cost. *TinyReef* is a register-based VM for WSNs, which has a smaller code size and higher processing speed compared with stack-based VM. However, the data processing unit used by the stack machine interacts only with the top elements of the stack. This workflow improves code processing speed and simplifies the hardware design significantly [142]. *TinyVM* uses compressed machine code, which avoids the CPU-intensive and memory-intensive decompression on motes. However, *TinyVM* was not designed for an IoT environment. The work undertaken is focused mainly on design and evaluation of code compression and the performance of the interpreter. Little information is available about this project, which is likely to have resulted in its low popularity among researchers and developers.

A workflow used by ASVMs is not a viable solution for supporting the heterogeneity of IoT infrastructure because it is heavyweight, which is not compatible with a vision for smaller and cheaper hardware [143]. Also, trading portability for performance reduces flexibility and the possibility of retasking [22]. Further research to address the heavyweight issues is ongoing, with Folliot *et al.* [144] proposing to virtualize the VM [137].

D. Agent-Based Middlewares

In the agent-based approach to middleware, applications are divided into modular programs to facilitate injection and distribution through the network using mobile agents. While migrating from one node to another, agents maintain (as shown in Fig. 8) their execution state. This facilitates the design of decentralized systems capable of tolerating partial failures [145]. Previous research in this area has presented a number of advantages for using mobile agents in generic distributed systems [146], [147]. In the context of the IoT middleware requirements, these are: resource management (network load reduction and network latency reduction), code management (asynchronous and autonomous execution and protocol encapsulation), availability and reliability (robustness and fault-tolerance), adaptiveness and heterogeneity [148]. Moreover, an agent can engage in dialogues with other software agents to proactively gather data and update only parts of the application. Additionally, agent-based approaches consider resource-constrained devices [134].

Impala [149] is a middleware solution for WSNs that enables application modularity, adaptivity, and repairability in WSNs. This middleware solution was part of the ZebraNet project, a mobile sensor network system for improving tracking technology via energy-efficient tracking nodes and P2P communication techniques. *Impala* adopts over-the-air programming for code management and describes a software architecture best suited for improving resource efficiency of resource-constrained nodes. Resource management, mobility, openness, and scalability requirements are supported by switching between different protocols and modes of operation depending on the applications and network conditions. However, *Impala* does not support data preprocessing, which is an important component of data management.

Smart messages [150] proposes an autonomous network architecture for large-scale embedded systems, which are known as resource-constrained, heterogeneous, volatile. *Smart messages* overcomes these limitations by migrating agents to nodes of interest, using application-controlled routing, instead of end-to-end communication between nodes. The main contribution of this middleware is high-flexibility in the presence of dynamic network configurations. However, *Smart messages* does not support multiple applications. Also, it considers only nodes with limited resources, and does not provide support for more complex computations possible in more resource-rich devices.

ActorNet [151] and *Agilla* [28] are also agent-based WSN middleware examples. *ActorNet* is a mobile agent platform for WSNs designed to improve code migration and offer support for interoperability. *ActorNet* introduces services like virtual memory, context switching, and multitasking to enable the execution of complex, highly dynamic mobile agent applications in severely resource-constrained environments. A drawback of *ActorNet* comes from the service discovery mechanism used, which is a broadcast protocol that introduces an extra overhead in the network. *Agilla* reduces its code size and offers support for self-adaptiveness within the WSN by deploying multiple autonomous mobile agents in each node when specific events are triggered. Each agent uses a tuple space structure to ensure consistency and scalability in a dynamic environment and to enable resource discovery. However, *Agilla* does not support a federated tuple space because of energy and bandwidth constraints. Like *Smart Messages*, *Agilla* considers only nodes with limited resources, and does not provide support for more complex computations possible in more resource-rich devices. Also, programmability and code management pose a challenge because of the low level of language abstraction. Moreover, the mobile agents are susceptible to message loss, which interferes with code migration tasks.

Ubiware [152] directly addresses the IoT requirements and domains, and since its inception has become popular in the research and development community. This middleware supports the creation of autonomous, complex, flexible, and extendible industrial systems. The main principles of *Ubiware* are to support automatic resource discovery, monitoring, composition, invocation, and execution of different applications.

A *Ubiware* agent is distributed over three layers: 1) a behavior engine implemented in Java; 2) a declarative middle-layer (behavior models corresponding to agent roles); and 3) a third layer, which contains shared and reusable resources interpreted as Java components (sensors, actuators, smart machines and devices, RFIDs, WS, etc.). *Ubiware* adds security policies to support the security requirement. Interoperability is achieved by semantic adaptation and by assigning a proactive agent to each of the resources. This is supported by using metadata and ontologies. However, support for interoperability is limited. For example, it does not cover the interoperability between different resource discovery protocols.

UbiROAD [153] is a semantic middleware for context-aware smart road environments. It deals with the interoperability between in-car and roadside heterogeneous devices. Semantic interoperability is achieved by two layers: 1) data-level interoperability and 2) functional protocol-level interoperability and coordination. *UbiROAD* is a specialized platform for smart traffic environments, but can also serve as an intelligent protocol between the smart road device layer and future SOAs. It is heterogeneous with respect to components, standards, data formats, and protocols. It is self-adaptive by deploying distributed agents and ensures context-awareness, and adaptive/reconfigurable composition. These requirements are achieved by customisation, personalization, dynamic behavior, and autonomy of services. Autonomous trust management is achieved via semantic annotation. *UbiROAD* guarantees a high level of safety.

AFME [154], *MAPS* [155], *MASPOT* [147], and *TinyMAPS* [156] are Java-based solutions that enable agent-oriented programming of WSN applications. *AFME* is a middleware solution designed for wireless pervasive systems to tackle the performance and code management issues associated with executing agents only on mobile devices. *MAPS* is based on a lightweight agent architecture and offers a set of services to support agent management. *MASPOT* extends the generality of *MAPS* and improves its code migration capabilities. However, the service discovery mechanism used in *MASPOT* employs a broadcast protocol, which introduces an extra overhead in the network. *TinyMAPS* ports *MAPS* onto devices much more resource-constrained than the ones used by *MAPS*. *TinyMAPS* is an ongoing effort to optimize the communication and code migration mechanisms. However, *TinyMAPS* does not consider mobility, which is an important characteristic of an IoT infrastructure.

The agent-based middleware solutions presented (i.e., *Impala*, *Smart Messages*, *Agilla*, *AFME*, *ActorNet*, *MAPS*, *MASPOT*, and *TinyMAPS*) do not address the heterogeneity of an IoT infrastructure. These solutions have been designed only for WSNs or mobile devices. All have been tested on a specific hardware/software platform (e.g., Mica2, MicaZ, TelosB running TinyOS, Hewlett-Packard/Compaq iPAQ Pocket PC running Linux, and Sun SPOTs). Moreover, these middleware solutions do not address issues such as hard real-time guarantees. Security and privacy are generally not considered. In general, these projects are popular, though early adopters are now asking for solutions designed for an IoT environment.

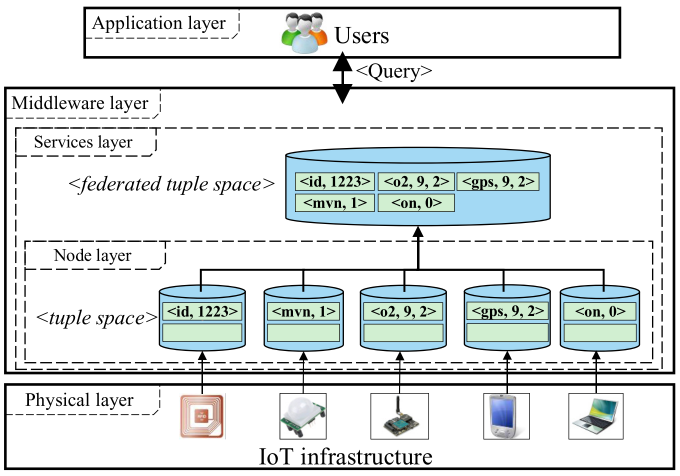

Fig. 9. General design model for a tuple-space middleware.

The IoT vision is to support the connection of various physical world objects to a common infrastructure, and designing a system that will enable this, is a complex process. The use of agent-based systems can reduce the complexity of designing such systems by defining some higher-level policies rather than direct administration. However, the autonomous characteristic of agents can lead to unpredictability in the system at runtime. The patterns and the effects of their interactions are uncertain [157]. Moreover, mobile agents are susceptible to message loss, especially in resource-constrained environments [158]. This imposes many limitations for an IoT middleware solution, including the ability to perform code management tasks.

