Internet of Things 19 (2022) 100529

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Review article

Designing and constructing internet-of-Things systems: An overview of the ecosystem

João Pedro Dias [a,d,*], André Restivo [a,b], Hugo Sereno Ferreira [a,c]

[a] Faculty of Engineering, University of Porto, Porto, Portugal
[b] LIACC, Porto, Portugal
[c] INESC TEC, Porto, Portugal
[d] BUILT CoLAB, Porto, Portugal

ARTICLE INFO

Keywords:
Internet-of-Things
Software engineering
Embedded systems
Large-scale systems
System design
System development

ABSTRACT

The current complexity of IoT systems and devices is a barrier to reach a healthy ecosystem, mainly due to technological fragmentation and inherent heterogeneity. Meanwhile, the field has scarcely adopted any engineering practices currently employed in other types of large-scale systems. Although many researchers and practitioners are aware of the current state of affairs and strive to address these problems, compromises have been hard to reach, making them settle for sub-optimal solutions. This paper surveys the current state of the art in designing and constructing IoT systems from the software engineering perspective, without overlooking hardware concerns, revealing current trends and research directions.

1. Introduction

In recent years, the explosion of Internet-connected devices having computing capabilities, also known as the Internet-of-Things (IoT) — the peak and *utopia* of ubiquitous connectivity and computing — made software systems even more complex and pervasive. Among other factors, IoT will not succeed if a way to interconnect the plethora of available devices and services seamlessly is not found.

Developing software systems is hard; not only due to the complex nature of these systems, but also because developers are humans, and humans tend to make mistakes. And this paradigm-shift comes with an increase in complexity[1] that might impact the already low success rate of software development; and historically, this has always been a problem with software engineering [2].

The inherent nature of IoT systems is characterized by being ultra-large-scale, having highly-dynamic topologies, being highly heterogeneous, and being cross-domain in terms of its application scenarios [3]. These characteristics, together with others, pose new challenges to the software development life-cycle of IoT, including the design and construction phases. Brian Fitzgerald envisions this as a new crisis [2]:

The demand for data from digital natives, coupled with the huge volume of data now generated through ubiquitous mobile devices, sensors, and applications, has led to a new software crisis.

* Corresponding author at: Faculty of Engineering, University of Porto, Porto, Portugal.
E-mail addresses: jpmdias@fe.up.pt (J.P. Dias), arestivo@fe.up.pt (A. Restivo), hugosf@fe.up.pt (H.S. Ferreira).

[1] A Microsoft study points out that 38% of the industry practitioners consider complexity and other technical challenges as the top concern for IoT development [1].

https://doi.org/10.1016/j.iot.2022.100529
Received 4 January 2022; Received in revised form 18 February 2022; Accepted 5 April 2022
Available online 20 April 2022

On the one hand, there has been an unprecedented increase in the number of new devices, resulting from both the significant advancement of hardware capabilities and the dramatic reduction in their cost, leading to the dissemination of data collection practices.[2] On the other hand, the desire to consume new technologies by the *digital natives* has been growing steadily. These are the disruptive factors for this crisis, as the advances in software engineering practices have not improved at the same pace as needed for it to tend to this new landscape [2].

Like any other kind of paradigm shift, and, in this case, from the software engineering community's perspective, there are no consensus or standards on the best practices for developing IoT systems [3]. Practitioners have been calling others into action, through manifestos, to tame this chaotic state of the practice [5]. Among others, several questions stand unanswered such as, how to select the more suitable architecture, which are the more reliable and optimal communications protocols for each scenario, and what are the best approaches in terms of security and privacy.

Developing software for IoT systems implies considerations that go beyond the everyday challenges of developing *software-only* systems (*e.g.*, mobile and web applications), such as the ones pointed by Taivalsaari et al. [6]: (1) multi-device programming (from tiny devices to cloud servers), (2) the reactive and always-on nature of the system (real-time and *human-in-the-loop*), (3) heterogeneity and diversity, (4) the distributed, highly dynamic, and potentially migratory nature of software and hardware components, and (5) the general need to write software in a fault-tolerant and defensive manner (from a safety and security perspective). These specific challenges, along with the unsuitability of the existing languages and tools, have a most direct impact on the software development of these systems and their life-cycle.

The application of the comprehensive Software Development Life-Cycle (SDLC) processes and methodologies to design, construct, test, deploy and maintain IoT systems faces different challenges than the ones met when developing traditional software systems, due to the inherent peculiarities of the IoT ecosystem (*e.g.*, dealing with both hardware and software limitations). Although well-known software practices can be harnessed and adapted to the IoT development challenges, there is still a need for new approaches, environments and best practices [7]. Zambonelli et al. [8] goes further by proposing a novel *IoT-oriented software engineering* field that brings together the overlapping engineering areas involved in IoT which eventually could *lead to the identification of general models, methodologies and tools* for developing these systems. As the typical user of IoT systems has little to no technical expertise, development solutions that empower them to configure and program their systems are also required [9]. Fortino et al. review of twelve different methodologies focused on IoT development concludes that most of them "mainly cover the analysis and design processes" from a stakeholder perspective, being typical general purpose, and with only a few "existing IoT platforms and developing large-scale decentralized IoT applications" [10].

While there are already plenty of surveys that focus on IoT, they are typically focused on specific aspects of the systems (*e.g.*, protocols [11], security [12,13], or privacy [13]) or application domains and their intricacies (*e.g.*, smart farming [14], smart cities [15,16] and industry and manufacturing [17]). Nonetheless, we are aware that different surveys focus on specific aspects of design and constructions, *e.g.*, low-code development solutions for IoT [18], cloud IoT platforms [19], design patterns [20], or architectures [21], and these works, as well as the approaches they identify, are contemplated as part of this study. As far as we could find, there are no surveys that contemplate the different approaches for designing and constructing IoT systems.

While one can argue that most of the knowledge about Software Engineering practices and IoT is readily available — captured and documented in books, novel research papers and surveys (including systematic studies), (open-source) implementations, web pages, and a myriad of other types of resources — the continuous growth in the body of knowledge of IoT across different media creates an intangible amount of literature that one must contemplate before being able to make decisions about the system they want to design. This growth can be observed by the increasing literature entries with the keywords IoT and Software Engineering being published in different conferences and journals, and indexed by Google Scholar, *cf.* Fig. 1.

