Middleware supporting PIS : requirements, solutions, and challenges

Chantal Taconet, Thais Batista, Pedro Victor Borges Caldas da Silva, Georgios Bouloukakis, Everton Cavalcante, Sophie Chabridon, Denis Conan, Thierry Desprats, Denisse Muñante

Middleware supporting PIS: Requirements, solutions, and challenges

Chantal Taconet, Thais Batista, Pedro Borges, Georgios Bouloukakis, Everton Cavalcante, Sophie Chabridon, Denis Conan, Thierry Desprats, and Denisse Muñante

Abstract In this chapter, we consider the requirements for middleware to support Pervasive Information Systems (PIS) in the context of the Internet of Things (IoT). With the IoT, PIS architectures become more and more distributed and need to be supported by middleware that provides applications with an easy integration of contextual data collected from connected objects spread over the Internet. This comes with new challenges and requirements for PIS middleware. In addition to context-awareness, middleware should tackle scalability, security, privacy and interoperability and provide applications with new abstractions representing the physical environment and ensuring the quality of the data that may be used for decision-making, while keeping PIS sustainable. Through the study of the state of the art regarding PIS middleware, we show in this chapter that the middleware community still faces new challenges, such as providing high-level programming models for PIS, supporting PIS dynamic adaptation, disseminating and filtering large volumes of data, end-to-end privacy and interoperability handling, as well as enabling to deploy sustainable applications.

Chantal Taconet, Pedro Borges, Georgios Bouloukakis, Sophie Chabridon, and Denis Conan
SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Évry and Palaiseau, France
e-mail: firstname.lastname@telecom-sudparis.eu

Thais Batista and Everton Cavalcante
Federal University of Rio Grande do Norte, Natal, Brazil
e-mail: thaisbatista@gmail.com,everton.cavalcante@ufrn.br

Thierry Desprats
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France
e-mail: Thierry.Desprats@irit.fr

Denisse Muñante
SAMOVAR, ENSIIE, Évry, France
e-mail: denisse.munantearzapalo@ensiie.fr

1 Introduction

Pervasive computing, the computing that disappears, has been introduced by Weiser (1991). It has been followed by ubiquitous computing, the computing appearing everywhere and anytime introduced by Satyanarayanan (2001a). They both have profoundly changed Information Systems (IS[1]) in the three last decades. Those IS are sometimes qualified as *Pervasive Information Systems* (PIS[2]) (Kourouthanassis and Giaglis, 2007). With PIS, IS features are enriched while their architecture becomes more and more distributed. In addition to the traditional databases they were built upon, they include data coming from the physical environment and should be accessible anytime and from any (mobile) device.

To illustrate the complexity of PIS throughout this chapter, we consider the case of a logistic chain traceability system related to the transport operations of shipments (Ahmed et al., 2021). Each shipment transport involves at least three types of stakeholders: (i) *the shipper* at the origin of the transport request; (ii) *the carriers* in charge of transport operations; and (iii) *the consignee* that receives the transported shipment. Other stakeholders can also be involved in this process, e.g., logistic service providers, customs, insurance companies, and banks. These traceability IS were centralized in the past, but next-generation IS in this domain are going to be more and more distributed. The system is deployed at each stakeholder infrastructure locally and in the cloud. IS includes data collected from the Internet of Things (IoT) with wireless connected devices (such as a temperature sensor) deployed on the shipment and in the stakeholders' infrastructures. Furthermore, traceability data may be used dynamically for decision-making purposes, e.g., a change in a transport company, notification of transport delays to the consignee and the carriers, and the early identification of transport default such as the non-respect of temperature conditions.

PIS software architecture comprises several layers: a *business layer*, a *service/middleware layer*, and *context-management data layer* (a.k.a. IoT layer). Each layer might be composed of several software components provided by different organizations and deployed on a large-scale, heterogeneous, and distributed infrastructure. A PIS may be abstracted by a distributed software architecture in which data and actions are transmitted among components, both inside and between layers. The so-called *middleware* is an essential part of the design and execution of this software architecture.

In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. Its role is to make application development easier, by providing common programming abstractions, by masking the heterogeneity and the distribution of the underlying hardware and operating systems, and by hiding low-level programming details (Krakowiak, 2009). Middleware has provided a key set of features enabling distributed architectures to expand. In the 1990s, middleware started by offering the

[1] The term IS will be herein interchangeably used to express both singular and plural.

[2] The term PIS will be herein interchangeably used to express both singular and plural.

basic client-server model that has been extensively used by IS. Since then, there have been extensive innovations in middleware capabilities. We can mention the persistence capability that enables transparent interactions between applications and databases and the publish-subscribe interaction pattern that enables designers to decouple system components.

PIS have specific requirements concerning middleware. Biegel and Cahill (2007) have identified some of these requirements, such as loosely coupled communication and sensor and actuator abstractions. Raychoudhury et al. (2013) surveyed the literature on middleware for pervasive systems and highlighted new requirements for PIS, such as context management, i.e., how to consume high-level context information obtained after processing, fusing, and filtering a large amount of low-level context data collected from the environment. They also draw attention to the service-oriented paradigm, the common middleware abstraction in this decade, which comes with service discovery and service composition issues. In this chapter, we focus on presenting the state of the art on requirements concerning middleware for PIS in the context of the IoT, i.e., the integration of connected devices that interact with the environment into the Internet. As stated by Blair et al. (2016), the IoT ensues with new requirements and challenges for PIS middleware such as scalability and heterogeneity.

At the same time, as PIS grow in terms of complexity and distribution and become ubiquitous, they raise a new concern in terms of energy consumption. According to Ferreboeuf et al. (2021), the energy demand of Information Technology (IT) in 2019 was estimated to be 4,184 TWh (IT represents 4.2% of the energy consumption and 3.5% of greenhouse gas in the world). If the energy consumption continues to rise by 6.2% by year as it has had since 2015, both energy and greenhouse gas could double in ten years, a non-sustainable scenario. As middleware has a central position in IS and as it is used by many of them, middleware platforms might play a key role in making systems developed atop of them become energy-aware and energy-efficient. These requirements are even more relevant considering that programmers often have limited knowledge on how much energy their software consumes and which parts use most energy (Pang et al., 2016). Consequently, energy consumption is a first-class concern for PIS middleware that we address in this chapter.

The remainder of this chapter is organized as follows. Section 2 describes the requirements imposed by PIS to middleware. Section 3 presents how some of those requirements are handled by middleware in the literature. Section 4 details how platforms proposed in our research respond to some of the identified requirements concerning PIS middleware. Next, Section 5 draws open challenges to be handled in the future. Section 6 concludes the chapter with final remarks.

2 Requirements for PIS middleware

This section gives an overview of the requirements for PIS middleware in the context of the IoT. As the aim of a middleware layer is to bridge the gap between

Table 1 Requirements for PIS in the context of the IoT

Requirement	Type	SotA[a]	Proposals
Context data management			
2.1 Sensing and actuation support	FR[b]		IoTvar (4.5)
2.2 Context-awareness	FR		
2.3 Dynamic adaptation capabilities	FR		
2.4 Quality of Context management	NFR[c]	3.1	QoCIM (4.1), QoDisco (4.4)
Application support			
2.5 Application development support	FR		IoTvar (4.5)
2.6 Support for multiple interaction patterns	FR	3.2	
Exacerbated in IoT systems			
2.7 Enabling interoperability	NFR	3.3	DeX Mediators (4.3), QoDisco (4.4), IoTvar (4.5)
2.8 Security and privacy	FR/NFR	3.4	muDEBS (4.2), QoCIM (4.1)
2.9 Scalability	NFR	3.5	muDEBS (4.2)
2.10 Energy efficiency and energy-awareness	FR/NFR	3.6	

[a] SotA = State of the Art
[b] FR = Functional Requirement
[c] NFR = Non-Functional Requirement

the pervasive elements spread over the physical environment and the applications, the requirements for PIS middleware include the provision of several services to allow applications to gather contextual information from heterogeneous distributed devices. We present the main functional requirements (i.e., driven by application constraints such as interacting with a given sensor or defining application adaptation rules) and non-functional requirements essential in a PIS scenario (such as handling interoperability, scalability, and the need for supporting energy-efficiency). Table 1 summarizes the presented requirements by organizing them in three categories: requirements necessary for *Context data management in the IoT*, *Application support*, and requirements *Exacerbated in IoT systems*. Table 1 also maps the middleware proposals that will be presented in Section 4 with the requirements they tackle and for which we discuss the state of the art in Section 3.

2.1 Sensing and actuation support

PIS middleware needs to deal with small, often battery-powered devices such as sensors and actuators, the physical elements that the system needs to interact with the environment. Sensors typically obtain information from entities of interest or their environment, whereas actuators act on an entity or the environment or provide feedback to the user. A relevant requirement for PIS middleware is to provide programming abstractions that enable event-driven programming at a high level, thereby

significantly simplifying the use of sensors and actuators by hiding the complexity of accessing heterogeneous devices that use different communication protocols.

In the example of the logistic chain traceability related to the transport operations of shipments, all the data collected by sensors providing the temperature in the compartments of a ship during the transportation need to be received by the application. Similarly, the application needs to set the desired temperature remotely by sending a message to some temperature actuators. The middleware layer should provide programming abstractions for the communication with heterogeneous sensors and actuators to support high-level interaction with them.

2.2 Context-awareness

Several works have defined the terms *context* and *context-awareness*. In this chapter, we rely on a generic, well-known definition from Dey and Abowd (2000): *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.* Context-awareness is one of the most notorious characteristics of PIS as it is related to the pervasive capability of collecting, processing, storing, and reasoning about environmental information on a real-time basis. This requirement is essential to support PIS self-adaptation to any environmental condition. For instance, users' mobility, or any environmental disruption, such as temperature increase, that can impact the quality of the application.

In essence, middleware should provide a well-defined interface to generic context management solutions to prevent PIS from dealing with the burden of context-awareness management. Middleware for PIS typically should offer system-level services to deal with context data acquisition, storage, reasoning, discovery, and query processing, as well as automated context-aware adaptation.

In the logistic chain traceability system example, context-awareness is essential to provide specific information to the different stakeholders involved. For instance, the insurance companies do not need to receive the same information as the customs services. Insurance companies are often interested only in information important to the insurance context, which differs from the information of interest to the customs services. Another example of context-awareness, is to automatically trigger alert and reconfiguration in case of an inappropriate temperature detection.