E. Tuple-Space Middlewares

In tuple-space middlewares, each member of the infrastructure holds a local tuple space structure. A tuple space is a data repository [159] that can be accessed concurrently. All the tuple spaces form a federated tuple space (shown in Fig. 9) on a gateway. This approach suits mobile devices in an IoT infrastructure, as they can transiently share data within gateway connectivity constraints.

Applications communicate by writing tuples in a federated tuple space, and by reading them through specifying the pattern of the data they are interested in.

LIME [160], *TinyLIME* [161], and *TeenyLIME* [162] are tuple-space middleware solutions, each tailored for a specific environment, ranging from mobile ad hoc networks to sensor networks. *LIME* is a middleware for MANETs developed to address mobile devices' energy limitations. *LIME* borrows and adapts the coordination model from Linda [163], and breaks up the centralized tuple space on the gateway into multiple tuple spaces, each permanently attached to a mobile component. Access to the tuple space is carried out using an extended set of tuple space operations, including several constructs designed to facilitate flexible and real-time responses to changes. *LIME* supports good programming abstractions for exploiting a dynamically changing context. However, its context-awareness is limited (e.g., it is not aware of the system configuration). It offers limited support for resource management, event management, and scalability and does not provide any mechanism for security or privacy. Another limitation is that an application can access only the federated tuple-space of the sensors in proximity. *TinyLIME* builds on *LIME* by adding specialized components for sensor networks. However, *TinyLIME* offers limited support for adaptation and does not have any built in security mechanism. *TeenyLIME* is an extension of *LIME* and *TinyLIME*. It provides a more general programming abstraction model by deploying both proactive and reactive operations. It limits the number of application-level uses by controlling a device's one-hop neighborhood. This is done to reduce power usage and improve collection context-sensitive data. A drawback of *LIME*, *TinyLIME*, and *TeenyLIME* is that they are designed for environments in which clients typically only need to query data from local sensors. The sensed data is collected only if the devices are within connectivity limits of a gateway. In an IoT environment, this approach is not sufficient to support distributed services or applications.

TS-Mid [164] is another tuple-space middleware for WSNs, which deploys an asynchronous and decoupled communication style in both time and space. Like in *LIME*, *TinyLIME* and *TeenyLIME*, *TS-Mid* follows the same approach of collecting data on a gateway. However, *TS-Mid* improves the hierarchy of node structure by creating logical regions (or groups) for nodes in proximity. The tasks that were performed previously on the gateway are now performed on an elected leader node in the group. Each leader node is responsible for data aggregation and forwarding to the sink node. The sink node is queried by clients. This model is heterogeneous with respect to programming languages, network, and operating systems. It supports data management through data aggregation and storage. However, it does not support hard real time, security, or privacy. Moreover, the leader node becomes a bottleneck in the group and does not provide uniformity of power usage.

A3-TAG [165] follows the hierarchical node structure used in *TS-Mid*, to address self-adaptation and dissemination of new configuration tasks through group communication. However, *A3-TAG* exhibits the same drawbacks: the leader node become a bottleneck in the group and does not provide uniformity of power usage. Moreover, because of the communication mechanism employed, *A3-TAG* addresses resource-rich devices.

The tuple-spaces middleware solutions presented here (i.e., *LIME*, *TinyLIME*, *TeenyLIME*, *TS-Mid*, and *A3-TAG*) have been designed only for WSNs or mobile devices. Tuple-space middlewares were originally proposed to address the problem of frequent disconnections, and to improve asynchronous communication. Although they have a flexible architecture that allows middleware to be used in different environments, the overheard due to its cross-layer design may be prohibitive in the IoT. Their programming model generally is not reprogrammable and they provide limited support for adaptability or scalability. Nonetheless, they are popular among researchers.

F. Database-Oriented Middlewares

In database-oriented middleware, a sensor network is viewed as a virtual relational database system (as shown in Fig. 10).

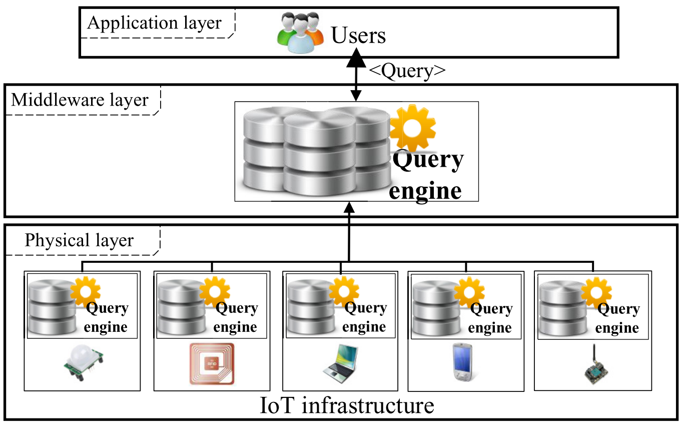

Fig. 10. General design model for a database-oriented middleware.

An application can query the database using an SQL-like query language, which enables the formulation of complex queries [22]. Research in this area has been focused on developing a distributed database approach to interoperating systems.

SINA [166] handles events and can also cope with the mobility of the querying (sink) node [167]. It allows sensor applications to issue queries and command tasks, collect replies and results, and monitor changes within the networks. *SINA* supports resource management though resource monitoring. It supports data preprocessing aggregation. *SINA* modules, running on each sensor node, provide adaptive organization of sensor information, and facilitate query, event monitoring, and tasking capabilities. Sensor nodes are autonomously clustered, which supports energy-efficiency and scalable operations. Although adaptive and autonomous, interoperability and context-awareness requirements are not resolved in *SINA*. *SINA* is not secured or private.

COUGAR [168] is another database-oriented middleware. It is an extension to the cornell predator object-relational database system. In *COUGAR*, there are two types of data: 1) stored data and 2) sensor data. Signal processing functions in each sensor node generate sensor data, which is communicated or stored as relations in a database system. Signal processing functions are modeled by using abstract data types. Long-running queries are formulated using an SQL-like language. Data aggregation refers to delivering data from distributed source sensor nodes to a central node for computation. *COUGAR* provides flexible and scalable access to large collections of sensors. From the functional and nonfunctional requirements aspect, it does not support event or code management.

IrisNet [169] is a database-oriented platform, which deploys heterogeneous services on WSNs. *IrisNet* supports the control of a global, wide-area sensor network by performing internet-like queries on this infrastructure. Each query operates over data collected from the global sensor network, and supports simple and more complex queries involving arithmetic and database operators. It is distributed and lightweight. It uses a database centric approach to publish generated data. The architecture of *IrisNet* is two-tiered. Heterogeneous sensors implement a common shared interface and are called sensing agents (SA). The data produced by sensors are stored in a distributed database that is implemented on organizing agents (OAs). Different sensing services run simultaneously on the architecture. As the processing nodes are always powered, *IrisNet* is not optimized for energy usage. Many architectural challenges are not resolved, such as: interoperability, context-awareness, autonomous behavior, adaptiveness.

Sensation [170] is database-oriented middleware developed for WSN applications, and designed to provide support for different sensors, network infrastructures, and middleware technologies. This level of heterogeneity is supported through an abstraction layer. *Sensation* provides a high-level and intuitive programming model for context-aware pervasive applications. It supports energy-awareness and scalability. Through its synchronous requests (queries), it retrieves requested data, and returns the corresponding responses in real time. *Sensation* is designed for periodic monitoring of sensor values. Context-aware applications use event-driven programming to trigger actions after events have been generated from the WSN.

TinyDB [69], [171] is a distributed query processing middleware system based on TinyOS. *TinyDB* provides power-efficiency in network query processing systems that collect data from individual sensor nodes. Reduced energy consumption is enabled through the reduced number of messages that must be exchanged. While *TinyDB* provides programming abstraction support and a data aggregation model, it does not provide much middleware service functionality, so applications must handle such functions themselves. It has good data management, minimizing expensive communication by applying aggregation and filtering operations inside the sensor network. It supports event-based processing and its processes can be optimized for energy usage.

GSN [172] is a popular project among developers and researchers, and it has been integrated in other projects (e.g., OpenIoT). It uses virtual sensors to control processing priority, management of resources, and stored data. Using declarative specifications, virtual sensors can be deployed and reconfigured in *GSN* containers at runtime. *GSN* creates highly dynamic processing environments and allows the system to quickly react to changing processing needs and environmental conditions. Dynamic resource management accomplishes three main tasks: 1) resource sharing; 2) failure management; and 3) explicit resource control. As the number of clients increases, the average processing time for each client decreases, which caters for scalability [26]. *GSN* provides simple and uniform access to the host of heterogeneous technologies available and is easy to deploy. *GSN* is adaptive, but it is not autonomous and it does not offer support for interoperability, security, or privacy.