In this paper, we delve into the current state of software engineering practices for IoT, focusing on the design and construction of such systems. This analysis will be framed in terms of the technological landscape and cross-cutting research challenges and opportunities, providing the reader with a common knowledge base supported by existent works. While we concur that there is a certain overlap with existing surveys and other systematic literature studies — which are mentioned and cited along the paper when appropriate — in this work we attempt to make a summary of the existent solutions without limiting to a specific approach or methodology (*e.g.*, analyzing different solutions under different approaches for developing IoT systems). This can be valuable to newcomers to the field that want to have a brief and wide vision of the different approaches being used in IoT and some of the solutions under those approaches.

We consider the main contribution of this paper to be **a broad overview of the current state of the art on how to design and construct IoT systems**, which can be used by both researchers and practitioners, especially those new to the field, providing ground and guide for further research — slightly pointing towards open research challenges and other pending issues of the IoT ecosystem.

2. Methodology and organization of the article

The approach used for this survey follows that of a *hybrid narrative review*, a framework more flexible than *systematic reviews*, though still providing a means for the reader to determine the authenticity of the author's findings [22]. We believe that this methodology is valuable given the current state of maturity of this field as IoT and Software Engineering related peer-reviewed

[2] One can easily associate the birth of the term Internet-of-Things as a direct consequence of this dissemination [4].

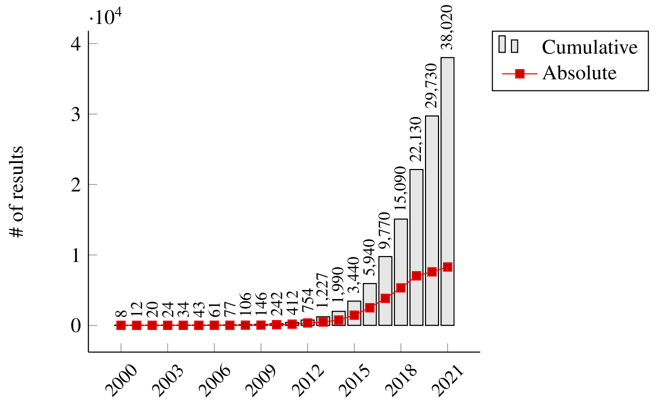

Fig. 1. The growing interest in the Software Engineering field for IoT can be observed by the increase in the literature that mentions both keywords, as it can be seen by the number of results on Google Scholar with the query `‘‘Software Engineering’’ AND (‘‘IoT’’ or ‘‘Internet-of-Things’’)`.

literature is yet in its early stages (most publications with such keywords started appearing back in 2014, with a priori publications being residual, *cf.* Fig. 1), with several studies lacking rigorous experimental validations, and vast amounts of knowledge still being disseminated using grey literature such as websites and technical reports. This is also supported by the fact that most IoT-related conferences and journals have only a few editions, with some of the oldest wide-range IoT venues appearing in 2014 (7 years as of 2021) [23], and with the first IoT Software Engineering focused workshop appearing in 2019 [24].

Thus, we employ an approach of identifying relevant articles through the *Google Scholar* search engine that indexes an extensive list of multi-disciplinary conferences, journals, white papers, and reports across multiple sources. While other bibliography sources were used for the first queries of this research (*e.g.*, Scopus, ACM Digital Library, and IEEEXplore) we found out that most of the works also appeared in *Google Scholar*, which led us to use it to the detriment of the other sources. This decision happened before any data collection, so we have no data about such early queries.

This methodology, however, still requires the definition of inclusion and exclusion criteria (IC/EC) as a way to provide the information necessary to understand how certain information was selected, even given their broad nature. These criteria are the following:

- Included resources must be written in English language (resources in other languages are excluded);
- Must be relevant for the IoT area;
- Must focus on systems’ design and/or construction aspects;
- Must present content beyond bare ideas, with reference implementations and/or evaluations; or be literature surveys of relevance to the study;
- Full text must be accessible (papers that are not available or behind a paywall are excluded);
- Must include the word `IoT` or `Internet-of-Things` or `Internet of Things` in the document text (title, abstract or body);
- Resources that focus on business models and similar aspects are also excluded.

The search criteria included different keywords due to the broadness of the holistic key concepts *Internet-of-Things* and *Software Engineering* (*cf.* Fig. 1). Specific queries were used for some parts of this work and are detailed when appropriate. An in-depth analysis was done for the most cited papers, and relevant articles cited by them were also analyzed.

A set of reference works in the area of *Software Engineering*, more precisely in the fields of software design and construction were used as both a baseline and a way to gather and present valuable insights [25–29].

This methodology, beyond the limitations, already presented, has inherent validity threats. These threats, along with the efforts that were made in order to mitigate or reduce their impact, are further discussed in Section 7.

The remaining article is organized as follows: Section 3 starts by introducing the IoT paradigm. This includes a brief overview of its history and evolution, an analysis of the domains of application, and a revision of some of the IoT key concepts. Some notes on software engineering for the Internet-of-Things are also given. Section 4 delves into the design of IoT systems, introducing the enabling technologies, in terms of hardware and software, as well as the current design practices, in terms of system architectures and software design. Section 5 presents the current state of the available approaches for building such systems at different levels of abstraction.

These two main sections (Sections 4 and 5) are followed by Section 6 presenting an overall discussion taking into account the research conducted and reviewed literature, pointing to new research directions and exploratory literature. Additional discussion on the threats to the validity of this work are presented in Section 7 and, finally, some closing remarks are given in Section 8.

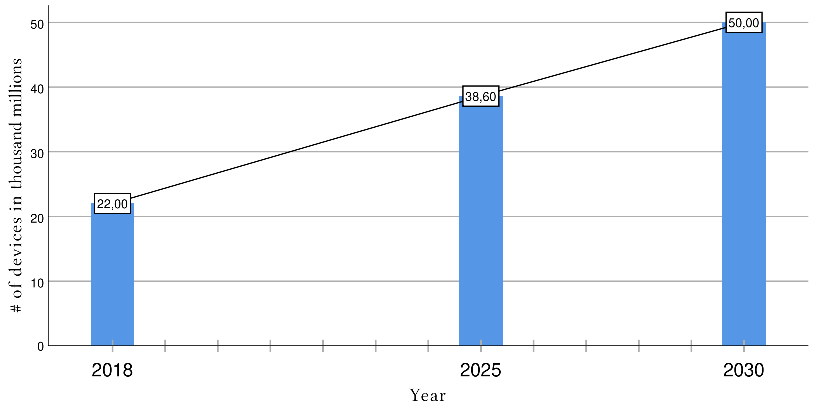

Fig. 2. Actual count and estimation of the number of IoT devices per year (from Statista) [31].