2.3 Dynamic adaptation capabilities

PIS have to be dynamic for diverse reasons, such as failure management, energy budget, network unavailability, user mobility, and unpredictable interactions. In the face of these situations, PIS middleware should hence provide dynamic adaptation

capabilities to ensure the quality and availability of applications at runtime. Dynamic adaptation means the ability of an application to reconfigure its structure, behavior, protocols, etc. without interrupting its execution, ideally with minimal or no human intervention or disruption.

PIS should possess inherent characteristics that make dynamic adaptation particularly relevant. Context-awareness is related to the ability of a system to perceive information about the context where it is inserted into. By sensing environmental conditions, the system can recognize the current context and adapt itself according to changes in it. Another sort of dynamic adaptation in PIS is device mobility, e.g., a user with a mobile device in the environment at a given moment and leaving that location at another, so that PIS needs to transparently discover and (un)link participating devices into the network. Kourouthanassis and Giaglis (2015) also raises opportunistic user interaction as a challenge to the development of PIS, in the sense that it may not be possible to know in advance the users who will interact with the system or the frequency of such interactions. All these features need to be adequately supported by PIS middleware components to enable building applications atop them that can have their structure and behavior adapted at runtime while maintaining their availability and quality.

In the logistic chain traceability system example, dynamic adaptations may be required due to communication latency issues (e.g., changing protocols for the sake of reliability and performance), anomalous operation, unavailability of connected devices due to a low power level or even failure, or measures to improve the accuracy of gathered data. These scenarios point out PIS middleware to maintain availability and work properly in such a dynamic environment while collecting, analyzing, planning and reacting to changes.

2.4 Quality of Context Management

An important requirement concerns monitoring and managing the quality of the context information received by applications. International standardization bodies underline the importance of uncertainty in metrology (Joint Committee for Guides in Metrology, 2008). When reporting the result of a measurement of a physical quantity, some quantitative indication of the quality of the result should be given so that those who use it can assess its reliability.

Regarding context information, Henricksen and Indulska (2004) acknowledge that it may be inherently *ambiguous*, when two different sources provide contradictory information, *inaccurate*, when too little information is available about a situation, or even *erroneous* when it does not reflect reality. For information provided by open data or human beings, e.g., data from social networks, the latter may be incomplete or erroneous, whether voluntarily or not. In general, context information sources are numerous and diverse. They do not all share the same formats or units of measurement, which means that conversion operations are necessary and potentially add new errors.

Quality of Context (QoC) has first been defined by Buchholz et al. (2003) as *any information that describes the quality of information that is used as context information* and can be represented as a set of parameters that reflects the quality of context data (Bellavista et al., 2012) We consider that QoC parameters, such as accuracy or currentness as defined in ISO/IEC 25012 (2008), should therefore be associated with context information in the form of metadata and be used to compute the quality level of context information.

In the case of logistic chains, at least four quality parameters should be considered in these metadata (Ahmed et al., 2021): (i) the *accuracy*, to ensure that the collected data represent the reality of the shipment conditions, (ii) the *completeness*, to ensure that there is no gap in the collected data, (iii) the *consistency*, to ensure the users' agreement on the traceability data collected from multiple sources, and (iv) the *currentness*, to ensure that the collected data are timely valid.

Information provided to context-aware applications is derived through analysis operations and various transformations. However, if these operations are performed on erroneous information, the new information produced is also erroneous. QoC management must hence be carried out throughout the entire information life cycle, from its collection to its dissemination to the applications through all the intermediate transformation steps. Middleware should enable applications to become QoC-aware and provide PIS developers with QoC management facilities.

2.5 Application development support

Middleware platforms are a key element in leveraging application development by abstracting away the specificities of the underlying distributed components from users and exposing valuable reusable services to applications. Besides an accessible programming model that adequately supports application developers by taking advantage of abstractions exposed by PIS middleware, it is relevant to come up with interoperable environments that could assist those developers to effortlessly build their applications while orchestrating the diversity of existing devices, platforms, and services. Inspired by a cloud-based IoT scenario (Truong and Dustdar, 2015), the life cycle of developing a PIS may comprise (i) selecting, composing, and integrating components across the system for specifying and developing possible governance and control operations, (ii) deploying several types of software components at different levels of abstraction and capabilities to configure deployments and continuous resource provisioning, and (iii) capabilities to monitor end-to-end metrics and perform governance processes across the system. Transversally, it is necessary to provide environments supporting the development of applications based on data streams generated by devices and available through the underlying deployment infrastructure (i.e., cloud, edge).

2.6 Support for multiple interaction patterns

To facilitate the development of applications that exchange data between devices and services, PIS middleware platforms rely on IP-based protocols. These protocols abstract distributed peers that interact with each other based on different interaction patterns, such as request-reply, publish-subscribe and event-based. Middleware protocols are typically available through an API, and each protocol supports several characteristics (synchronous/asynchronous interactions, QoS guarantees, etc.). In general, each interaction pattern can be characterized by (i) its semantics, which expresses the different dimensions of coupling among interacting peers, and (ii) its API, with a set of primitives expressed as functions provided by the middleware.

The *request-reply* pattern is commonly used for Web Services and followed by popular middleware protocols such as HTTP, XMPP, etc. A client interacts directly (without intermediate components) with a server either by direct messaging (one-way) or through remote procedure calls (RPC). Request-reply protocols usually support both synchronous and asynchronous interactions. In turn, the *publish-subscribe* pattern is commonly used for content broadcasting. Middleware protocols such as MQTT and AMQP, APIs (e.g., JMS) and message brokers such as RabbitMQ, EMQx, and Mosquitto follow this pattern. Multiple publisher-consumer peers interact via an intermediate broker. Consumers subscribe to a specific filter (e.g., topic-based filters) on the broker while publishers produce events to that filter, whereas consumers receive events in a FIFO order. Publish-subscribe protocols commonly support asynchronous interactions. In Section 3.2, we provide an overview of existing protocols that can be classified into the request-reply and publish-subscribe patterns.

PIS are characterized by diverse entities (devices, systems, users, etc.) that are pervasively inserted into the environment and provide context information about this environment. These entities are also inherently mobile, i.e., they may be present in the surroundings at a given instant of time and no longer be there at another one, and they may be unknown a priori at design time. Such characteristics lead the communication in a PIS to be preferably loosely coupled due to the inherent dynamicity of interaction among the system constituents and scale well upon the many entities envisioned in PIS environments. In this perspective, Biegel and Cahill (2015) especially advocate using an *event-based pattern* (Bacon et al., 2000) in PIS middleware as a means of providing asynchronous communication in a many-to-many, loosely-coupled interaction among the distributed application components.

2.7 Enabling interoperability

It is essential to tackle heterogeneity across multiple layers to enable interoperability between IoT devices and other PIS components. For instance, in the logistic chain traceability scenario, a shipment may provide information regarding its state through the following application layer operation: `get_shipment_state (id)`. However, a carrier may require the shipment status via `query_shipment (shipment_id,`

`state)`. Such issues at the application layer can be qualified as semantic heterogeneity issues. Ensuring end-to-end data consistency is one of the goals of semantic interoperability. There are two basic solutions for achieving semantic interoperability between two IoT devices. The first solution is a one-to-one model mapping. Another more suitable approach is to use shared data meta-models that can be used to unambiguously define the meaning of terms in existing models, such as ontologies. In Section 3.3, we discuss some existing semantic interoperability approaches in the literature.

Semantic interoperability ensures mapping between diverse data models employed by IoT systems. However, this alone does not make the interacting devices fully interoperable. Different APIs and data representations and primitives used by IoT devices must be mapped with each other at the middleware layer. Solving the middleware interoperability issue is challenging, mainly due to the fast development of protocols and APIs. Existing efforts address the middleware interoperability issue by relying on service-oriented architectures (SOA), IoT gateways, cloud computing platforms, and model-driven engineering. In Section 3.3, we discuss some middleware interoperability approaches in the literature.

2.8 Security and privacy

To promote the user acceptability of new IoT-enabled PIS applications, it is essential to provide mechanisms to ensure the privacy of users and the protection of the handled data. With the heterogeneity and amount of connected things and the unprecedented amount of collected data, security and privacy are no longer an option in PIS. They should be enforced throughout the entire software life cycle. PIS middleware is the right layer to intercept the information flow of applications and integrate security and privacy mechanisms. Such mechanisms can then benefit all applications by default, with the possibility to configure some specific business rules to take into account applications needs.

Security corresponds to *the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization* (ISO/IEC 25010, 2011). More specifically, cybersecurity is about ensuring three properties of information, services, and systems, namely confidentiality, integrity, and availability. Securing an information system means preventing an unauthorized entity from accessing information, services, and systems, modifying them, or making them unavailable. *Privacy* can be thought of as the confidentiality of the relationship between people and data. Therefore, it is important to notice that privacy can be guaranteed only when a security strategy is enforced in an end-to-end way. While relying on cryptographic primitives and protocols, privacy protection involves its own properties, techniques, and methodologies.

Cavoukian and Dixon (2013) recommend aligning seven principles for both security-by-design and privacy-by-design. These principles are: (i) proactive and

preventative, not reactive and remedial, to anticipate and prevent invasive events before they happen; (ii) default setting as no action should be required on the part of individuals for their protection; (iii) embedded into the design, not bolt after the fact; (iv) a positive-sum, not zero-sum but full functionality by accommodating all legitimate interests and objectives; (v) an end-to-end approach, by ensuring secure life-cycle management of information with confidentiality, integrity, and availability of all information for all stakeholders; (vi) visibility and transparency, by keeping IT systems' internal parts transparent to users and providers and by following open standards; and (vii) respect for the user in a user-centric approach to protecting the interests of all information owners.

PIS technology is still in its infancy and does not have utterly standardized security and privacy requirements (Chaudhuri and Cavoukian, 2018). Alhirabi et al. (2020) recommend using threat modelling techniques during the design stage, like STRIDE for security threats and LINDDUN for privacy threats. The STRIDE framework (Howard and Lipner, 2006) is an acronym for *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service*, and *Elevation of Privilege*. The LINDDUN framework (Deng et al., 2011) is an acronym for *Linkability*, *Identifiability*, *Non-repudiation*, *Detectability*, *information Disclosure*, *content Unawareness*, and *policy and consent Non-compliance*.