$KSpot^{+}$ [173] is a data-centric distributed middleware architecture for WSN. It is network-aware and supports advanced query semantics for data aggregation. $KSpot^{+}$ is an open-source middleware framework that can be used in numerous application domains including environmental monitoring, structural monitoring, urban monitoring, and health monitoring. $KSpot^{+}$ provides a decentralized resource discovery mechanism. Several challenges have been taken into consideration, such as modularity, energy-efficiency, availability, distributed and autonomous behavior, and scalability and failure tolerance. Special attention was given to scalability, to ensure that the performance of $KSpot^{+}$ maintains acceptable QoS standards regardless of increasing network size. It does not support privacy, or code or event management. It is not real time, context-aware, dynamic, or adaptive.

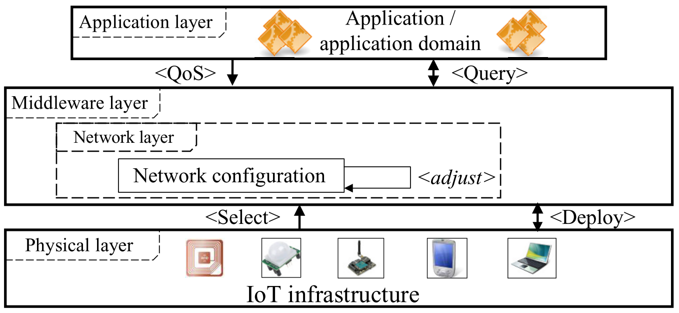

Fig. 11. General design model for an application-specific middleware.

HyCache [174] is an application-level caching middleware for distributed file systems based on database-oriented design. Distributed file systems are deployed on top of *HyCache* on all data nodes. *HyCache*'s strategy is to achieve straightforward high, and scalable, writing flow. This is achieved if the client writes data only to its local storage, which provides data storage management. *HyCache* supports data preprocessing aggregation and achieves an optimal flow by associating all the writes with the local I/O flow. *HyCache* supports resource distributed discovery and resource monitoring management. It provides dynamic programming abstractions. It uses heterogeneous storage devices for distributed file systems and works completely in the user space. It does not deal with security and privacy issues, or with code management. It does not provide real-time services.

A database approach to middleware views the whole network as a virtual database system. Easy-to-use interfaces support user queries to sensor networks to extract data of interest. However, only approximate results are returned. Most IoT applications are real time, where time and space are important. Database middlewares do not support timeliness. Energy consumption is reduced by collecting data from individual nodes. While database middlewares can provide good programming abstraction support and have good data management support, the rest of IoT middleware requirements are mostly ignored. Moreover, database middleware approach uses a centralized model, which makes it difficult to handle large-scale sensor networks dynamics. Furthermore, database middlewares generally do not support hard real-time applications, and are not popular with technology adopters, whose main interests tends to be in safety critical systems.

G. Application-Specific Middlewares

An application-specific (i.e., application-driven) approach to middleware focuses on resource management support (i.e., QoS support) for a specific application or application domain by implementing an architecture that fine-tunes the network or infrastructure (as shown in Fig. 11) based on the application or application domain requirements.

AutoSec [175] and *Adaptive Middleware* [176] are some examples of this approach. *AutoSec* uses a dynamic service broker for resource management in a distributed system. This is done by appropriate combination of information collection and resource provisioning policies based on current system conditions and application requirements. *AutoSec* is application-specific in the sense that it supports only one application at a time. *Adaptive Middleware* explores the tradeoff between resource spending and quality during information collecting. The main goal is to decrease the transmissions between sensor nodes without compromising the overall result. *Adaptive middleware* is autonomous and offers support for adaptation, though has been designed particularly for smart-home context-aware applications.

MiLAN [177] is similar to *Adaptive Middleware*, though *MiLAN* explores the concept of proactive adaptation in order to respond to application needs. *MiLAN* allows applications to specify their QoS requirements and adjust the network configuration at runtime. The adjustments are made based on information collected from the application, the user, the network, and the overall system. Both *Adaptive Middleware* and *MiLAN* require knowledge about the exact sensors. In dynamic and pervasive computing environments, the number and types of sensors available to the applications may vary. It is impractical to include knowledge about all the available sensor nodes that an application can potentially use. Moreover, *MiLAN* does not consider the information acquisition cost. Also, it does not address mobility. *MiLAN* was designed for medical advising and monitoring.

TinyCubus [178] is a cross-layer framework implemented on top of TinyOS. It proposes a generic, extensible and flexible framework that can manage new application requirements. The application-specific requirements are satisfied by customising generic components. However, the cross-layer design produces an extra overhead, which is detrimental for energy usage. Also, this software solution is not scalable. *TinyCubus* was designed for monitoring bridges for structural defects and for driver assistance systems.

MidFusion [179] builds on the concepts presented in *MiLAN* and *Adaptive Middleware*. The purpose of this middleware solution is to avoid maintaining knowledge about the exact sensors available by using Bayesian and decision theory to provide a portable abstraction of the infrastructure to the application. Compared to *MiLAN* and *Adaptive Middleware*, *MidFusion* uses the cost of information acquisition as the selection criterion of the best set of sensors or sensor agents. *MidFusion* was designed for applications that perform information fusion (e.g., an intruder detection system).

Application-specific solutions do not address the heterogeneity of an IoT infrastructure as there is tight coupling between applications and middleware layer. Moreover, the application-specific approach creates only specialized middleware solutions [30] instead of general purpose solutions. This does not satisfy the IoT middleware requirements since an IoT solution should support multiple applications. Furthermore, all the presented application-specific middleware solutions use a centralized resource discovery mechanism, which is not a viable approach for a distributed fault-tolerant IoT solution. Moreover, these drawbacks make this type of middleware solutions unattractive to technology adopters.

Tables I–III summarize the functional, nonfunctional, and architectural capabilities of the surveyed middlewares. In populating the tables, a few common legends are used [e.g., supported (S), not supported (NS), no information (NI)—if no information available about the requirement] along with requirement-specific legends [e.g., for lightweight requirements: memory needed (M) and energy efficiency (E)].

IV. Open Research Challenges and Future Directions

Although the middlewares presented herein address many issues and requirements in IoT, there are still some open research challenges. In particular, research is needed in the area of dynamic heterogeneous resource discovery and composition, scalability, reliability, interoperability, context-awareness, security, and privacy with IoT middleware. Importantly, most current middlewares address WSNs, while other perspectives (e.g., M2M, RFID, and SCADA) are rarely addressed. This survey indicates that there have been significant advances in addressing many challenges for middleware in an IoT environment, with the following open challenges remaining.

A. Challenges Related to Functional Requirements

Resource discovery: The dynamic and ultra-large-scale nature of the IoT infrastructure invalidates centralized resource registries and discovery approaches. However, deciding between purely distributed and hybrid solutions is complicated. A tradeoff is necessary between registry distribution and the number of registries. Fewer registries provide consistent and fast discovery of resources under normal circumstances, but will not scale well when there is a large number of service discovery queries in IoT applications. Probabilistic resource (e.g., service) registries and discovery [45], [97], [180] can be scalable, though may not work well in applications (e.g., mission critical applications) that need guaranteed discovery of resources with high accuracy. Further research is necessary for improved and highly accurate probabilistic models to make them suitable for diverse applications of IoT.

Resource management: Frequent resource conflicts occur in IoT applications that share resources (e.g., actuators). Conflict resolution will be required to resolve conflicts in resource allocation among multiple concurrent services or applications. This is not considered in most existing middleware solutions, except *ubiSOAP* [94] (Tables I and IV). There is clearly significant scope for future work in this area. Agent-based cooperative approach for conflict resolution [181] could be a good starting point for autonomous conflict management.

Data management: A vast amount of raw data continuously collected needs to be converted into usable knowledge, which implies aggregated and filtered data. Most of the surveyed middlewares offer support for data aggregation, but do not consider data filtering. Data filtering is likely to be found in application-specific approaches since the middleware is tailored for a specific application or group of applications. Moreover, no approach offers data compression. This remains an important issue for research since many IoT devices are resource-constrained and transmission of data is more expensive than local processing [87].

Event management: A large number of events are generated proactively and reactively in IoT. Because of this, it is expected that middleware components may become bottlenecks in the system. Most of the middleware surveyed cannot handle or have not been tested against this requirement. Also, events can be primitive (i.e., simple) or complex. Most middlewares statically predefine how an event is handled. Further work should consider complex events and how to handle unknown events. Moreover, the work presented does not consider the difference between discrete (e.g., a door opens, switch ON a light) and continuous events (e.g., driving a car).