3. Preliminaries

The term Internet-of-Things is claimed to have been coined by Kevin Ashton circa 1999 during a presentation about supply-chain management and the use of Radio-Frequency Identification (RFID) technology to enable computers to observe, identify and understand the world (without the limitations of human entered data), long before it was typical for anything but computers to be connected to the Internet [4].

3.1. Key concepts

Encompassing the opportunities of IoT, the joint technical committee of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), *cf.* ISO/IEC JTC 1, defined[3] Internet-of-Things as [30]:

> *An infrastructure of interconnected objects, people, systems and information resources together with intelligent services to allow them to process information about the physical and the virtual world and react.*

IoT shifts computing from traditional devices (IoC) towards a ubiquitous and pervasive computing world of systems, and, Systems of Systems (SoS), where heterogeneous and highly-distributed objects (*things*) are computing-capable and Internet-connected [10,32]. Even further, Fortiono et al. states that "IoT systems follow the SoS paradigm and face similar issues, thus demanding for a comprehensive engineering approach" [10]. The number of Internet-connected devices under the IoT umbrella is growing (see Fig. 2), expected to reach around 50 thousand million connected devices by 2030 [31].

From its birth, a crucial requirement for IoT, beyond ubiquitous computing, was ubiquitous connectivity. Any object that is aware of its context and that can communicate with other entities fall under the umbrella of IoT. Initially, RFID was the dominant technology behind IoT development, but, as of today, wireless sensor networks (WSN) and Bluetooth-enabled devices augmented the mainstream adoption of the IoT trend [4].

From an innovation point-of-view, IoT has been enhancing the interactions among *things* and humans, enabling the realization of smart cities, infrastructures, and services for enhancing the quality of life and utilization of resources. Thus, IoT envisions a new world of connected devices and humans in which the quality of life is enhanced because the management of the city and its infrastructure is less cumbersome, health services are conveniently accessible, and disaster recovery is more efficient [33]. Regarding the fourth industrial revolution, so-called Industry 4.0, IoT, more specifically known as Industrial IoT (IIoT), is considered the *supporting backbone* of the vision of "*a fully describable, manageable, context-sensitive and controllable or self-regulating manufacturing systems*", making it possible by embedding Internet-powered devices in the production systems that are part of the life cycle of a product [34].

From a technical point of view, one can consider that a major role of the IoT consists of the delivery of highly complex knowledge-based and action-oriented applications in real-time. To be able to reach such an end, several considerations should be done regarding the full lifecycle of these systems, from conceptualization to development, from test to deployment and maintenance. These include, but are not limited to, (1) development of scalable architectures, (2) moving from closed systems to open systems, (3) dealing with privacy and ethical issues (due to data collection and storage practices), (4) heterogeneity support, (5) data storage, (6) data processing, decision-making, (7) designing interaction protocols, (8) autonomous management, (9) communication protocols, (10) smart objects and service discovery, (11) programming frameworks and languages, (12) resource management, (13) data and network management, (14) real-time needs, (15) power and energy management, (16) governance, and (17) interoperability [33].

[3] Note that the precise definition of IoT, and its relationship/boundaries with concepts such as Cyber–physical Systems (CPS), Intelligent Internet-of-Things or Internet-of-Everything (IoE), is yet an open discussion [7].

Table 1
IoT application domains [35].

Domain	Description	Indicative Examples
Industry	Activities involving financial or commercial transactions between companies, organizations and other entities.	Manufacturing, logistics, service sector, banking, financial governmental authorities, and intermediaries.
Environment	Activities regarding the protection, monitoring and development of all natural resources.	Agriculture & breeding, recycling, environmental management services, and energy management.
Society	Activities and initiatives regarding the development and inclusion of societies, cities and people.	Governmental services towards citizens and other society structures (e-participation), and e-inclusion (*e.g.*, aging, disabled people).

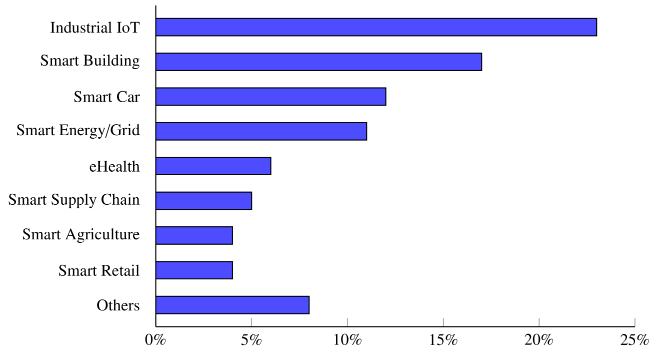

Fig. 3. Statistics based upon 1600 public known enterprise IoT projects circa 2018 (not including consumer level IoT projects such as wearables and smart homes) [36].

3.2. Application domains

IoT has been the main booster of technological innovation in different contexts and scenarios as it is the foundation for any *smart space*. This role is visible through the work done by the Cluster of European Research Projects on the Internet of Things (CERP-IoT) which has identified a large number of application domains for IoT [35]. A study from Microsoft points out that 85% of the surveyed enterprise decision-makers have at least one IoT project in either the learning, proof of concept, purchase, or use phase [1].

The CERP-IoT report [35] defines three IoT application domains, as described in Table 1. Within these application domains, several fields with open opportunities are presented such as aerospace and aviation (*e.g.*, systems status monitoring, green operations), automotive (*e.g.*, systems status monitoring, vehicle-to-vehicle and vehicle-to-infrastructure communication), telecommunications, intelligent buildings (*e.g.*, automatic energy metering, home automation, wireless control), healthcare (*e.g.*, personal area networks, monitoring of parameters, positioning, real-time location systems), independent living (*e.g.*, wellness, mobility, monitoring of an elder population), retail, logistics, supply chain management, people and goods transportation, media, entertainment and insurance.

The IoT Analytics GmbH report [36] points that the most relevant enterprise-level IoT segments are Smart City, Industrial IoT, Smart Building, Smart Car, Smart Energy/Grid, eHealth, Smart Supply Chain, Smart Agriculture, and Smart Retail; their relevance is shown in the chart on Fig. 3. However, this report does not take into account the consumer-level IoT segment (*e.g.*, wearables and smart homes).