Even though PIS mainly rely on an event-driven data reporting method (see Section 2.6), there may be situations when a query-driven approach is more relevant to get insights about some phenomenon at a given time. For instance, a query would get a particular set of sensor readings satisfying some condition. Data query privacy (López et al., 2017) is hence an important requirement of PIS in order to reduce the risk of exposing sensitive information to attackers when issuing queries.

In the logistic chain traceability system example, the collected data should be kept confidential and not be transferred to or stored by untrustworthy third parties. Anonymity or pseudonymity should also be enforced so that untrustworthy third parties can distinguish location information from fake locations. These are just examples of some issues. Many more security and privacy aspects should be considered in a PIS middleware all along the data life cycle and at all the system architecture.

2.9 Scalability

The IoT paradigm calls for exchanging data among dynamic, heterogeneous sensors and client applications at unprecedented scales. We follow the framework from Duboc et al. (2007) for characterizing the scalability of PIS middleware. For instance, when considering an IoT-based solution, the scaling dimensions, which represent the scaling aspects, are the number of queries per second and the number of machines in the cluster. The non-scaling variables are the network conditions (e.g., available bandwidth). The dependent variables, which represent the aspects of the system behavior affected by changes in the scaling dimensions, are the response time for a query, bandwidth usage, and cluster load. In this first example, the requirement

can then be formulated as follows: "the studied system shall scale with respect to latency" because it can maintain a maximum given response time as the number of requests per second scales by varying the number of machines in the cluster. In another architectural style, such as a highly-distributed publish-subscribe system for the PIS middleware, the scaling dimensions shall include the number of intermediary entities (i.e., brokers of the overlay network) that route data from sensors to client applications. We shall then measure the total resource consumption for filtering data records through the multiple brokers from the sensor to the client application.

When considering the logistic chain traceability illustrative application domain, architects may differentiate PIS systems deployed in relatively small areas mainly managed by one administrative entity, such as merchandise warehouses or ocean liners, from more extensive areas with many stakeholders, e.g., in port cities. In the former scenario, IoT solutions, including ones enhanced with cloud computing, may be appropriate. In the latter configuration, more distributed, decoupled solutions involving several brokers along with distributed routing and filtering might be required.

2.10 Energy efficiency and energy-awareness

Penzenstadler (2015) point out that new quality attributes have recently been studied by the research community in the objective to keep systems sustainable. In the past, resource utilization mainly referred to the efficiency of the use of the available processing, storage, and network. For energy-efficiency purposes, the resource to be monitored is the energy consumption.

While *energy efficiency* means using less energy to perform a given task, *energy-awareness* represents knowing the energy consumption for a given task. The middleware can use energy-awareness to reduce energy consumption through energy-saving strategies, e.g., protocol, scheduling and the volume of exchanged data. Energy-awareness can also be shared with upper layers of applications. Applications may adjust their behavior for energy-saving purposes, e.g., reducing some requirements to remain within the limits of a given energy budget. Applications can also share energy consumption reports with end-users who could adapt their usage based on energy consumption knowledge. Indeed, energy-awareness is expected to have a positive impact in terms of energy efficiency (Hassan et al., 2009).

In the logistic chain traceability system example, to achieve energy efficiency, the PIS middleware could: (i) at design time, choose the most energy-efficient consensus algorithm for sharing securely and transparently data between the stakeholders (Sedlmeir et al., 2020); (ii) at runtime, reduce the volume of exchanged data by filtering data based on their content or minimizing the frequency of data transmissions (de Oliveira et al., 2020). For energy-awareness, the middleware could adapt the frequency of data transmissions to keep energy consumption above a certain level of energy budget or transmit energy consumption information to the applica-

tion level, for example, to inform the end-user about the consumption of the energy budget.

PIS middleware should integrate architectural tactics for energy efficiency, e.g., energy monitoring, resource allocation, and resource adaptation (Paradis et al., 2021). It is reasonable seeing middleware as the good level for integrating energy management strategies due to its operation at the protocol level and high reusability (Noureddine et al., 2013) Additionally, middleware should provide energy-awareness mechanisms (Verdecchia et al., 2021) that allow future PIS providers to master energy consumption.

3 State of the art on middleware supporting PIS requirements

In Section 2, we have identified and defined the requirements for PIS middleware in the context of the IoT. Context-awareness state-of-the-art is covered in Chapter of this book. In this section, we present only the state of the art concerning the most pregnant requirements in the context of the IoT.

3.1 QoC management

Even though the management of the quality of context data has long been recognized as a requirement of PIS and context-aware applications (Buchholz et al., 2003), only a few middleware actually provide the necessary support for QoC. We herein present some recent initiatives and summarize the provided mechanisms.

The ContextNet middleware (Endler and Silva, 2018) integrates QoC management through the Context Data Distribution Layer (CDDL) (Gomes et al., 2017a). A set of QoC parameters is available, including accuracy, measurement time, age, completeness, and numeric resolution. ContextNet targets the Internet of Mobile Things (IoMT) and takes dynamicity into account at different levels. QoC parameters may also exhibit dynamic variability (they oscillate over time), and CDDL can monitor the variation of a given QoC parameter. CDDL also offers filtering based on context data and their QoC metadata.

The LAURA architecture (Teixeira et al., 2020) was designed to support the deployment of decoupled IoT applications. LAURA provides a fog layer that plays the role of an intermediate between applications and the network or sensor nodes and can be regarded as a middleware. This fog layer, still under development, is designed to filter or aggregate data received from the physical layer to prevent unnecessary or poor quality data from being sent to upper layers. QoC parameters are associated with the sensed data, allowing user applications to verify the context data's usefulness or temporal relevance. QoC-based filtering and aggregation are seen as important features of LAURA.

Jagarlamudi et al. (2021) proposed a Service Level Agreement (SLA) template integrating a QoC-aware mechanism, called the Relative Reputation (RR), to select context providers with high RR values. The QoC evaluator generates the RR unit representing the match between QoC outcomes and QoC requirements. A mechanism of penalties also exists to indicate the applicable penalties with each QoC indicator's degradation in the context response compared to its guarantees.

3.2 Protocols for multiple interaction patterns

As mentioned in Section 2.6, PIS middleware platforms leverage communication protocols upon different interaction patterns. We herein provide an overview of existing middleware-based IoT protocols. These protocols offer middleware primitives that aim to facilitate the development of IoT applications that include resource-constrained IoT devices. Karagiannis et al. (2015) compare the most promising IoT middleware protocols (more specifically, the ones mentioned here). Even though there are multiple IoT protocols, no single protocol has been adopted yet for IoT system development. This is mainly because the IoT is too diverse, including multiple data formats and (possibly highly) resource-constrained devices.

Protocols such as DPWS, OPC UA, CoAP, and XMPP have been introduced to support data exchange among peers based on the request-reply interaction pattern. OASIS introduced DPWS (Zeeb et al., 2007) in 2004 as an open standard, and it is suitable for supporting large-scale deployments and mobile devices. Nevertheless, the induced protocol overhead is noticeable and requires a large amount of RAM. The OPC Foundation designed OPC UA (Mahnke et al., 2009) in 2008 to target resource-constrained devices, but it implies a large payload unsuitable for IoT applications. IETF designed CoAP (Shelby et al., 2014), a lightweight protocol that supports highly resource-constrained devices and the delivery of small message payloads. Finally, XMPP (Saint-Andre, 2011) is now a suitable protocol for IoT real-time communications, even though it uses XML data formats that create a significant computational overhead.

The publish-subscribe interaction pattern is an alternative to request-reply and offers time and space decoupled interactions. The Sun Microsystems' JMS standard has been one of the most successful asynchronous messaging technologies available by defining an API for building messaging systems. DDS (OMG, 2015) is a messaging protocol designed for brokerless architectures and real-time applications. AMQP (OASIS, 2012) is another messaging protocol designed to support applications with high message traffic rates. To support highly resource-constrained devices, MQTT (Banks and Gupta, 2014) offers a publish-subscribe centralized architecture, but its performance decreases significantly when sending large message payloads. WebSockets (Fette, 2011) were introduced to support real-time full-duplex interactions using only two bytes of overhead in message payloads.

3.3 Enabling Interoperability

Different data representations and APIs among IoT devices, platforms, and applications can be mapped with each other at the middleware layer. However, this alone does not make the interacting peers fully interoperable. There are indeed incompatibilities of IoT devices at the application layer, e.g., operation/resource names, data semantics, etc.

Ontologies (Gruber, 1993) provide a common model for annotating content and thus help systems to interoperate. We review well-known ontologies for general sensor modeling. The W3C Semantic Sensor Network (SSN) ontology (Compton et al., 2012) presents a vocabulary to describe sensors and their observations, actuators, and their association to features of interest. Its central building block is the SOSA (Sensor, Observation, Sample, and Actuator) ontology (Janowicz et al., 2019), a standalone light-weight ontology that offers the core vocabulary for the descriptions. The Smart Appliances REFerence (SAREF) ontology (Daniele et al., 2015) follows a similar design to describe concepts required by smart applications. In SAREF, devices make measurements related to properties of interest (similar to sensors making observations in SSN). Depending on the application under development, developers must use the appropriate ontology. For example, the SAREF ontology is commonly used to model information of appliances in smart homes.

Several approaches to bridge middleware-based protocols have been proposed concerning APIS, protocols, and data representations, e.g., the QEST broker for CoAP and RESTful APIs (Collina et al., 2012), HTTP-CoAP proxy (Castellani et al., 2012), and Ponte for REST, CoAP, and MQTT (Banks and Gupta, 2014). These approaches implement one-to-one mappings between existing protocols. Despite the simplicity, this is highly inefficient due to the vast development of IoT protocols. Negash et al. (2015, 2016) introduces the Lightweight Internet of Things Service Bus (LISA) for tackling IoT heterogeneity. Derhamy et al. (2017) introduced a protocol translator that utilizes an intermediate format to capture all protocol-specific information. XWARE (Roth et al., 2018) implements mediators to translate messages of IoT protocols by using an intermediate format. Finally, Georgantas et al. (2013) extended the Bouloukakis et al. (2019)'s work to deal with IoT heterogeneity using software abstractions and code generation.