Code management: Reprogrammability is one of the major challenges not only in IoT, but also in software development. Updates or changes in business logic should be supported by any IoT component. Agent-based, VM-based, and application-specific middlewares offer support for code management. However, their support for code allocation and code migration is limited. Many do not distinguish between business logic code (i.e., application code) or firmware code. Moreover, none handles both cases. Many middlewares considered only homogeneous devices, though VM approaches address this issue through migration and allocation of interpreted code, rather than compiled code. However, reducing the size of the interpreted code compared with the compiled code is still a challenge.

B. Challenges Related to Nonfunctional Requirements

Scalability: Since most existing middlewares are WSNs centric, their network level scalability is also limited to WSNs. They are likely perform poorly in IoT's ultra-large-scale network. Importantly, scalability is a system-wide requirement, every component (e.g., resource discovery, security solution, context-awareness) of middleware needs be scalable to achieve system-wide scalability.

Real time: Applications and services rely on being directly connected to the physical world. Getting real-time information about the state of the real world is still a challenging task. Some middleware approaches are by nature non-real time (e.g., database or tuple-space middlewares), while the rest provide at least soft real-time services. Hard real time can be provided by application-specific middleware approach and a few event-based middlewares. Current middleware solutions need to consider real-time service composition or self-adaptivity.

Reliability: It is not addressed in most existing proposals. To achieve middleware reliability, every component or service of a middleware needs to be seamlessly replaceable. There is a clear dependency between reliability and other requirements (e.g., compression of data management, lightweight/energy efficiency), which should be better understood and exploited. There is significant scope for future work in this area.

Availability: Maximising system availability and fast recovery from failures are challenges that are not specific to IoT, but to any distributed system. In the context of IoT, availability of things and services offered is important. Hardware devices fail periodically and any service they provide will be unavailable when they fail. Service provision should be seamless by obtaining the required service from a different device.

Security and privacy: All the concerns of security, privacy, and trust in all the technologies (e.g., traditional Internet, WSNs, M2M communications, RFID, SCADA, and cloud computing) used in IoT are clearly present in the context of the IoT. However, security, privacy, and trust are not completely resolved in these technologies. Most existing middlewares' authentication-based partial security solutions are insufficient for a number of IoT applications. Research for a holistic security solution that takes care of system as well as middleware level security and privacy aspects is necessary.

Ease-of-deployment: Deployment, postdeployment, and reprogrammability are important tasks in an IoT middleware lifecycle. Reducing human interaction at these stages and having the possibility to remotely deploy the middleware without any preconfiguration of the device still remains an interesting challenge.

Popularity: Growing an active online community around a software project is a hard task. The project needs to provide a clear value (e.g., material, intellectual, recognition, competitive) to the contributors. For example, *Xively* [99] has become popular following the nuclear accident in Fukushima, when individuals used this platform to monitor the radiation levels across the country. In this paper, the methodology for assessing the popularity relied on: the existence of an online repository where the source code is made available to users, the number of citations of this paper that describes the middleware, and the existence of active communities around these middlewares. A large number of citations (i.e., more than 50) and the existence of a source code repository or a dedicated website reflects a highly popular middleware. A few (i.e., less than 5) or no citations, no repository, and no dedicated website indicates a less popular middleware. The works in between are moderately popular. However, a full analysis of the number of users is beyond the scope of this paper.

C. Challenges Related to Architectural Requirements

Programming abstraction: Most middlewares offer programming abstraction support. However, the new languages and tools that need to be adopted have a steep learning curve for developers and users. Support for this requirement can be improved.

Interoperability: Network interoperability is well supported by most existing middlewares, but many lack support for semantic and syntactical interoperability. Semantic interoperability is very challenging in IoT because of heterogeneity and the lack of standard in ontologies. From all middleware categories, the service-oriented approach offers the best support for semantic interoperability. However, support for syntactic interoperability is limited. For example, in service-oriented approaches, only *Hydra* [101] offers support for this kind of interoperability. Research on global, scalable, understanding of IoT services' syntax, and semantics is required.

TABLE IV
SUMMARY OF THE IoT MIDDLEWARE APPROACHES: IoT MIDDLEWARE REQUIREMENTS

	Functional	Non-functional	Architectural
Event-based	CD-DeD, RM, DPA, SS, CA	NLWSNS, A, CR, HRT	ABS, CW, Sb, E, M , DIST
Service-oriented	DD-DeD, DD-SD, RA, RM, RCA, DS, DPA, SS	AL, NLWSNS, NLIoTS, C, A, SRT	ABS, NeI, CW, DA, Sb, E, M, DIST
VM-based	DD-DeD, RA, RM, RCA, DS, DPA, LS, CA	AL, NLWSNS, A, CR, SRT	ABS, NeI, AUTO, DA, Sb, M, DIST
Agent-based	DD-DeD, RA, RM, DPA, LS, CA, CM	AL, NLWSNS, A, CR, DR, SRT	ABS, NeI, SeI, CW, AUTO, DA, Sb, M, DIST
Tuple-space	CD-DeD, CD-SD, RM, DS, DPA, SS, CM	NLWSNS, SRT	ABS, CW, M, DIST
Database-oriented	DD-DeD, RM, DS, DPA, SS	NLWSNS, A, NRT	ABS, E, DIST
Application-specific	CD-SD, RA, RM, DS, DPA, DPF, LS, CA	NLWSNS, A, CR, HRT	NeI, CW, AUTO, DA, Sb, E, DIST
Legend	CA/CM (code allocation/migration) CD/DD (centralised/distributed discovery) DeD/SD (device/service discovery) DPA/DPC/DPF (data preprocessing - aggregation, compression, filtering) DS (data storage) LS/SS (large/small scale event management) ND (network discovery) RA (resource allocation) RCA (adaptive resource composition) RCL (resource conflict) RCP (predefined resource composition) RM (resource monitor)	A (security: availability) AL (scalability: application level) AS (availability supported) C (security: confidentiality) CR (reliability: communication) DR (reliability: data) I (security integrity) HRT (hard real time) NLIoTS/NLWSNS (scalability: network level IoT/WSN scale) NRT (non-real time) P (privacy supported) SRT (soft real time)	ABS (abstraction supported) AUTO (autonomous) CW (context aware) DA (dynamically adaptive) DIST (distributed) E (lightweight: energy) M (lightweight: memory) NeI (network interoperability) SA (statically adaptive) SeI (semantic interoperability) SI (syntactic interoperability) Sb (service-based)

Service-based: Most of the middlewares are service-based. Service description needs to be comprehensive but energy efficient to become suitable for resource constrained devices. Also, service discovery and composition should be autonomous and dynamically adaptive.

Adaptive: In a number of approaches, adaptation decision-making is hard-coded and requires recompiling and redeploying the system or a part of the system. Where adaptation is more dynamic, policies, rules, or QoS definitions are used, which can be changed during runtime to create new behavior. Even though most middlewares use a dynamic approach, the rules, policies, and QoS definitions are mostly hard-coded and are not context-aware. In IoT, this approach is not scalable. Moreover, only application-specific middlewares dynamically adapt according to the QoS requirements, but this introduces a coupling between middleware components. Research is required for a more flexible, dynamic, and context-aware adaptation model.

Context-awareness and autonomous behavior: Different types of middlewares have exploited some level of context-awareness. For instance, *MUSIC* [70] exploits context for self-adaptation to maintain a satisfactory QoS. Common uses of context (e.g., context-aware resource discovery, context-aware composition, and context-aware data management) [182], [183] from other fields are missing. Also, the context lifecycle approach needs to be standardized. This will improve the interoperability between different middleware components as well as reusability and applicability of extracted context information.

Most existing middlewares are unsuitable for systems with self-* properties (e.g., self-adaptive) including M2M communications. Along with the wider exploitation of context, integration and exploitation of intelligence and self-* properties in IoT middleware system is a rich, open research area.

V. SUMMARY AND FUTURE WORK

Middleware is necessary to ease the development of the diverse applications and services in IoT. Many proposals have focused on this problem. The proposals are diverse and involve various middleware design approaches and support different requirements. This paper puts these works into perspective and presents a holistic view of the field. In doing this, the key characteristics of IoT and the requirements of IoT's middleware are identified. Based on these requirements, a comprehensive survey of these middleware systems focusing on current, state-of-the-art research has been presented. Finally, open research issues, challenges and recommended possible future research directions are outlined.

This survey categorizes the existing middlewares according to their design approaches: event-based, service-oriented, agent-based, tuple-space, VM-based, database-oriented, and application-specific. Each category has many middleware proposals, which are presented accordingly. Most of these proposals have been reviewed and summarized in terms their supported functional, nonfunctional, and architectural requirements (Tables I–III). The summaries show that each middleware fully or partially supports two or more of the listed requirements from each requirement type (e.g., *PRISMA* partially supports code management through code allocation). None supports all the listed requirements.