IoT enterprise applications can also be aggregated in three major categories, depending on their role; namely: (1) monitoring and actuating, (2) business process and data analysis, and (3) information gathering and collaborative consumption [33].

The IIoT, the core technological component of the Industry 4.0 initiative, is a form of IoT application, favored by big high-tech companies, envisioning the sensing and actuating capabilities of *things* as a way to gather more data about their processes. This enables companies to detect and resolve problems faster, thus resulting in overall money and time savings [33]. As an example, in a manufacturing company, IIoT can be used to efficiently track and manage the supply chain, perform quality control and assurance.

Together with the recognized impact that IoT can have on the industry, the impact it can have on improving the quality of life is also envisioned [33]. From a healthcare perspective, IoT can be a facilitator of data collecting (*e.g.*, heart rate) which enables remote patient monitoring, *viz.* ambient assisted living [37]. Further, monitoring hazard environmental conditions can give data insights for authorities to act better and alert the population [38,39].

Holistically, exploiting the open IoT opportunities and different application domains can lead to improvement not only in people's quality of life but also in the industry and the enterprise world.

3.3. Sociotechnological context

The initial vision of IoT — enabled by achieving ubiquitous computation[4] and ubiquitous connectivity — has now become a reality. Our mobile phones, our security cameras, our watches, our fridge, toaster and coffee machine, all have (micro-)processors more powerful than the one used to send the first man to the moon [40]. They all tend to be (inter-)connected for sending alerts, transferring new updates, or even downloading new *mocha* recipes from the Internet [41]. We continuously push annotated geo-location (collected almost simultaneously by the GPSs present in our mobile phone, watch, fitness band and car) to a multitude of recipients that consume vast volumes of personal data for the sole purpose of displaying ads [42]. Even wired internet access is becoming obsolete due to faster, cheaper, convenient, and more energy-efficient wireless technologies [43]. The *status quo* is ubiquitous connectivity, available to everyone, everything, every time, everywhere.

From the above examples, one can understand why IoT has been identified as an enabler of *machine-to-machine*, *human-to-machine*, and *human-with-environment* interactions [33]. IoT empowers the so-called *human-in-the-loop* systems, in which humans and *things* operate synergistically and cooperatively. As new applications intimately involve humans, a range of new opportunities for a broader range of applications will appear.

Munir et al. categorized the *human-in-the-loop* applications based on the controls that they employ, namely, (1) applications where humans directly control the system (*e.g.*, "toggle the room lights"), (2) applications where the system passively monitors humans and takes appropriate actions (*e.g.*, surveillance systems or sleep monitors), and (3) hybrid approaches based on the aforementioned (*e.g.*, energy optimization systems that combine information from sensors and operators) [44].

However, as pointed by John Stankovic, several challenges also arise from the *human-in-the-loop* condition: (1) the need for a comprehensive understanding of the complete spectrum of types of *human-in-the-loop* controls, and (2) the need to extend current identification systems, or other techniques to derive models of human behaviors and determine how to incorporate human behavior models into the formal methodology of feedback control [45].

3.4. Technology maturity

A report from Microsoft circa 2019 provides some insights on the challenges faced by practitioners that want to utilize IoT-based products and services, identifying the complexity of the field and other technical challenges (38%), lack of budget and resources (29%), lack of knowledge (29%), lack of suitable IoT solutions (28%) and security considerations (19%) as driving issues [1].

Focusing on the complexity of the field and other technical challenges, the wide range of application domains and technological advancements that empower IoT to make it depend on several areas of knowledge, thus inheriting their particularities and challenges while also posing new ones. Inherent particularities such as high-heterogeneity, logical and geographical distribution, human-in-the-loop concerns, real-time needs, and power constraints play a key role in the design, development, testing, and maintenance of IoT systems, but are also impacted by other, more common, concerns such as the ones of large-scale systems such as interoperability, security, and scalability [46].

The lack of maturity of several technologies, architectures, and approaches used in IoT (as the lack of the maturity of the field by itself), makes it hard to know how these will evolve, survive or suffice in the long run. Both the technological challenges (complexity) and unpredictability of the field makes it hard for practitioners to find workers with the right skills and experience to embrace IoT.[5] [1].

Given the high data volume being generated by IoT, plus the variety of objects that are part of it, several issues/requirements need to be addressed to provide a good Quality of Service (QoS). Examples of these are minimizing the latency (milliseconds matter for many types of industrial systems, such as when you are trying to prevent manufacturing line shutdowns or restore electrical service), conserving network bandwidth (since it is not practical to transport vast amounts of data from thousands of edge devices, as in sensors and actuators, to the cloud), and increasing local efficiency (*e.g.*, collecting and securing data across a wide geographic area with different environmental conditions may not be useful) [47–49].

3.5. Architectural context

Traditional IT computing models (*e.g.*, a direct connection between end-devices and the cloud[6]) do not suffice for the QoS needs of IoT systems. For most of the scenarios, these requirements are not met due to limitations in bandwidth in last-mile IoT networks, high latency, network unreliability, and increasing volume of the data being generated, transmitted and, *a posteriori*, analyzed [48,49,49].

Fog computing has been proposed and used as a way to mitigate some of the challenges mentioned earlier. The concept coined by Cisco Systems, circa 2015, focuses on distributed data management throughout the IoT system, as close to the edge of the network as possible [49]. The National Institute of Standards and Technology (NIST) [50] states that the fog computing model *facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes residing between smart end-devices and centralized (cloud) services*.

Fog computing provides contextual location awareness, low latency, geographical distribution, heterogeneity support, interoperability, real-time interactions support, scalability and agility of federated, *fog-node* clusters [49,50].

[4] Ubiquitous computation is also known as *ubicomp* or pervasive computing.

[5] "*47% of companies that have adopted IoT report that they do not have enough skilled workers, and 44% do not have enough available resources to train employees*" [1].

[6] Approach popularized by the commercialization of Platforms as a Services (PaaS) IoT solutions, also known as CoT (Cloud of Things) [48].