While the above approaches considerably reduce the development effort, they do not consider semantic layer incompatibilities prevalent in the IoT. IoT platforms such as SemIoTic (Yus et al., 2019) provide end-to-end IoT interoperability in smart buildings by leveraging the SSN/SOSA ontologies and mediating adapters. In addition, it leverages the middleware-based interoperability approach that is further presented in Section 4.3.

3.4 Security and privacy

In a comparison of 50 context-aware computing research projects, Perera et al. (2014) identified that only 11 projects (about 20%) provided security and privacy solutions. More recently, Alhirabi et al. (2020) reviewed the evolution of design notations, models, and languages that facilitate capturing the non-functional requirements of security and privacy. The majority of the requirement engineering efforts are focused on security. Among the 47 design notations analyzed in their study, security is supported by more than half (32 notations out of 47), while only three notations cover privacy. Even though a by-design approach has long been recommended for both security and privacy Cavoukian and Dixon (2013), it is still not sufficiently put into practice by developers. Aljeraisy et al. (2021) highlight that there is still a relevant gap between legislation and design patterns that can help to translate and implement them.

Aljeraisy et al. (2021) analyzed data protection laws used across different countries, namely the European General Data Protection Regulations (GDPR), the Canadian Personal Information Protection and Electronic Documents Act (PIPEDA), the California Consumer Privacy Act (CCPA), the Australian Privacy Principles (APPs), and the New Zealand's Privacy Act 1993. The authors then retained the fundamental principles and individuals' rights to define the Combined Privacy Law Framework (CPLF) by eliminating duplication. Finally, they mapped CPLF with privacy-by-design (PbD) schemes (e.g., privacy principles, strategies, guidelines, and patterns) previously developed by different researchers to investigate the gaps in existing schemes. The results of this extensive study helped to identify where new privacy patterns should be defined. More than 70 privacy patterns have already been proposed in the literature (Colesky et al., 2022; Kargl et al., 2022) and they are a relevant, concrete mechanism to handle data usage and protection in a specific context. However, some principles and rights of CPLF are not achieved by any existing privacy pattern and call for further research.

While security and privacy research is very active, its integration into operational middleware is still limited. Fremantle and Scott (2017) analysed 54 IoT middleware frameworks and observed that they address security and privacy in very different ways. A majority of these middleware frameworks provide access control and authentication mechanisms, and others focus on providing protection for the content shared on the network. However, very few middleware frameworks support a sufficient coverage of the features required to support security and privacy for PIS.

3.5 Scalability

Without middleware, i.e., when applications directly obtain IoT data from sensors, existing coupling significantly hampers the system's scalability. Therefore, as formulated by Bellavista et al. (2012), PIS middleware architectures are classically first organized according to the following question: is the middleware centralized or

decentralized? The centralized approach includes deploying middleware on a single host or cloud. The second approach has two subcategories depending on whether the distribution is hierarchical or not. Consequently, the basic solutions for scaling up follow these three classes of solutions.

The architectures of the first class of solutions have been referred to as Web of Things (Delicato et al., 2013) or, more recently, Cloud of Things (Dias et al., 2020). Scalability issues arising in these centralized architectures concern the complex processing of a huge quantity of data with many clients either producing or requesting data. Cugola and Margara (2012) surveyed solutions for complex event processing and stream processing.

The second class of solutions dealing with scalability targets this requirement at a local scale, a.k.a. localized scalability (Satyanarayanan, 2001b). A collection of small clouds, i.e., cloudlets, typically are brought to lower latencies between so-called co-located clients: these smaller clouds are physically distributed to form smaller groups of clients. This architectural style corresponds to what we know as fog computing. Perera et al. (2017) surveyed such solutions for smart cities.

To target scalability at a global scale, an architecture based on publish-subscribe is preferred as it favors decoupling. Eugster et al. (2003) distinguish three forms of decoupling, namely space, time, and synchronization decoupling. In this approach, some clients publish IoT data while others consume these data. As surveyed by Bellavista et al. (2012), a first set of solutions organize an overlay of brokers responsible for routing IoT data from producers to subscribers. Producers push data to their access broker, and brokers forward them to the consumers that have subscribed to these data. A data model and a filtering model define the non-scaling variables of publish-subscribe solutions: roughly speaking, topic-based filtering with opaque data scales better than content-based filtering with structured data or semi-structured data. In addition, the diameter of the overlay network of brokers is the other non-scaling variable. Kermarrec and Triantafillou (2013) surveyed a second set of solutions targeting non-broker-based routing and using topic-based filtering. These solutions are constructed as peer-to-peer systems: peer nodes simultaneously play the three roles, namely publishers, subscribers, and routers.

Finally, note that broker-based PIS middleware protocols such as AMQP and MQTT are topic-based and cloud-based, but without complex event processing or streaming. This is precisely the role of recent works such the one of Luckner et al. (2014), and of industrial platforms such as AWS IoT Core[3], Google IoT Core[4], Microsoft Azure IoT Hub[5], and FIWARE[6], to add complex event processing and streaming to publish-subscribe middleware standards. These platforms are proof, if any were needed, of the interest of major operators in working to integrate scalability into PIS middleware.

[3] https://aws.amazon.com/iot-core/

[4] https://cloud.google.com/iot-core

[5] https://docs.microsoft.com/azure/architecture/reference-architectures/iot

[6] https://www.fiware.org/

3.6 Energy efficiency and energy-awareness

Middleware has recently explored some strategies for the IoT for energy efficiency and energy-awareness purposes. The most used strategy for energy efficiency is network adaptation. Network adaptation refers to introducing new protocols, modifying existing ones, and making network optimizations. Akkermans et al. (2016) proposed adapting a publish-subscribe middleware by adding a layer between the broker and the client applications to send notifications via IPv6 multicast rather than using several point-to-point messages. Kalbarczyk and Julien (2018) proposed Omni, a device-to-device middleware with periodic adaptive discovery of neighbor devices using lightweight discovery mechanisms in wireless local area networks. Discovered devices are only connected when data needs to be transferred, and the communication technology adapts both to the network energy efficiency and the volume of data.

Task offloading stands for using the network to transfer software components to other locations. For example, an application running on a mobile phone could send data to a server in a cloud or to another computer in its vicinity for data processing purposes. Several authors such as Aazam et al. (2020), Pasricha (2018), Song et al. (2017), Ivarez-Valera et al. (2019), and Shekhar et al. (2019) proposed middleware to offload software components to other nodes in the cloud or the fog as a means of saving energy on the source nodes. All these proposals show that task offloading has a benefit in terms of energy consumption for at least one of the nodes of the system.

The data filtering capability offered by some middleware proposes processing data to reduce the number or the size of messages according to specific criteria. Adaptive sampling and adaptive filtering (Giouroukis et al., 2020a) are two techniques that have emerged over the last decade. These techniques dynamically reconfigure rates and filter thresholds to trade-off data quality against resource utilization. de Oliveira et al. (2020) proposed a data stream processing workflow to be deployed at the network's edge to perform data cleaning tasks.

Another strategy used by middleware is to temporarily reduce the activity of some nodes to reduce the infrastructure energy consumption. This strategy is used for a time in data centers. For example, Binder and Suri (2009) presented a dispatch algorithm to concentrate services on a reduced number of servers so that they put inactive servers in a sleeping mode to save energy in the data center. In the context of the IoT, this strategy is used as well and is known as active node selection. For example, Cecchinel et al. (2019) proposed determining an optimal configuration of sensors towards extending their battery life. (Sarkar et al., 2016) proposed to reduce interactions among the nodes of a wireless sensor network and hence the network's energy consumption. The data stream processing workflow proposed by de Oliveira et al. (2020) also includes active node selection. Active node selection can hence reduce energy consumption on some of the nodes of a PIS.

The second requirement concerning energy introduced in Section 2.10 is energy-awareness. The energy-awareness may be provided at the middleware or the application level (i.e., knowledge shared through middleware abstractions with the application components). At the former level, energy-awareness may be used to con-

strain the system's energy consumption through an energy budget configuration. For example, (Padhy et al., 2017) proposed a middleware to minimize the total energy consumption of an IoT application while ensuring that the requested accuracy is met. The middleware intends to find the sensors that consume the minor energy while satisfying the sensing requirements and maximizing the overall accuracy under an energy budget. For the latter level, we found some examples where applications express energy requirements (e.g. (Song et al., 2017)) for deployment purposes. However, middleware does not usually expose energy consumption to upper layers.

Energy consumption is a recent concern for the community working on IoT middleware. Some middleware has mainly handled energy efficiency to reduce energy consumption only on some systems parts. We noticed a few middleware proposals providing energy-awareness to the upper layers.

4 PIS middleware proposals

We have been working on middleware for the IoT and PIS for some years. Different software is available in open source (Conan et al., 2022; Bouloukakis et al., 2022; Gomes et al., 2017b) and some of these proposals are presented below. Table 1 summarizes the requirements tackled by each of them.

4.1 QoC management with QoCIM and processing functions

Based on the QoC criteria most frequently mentioned in the literature, it is possible to notice that no criteria can respond to all the needs of applications, each having its own method for computing the quality of context information. We have then focused our attention on realizing a model able to represent any type of QoC criteria. This resulted in QoCIM[7] (Quality of Context Information Model) (Marie et al., 2013), a meta-model dedicated to modeling QoC criteria and enforcing expressiveness, computability, and genericity of QoC management. QoCIM offers a flexible ideology, i.e., it defines a basis to design and represent any QoC criterion instead of providing a predefined list of supported QoC criteria. With QoCIM, a given QoC criterion can also be built upon other primitive or composed QoC criteria.

QoCIM is complemented with the specification and implementation of a set of functions for processing context information and its QoC metadata. The goal of these processing functions is to provide the developers of PIS with middleware programming facilities to process context information together with its associated QoC metadata efficiently. The functions manage three types of data: (i) context information sensed and collected from different sources; (ii) QoC metadata modeled with QoCIM, each piece of QoC metadata corresponding to an instance of a QoC

[7] QoCIM is part of the M4IoT platform: https://www-inf.it-sudparis.eu/m4iot/

indicator, and (iii) message encapsulating a piece of context information associated to a list of QoC metadata. There are functions for aggregation, filtering, inference, and fusion of context information with QoC metadata. These functions can be configured to determine what computing method to use and to indicate the number of messages to be taken as input. The configurability of the functions is based on a declarative solution.