Table IV summarizes each middleware category in terms of their supported functional, nonfunctional, and architectural requirements. In general, service-oriented, agent-based, and VM-based design approaches address more IoT requirements than others. The service-oriented and VM-based approaches support abstraction and network and application level scalability well. Also, these approaches support resource management through resource compositions, and most cases these compositions can be predefined, especially in VM-based approaches. However, predefined and deterministic composition mechanisms will not scale well in ultra large and dynamic IoT environments. The agent-based design approach is good at resource and code management because of its mobile and distributed nature, but this means that the security and privacy solutions are difficult. On the other hand, middlewares based on tuple-spaces are distributed and relatively more reliable than others because of their data redundancy characteristics. Like agent-based approaches, tuple-space-based middlewares will have difficulties with security and privacy. Database design approaches perform well in data management and respond quickly, assuming non-real time responses are sufficient. Generally, a database approach cannot provide real-time responses to real-time sensing. Event-based middlewares perform well in mobile and reactive applications, but have limited interoperability, adaptability, and context-awareness. Finally, application-specific middlewares are optimized for an application or a group of applications, and may not be suitable and effective for other applications.

Although the existing middleware solutions address many requirements associated with middleware in IoTs, some requirements and related research issues remain relatively unexplored, such as scalable and dynamic resource discovery and composition, system-wide scalability, reliability, security and privacy, interoperability, integration of intelligence, and context-awareness. There is significant scope for future work in these areas.

References

[1] M. Uusitalo, "Global vision for the future wireless world from the WWRF," *IEEE Veh. Technol. Mag.*, vol. 1, no. 2, pp. 4–8, Jan. 2006.

[2] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, "Future Internet research and experimentation: The FIRE initiative," *Comput. Commun. Rev.*, vol. 37, no. 3, pp. 89–92, 2007.

[3] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W. D. Meuter, "Middleware for the Internet of Things, design goals and challenges," *Electron. Commun. EASST*, vol. 28, 2010.

[4] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, "Convergence of MANET and WSN in IoT urban scenarios," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3558–3567, Oct. 2013.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things: A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

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

[7] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni, "Rapid prototyping of semantic mash-ups through semantic web pipes," in *Proc. 18th Int. Conf. World Wide Web*, 2009, pp. 581–590.

[8] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The Internet of Things for ambient assisted living," in *Proc. 7th Int. Conf. Inf. Technol. New Gener. (ITNG)*, Apr. 2010, pp. 804–809.

[9] *Contiki* [Online]. Available: http://www.contiki-os.org/

[10] *Brillo* [Online]. Available: https://developers.google.com/brillo/

[11] *mbed* [Online]. Available: http://mbed.org/technology/os/

[12] *RIOT* [Online]. Available: http://www.riot-os.org/

[13] *FreeRTOS* [Online]. Available: http://www.freertos.org/

[14] *Embedded Linux* [Online]. Available: http://elinux.org

[15] *OpenWSN* [Online]. Available: http://openwsn.atlassian.net

[16] *TinyOS* [Online]. Available: http://www.tinyos.net/

[17] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, *Smart Semantic Middleware for the Internet of Things*. INSTICC Press, 2008, pp. 169–178.

[18] Y. Ni, U. Kremer, A. Stere, and L. Iftode, "Programming ad-hoc networks of mobile and resource-constrained devices," *SIGPLAN Not.*, vol. 40, no. 6, pp. 249–260, Jun. 2005.

[19] H. Zhou, *The Internet of Things in the Cloud: A Middleware Perspective*, 1st ed. Boca Raton, FL, USA: CRC, 2012.

[20] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, May 2013.

[21] K. Aberer and M. Hauswirth, "Middleware support for the 'Internet of Things'," 2006.

[22] A. Azzara, S. Bocchino, P. Pagano, G. Pellerano, and M. Petracca, "Middleware solutions in WSN: The IoT oriented approach in the ICSI project," in *Proc. 21st Int. Conf. Softw. Telecommun. Comput. Netw.*, 2013, pp. 1–6.

[23] V. Issarny *et al.*, "Service-oriented middleware for the future Internet: State of the art and research directions," *J. Internet Serv. Appl.*, vol. 2, no. 1, pp. 23–45, 2011.

[24] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "A survey of middleware for Internet of Things," in *Recent Trends in Wireless and Mobile Networks*. New York, NY, USA: Springer, 2011, vol. 162, pp. 288–296.

[25] F. C. Delicato, P. F. Pires, and T. Batista, *Middleware Solutions for the Internet of Things*. New York, NY, USA: Springer, 2013.

[26] K. Aberer, M. Hauswirth, and A. Salehi, "The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks," Tech. Rep., 2006.

[27] M. Musolesi, C. Mascolo, and S. Hailes, "EMMA: Epidemic messaging middleware for ad hoc networks," *Pers. Ubiq. Comput.*, vol. 10, no. 1, pp. 28–36, 2005.

[28] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *ACM Trans. Auton. Adapt. Syst. (TAAS)*, vol. 4, no. 3, p. 16, 2009.

[29] J. R. Silva *et al.*, "PRISMA: A publish-subscribe and resource-oriented middleware for wireless sensor networks," in *Proc. Adv. Int. Conf. Telecommun. (AICT'14)*, 2014, pp. 87–97.

[30] S. Hadim and N. Mohamed, "Middleware: Middleware challenges and approaches for wireless sensor networks," *IEEE Distrib. Syst. Online*, vol. 7, no. 3, p. 1, Mar. 2006.

[31] M.-M. Wang, J.-N. Cao, J. Li, and S. Dasi, "Middleware for wireless sensor networks: A survey," *J. Comput. Sci. Technol.*, vol. 23, no. 3, pp. 305–326, 2008.

[32] D. C. Schmidt, "Middleware for real-time and embedded systems," *Commun. ACM*, vol. 45, no. 6, pp. 43–48, 2002.

[33] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," *Serv. Oriented Comput. Appl.*, vol. 5, no. 2, pp. 71–85, 2011.

[34] J. Radhika and S. Malarvizhi, "Middleware approaches for wireless sensor networks: An overview," *IJCSI Int. J. Comput. Sci. Issues*, vol. 9, no. 6, 2012.

[35] X. Li and S. Moh, "Middleware systems for wireless sensor networks: A comparative survey," *Contemp. Eng. Sci.*, vol. 7, no. 13, pp. 649–660, 2014.

[36] B. Bhuyan, H. K. D. Sarma, and N. Sarma, "A survey on middleware for wireless sensor networks," *J. Wireless Netw. Commun.*, vol. 4, no. 1, pp. 7–17, 2014.

[37] J. Al-Jaroodi, J. Aziz, and N. Mohamed, "Middleware for RFID systems: An overview," in *Proc. 33rd Annu. IEEE Int. (COMPSAC'09)*, Jul. 2009, vol. 2, pp. 154–159.

[38] N. C. Găitan, V. G. Găitan, Ş. G. Pentiuc, I. Ungurean, and E. Dodiu, "Middleware based model of heterogeneous systems for SCADA distributed applications," *Adv. Elect. Comput. Eng.*, vol. 10, no. 2, pp. 121–124, 2010.

[39] D. Niyato, L. Xiao, and P. Wang, "Machine-to-machine communications for home energy management system in smart grid," *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 53–59, Apr. 2011.

[40] "Combining heterogeneous service technologies for building an Internet of Things middleware," *Comput. Commun.*, vol. 35, no. 4, pp. 405–417, 2012.

[41] E.-J. Kim and S. Youm, "Machine-to-machine platform architecture for horizontal service integration," *EURASIP J. Wireless Commun. Netw.*, vol. 2013, no. 1, pp. 1–9, 2013.

[42] L. Roalter, M. Kranz, and A. Möller, "A middleware for intelligent environments and the Internet of Things," in *Proc. 7th Int. Conf. Ubiq. Intell. Comput.*, 2010, pp. 267–281.

[43] Z. Song, A. Cardenas, and R. Masuoka, "Semantic middleware for the Internet of Things," in *Proc. Internet Things (IOT)*, Nov. 2010, pp. 1–8.

[44] Y. Hong, "A resource-oriented middleware framework for heterogeneous Internet of Things," in *Proc. Int. Conf. Cloud Serv. Comput. (CSC)*, Nov. 2012, pp. 12–16.

[45] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, "Service oriented middleware for the Internet of Things: A perspective," in *Proc. 4th Eur. Conf. Towards Serv. Based Internet*, 2011, pp. 220–229.

[46] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "MOSDEN: An Internet of Things middleware for resource constrained mobile devices," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, 2014, pp. 1053–1062.

[47] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "Role of middleware for Internet of Things: A study," *Int. J. Comput. Sci. Eng. Survey*, vol. 2, no. 3, pp. 94–105, 2011.