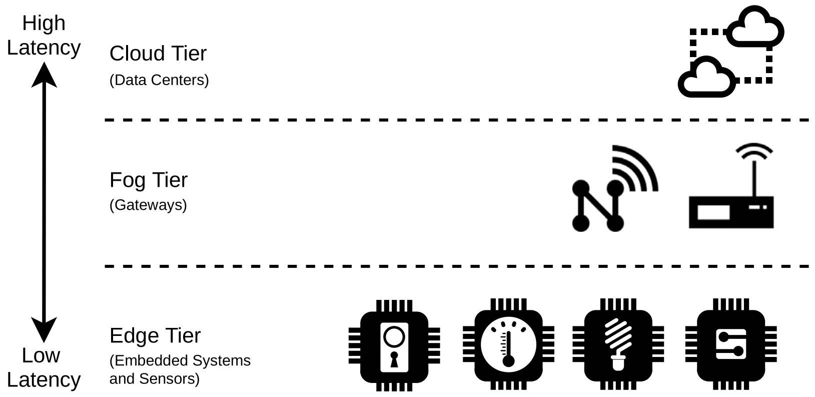

Fig. 4. Typical architectural tiers composition of an IoT system. The upper tiers have more latency and more computational power than the lower tiers of the stack.

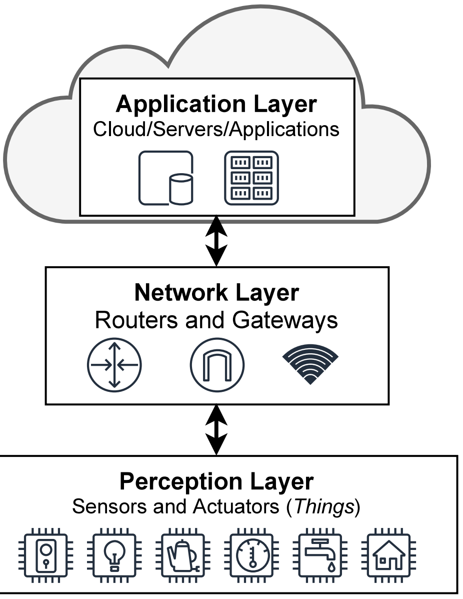

Fig. 5. Logical view of the common layers of IoT systems.

Fog computing presents several advantages in terms of latency (*i.e.*, real-time interactions support), local proximity (*i.e.*, contextual location awareness), geographical distribution (*i.e.*, potentially increasing resilience), and battery life extension. However, some disadvantages are also identified such as the increase in heterogeneity (*i.e.*, when compared to the homogeneousness of cloud computing) and the transience of their permanence in the network (high-dynamic topology due to, *e.g.*, device mobility) [48–50]. The *fog nodes* responsibilities can include data (pre-)processing, hosting middleware and other applications, and maintaining communications between end-devices and the cloud (including external third-party services) [48].

Another complementary concept is the one of edge computing (also known as mist computing). While both fog and mist/edge devices are at the *edge* of the network (on-premises), their differences reside on the device typology. Fog devices have, typically, more processing power, have a direct Internet connection, and can act as gateways for other devices. All the other devices that do not share these features fall under the mist category of devices. These devices, which are, mostly, the IoT sensors and actuators, have, typically, high computational constraints and leverage network protocols beyond the typical TCP/IP based ones [48].

However, as computing capabilities increase, some mist devices have enough computing capabilities to perform at least low-level analytics and filtering to make fundamental decisions, allowing for distributing computing tasks among them (making these constrained devices more autonomous and fault-tolerant) [48,49].

Cloud, fog, and edge computing paradigms gave rise to the widespread dissemination of the three-tier architectural hierarchy common in IoT systems, as depicted in Fig. 4. This architecture is the foundation behind the architectural pattern *Fogxy* [51] (where these *tiers* are considered *layers*) and is observable in diverse IoT systems [52]. A complementary layered architectural can be seen in Fig. 5.

In the lower level are the IoT devices (*i.e.*, embedded systems and sensors/actuators), the edge/mist tier devices. After, and close to the edge-devices are the *fog nodes* which together make the fog tier. On the top are the data centers, the cloud tier. The applicability of this architectural hierarchy is visible in enterprise solutions such as the RedHat IoT enterprise architecture [52].

Despite the appearance of the presented concepts of fog and edge computing in several scientific works and practitioners' communications, there is still (1) a lack of consensus on the concrete definitions of what constitutes a fog or edge device and (2) a certain degree of ambiguity for the concept of both Fog Computing and Edge Computing (as an example, Edge Computing appears in the Cloud Computing literature as the use of computational resources geographically-closer to their clients/users). In this work, the presented concepts of fog and edge will be the ones used.

4. Designing IoT systems

The design of software systems is the *creative activity in which you identify software components and their relationships* based on the system's requirements (*i.e.*, *design is about how to solve a problem*), and the most common outputs of this process are the system's models and documentation [29].

Most of the logical architectural concerns of IoT systems are unique to each application domain, such as application features, classes, and relationships, timing constraints, and states. However, the physical architecture tends to exhibit more commonalities across different IoT systems as it is mainly concerned with [53]:

1. *On which devices do the system's components live?*
2. *How are the computers and hardware devices connected and how do they communicate?*
3. *In which software modules, packages, or components (physical containers) is the system divided? And what are the system dependencies?*

The physical architecture is, in this way, highly influenced by the advancements of hardware capabilities, communication features, computational resources, energy management, and reliability among others.

4.1. Physical platforms and devices

The hardware (*e.g.*, devices, computers) in which these systems (or components) *live*, ranges from low computing power edge devices to high-performance cloud servers.

Most of the cloud part of IoT systems use software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) provided by several vendors such as Microsoft Azure, Amazon Web Services (Amazon AWS) or Google Cloud Platform (Google Cloud) [16,19,54,55]. These providers and others have been developing new and tailored solutions (*i.e.*, computational platforms) to meet the requirements of IoT in terms of cloud resources.

Different surveys on IoT cloud providers identify and compare the different available solutions, including the AWS IoT Platform, Kaa IoT Platform, IBM Watson IoT Platform, Microsoft Azure IoT Platform, Bosch IoT Suite, Google Cloud IoT, ThingSpeak and ThingWorx. P.P. Ray survey of 26 different IoT clouds exhaustively compares the available solutions in terms of suitability and applicability in terms of support for application development, device management, system management, heterogeneity management, data management, analytics, deployment management, monitoring, and visualization [19]. In his work, he points out several issues across different solutions, namely, standards conformance, heterogeneity adaptation, context awareness, incentives to the proliferation of vertical silos, issues dealing with things identities, and lack of considerations for fault-tolerance and energy management.

While different cloud providers favor different protocols (*c.f.*, Sections 4.2 and 4.3), operating systems (*e.g.*, Amazon FreeRTOS and Microsoft Windows 10 IoT), and provide different Software Development Kits (SDK), they also provide a common set of services thus reducing the heterogeneity between them (although having different commercial names).