The *aggregation* function applies an aggregation operator onto a list of messages. The result is a message with the same abstraction level. The choice of the aggregation operator (arithmetical average, for instance) is specified in a configuration file. There is also a distinction between temporal aggregation and spatial aggregation. The former handles information coming from a single context source and produced during some time. The latter handles information coming from several context sources that periodically produce the same type of context information. The *filtering* function analyzes the message and decides to remove it or not, but the content of the message itself is never modified. The *inference* function applies an inference operator onto a list of messages. The result is only one message with a higher abstraction level. The *fusion* function executes a set of functions sequentially. The result is a list of messages with a higher abstraction level.

QoC management must take place throughout the whole chain of processing context information. A declarative programming approach allows qualifying context information and self-adapting QoC management due to potential physical limitations of the processing entities (Marie et al., 2016).

4.2 muDEBS

Distributed-based event systems (DEBS) for broad IoT face unprecedented scales regarding the volume of exchanged data, number of participants, and communication distance. As many brokers may be involved, a high amount of messages may be exchanged when installing subscription filters and, most importantly, when routing numerous events from producers to consumers. muDEBS[8] (Conan et al., 2017) take advantage of the inherently heterogeneous nature of broad IoT systems to control and limit the amount of exchanged data. Some sources of heterogeneity, such as geographical and group membership heterogeneity, may delimit visibility scopes for data distribution, with notifications being visible only in certain scopes. More precisely, Fiege et al. (2002) define scope as *an abstraction that bundles a set of clients (producers and consumers) in that the visibility of notifications published by a producer is confined to the consumers belonging to the same scope as the producer; a scope can recursively be a member of other scopes*. In muDEBS, filtering is impacted by the visibility of notifications that are analyzed according to several dimensions of scopes. A client advertises or subscribes by providing a filter tagged with a set of scopes, with at most one scope per dimension, e.g., interest in geographical scopes

[8] muDEBS is part of the M4IoT platform: https://www-inf.it-sudparis.eu/m4iot/

or areas belonging to end-users scopes or groups. A notification is visible to a client if it is visible in all the dimensions. In summary, muDEBS targets scalability by scoping the distribution of data between producers and consumers.

IoT data can be exploited by pervasive applications to detect the users' current situation and provide them with the relevant services corresponding to their precise needs. The threats to the users' privacy appear more clearly and Chabridon et al. (2014) have shown that QoC and privacy are closely related and must be addressed together in order to find a workable solution. As a first step, Lim et al. (2015) identified models for a first set of attributes to be specified in privacy policies, namely purpose (intention of use), visibility (who has access), and retention (for how long data may be retained). Following these models, IoT producers specify privacy requirements and QoC guarantees in producer context contracts that are then registered in muDEBS as XACML policies[9]. On their side, IoT consumers express their QoC requirements and the privacy guarantees that they are committed to fulfilling in consumer context contracts, mentioning at least for what purpose they are requesting access to some specific IoT data. Privacy guarantees take the form of ABAC information registered with the subscription filters. QoC guarantees and requirements are expressed by following the QoCIM model (see Section 4.1). As a second step, Denis et al. (2020) studied confidentiality under the semi-trusted broker assumption in which brokers are considered honest-but-curious, i.e., brokers route the publications to the interested consumers, but they can make use of the data for their own interest. More precisely, confidentiality concerns encompass (i) part or all of the constraints of the subscriptions, (ii) part or all the information in the publication that is used for routing against subscriptions, and (iii) the payload of the publications. The solution proposed in muDEBS adapts an existing attribute-based encryption scheme and combines it with data splitting, a non-cryptographic method called for alleviating the cost of encrypted matching. Data splitting enables forming groups of attributes sent apart over several independent broker networks. It also prevents the identification of an end-user, and only attributes are encrypted to prevent data leakage.

4.3 DeX Mediators

IoT devices employ middleware-layer protocols such as MQTT, CoAP, ZeroMQ, and more to interact with each other. These protocols support different Quality of Service (QoS) semantics. They define multiple data-serialization formats (e.g., JSON, XML, protobuf, etc.) and different payloads suitable for constrained or healthy devices and follow different interaction patterns such as request-reply and publish-subscribe. IoT systems include heterogeneous IoT devices employing any of those protocols. In many cases, new heterogeneous IoT devices may be added to an IoT system in an on-demand fashion. For instance, in the logistic chain traceability scenario, IoT devices

[9] https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

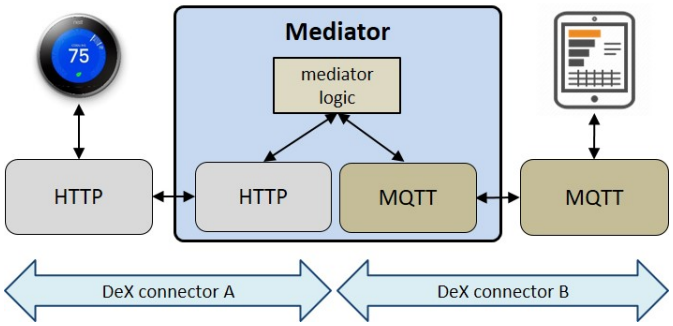

Fig. 1 Enabling data exchange via mediators.

in the shipment must interact with the services of the IS that dynamically collect information for decision-making purposes. Therefore, generic, automated solutions must enable data exchange in such IoT systems.

The Data eXchange Mediator Synthesizer (DeXMS)[10] (Bouloukakis et al., 2019) addresses the heterogeneity of IoT devices and services by synthesizing software mediators. As depicted in Fig. 1, DeXMS relies on the Data eXchange (DeX) API, which implements POST and GET primitives for sending/receiving messages using existing IoT protocols such as CoAP, MQTT, XMPP, etc. In the illustration, the mediator converts temperature data from a package (in JSON format through the HTTP protocol) to be received from an IS dashboard (in XML format through the MQTT protocol). Considering a set of heterogeneous IoT devices that have to interconnect with devices deployed in an IoT system, DeXMS accepts their input/output data representation models as input and synthesizes the required mediators. Based on the requirements defined in Section 2, DeXMS provides a semi-automated manner to tackle interoperability among devices employing middleware-layer protocols (classified to diverse interaction patterns). Regarding the application-layer, DeXMS enables developers to manually perform data mappings between applications semantics. More details on DeXMS can be found in the work of Bouloukakis et al. (2019).

4.4 QoDisco

A pervasive context encompasses a distributed plethora of heterogeneous resources (sensors, actuators, services) with different functionalities and communication protocols. In this scenario, a well-known challenge for both machines and users is finding, selecting, and using these resources. Discovering services play a significant role in addressing this issue by enabling clients (applications, middleware, end-users) to

[10] https://gitlab.inria.fr/dexms

retrieve available resources based on complex search criteria considering contextual information essential in a pervasive environment.

QoDisco[11] is a QoC-aware federated discovery service supporting multiple-attribute searches, range queries, and synchronous/asynchronous operations. It encompasses an ontology-based information model for semantically describing resources, services, and QoC-related information. QoDisco is structured upon a distributed architecture composed of a federation of autonomous repositories cooperating with each other to perform data and service discovery tasks. It provides an API to perform discovery tasks in such repositories, and each repository provides operations for querying and updating records. Clients are responsible for semantically annotating resource data (such as the ones provided by sensors) by using the concepts of the QoDisco information model. When receiving a discovery request, QoDisco searches for resource descriptions or data stored in the available repositories, thereby hiding the heterogeneity.

The semantic description of resources defined by the QoDisco information model relies on: (i) the SAN ontology (Spalazzi et al., 2014), an extension of the W3C's SSN ontology (Barnaghi et al., 2011) that provides concepts, attributes, and properties to model both sensors and actuators; (ii) part of the SOUPA ontology (Chen et al., 2005) aiming at including location-related concepts to describe spatial locations of entities in terms of latitude, longitude, altitude, distance, and surface, as well as symbolic representations of space and spatial relationships; (iii) the OWL-S ontology for semantically modeling services exposed by the resources; and (iv) part of the QoCIM meta-model (Marie et al., 2013) to describe QoC-related concerns (see Section 4.1). This information model supports the QoC management requirement and tackles data format heterogeneity using ontologies.

Due to the dynamic context in which the IoT resources operate, QoDisco handles both synchronous calls and asynchronous notifications. The former relies on request-reply interactions towards providing resource information at the moment of the search. The latter is based on publish-subscribe interactions to notify clients in case of resource removal, insertion, or update. More details on QoDisco can be found in the work of Gomes et al. (2019).

4.5 IoTVar

IoTVar[12] is a middleware that provides developers with abstractions for IoT variables. From a variable declaration, IoTVar automatically discovers matching data-producer objects and transparently deals with updates to these variables thanks to transparent interaction with IoT systems. IoTVar offers an abstraction level to interact with virtualized sensors. It drastically minimizes the number of lines of code to be written

[11] https://github.com/porfiriogomes/qodisco

[12] IoTVar is part of the M4IoT platform: https://www-inf.it-sudparis.eu/m4iot/

by the client application developer to obtain up-to-date sensor data from several hundreds of lines of code to a single dozen.

The IoTVar architecture has been designed to integrate new IoT platforms and IoT systems. For this purpose, it exposes an interface that can easily be implemented for integrating with new platforms. The architecture was focused not only on developing the IoT applications but also on expanding the middleware to support multiple IoT platforms. IoTVar is currently integrated with FIWARE, OneM2M (oneM2M Partners, 2019), and MuDEBS (Conan et al., 2017) IoT platforms. More details on IoTVar can be found in the work of Borges et al. (2019).

IoTVar responds to some of the previously mentioned PIS requirements. The multiple IoT platforms supported by IoTVar have different data models and API access and use different protocols to retrieve sensor data. For the sake of interoperability, IoTVar includes data unmarshallers, IoT protocols handlers, and IoT API handlers, as well as it supports both publish-subscribe and request-reply interaction patterns to be chosen according to efficiency considerations. IoTVar also supports application development by providing an API accessible through code in the Java programming language and enabling IoT developers to access sensor data easily. The developer will declare environment variables by providing a simple IoT variable declaration. Those IoT variables will be automatically updated.