[48] Moor Insights and Strategy. (2013). "Behaviorally segmenting the Internet of Things (IoT)," [Online]. Available: http://www.moorinsightsstrategy.com/wp-content/uploads/2013/10/Behaviorally-Segmenting-the-IoT-by-Moor-Insights-Strategy.pdf

[49] C. P. Greg Gorbach and A. Chatha. (2014). "Planning for the industrial Internet of Things," [Online]. Available: http://www.arcweb.com/brochures/planning-for-the-industrial-internet-of-things.pdf

[50] M. Scott and R. Whitney. (2014). "The industrial Internet of Things," [Online]. Available: http://www.mcrockcapital.com/uploads/1/0/9/6/10961847/mcrock_industrial_internet_of_things_report_2014.pdf

[51] TE Commission. (2008). "Internet of Things in 2020," [Online]. Available: http://www.caba.org/resources/Documents/IS-2008-93.pdf

[52] P. F. Harald Sundmaeker, P. Guillemin, and S. Woelfflé, *Vision and Challenges for Realising the Internet of Things*. Pub. Office EU, 2010 [Online]. Available: http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf

[53] V. Ovidiu and F. Peter, *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*, 2013.

[54] IT Union, "ITU Internet report 2005: The Internet of Things," 2005.

[55] ECHELON. (2013). "Requirements for the industrial Internet of Things," [Online]. Available: http://media.digikey.com/Resources/Echelon/IIoT-WP.PDF

[56] Gartner, "Gartner says the Internet of Things installed base will grow to 26 billion units by 2020," 2013.

[57] "More than 30 billion devices will wirelessly connect to the Internet of Everything in 2020," [Online]. Available: https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne

[58] V. Cristea, C. Dobre, and F. Pop, "Context-aware environments for the Internet of Things," in *Internet of Things and Inter-Cooperative Computational Technologies for Collective Intelligence*. New York, NY, USA: Springer, 2013, vol. 460, pp. 25–49.

[59] IIS Group. (2014). "The Internet of Things starts with intelligence inside," [Online]. Available: http://www.intel.com/newsroom/kits/internetofthings/pdfs/IoT_Day_presentation.pdf

[60] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of Things," *IEEE Internet Comput.*, vol. 14, no. 1, pp. 44–51, Jan./Feb. 2010.

[61] D. Kyriazis and T. Varvarigou, "Smart, autonomous and reliable Internet of Things," *Proc. Comput. Sci.*, vol. 21, pp. 442–448, 2013.

[62] P. Banerjee *et al.*, "Everything as a service: Powering the new information economy," *Computer*, vol. 44, no. 3, pp. 36–43, Mar. 2011.

[63] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," 2013.

[64] D. Tracey and C. Sreenan, "A holistic architecture for the Internet of Things, sensing services and big data," in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid)*, May 2013, pp. 546–553.

[65] TE Commission. (2012). "Protection of personal data," [Online]. Available: http://ec.europa.eu/justice/data-protection/

[66] *Vivante Internet of Things (IoT) Solutions* [Online]. Available: http://bensontao.wordpress.com/2013/10/06/vivante-internet-of-things/

[67] S. Neely, S. Dobson, and P. Nixon, "Adaptive middleware for autonomic systems," *Ann. Técommun.*, vol. 61, nos. 9–10, pp. 1099–1118, 2006.

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

[69] Tinydb [Online]. Available: http://telegraph.cs.berkeley.edu/tinydb/

[70] R. Rouvoy *et al.*, "Middleware support for self-adaptation in ubiquitous and service-oriented environments," in *Software Engineering for Self-Adaptive Systems*. New York, NY, USA: Springer, 2009, pp. 164–182.

[71] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, "Smart semantic middleware for the Internet of Things," in *Proc. ICINCO-ICSO*, 2008, pp. 169–178.

[72] M. Bakillah, S. H. L. Liang, A. Zipf, and M. A. Mostafavi, "A dynamic and context-aware semantic mediation service for discovering and fusion of heterogeneous sensor data," *J. Spatial Inf. Sci.*, vol. 6, no. 1, pp. 155–185, 2013.

[73] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, "Adaptive service composition based on reinforcement learning," in *Service-Oriented Computing*. New York, NY, USA: Springer, 2010, pp. 92–107.

[74] G. Bin, Z. Daqing, and W. Zhu, "Living with Internet of Things: The emergence of embedded intelligence," in *Proc. 4th Int. Conf. Cyber Phys. Soc. Comput. Internet Things (iThings/CPSCom)*, Oct. 2011, pp. 297–304.

[75] K. Modukuri, S. Hariri, N. V. Chalfoun, and M. Yousif, "Autonomous middleware framework for sensor networks," in *Proc. Int. Conf. Pervasive Serv. (ICPS'05)*, 2005, pp. 17–26.

[76] R. Meier and V. Cahill, "Steam: Event-based middleware for wireless ad hoc networks," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst. Workshops*, 2002, pp. 639–644.

[77] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[78] V. Issarny, M. Caporuscio, and N. Georgantas, "A perspective on the future of middleware-based software engineering," in *Proc. Future Softw. Eng.*, 2007, pp. 244–258.

[79] P. R. Pietzuch, "Hermes: A scalable event-based middleware," Univ. Cambridge, Comput. Lab., Tech. Rep. UCAM-CL-TR-590, Jun. 2004 [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-590.pdf

[80] T. Yamashita *et al.*, "Emma middleware: An application-level multicast infrastructure for multi-party video communication," in *Proc. 15th Int. Conf. Parallel Distrib. Comput. Syst. (PDCS'03)*, 2003, pp. 416–421.

[81] T. Sivaharan, G. Blair, and G. Coulson, "Green: A configurable and reconfigurable publish-subscribe middleware for pervasive computing," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*. New York, NY, USA: Springer, 2005, pp. 732–749.

[82] P. Costa *et al.*, "The runes middleware for networked embedded systems and its application in a disaster management scenario," in *Proc. 5th Annu. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom'07)*, 2007, pp. 69–78.

[83] K. K. Khedo and R. Subramanian, "Misense: A generic energy efficient middleware architecture for wireless sensor networks," in *Proc. 2nd Int. Conf. Wireless Commun. Sensor Netw. (WCSN'06)*, 2006, pp. 207–215.

[84] K. K. Khedo and R. Subramanian, "Meeca: Misense energy efficient clustering algorithm," in *Proc. 3rd Int. Conf. Wireless Commun. Sensor Netw. (WCSN'07)*, 2007, pp. 31–35.

[85] S. Lai, J. Cao, and Y. Zheng, "Psware: a publish/subscribe middleware supporting composite event in wireless sensor network," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom'09)*, 2009, pp. 1–6.

[86] P. Boonma and J. Suzuki, "TinyDDS: An interoperable and configurable," *Principles and Applications of Distributed Event-Based Systems*, 2010, p. 206.

[87] A. R. Ribeiro, F. Silva, L. C. Freitas, J. C. Costa, and C. R. Francês, "Sensorbus: A middleware model for wireless sensor networks," in *Proc. 3rd Int. IFIP/ACM Latin Amer. Conf. Netw.*, 2005, pp. 1–9.

[88] E. Souto *et al.*, "Mires: A publish/subscribe middleware for sensor networks," *Pers. Ubiq. Comput.*, vol. 10, no. 1, pp. 37–44, 2006.

[89] Websphere mq [Online]. Available: http://www-03.ibm.com/software/products/en/websphere-mq

[90] Mosquitto [Online]. Available: http://mosquitto.org/

[91] Ibm mq [Online]. Available: http://www-03.ibm.com/software/products/en/ibm-mq

[92] M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. 4th Int. Conf. Web Inf. Syst. Eng. (WISE'03)*, Dec. 2003 pp. 3–12.

[93] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 223–235, Jul. 2010.

[94] M. Caporuscio, P. G. Raverdy, and V. Issarny, "UbiSOAP: A service-oriented middleware for ubiquitous networking," *IEEE Trans. Serv. Comput.*, vol. 5, no. 1, pp. 86–98, Jan./Mar. 2012.

[95] C. L. Fok, G. C. Roman, and C. Lu, "Servilla: A flexible service provisioning middleware for heterogeneous sensor networks," *Sci. Comput. Programm.*, vol. 77, no. 6, pp. 663–684, 2012.

[96] N. Reijers, K.-J. Lin, Y.-C. Wang, C.-S. Shih, and J. Y. Hsu, "Design of an intelligent middleware for flexible sensor configuration in M2M systems," in *Proc. SENSORNETS*, 2013, pp. 41–46.

[97] S. Hachem, "Service-oriented middleware for the large-scale mobile Internet of Things," M.S. theses, Université de Versailles-Saint Quentin en Yvelines, Versailles, France, Feb. 2014.