Regarding the lower tiers of the IoT system (*viz.* Fig. 4), namely the *fog* and *edge* tiers, the device and platform heterogeneity sharply increases. Most of these devices can be placed into three groups, which are presented next, along with some examples of hardware devices that were later searched on Google Scholar for a reference metric of popularity (*i.e.*, number of results) as plotted in Fig. 6. The most common device taxonomy is (1) Single-Board Computers, (2) Single-Board Microcontrollers, and (3) Networking Devices. Most of these devices have several aspects that should be considered when choosing the most appropriate for a given scenario, such as (a) computing capabilities, (b) power consumption, and (c) connection range [56].

4.2. Lower-layers communication protocols

Although wire-based protocols are still widely used due to their reliability, with one of the most common ones being Ethernet (*e.g.*, gateways and other fog devices are, typically, connected via Ethernet to routers or other local network devices), these protocols are limited to physical installations and limit mobility. Other wired protocols[7] that still play a role in IoT include the Controller Area Network (CAN) bus protocol (which is widely used in the automotive industry), Modbus (which has become a *de facto* standard

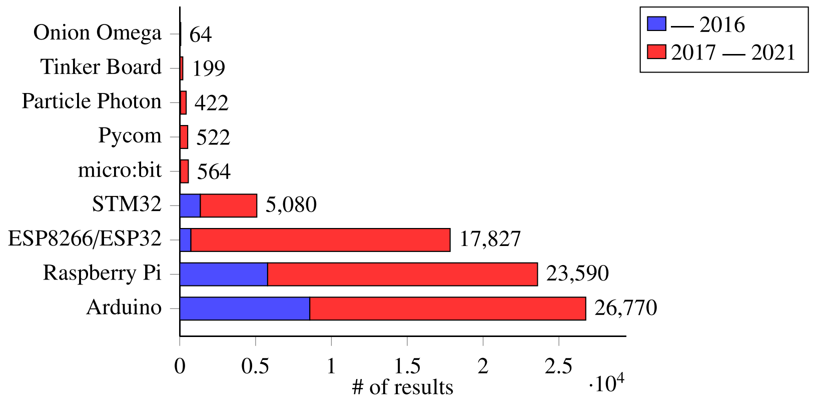

Fig. 6. Number of results on Google Scholar for most-known hardware boards. Query (``<board_name>'') AND (``IoT'' OR ``Internet-of-Things'' OR ``Internet of Things'') circa November 2021.

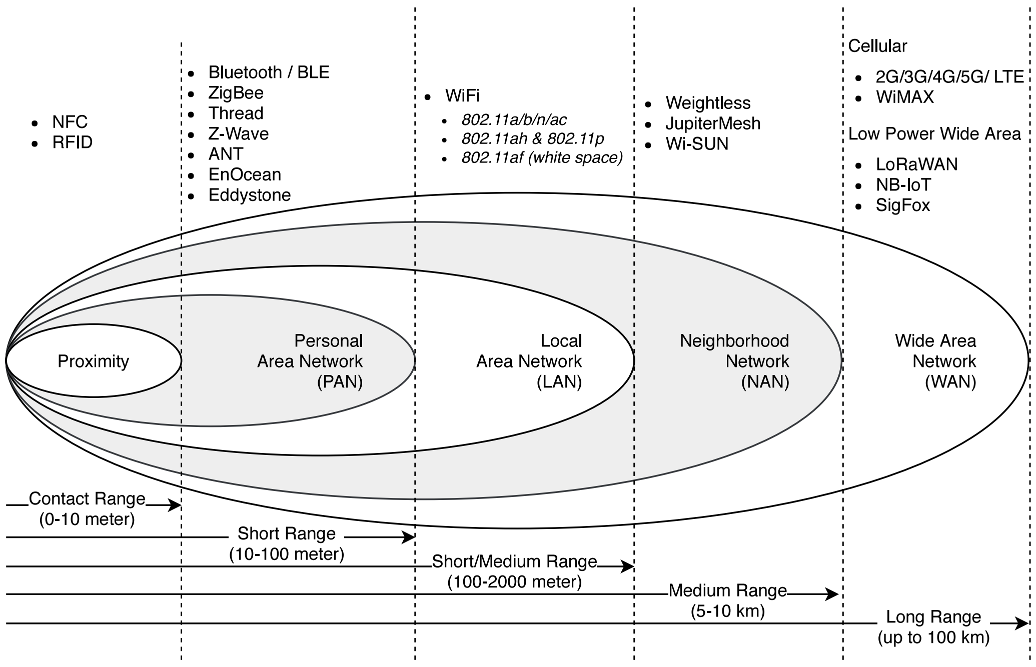

Fig. 7. An overview of the IoT-enabling network protocols pointing their fitness to each spatial scope (*i.e.*, coverage range), namely: Personal Area Network (PAN), Local Area Network (LAN), Neighborhood Area Network (NAN) and Wide Area Network (WAN).

for industrial PLC communications), and power-line communication (also known as power-line carrier or PLC) widely used for transmitting data using AC power lines [57].

Protocols targeting lower layers (per the OSI model) have been specifically designed to address necessities shared by IoT systems. Some of these are transparent to mechanisms used in higher layers while others pose practical limitations or enforce specific strategies (see Fig. 7). A few of these protocols are the following:

RFID *Radiofrequency identification* is the foundation technology for several solutions that provide automated wireless identification and tracking capabilities. It is more robust than the barcode system and consists at least of a reader (interrogator) with a suitable antenna and tags (transponders) which are microchips combined with an antenna in a compact package. A

[7] While there are several other lower-level wired communication protocols used for establishing communication between sensors, actuators, and microcontrollers — *e.g.*, I2C, JTAG, SPI, RS232, and several other serial-based communication protocols — these are not of relevance for this work as their complexity impact is, typically, limited to the embedded systems' development.

popularized RFID-based solution is the high-frequency Near Field Communication (NFC) which operates at a close range of about 10 cm or less [58].

Bluetooth/BLE *Bluetooth* objective was to replace short-range wired communication, typically used in point-to-point or star topologies. However, it has a weak security layer. More recently, *Bluetooth Low Energy* (BLE) appeared, providing considerably reduced power consumption and cost while maintaining a similar communication range [59]. Beacon protocols such as Eddystone, iBeacon, AltBeacon and GeoBeacon are all based on BLE [60].