5 Open challenges for future PIS middleware

Next-generation PIS are deployed at an unprecedented scale with components on connected mobile devices and remote servers in cloud and fog intermediaries. In this context, handling requirements from an end-to-end perspective is challenging. At the same time, mastering requirements such as privacy and sustainability become essential and even more complex. This section highlights some open challenges that can commission research on future PIS middleware.

5.1 Enabling end-to-end interoperability

As mentioned in Section 3.3, existing middleware approaches enable interoperability at each layer (i.e., application, middleware, network) independently. However, enabling IoT interoperability requires introducing end-to-end approaches. This is challenging due to: (i) the difficulty to select a unique data model and IoT protocol to develop cross-domain IoT applications, which results in composing multiple IoT protocols and data models; (ii) the existence of numerous IoT protocols to support diverse types of devices (healthy/constrained/tiny in terms of resources); (iii) the diversity of data models to cover multiple application domains (healthcare, autonomous driving, etc.); and (iv) end-to-end approaches are usually developed for specific application domains (e.g., smart buildings) and it is difficult to adapt them

to other domains. Therefore, advanced end-to-end interoperability approaches must be introduced while considering those challenges.

5.2 PIS adaptive middleware

Previous research on the so-called adaptive middleware can indeed contribute to support dynamic adaptation in PIS, including proposals on context-aware applications (Huebscher and McCann, 2006), ubiquitous computing (Yau and Karim, 2004), wireless sensor networks (Portocarrero et al., 2016), IoT (Cavalcanti et al., 2021), cyber-physical systems (García-Vallis and Baldoni, 2015), and cloud computing (Rafique et al., 2017). *Adaptive middleware* can be defined as a kind of middleware that enables modifying the behavior of a distributed application in response to changes in requirements or operating conditions (Sadjadi and McKinley, 2003). To the best of our knowledge, the literature has still not explored building adaptive middleware to support PIS and provide these systems with dynamic adaptation capabilities.

Designing adaptive middleware needs to consider some 5W1H (What? Who? Where? When? Why? How?) issues typically associated with self-adaptive software systems (Salehie and Tahvildari, 2009). It is necessary to understand (i) the need for adapting the middleware to changes in application requirements and context, (ii) the time at which the adaptation needs to be triggered, whether proactively or reactively, (iii) the extent of the adaptation in terms of how many components should be subjected to the adaptation, and (iv) how the adaptation actions can be executed and implemented (Rosa et al., 2020). Designing PIS middleware with adequate support for dynamic adaptation should hence cope with these issues.

5.3 Support to develop PIS relying on middleware

Middleware platforms are well-acknowledged to leverage the development of distributed applications, but this does not seem to be the case for PIS yet. Indeed, there is still no available programming model for PIS relying on middleware while coping with the characteristics of this class of systems. (Biegel and Cahill, 2015) highlight that existing solutions and approaches in the literature are not currently able to address the requirements for PIS middleware comprehensively, but rather only a subset of them. The authors also point out the significant effort necessary from application developers to deal with these requirements, an issue that hampers a broader adoption of PIS middleware in industrial settings. Therefore, a programming model able to ease the development of PIS relying on middleware is desirable.

The development of PIS relying on middleware faces other challenges. On the one hand, the proliferation of physical devices and platforms to support PIS may lead these systems to become primarily vendor/platform- and hardware-specific (Taival-

saari and Mikkonen, 2017). This may also pose difficulties in finding the most suitable solution (or set of solutions) for a specific application and deepen users' lack of experience and knowledge on understanding the implications for current and future needs. PIS middleware should hence enable applications to benefit from using different devices and platforms while relieving developers from dealing with their specificities through proper high-level abstractions.

5.4 Privacy and security

Security for PIS is still a significant challenge as attacks are relatively easy in an open, connected world. Many devices were not designed for security, and their high number increases the attack surface, as well as their integration within the Internet that exposes them to numerous potential attackers. We underline some specific areas where research challenges need to be addressed by PIS middleware in the short term: (i) the need for low-cost cryptography primitives suitable for devices with limited resources; (ii) security analysis of new low-power wireless wide area network technologies; and (iii) the need for frameworks and protocols to facilitate the development of devices where security is considered from the design stage.

Considering privacy, our connected world has allowed unprecedented growth in personal data collection practices, with intrusion in our private life. The lack of transparency, the fact that many services and devices behave like black boxes, and the lack of user control raise major research challenges to enable PIS middleware to enforce data protection and privacy patterns. In addition, robust anonymization, which effectively resists deanonymization attacks while preserving data utility, remains an open research topic.

With resource-constrained devices and sustainability objectives, resource consumption of security and privacy solutions is gaining importance. We consider that this also opens some new research directions where concerns for security, privacy, and sustainability can be addressed jointly in PIS middleware.

5.5 Context data sampling and filtering

As discussed in Section 3.5, many contributions exist that enable scaling PIS solutions deployed in Clouds. Among the next challenges, for scaling PIS deployed in highly distributed environments such as connected mobile devices and with fog intermediaries, the contextual data filtering module of a PIS middleware should strive to increase the system's scalability by controlling and reducing the amount of transmitted data. Giouroukis et al. (2020b) classify filtering techniques into (i) time-based, i.e., sending data is suppressed until certain time conditions become true, and (ii) change-based, i.e., sending data is suppressed as long as the contextual data are equal or similar to that previously transmitted. Of course, any combination

of time-based and change-based filtering techniques is possible. For example, in the illustrative logistic chain traceability system, some applications may request to receive location updates only if the new location is not identical to the previous one and if an interval of at least ten minutes has elapsed.

Adaptive sampling is, of course, closely related to adaptive filtering. For instance, tuning sensor sampling frequency enforces network usage optimization and can be performed according to the frequency of requests from deployed software applications. As another significant outcome, PIS middleware obtains a self-adaptive platform with an extended sensor battery life while ensuring good data quality and freshness.

Put together, selection-based filtering of publish-subscribe systems enables the system to limit dissemination to some scopes, contrary to system-wide scoping. Context-based filtering uses context data of different context dimensions to route IoT data at the application layer. In contrast, adaptive filtering enables the system to decide whether some IoT data are worth passing on intermediaries, depending on whether a sensor value is similar to previous values or evolves predictably. These issues are still not solved and are certainly a very fruitful area for future research.

5.6 PIS sustainability

In the last decade, the number of existing PIS has grown, coming with new facilities for the end-users and rising computer power demand. However, sustainability in IT is from now on a first-class concern for enterprises. This demand has to be taken into account by PIS middleware designers.

As seen in Section 3.6, many strategies have been proposed so far by middleware targeting energy efficiency. However, those strategies mainly target one of the components of the system. Considering energy efficiency at the scale of the whole system is still a challenge.

Even though middleware eases the task of application developers when dealing with energy efficiency, a developer may face difficulties in evaluating the energy consumption of the system. An important research direction to foster energy efficiency in PIS is providing energy-awareness at the middleware level. Some techniques such as static code analysis (Vekris et al., 2012) and profilers to detect software energy and performance bugs (Nistor and Ravindranath, 2014) have been proposed in the last years aiming at statically easing the identification of energy-consuming components. Energy awareness may also be provided at runtime through abstractions expressing energy requirements and evaluating energy consumption. These abstractions based on measures and energy consumption models have yet to be integrated in middleware. We believe that energy-awareness may significantly increase the efficiency of the systems as the awareness brings a broader view of where and how the many resources (CPU, network, energy, etc.) used by an application are behaving in terms of energy consumption.

6 Conclusion

In this chapter, we have considered PIS middleware in the context of the IoT. This middleware provides applications with an easy integration of context data collected from connected objects spread over the Internet. This context comes with new challenges and requirements. In addition to context-awareness, middleware should tackle scalability, privacy and interoperability and provide applications with new abstractions representing the physical environment and ensure the quality of the data that may be used for decision-making.

We have shown through the state of the art that middleware has proposed semantic interoperability for handling heterogeneities and large-scale publish-subscribe architectures to tackle scalability. However, while middleware has already enabled new kinds of PIS in various domains such as transport traceability, healthcare, and smart cities, the middleware community still faces new challenges, such as providing high-level programming model for PIS, supporting PIS dynamic adaptation, disseminating and filtering large volumes of data, end-to-end privacy and interoperability handling, as well as enabling the deployment of sustainable applications.

Acknowledgments

This work is a contribution to the Energy4Climate Interdisciplinary Center (E4C) of IP Paris and École des Ponts ParisTech, supported by 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02]. It has been partially funded by the "Futur & Ruptures" program from Institut Mines-Télécom, Fondation Mines-Télécom, and Institut Carnot.

References

Aazam M, Islam SU, Lone ST, Abbas A (2020) Cloud of things (cot): Cloud-fog-iot task offloading for sustainable internet of things. IEEE Transactions on Sustainable Computing pp 1–1, DOI 10.1109/TSUSC.2020.3028615

Ahmed M, Taconet C, Ould M, Chabridon S, Bouzeghoub A (2021) IoT Data Qualification for a Logistic Chain Traceability Smart Contract. Sensors 21(6):2239

Akkermans S, Bachiller R, Matthys N, Joosen W, Hughes D, Vučinić M (2016) Towards efficient publish-subscribe middleware in the iot with ipv6 multicast. In: 2016 IEEE International Conference on Communications (ICC), pp 1–6

Alhirabi N, Rana O, Perera C (2020) Security and Privacy Requirements for the Internet of Things: A Survey. ACM Trans Internet Things 2(1):6:1–6:37

Aljeraisy A, Barati M, Rana O, Perera C (2021) Privacy Laws and Privacy by Design Schemes for the Internet of Things: A Developer's Perspective. ACM Comput Surv 54(5):102:1–102:38

Bacon J, Moody K, Bates J, Ma C, McNeil A, Seidel O, Spiteri M (2000) Generic support for distributed applications. Computer 33(3):68–76, DOI 10.1109/2.825698

Banks A, Gupta R (2014) Mqtt version 3.1. 1

Barnaghi P, et al. (2011) Semantic Sensor Network XG Final Report. Tech. rep., W3C, URL http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

Bellavista P, Corradi A, Fanelli M, Foschini L (2012) A Survey of Context Data Distribution for Mobile Ubiquitous Systems. ACM Computing Survey 44(4):24:1–24:45