[98] Carriots [Online]. Available: https://www.carriots.com/

[99] Xively [Online]. Available: https://xively.com/

[100] Sicsthsense [Online]. Available: http://sense.sics.se/

[101] M. Eisenhauer, P. Rosengren, and P. Antolin, "Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *The Internet of Things*. New York, NY, USA: Springer, 2010, pp. 367–373.

[102] *Linksmart Middleware Portal* [Online]. Available: https://linksmart.eu

[103] P. L. Evensen and H. Meling, "A service oriented middleware with sensor virtualization and self-configuration," in *Proc. Int. Conf. Intell. Sens. Sensor Netw. Inf. Process. (ISSNIP)*, 2009, p. 261.

[104] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, 2005.

[105] E. Avils-Lpez and J. Garca-Macas, "TinySOA: A service-oriented architecture for wireless sensor networks," *Serv. Oriented Comput. Appl.*, vol. 3, no. 2, pp. 99–108, 2009.

[106] H. Bohn, A. Bobek, and F. Golatowski, "Service infrastructure for real-time embedded networked devices: A service oriented framework for different domains," in *Proc. Int. Conf. Netw./Int. Conf. Syst. Int. Conf. Mobile Commun. Learn. Technol. (ICN/ICONS/MCL)*, Apr. 2006, p. 43.

[107] S. de Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker, "SODA: Service oriented device architecture," *IEEE Pervasive Comput.*, vol. 5, no. 3, pp. 94–96, Jul. 2006.

[108] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, "A service-oriented architecture for QoS configuration and management of wireless sensor networks," in *Proc. IEEE Conf. Emerging Technol. Factory Autom. (ETFA)*, 2010, pp. 1–8.

[109] V. Tsiatsis *et al.*, *The SENSEI Real World Internet Architecture, "Towards the Future Internet—Emerging Trends From European Research"*. Amsterdam, The Netherlands: IOS Press, 2010, pp. 247–256.

[110] I. Corredor, J. F. Martnez, M. S. Familiar, and L. López, "Knowledge-aware and service-oriented middleware for deploying pervasive services," *J. Netw. Comput. Appl.*, vol. 35, pp. 562–576, 2012.

[111] A. Ben Hamida *et al.*, "The future Internet," pp. 81–92, 2012.

[112] A. Ben Hamida *et al.*, "Integrated CHOReOs middleware—Enabling large-scale, QoS-aware adaptive choreographies," Sep. 2013.

[113] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by Internet of Things," *Trans. Emerging Telecommun. Technol.*, vol. 25, no. 1, pp. 81–93, 2014.

[114] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," *Comput. Netw.*, vol. 52, no. 11, Aug. 2008.

[115] "Top 49 tools for the Internet of Things," [Online]. Available: http://blog.profitbricks.com/top-49-tools-internet-of-things/

[116] "OpenIoT," [Online]. Available: http://www.openiot.eu/

[117] S. Satyadevan, B. Kalarickal, and M. Jinesh, "Security, trust and implementation limitations of prominent IoT platforms," in *Proc. Front. Intell. Comput. Theory Appl. (FICTA'14)*, 2015, vol. 328, pp. 85–95.

[118] *Industrial IoT Platform* [Online]. Available: http://www.echelon.com/

[119] M. M. Faghih and M. E. Moghaddam, "SOMM: A new service oriented middleware for generic wireless multimedia sensor networks based on code mobility," *Sensors*, vol. 11, pp. 10343–10371, 2011.

[120] A. Puliafito, A. Cucinotta, A. Minnolo, and A. Zaia, "Making the Internet of Things a reality: The wherex solution," in *The Internet of Things*. New York, NY, USA: Springer, 2010, pp. 99–108.

[121] N. Mohamed and J. Al-Jaroodi, "A survey on service-oriented middleware for wireless sensor networks," *Serv. Oriented Comput. Appl.*, vol. 5, no. 2, pp. 71–85, 2011.

[122] N. Costa, A. Pereira, and C. Serodio, "Virtual machines applied to WSN's: The state-of-the-art and classification," in *Proc. Int. Conf. Syst. Netw. Commun. (ICSNC'07)*, Aug. 2007, p. 50.

[123] I. Marques, J. Ronan, and N. Rosa, "A register-based virtual machine for wireless sensor networks," in *Proc. Sensors*, 2009, pp. 1423–1426.

[124] P. A. Levis, "Application specific virtual machines: Operating system support for user-level sensornet programming," Ph.D. dissertation, Berkeley, CA, USA, 2005.

[125] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," *SIGARCH Comput. Archit. News*, vol. 30, no. 5, Oct. 2002.

[126] P. Stanley-Marbell and L. Iftode, "Scylla: A smart virtual machine for mobile embedded systems," in *Proc. 3rd IEEE Workshop Mobile Comput. Syst. Appl.*, 2000, pp. 41–50.

[127] F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "Middleware platforms: State of the art, new issues, and future trends," in *The Art of Wireless Sensor Networks*. New York, NY, USA: Springer, 2014, pp. 645–674.

[128] J. Koshy and R. Pandey, "Vm*: Synthesizing scalable runtime environments for sensor networks," in *Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst. (Sensys)*, 2005, pp. 243–254.

[129] A. Boulis, C.-C. Han, R. Shea, and M. B. Srivastava, "Sensorware: Programming sensor networks beyond code update and querying," *Pervasive Mobile Comput.*, vol. 3, no. 4, pp. 386–412, 2007.

[130] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proc. 4th Int. Conf. Embedded Netw. Sensor Syst.*, 2006, pp. 139–152.

[131] W. Horré, N. Matthys, S. Michiels, W. Joosen, and P. Verbaeten, "A survey of middleware for wireless sensor networks," CW Reports, 2007.

[132] C. M. Kirsch, M. A. Sanvido, and T. A. Henzinger, "A programmable microkernel for real-time systems," in *Proc. 1st ACM/USENIX Int. Conf. Virtual Execution Environ.*, 2005, pp. 35–45.

[133] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java on the bare metal of wireless sensor devices: The Squawk Java virtual machine," in *Proc. 2nd Int. Conf. Virtual Execution Environ.*, 2006, pp. 78–88.

[134] J. Ceclio and P. Furtado, "Existing middleware solutions for wireless sensor networks," in *Proc. Wireless Sens. Heterogen. Netw. Syst.*, 2014, pp. 39–59.

[135] J. Ellul and K. Martinez, "Run-time compilation of bytecode in sensor networks," in *Proc. 4th Int. Conf. Sensor Technol. Appl. (SENSORCOMM)*, Jul. 2010, pp. 133–138.

[136] Z. Song, M. Lazarescu, R. Tomasi, L. Lavagno, and M. Spirito, "High-level Internet of Things applications development using wireless sensor networks," in *Internet of Things*, 2014, vol. 9, pp. 75–109.

[137] P. Levis, D. Gay, and D. Culler, "Active sensor networks," in *Proc. 2nd Conf. Symp. Netw. Syst. Des. Implement.*, 2005.

[138] R. Balani, C.-C. Han, R. K. Rengaswamy, I. Tsigkogiannis, and M. Srivastava, "Multi-level software reconfiguration for sensor networks," in *Proc. 6th ACM IEEE Int. Conf. Embedded Softw.*, 2006, pp. 112–121.

[139] S. Michiels, W. Horré, W. Joosen, and P. Verbaeten, "DAViM: A dynamically adaptable virtual machine for sensor networks," in *Proc. Int. Workshop Middleware Sensor Netw.*, 2006, pp. 7–12.

[140] R. Mueller, G. Alonso, and D. Kossmann, "SwissQM: Next generation data processing in sensor networks," in *Proc. Conf. Innov. Data Syst. Res. (CIDR)*, 2007, pp. 1–9.

[141] K. Hong *et al.*, "TinyVM: An energy-efficient execution infrastructure for sensor networks," *Softw. Practice Exp.*, no. 10, 2012.

[142] S. Bosse, "Distributed agent-based computing in material-embedded sensor network systems with the agent-on-chip architecture," *IEEE Sens. J.*, vol. 14, no. 7, pp. 2159–2170, Jul. 2014.

[143] P. Winterbottom and R. Pike, "The design of the inferno virtual machine," in *Proc. IEEE Compcon*, 1997, vol. 97, pp. 241–244.

[144] B. Folliot, I. Piumarta, and F. Riccardi, "A dynamically configurable, multi-language execution platform," in *Proc. 8th ACM SIGOPS Eur. Workshop Support Compos. Distrib. Appl.*, 1998, pp. 175–181.

[145] M. Mamei and F. Zambonelli, *Field-Based Coordination for Pervasive Multiagent Systems*. Berlin, Germany: Springer-Verlag, 2005.