LoRa and LoRaWAN *LoRa Low Power Wide Area* proprietary technology that supports the use of low power, low data requirements, and long-range operation devices. LoRaWAN defines the communication protocol and system architecture for the network while the LoRa physical layer enables the long-range communication link [61]. A single LoRaWAN gateway can collect data from thousands of nodes deployed kilometers away [62].

SigFox SigFox aims to deploy a controlled network dedicated to IoT, much like a cellular network. A Sigfox certified transmitter must be added to the devices, then the data transmitted by the device is first routed to the Sigfox server to scrutinize for integrity and security of the data, following which, it is routed back to applications IT network [63].

ZigBee A common wireless networking standard for building sensor networks. It consists of a packet-based open and global protocol that has been designed to provide a secure and reliable communication architecture transmitting small data at low power over short distances [63]. The next generation of ZigBee is under development as an attempt to unify some of the available protocols that follow the IEEE 802.15.4 specification, so-called Matter [64].

Thread Based upon the IPv6 and 6LoWPAN protocols, *Thread* is a low-power mesh network protocols. It provides suppliers with the freedom to use proper application layers according to their application needs [65].

EnOcean A wireless energy harvesting technology (self-powered) that delivers a high data rate at low energy consumption, adequate for short-range, less data-heavy applications. Thus, *EnOcean*-compliant devices employ energy converters which reap power from surroundings (*e.g.*, harnessing temperature difference), thus requiring no batteries [63].

ANT Proprietary wireless sensor network technology that enables semiconductor radios operating in the Industrial, Scientific and Medical allocation of the RF spectrum (*ISM band*) to communicate by establishing standard rules for co-existence, data representation, signaling, authentication, and error detection.

GPRS/2G/3G/4G/5G cellular Using the already-in-place ubiquitous cellular networks for machine-to-machine communications in the IoT domain has been explored in the last few years, with several technologies being introduced for cellular IoT networks, such as Extended Coverage GSM IoT (EC-GSM-IoT), Long Term Evolution (LTE-M), Narrowband Internet of Things (NB-IoT) [63,66]. The next fifth-generation (5G) Radio Access technology, is promised to be a key component of the *Networked Society*. 5G will support massive numbers of connected devices and meet the real-time, high-reliability communication needs of mission-critical applications [61].

Wi-Fi Is one of the most well-established protocols for wirelessly connecting local area devices, typically used to provide Internet connection to them. It is based on the IEEE 802.11 family of standards and operates in the 2.4 GHz and 5 GHz bands (Wi-Fi 6E also uses the 6 GHz band) [67]. Some extensions to the protocol have been proposed, including the support for mesh networking [68]. Another related protocol is the WiMAX one, based on the IEEE 802.16, which focus on providing last-mile wireless broadband access [69].

Less common protocols examples include Z-Wave, JupiterMesh and Wi-SUN. And, despite this extensive list of protocols enabling communication between different parts of the IoT systems across different tiers (*viz.* Fig. 4), new protocols are still being designed and implemented due to limitations imposed by the scope of operation of these systems. As an example, most of the presented communication protocols are not suitable for Vehicle-to-Vehicle Communication (V2V), and several solutions such as vehicle ad hoc networks (VANETs), and mobile ad-hoc networks (MANETs) have been studied as a solution for such issues [70].

Several works have been focused on using ultrawideband (UWB) as a low energy level for short-range and high-bandwidth communications, with some commercial-grade devices (*e.g.*, mobile phones) already using it for precision locating and tracking purposes [71]. Some sub-GHz ISM bands (license-free Industrial, Scientific, and Medical frequency bands) are also popular amongst IoT, specifically 433 MHz, 868 MHz, and 915 MHz — with more than 5270 works on Google Scholar by November 2021. The data transmitted in these frequencies can be modulated using a wide range of protocols (*e.g.*, LoRa runs on 433 MHz in the USA and 868 MHz in Europe) and they are typically low-powered.

Additionally, other *wireless* protocols, including light-based ones, have been proposed as a way of mitigating some of the issues (*e.g.*, electromagnetic interference) that affect most of the protocols above to some degree. Among such initiatives, one can identify Li-Fi (Light-Fidelity), a derivative of optical wireless communications (OWC) technology, that uses light-emitting diodes (LED) for transmitting data [72]. Another widely-used light-based communication technology is infrared (IR), which has been historically used (standardized by IrDA and IEEE in the early 90s) as a low-range communication protocol solution and still plays a role in IoT due to its low power consumption [73,74].

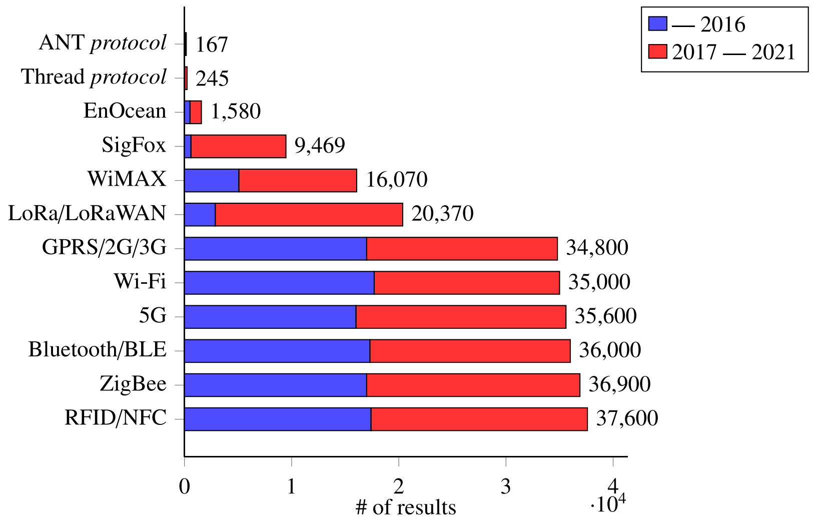

Fig. 8. Number of occurrence of each protocol in the literature gathered using Google Scholar with the search query `<protocol_name> AND (''IoT'' OR ''Internet-of-Things'' OR ''Internet of Things'')`, where `<protocol_name>` is the name of the protocol or a combination of two versions of it (*e.g.*, `Bluetooth OR BLE`). Some protocols required the use of the word *protocol* in the query to disambiguate the term. Data collected in November 2021.