Biegel G, Cahill V (2007) Requirements for middleware for pervasive information systems. In: Pervasive Information Systems, M.E. Sharpe, Armonk, NY, pp 102–118

Biegel G, Cahill V (2015) Requirements for middleware for pervasive information systems. In: Kourouthanassis PE, Giaglis GM (eds) Pervasive information systems, Routledge, USA, pp 86–102

Binder W, Suri N (2009) Green computing: Energy consumption optimized service hosting. In: 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), Spindleruv Mlýn, Czech Republic, Springer, Lecture Notes in Computer Science, vol 5404, pp 117–128

Blair GS, Schmidt DC, Taconet C (2016) Middleware for Internet distribution in the context of cloud computing and the Internet of Things - Editorial Introduction. Ann des Télécommunications 71(3-4):87–92

Borges PV, Taconet C, Chabridon S, Conan D, Batista T, Cavalcante E, Batista C (2019) Mastering Interactions with Internet of Things Platforms through the IoTVar Middleware. In: 13th Int. Conf. on Ubiquitous Computing and Ambient Intelligence (UCAmI), MDPI Proceedings, vol 31, p 78

Bouloukakis G, Georgantas N, Ntumba P, Issarny V (2019) Automated Synthesis of Mediators for Middleware-layer Protocol Interoperability in the IoT. Future Generation Computer Systems 101:1271 – 1294

Bouloukakis G, et al. (2022) DeXMS, The Data eXchange Mediator Synthesizer Framework. https://gitlab.inria.fr/dexms

Buchholz T, Kupper A, Schiffers M (2003) Quality of context information: What it is and why we need it. In: 10th Int. Workshop of the HP OpenView University Association (HPOVUA), Geneva, Switzerland

Castellani AP, Fossati T, Loreto S (2012) HTTP-CoAP cross protocol proxy: an implementation viewpoint. In: 9th IEEE Int. Conf. on Mobile Ad-Hoc and Sensor Systems, (MASS)

Cavalcanti D, Carvalho R, Rosa N (2021) Adaptive middleware of things. In: Proceedings of the 2021 IEEE Symposium on Computers and Communications, IEEE, USA

Cavoukian A, Dixon M (2013) Privacy and security by design: An enterprise architecture approach. Tech. rep., Information and Privacy Commissioner of Ontario, Canada, https://www.ipc.on.ca

Cecchinel C, Fouquet F, Mosser S, Collet P (2019) Leveraging live machine learning and deep sleep to support a self-adaptive efficient configuration of battery powered sensors. Future Generation Computer Systems 92:225–240

Chabridon S, Laborde R, Desprats T, Oglaza A, Marie P, Machara Marquez S (2014) A Survey on Addressing Privacy together with Quality of Context for Context Management in the Internet of Things. Annals of Telecommunications 69(1):47–62

Chaudhuri A, Cavoukian A (2018) The Proactive and Preventive Privacy (3P) Framework for IoT Privacy by Design. EDPACS 57(1):1–16

Chen H, Finin T, Joshi A (2005) The SOUPA ontology for Pervasive Computing. In: Ontologies for agents: Theory and experiences, Whitestein Series in Software Agent Technologies, Switzerland, pp 233–258

Colesky M, Hoepman JH, Boesch C, Kargl F, Kopp H, Mosby P, Métayer DL, Drozd O, del Álamo JM, Martín YS, Caiza JC, Gupta M, Doty N (2022) Privacy Patterns. https://privacypatterns.org

Collina M, Corazza GE, Vanelli-Coralli A (2012) Introducing the QEST broker: Scaling the iot by bridging MQTT and REST. In: 23rd IEEE Int. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)

Compton M, Barnaghi P, Bermudez L, García-Castro R, Corcho O, Cox S, Graybeal J, Hauswirth M, Henson C, Herzog A, Huang V, Janowicz K, Kelsey WD, Le Phuoc D, Lefort L, Leggieri M, Neuhaus H, Nikolov A, Page K, Passant A, Sheth A, Taylor K (2012) The SSN ontology of the W3C semantic sensor network incubator group. Journal of Web Semantics 17

Conan D, Lim L, Taconet C, Chabridon S, Lecocq C (2017) A Multiscale Approach for a Distributed Event-Based Internet of Things. In: Proc. of 15th IEEE Int. Conf. on Pervasive Intelligence and Computing (PICOM), Orlando, USA, pp 844–852

Conan D, et al. (2022) M4IoT Frameworks, Middleware for the Internet of Things. https://www-inf.it-sudparis.eu/m4iot/

Cugola G, Margara A (2012) Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Survey 44(3):15:1–15:62

Daniele L, den Hartog F, Roes J (2015) The Smart Appliances REFerence (SAREF) Ontology. In: Proc. of International Workshop Formal Ontologies Meet Industries

Delicato F, Pires P, Batista T (2013) Middleware Solutions for the Internet of Things. Springer Briefs in Computer Science, Springer

Deng M, Wuyts K, Scandariato R, Preneel B, Joosen W (2011) A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requirements Engineering 16(1):3–32

Denis N, Chaffardon P, Conan D, Laurent M, Chabridon S, Leneutre J (2020) Privacy-preserving Content-based Publish/Subscribe with Encrypted Matching and Data Splitting. In: 17th Int. Joint Conf. on e-Business and Telecommunications (SECRYPT), INSTICC, SciTePress, Paris, France, pp 405–414

Derhamy H, Eliasson J, Delsing J (2017) Iot interoperability—on-demand and low latency transparent multiprotocol translator. IEEE Internet of Things Journal 4(5)

Dey A, Abowd G (2000) Towards a better understanding of context and context-awareness. In: Proceedings of the PrCHI 2000 Workshop on the What, Who, Where, When and How of Context-Awareness

Dias D, Delicato F, Pires P, Rocha A, Nakagawa E (2020) An Overview of Reference Architectures for Cloud of Things. In: Proc. of the 35th ACM Symposium on Applied Computing, New York, NY, USA, pp 1498–1505

Duboc L, Rosenblum D, Wicks T (2007) A Framework for Characterization and Analysis of Software System Scalability. In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnik, Croatia, pp 375–384

Endler M, Silva F (2018) Past, Present and Future of the ContextNet IoMT Middleware. Open Journal of Internet Of Things (OJIOT) 4(1):7–23, Special Issue: Int. Workshop on Very Large Internet of Things (VLIoT), in conjunction with the VLDB Conference in Rio de Janeiro, Brazil

Eugster P, Felber P, Guerraoui R, Kermarrec AM (2003) The Many Faces of Publish/Subscribe. ACM Computing Survey 35(2)

Ferreboeuf H, Efoui-Hess M, Verne X (2021) Impact environnemental du numérique : Tendances à 5 ans et gouvernance de la 5G. Tech. rep., The Shift project

Fette I (2011) The websocket protocol

Fiege L, Mezini M, Mühl G, Buchmann A (2002) Engineering Event-Based Systems with Scopes. In: Magnusson B (ed) Proc. 16th European Conference on Object-Oriented Programming, Springer, Málaga, Spain, Lecture Notes in Computer Science, vol 2374, pp 309–333

Fremantle P, Scott PJ (2017) A survey of secure middleware for the internet of things. PeerJ Comput Sci 3:e114

García-Vallis M, Baldoni R (2015) Adaptive middleware design for CPS: Considerations on the OS, resource managers, and the network at run-time. In: 14th Int. Workshop on Adaptive and Reflective Middleware, ACM, USA, DOI 10.1145/2834965.2834968

Georgantas N, Bouloukakis G, Beauche S, Issarny V (2013) Service-oriented distributed applications in the future internet: The case for interaction paradigm interoperability. In: Lau K, Lamersdorf W, Pimentel E (eds) 2nd European Conf. on Service-Oriented and Cloud Computing, ESOCC, vol 8135

Giouroukis D, Dadiani A, Traub J, Zeuch S, Markl V (2020a) A Survey of Adaptive Sampling and Filtering Algorithms for the Internet of Things. In: Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems, Association for Computing Machinery, New York, NY, USA, DEBS '20, p 27–38, DOI 10.1145/3401025.3403777, URL https://doi.org/10.1145/3401025.3403777

Giouroukis D, Dadiani A, Traub J, Zeuch S, Markl V (2020b) A Survey of Adaptive Sampling and Filtering Algorithms for the Internet of Things. In: Proc. 14th ACM International Conference on Distributed Event-Based Systems, Montreal, Quebec, Canada, pp 27–38

Gomes B, Muniz LCM, da Silva e Silva FJ, dos Santos DV, Lopes RF, Coutinho LR, Carvalho FO, Endler M (2017a) A Middleware with Comprehensive Quality of Context Support for the Internet of Things Applications. Sensors 17(12):2853

Gomes P, Cavalcante E, Batista T, Taconet C, Conan D, Chabridon S, Delicato F, Pires P (2019) A semantic-based discovery service for the internet of things. Journal of Internet Services and Applications 10

Gomes P, et al. (2017b) QoDisco. https://github.com/porfiriogomes/qodisco

Gruber TR (1993) A translation approach to portable ontology specifications. Knowledge Acquisition 5(2)

Hassan MG, Hirst R, Siemieniuch C, Zobaa A (2009) The impact of energy awareness on energy efficiency. Int Journal of Sustainable Engineering 2(4):284–297

Henricksen K, Indulska J (2004) Modelling and using imperfect context information. In: Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on, pp 33–37

Howard M, Lipner S (2006) The Security Development Lifecycle. Microsoft Press, USA

Huebscher MC, McCann JA (2006) An adaptive middleware framework for context-aware applications. Pervasive and Ubiquitous Computing 10:12–20

ISO/IEC 25010 (2011) Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Tech. rep., ISO

ISO/IEC 25012 (2008) Data Quality model. URL https://iso25000.com/index.php/en/iso-25000-standards/iso-25012

Ivarez-Valera HH, Dalmau M, Roose P, Herzog C (2019) The architecture of kali-green V2: A middleware aware of hardware opportunities to save energy. In: Alsmirat MA, Jararweh Y (eds) Sixth International Conference on Internet of Things: Systems, Management and Security, IOTSMS 2019, Granada, Spain, October 22-25, 2019, IEEE, pp 79–86