[146] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Commun. ACM*, vol. 42, no. 3, pp. 88–89, Mar. 1999.

[147] R. Lopes, F. Assis, and C. Montez, "MASPOT: A mobile agent system for sun spot," in *Proc. Int. Symp. Auton. Decentralized Syst. (ISADS)*, Mar. 2011, pp. 25–31.

[148] F. Aiello, G. Fortino, and A. Guerrieri, "Using mobile agents as enabling technology for wireless sensor networks," in *Proc. SENSORCOMM*, Aug. 2008.

[149] T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," *ACM SIGPLAN Notices*, vol. 38, no. 10, pp. 107–118, 2003.

[150] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, and L. Iftode, "Smart messages: A distributed computing platform for networks of embedded systems," *Comput. J. Spec. Focus Mobile Pervasive Comput.*, vol. 47, pp. 475–494, 2004.

[151] Y. Kwon, S. Sundresh, K. Mechitov, and G. Agha, "Actornet: An actor platform for wireless sensor networks," Tech. Rep., 2005.

[152] N. Michal, K. Artem, K. Oleksiy, N. Sergiy, S. Michal, and T. Vagan, "Challenges of middleware for the Internet of Things," in *Automation Control—Theory and Practice*. InTech, 2009.

[153] V. Terziyan, O. Kaykova, and D. Zhovtobryukh, "Semantic middleware for context-aware smart road environments," in *Proc. 5th Int. Conf. Internet Web Appl. Serv. (ICIW)*, 2010, pp. 295–302.

[154] C. Muldoon, G. OHare, R. Collier, and M. OGrady, "Agent factory micro edition: A framework for ambient applications," in *Proc. Int. Conf. Comput. Sci. (ICCS)*, 2006, vol. 3993, pp. 727–734.

[155] F. Aiello, A. Carbone, G. Fortino, and S. Galzarano, "Java-based mobile agent platforms for wireless sensor networks," in *Proc. Int. Multiconf. Comput. Sci. Inf. Technol. (IMCSIT)*, Oct. 2010, pp. 165–172.

[156] F. Aiello, G. Fortino, S. Galzarano, and A. Vittorioso, "TinyMAPS: A lightweight java-based mobile agent system for wireless sensor networks," in *Intelligent Distributed Computing V*. New York, NY, USA: Springer, 2012, vol. 382, pp. 161–170.

[157] N. R. Jennings, "An agent-based approach for building complex software systems," *Communications*, pp. 35–41, 2001.

[158] E. Pignaton de Freitas, "A survey on adaptable middleware for wireless sensor networks," 2008.

[159] L. Mottola, A. L. Murphy, and G. P. Picco, "Pervasive games in a mote-enabled virtual world using tuple space middleware," in *Proc. 5th ACM SIGCOMM*, 2006, p. 29.

[160] A. L. Murphy, G. P. Picco, and G. Roman, "LIME: A middleware for physical and logical mobility," in *Proc. 21st Int. Conf. Distrib. Comput. Syst.*, 2001, pp. 524–533.

[161] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco, "The TinyLime middleware," *Pervasive Mobile Comput.*, vol. 1, no. 4, pp. 446–469, 2005.

[162] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "TeenyLIME: Transiently shared tuple space middleware for wireless sensor networks," in *Proc. Middleware Sensor Netw.*, 2006, pp. 43–48.

[163] D. Gelernter, "Generative communication in Linda," *TOPLAS*, vol. 7, no. 1, pp. 80–112, 1985.
[164] R. de Cassia Acioli Lima, N. S. Rosa, and I. R. L. Marques, "TS-Mid: Middleware for wireless sensor networks based on tuple space," in *Proc. 22nd Int. Conf. Adv. Inf. Netw. Appl. Workshops (AINAW)*, 2008, pp. 886–891.
[165] L. Baresi, S. Guinea, and P. Saeedi, "Achieving self-adaptation through dynamic group management," in *Assurances for Self-Adaptive Systems*. New York, NY, USA: Springer, 2013, pp. 214–239.
[166] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," I*EEE Pers. Commun.*, vol. 8, no. 4, pp. 52–59, Aug. 2001.
[167] K. Henricksen and R. Robinson, "A survey of middleware for sensor networks: State-of-the-art and future directions," in *Proc. Middleware Sensor Netw.*, 2006, pp. 60–65.
[168] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Mobile Data Management*. New York, NY, USA: Springer, 2001, vol. 1987, pp. 3–14.
[169] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "IrisNet: An architecture for a worldwide sensor web," *IEEE Pervasive Comput.*, vol. 2, no. 4, pp. 22–33, Oct./Dec. 2003.
[170] P. Hasiotis, G. Alyfantis, V. Tsetsos, O. Sekkas, and S. Hadjiefthymiades, "Sensation: A middleware integration platform for pervasive applications in wireless sensor networks," in *Proc. Wireless Sensor Netw.*, 2005, pp. 366–377.
[171] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.
[172] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *Proc. 32nd Int. Conf. Very Large Data Bases*, 2006, p. 1199.
[173] P. Andreou, D. Zeinalipour-Yiazti, P. Chrysanthis, and G. Samaras, "Towards a network-aware middleware for wireless sensor networks," in *Proc. Int. Workshop Data Manage. Sensor Netw. (DMSN)*, 2011.
[174] D. Zhao and I. Raicu, "HyCache: A user-level caching middleware for distributed file systems," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process. Workshops (IPDPSW)*, 2013, pp. 1997–2006.
[175] Q. Han and N. Venkatasubramanian, "Autosec: An integrated middleware framework for dynamic service brokering," *IEEE Distrib. Syst. Online*, vol. 2, no. 7, pp. 22–31, Oct. 2001.
[176] M. C. Huebscher and J. A. McCann, "Adaptive middleware for context-aware applications in smart-homes," in *Proc. Middleware Pervasive Ad-Hoc Comput.*, 2004, pp. 111–116.
[177] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *Network*, pp. 6–14, 2004.
[178] A. Lachenmann, "Tinycubus: Publications," 2011.
[179] H. Alex, M. Kumar, and B. Shirazi, "Midfusion: An adaptive middleware for information fusion in sensor network applications," *Inf. Fusion*, vol. 9, no. 3, pp. 332–343, Jul. 2008.
[180] A. Nedos, K. Singh, R. Cunningham, and S. Clarke, "Probabilistic discovery of semantically diverse content in MANETs," *IEEE Trans. Mobile Comput.*, vol. 8, no. 4, pp. 544–557, Apr. 2009.
[181] H. S. Nwana, L. Lee, and N. R. Jennings, "Co-ordination in software agent systems," 1996.
[182] K. Fujii and T. Suda, "Semantics-based context-aware dynamic service composition," in *Remote Instrumentation and Virtual Laboratories*. New York, NY, USA: Springer, 2010, pp. 293–311.
[183] W. Rong and K. Liu, "A survey of context aware web service discovery: From user's perspective," in *Proc. Int. Symp. Serv. Oriented Syst. Eng. (SOSE)*, 2010, pp. 15–22.

Mohammad Abdur Razzaque (M'12) received the B.Sc. degree in applied physics, M.Sc. degree in electronics and communication engineering, and Ph.D. degree in computer science.

He is currently a Senior Research Fellow with the School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland. His research interests include the Internet of Things, wireless and mobile computing, and communications and wireless security.

Mr. Razzaque is a Member of the Association for Computing Machinery.

Marija Milojevic-Jevric received the B.Sc. and M.Sc. degrees in information technologies and Ph.D. degree in applied mathematics.

She is a Research Fellow with the School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland, where she is a Member of the Distributed System Group and Future Cities. She also worked with the Mathematical Institute, Serbian Academy of Science and Arts, Serbia, as a Research Assistant. Her research work resulted in various papers published in peer-reviewed journals and conferences. Her research interests include optimization, heuristics, metaheuristics, numerical algorithms, Internet of Things, and middleware.

Andrei Palade received the M.Sc. degree in computer science from the University of Glasgow, Glasgow, U.K., and is currently working toward the Ph.D. degree in computer science at Trinity College Dublin, Dublin, Ireland.

His research is funded by the Science Foundation Ireland (SFI). His research interests include service discovery, selection and composition, and middleware for Internet of Things.

Siobhán Clarke received the B.Sc. and Ph.D. degrees in computer science from Dublin City University, Dublin, Ireland.

She is a Professor and a Fellow with Trinity College Dublin, Dublin, Ireland, where she leads the Distributed Systems Group and is the Director of Future Cities, the Trinity Centre for Smart and Sustainable Cities. Her research interests include software engineering models for the dynamic provision of smart and dynamic software services to urban stakeholders in large-scale mobile environments.