Jagarlamudi KS, Zaslavsky A, Loke SW, Hassani A, Medvedev A (2021) Quality and Cost Aware Service Selection in IoT-Context Management Platforms. In: Int. Conferences on Internet of Things (iThings), Green Computing & Communications (GreenCom), Cyber, Physical & Social Computing (CPSCom), Smart Data (SmartData) and Congress on Cybermatics (Cybermatics), IEEE, pp 89–98

Janowicz K, Haller A, Cox SJ, Le Phuoc D, Lefrançois M (2019) SOSA: A lightweight ontology for sensors, observations, samples, and actuators. Journal of Web Semantics 56

Joint Committee for Guides in Metrology (2008) Evaluation of measurement data - guide to the expression of uncertainty in measurement. https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf

Kalbarczyk T, Julien C (2018) Omni: An Application Framework for Seamless Device-to-Device Interaction in the Wild. In: 19th Int. Middleware Conf., ACM, Rennes, France, p 161–173

Karagiannis V, et al. (2015) A Survey on Application Layer Protocols for the Internet of Things. Transaction on IoT and Cloud Computing 3:11–17

Kargl F, Métayer DL, Gupta M, Colesky M, Hoepman JH, del Álamo JM, Martín YS, Boesch C, Kopp H, Mosby P, Doty N, Drozd O (2022) Privacy Patterns, Collecting Patterns for Better Privacy. https://privacypatterns.eu

Kermarrec AM, Triantafillou P (2013) XL Peer-to-Peer Pub/Sub Systems. ACM Computing Survey 46(2):16:1–16:45

Kourouthanassis PE, Giaglis GM (2007) Pervasive Information Systems. Advances in Management Information Systems (AMIS) Vol. 10:. M.E. Sharpe, Armonk, NY

Kourouthanassis PE, Giaglis GM (2015) Toward pervasiveness: Four eras of information systems development. In: Kourouthanassis PE, Giaglis GM (eds) Pervasive information systems, Routledge, USA, pp 3–25

Krakowiak S (2009) Middleware Architecture with Patterns and Frameworks. https://lig-membres.imag.fr/krakowia/Files/MW-Book/Chapters/Preface/preface.html

Lim L, Marie P, Conan D, Chabridon S, Desprats T, Manzoor A (2015) Enhancing context data distribution for the internet of things using qoc-awareness and attribute-based access control. Annals of Telecommunications pp 1–12

López J, Rios R, Bao F, Wang G (2017) Evolving privacy: From sensors to the internet of things. Future Gener Comput Syst 75:46–57

Luckner M, Grzenda M, Kunicki R, Legierski J (2014) IoT Architecture for Urban Data-Centric Services and Applications. ACM Transactions on Internet Technology 20(3):29:1–29:30

Mahnke W, Leitner SH, Damm M (2009) OPC unified architecture. Springer Science & Business Media

Marie P, Desprats T, Chabridon S, Sibilla M (2013) QoCIM: A meta-model for Quality of Context. In: Modeling and Using Context, LNCS, vol 8175

Marie P, Desprats T, Chabridon S, Sibilla M (2016) Enabling Self-Configuration of QoC-Centric Fog Computing Entities. In: Intl IEEE Conf. on Advanced and Trusted Computing, Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France

Negash B, Rahmani AM, Westerlund T, Liljeberg P, Tenhunen H (2015) Lisa: Lightweight internet of things service bus architecture. Procedia Computer Science 52

Negash B, Rahmani AM, Westerlund T, Liljeberg P, Tenhunen H (2016) Lisa 2.0: lightweight internet of things service bus architecture using node centric networking. Journal of Ambient Intelligence and Humanized Computing 7(3)

Nistor A, Ravindranath L (2014) SunCat: Helping developers understand and predict performance problems in smartphone applications. In: Int. Symp. on Software Testing and Analysis, ACM, USA, p 282–292

Noureddine A, Rouvoy R, Seinturier L (2013) A review of middleware approaches for energy management in distributed environments. Softw Pract Exp 43(9):1071–1100

OASIS (2012) Advanced Message Queuing Protocol (AMQP) version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

de Oliveira EA, Delicato F, Mattoso M (2020) An energy-aware data cleaning workflow for real-time stream processing in the internet of things. In: Anais do IV Workshop de Computação Urbana, SBC, Porto Alegre, RS, Brasil, pp 71–83

OMG (2015) Data Distribution Service, v. 1.4. https://www.omg.org/spec/DDS/

Padhy S, Chang HY, Hou TF, Chou J, King CT, Hsu CH (2017) A Middleware Solution for Optimal Sensor Management of IoT Applications on LTE Devices. In: Quality, Reliability, Security and Robustness in Heterogeneous Networks (QSHINE), vol 199, Springer, pp 283–292

Pang C, Hindle A, Adams B, Hassan AE (2016) What do programmers know about software energy consumption? IEEE Software 33(03):83–89

Paradis CV, Kazman R, Tamburri DA (2021) Architectural tactics for energy efficiency: Review of the literature and research roadmap. In: 54th Hawaii International Conference on System Sciences (HICSS), pp 1–10

oneM2M Partners (2019) oneM2M Services Platform. Release 3

Pasricha S (2018) Overcoming Energy and Reliability Challenges for IoT and Mobile Devices with Data Analytics. In: 31st Int. Conf. on VLSI Design (VLSID)

Penzenstadler B (2015) From requirements engineering to green requirements engineering. In: Calero C, Piattini M (eds) Green in Software Engineering, Springer

Perera C, Zaslavsky AB, Christen P, Georgakopoulos D (2014) Context aware computing for the internet of things: A survey. IEEE Commun Surv Tutorials 16(1):414–454

Perera C, Qin Y, Estrella J, Reiff-Marganiec S, Vasilakos A (2017) Fog Computing for Sustainable Smart Cities: A Survey. ACM Computing Survey 50(3):32:1–32:43

Portocarrero JMT, Delicato FC, Pires PF, Rodrigues TC, Batista TV (2016) SAMSON: Self-adaptive middleware for wireless sensor networks. In: 31st Annual ACM Symposium on Applied Computing, ACM, USA

Rafique A, Van Landuyt D, Reniers V, Jossen W (2017) Towards an adaptive middleware for efficient multi-cloud data storage. In: 4th Workshop on CrossCloud Infrastructures & Platforms, ACM, USA

Raychoudhury V, Cao J, Kumar M, Zhang D (2013) Middleware for pervasive computing: A survey. Pervasive Mob Comput 9(2):177–200

Rosa N, Cavalcanti D, Campos G, Silva A (2020) Adaptive middleware in Go - a software architecture approach. Journal of Internet Services and Applications 11(3), DOI 10.1186/s13174-020-00124-5

Roth FM, Becker C, Vega G, Lalanda P (2018) XWARE - A customizable interoperability framework for pervasive computing systems. Pervasive Mob Comput 47

Sadjadi SM, McKinley PK (2003) A survey of adaptive middleware. Tech. rep., Michigan State University, USA

Saint-Andre P (2011) Extensible messaging and presence protocol (xmpp): Core

Salehie M, Tahvildari L (2009) Self-adaptive Software: Landscape and Research Challenges. ACM Transactions on Autonomous and Adaptive Systems 4(2)

Sarkar C, Rao VS, Venkatesha Prasad R, Das SN, Misra S, Vasilakos A (2016) Vsf: An energy-efficient sensing framework using virtual sensors. IEEE Sensors Journal 16(12):5046–5059, DOI 10.1109/JSEN.2016.2546839

Satyanarayanan M (2001a) Pervasive computing: vision and challenges. Personal Communications, IEEE 8(4):10–17, DOI 10.1109/98.943998

Satyanarayanan M (2001b) Pervasive Computing: Vision and Challenges. IEEE Personal Communications 8(4):10–17

Sedlmeir J, Buhl HU, Fridgen G, Keller R (2020) The energy consumption of blockchain technology: beyond myth. Business & Information Systems Engineering 62(6):599–608

Shekhar S, Chhokra A, Sun H, Gokhale A, Dubey A, Koutsoukos X (2019) URMILA: A Performance and Mobility-Aware Fog/Edge Resource Management Middleware. In: 22nd IEEE Int. Symposium on Real-Time Distributed Computing (ISORC), pp 118–125

Shelby Z, et al. (2014) The constrained application protocol (coap)

Song Z, Le M, Kwon YW, Tilevich E (2017) Extemporaneous micro-mobile service execution without code sharing. In: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), pp 181–186, DOI 10.1109/ICDCSW.2017.70

Spalazzi L, Taccari G, Bernardini A (2014) An internet of things ontology for earthquake emergency evaluation and response. In: Proceedings of the 2014 International Conference on Collaboration Technologies and Systems (CTS 2014), pp 528–534

Taivalsaari A, Mikkonen T (2017) A roadmap to the Programmable World: Software challenges in the IoT era. IEEE Software 34(1):72–80, DOI 10.1109/MS.2017.26

Teixeira S, Agrizzi BA, Filho JGP, Rossetto S, Pereira ISA, Costa PD, Branco AF, Martinelli RR (2020) LAURA architecture: Towards a simpler way of building situation-aware and business-aware IoT applications. Journal of Systems and Software 161:110494

Truong HL, Dustdar S (2015) Principles for engineering IoT cloud systems. IEEE Cloud Computing 2(2):68–76, DOI 10.1109/MCC.2015.23

Vekris P, Jhala R, Lerner S, Agarwal Y (2012) Towards verifying Android apps for the absence of no-sleep energy bugs. In: Proceedings of the 2012 Workshop on Power-Aware Computing and Systems, USENIX Association, USA

Verdecchia R, Lago P, Ebert C, de Vries C (2021) Green it and green software. IEEE Software 38(6):7–15, DOI 10.1109/MS.2021.3102254

Weiser M (1991) The Computer for the 21st Century. Scientific American, Special Issue on Communications, Computers, and Networks 265(3):66–75

Yau SS, Karim F (2004) An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments. Real-Time Systems 26:29–61

Yus R, Bouloukakis G, Mehrotra S, Venkatasubramanian N (2019) Abstracting interactions with iot devices towards a semantic vision of smart spaces. In: 6th ACM Int. Conf. on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys

Zeeb E, Bobek A, Bohn H, Golatowski F (2007) Service-oriented architectures for embedded systems using devices profile for web services. In: 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), IEEE, vol 1, pp 956–963