We are still far from a reference (standard) set of protocols a designer can objectively choose from when developing IoT systems. As can be observed in Fig. 8, several network protocols appear recurrently in the literature. While not every protocol can be used for all kinds of networks, there are some overlapping protocols in terms of range and functioning (*e.g.*, BLE and ZigBee). They are indistinguishable in terms of popularity, which goes accordingly with the current fragmentation (*i.e.*, heterogeneity) of the IoT ecosystem.

4.3. Higher-layers communication protocols

Some of the most common higher-layers communication protocols (per the OSI model[8]) that are commonly used in IoT systems are the following [75]:

HTTP *Hyper Text Transfer Protocol* is the most commonly used application layer. However, its inherent complexity and verbosity make it an undesirable candidate for IoT applications [63]. HTTP/2 further reduces the network latency, compresses the header-field data, and allows parallel data exchanges on the same network. While it is still complex and verbose, it provides advantages regarding bandwidth, reliability, and messaging mechanisms. Representational State Transfer (REST) architectural style is often used in both [76].

The third-version of HTTP (HTTP/3 or HTTP-over-QUIC[9]) improves the current version 2 by supporting multiplexing (reducing the impact of packet loss on data transfer speeds) and lessen the handshake scheme [78], however, there is still no references to the protocols in the context of IoT.

MQTT *Message Queue Telemetry Transport* was released by IBM and targets lightweight M2M communications. It is an asynchronous publish/subscribe protocol that runs on top of the TCP stack and a typical network configuration consists of a server and several clients [79]. Small modifications on this protocol originated the *Message Queuing Telemetry Transport for Sensor Networks* (MQTT-SN) [80], MQTT over WebSockets that leverage the use of MQTT on top of WebSockets [81], and MQTTw/QUIC which uses QUIC as the transport protocol [82].

AMQP *Advanced Message Queuing Protocol* provides asynchronous publish/subscribe communication with messaging. Its main advantage is its store-and-forward feature that ensures reliability even after network disruptions, which is a common issue of IoT networks. Further, it provides flexible routing and transactions support [83].

CoAP *Constrained Application Protocol* was designed for IoT devices being deployed in low-power lossy networks, working in a REST-style but using datagrams for data transmission [63]. CoAP holds various functional characteristics such as multicast

[8] Open Systems Interconnection model (OSI model) splits network communication protocols into seven layers — from the physical lower-level layer to the higher-level application layer.
[9] QUIC is a *multiplexed and secure general-purpose transport protocol* on top of UDP [77].

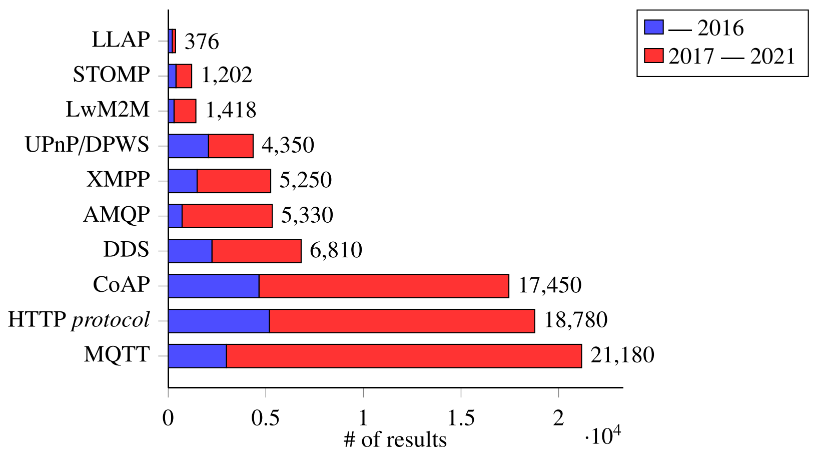

Fig. 9. Number of occurrence of each protocol in the literature gathered using Google Scholar with the search query `<protocol_name> AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<protocol_name>` is the name of the protocol or a combination of two versions of it (*e.g.*, `Bluetooth OR BLE`). Some protocols required the use of the word *protocol* in the query to disambiguate the term. Data collected in November 2021.

Table 2
Basic overview of some of the most common higher-layer protocols used in IoT systems.

Protocol	Message Pattern	QoS	Security	Addressing	REST	Constrained Devices
MQTT	Pub–Sub	•		•		•
CoAP	Request–Response	•	DTLS	•	•	•
XMPP	Pub–Sub		TLS	•		•
AMQP	Point-to-PointPub–Sub	•	TLS	•		
DDS	Pub–Sub	•		•		•
HTTP/2	Request–Response	•	TLS	•	•	•
STOMP	Pub–Sub	•	No			

support, small data overhead, Uniform Resource Indicator (URI) support, content-type support, subscription of a resource, and surfing push notifications [60]. Although it was originally designed to use UDP as an underlying transport protocol, it can be used over TCP, TLS, and WebSockets.

XMPP *Extensible Messaging and Presence Protocol* is an open protocol that provides facilities such as lightweight middleware, instant messaging, real-time communication, voice/video call, and multi-party chat, and content syndication of XML data [63]. XMPP runs over TCP, and it is suitable for IoT applications due to its publish/subscribe (asynchronous) and also request/response (synchronous) messaging systems, thus being extensible and flexible [76].

LwM2M *Light-weight Machine-to-Machine* is a standard that provides an interface between M2M devices and M2M Servers to build an application-agnostic scheme for management and communication between a variety of devices. It is built on top of the CoAP protocol [84].

LLAP *Lightweight Local Automation Protocol* is designed to facilitate the transmission of short messages among IoT devices [60].

UPnP Universal Plug and Play (UPnP) is built on top of the Plug and Play peripheral model and uses existing protocols (*e.g.*, UDP and HTTP) to enable devices to dynamically join a network (automatically setting all the necessary information, including IP addresses). UPnP also defines mechanisms for devices to announce themselves and their capabilities to other UPnP-enabled devices and automatically retrieve the relevant information about other compliant devices in the network, without requiring any manual configuration (*i.e.*, zero configuration or *zeroconf*). Device Profile for Web Services (DPWS) was introduced as a successor of UPnP but fully aligned with Web Services' vision. It defines a set of requirements that enable secure web-based messaging, discovery, description, and eventing, with a particular focus on resource-constrained devices [85].

An introductory comparative overview of some of these protocols given their relevance in the IoT ecosystem is given in Table 2. Other protocols include the Data-Distribution Service (DDS) and Streaming Text Orientated Messaging Protocol (STOMP). Several studies have been carried to evaluate metrics on some of the above-listed protocols, such as power consumption, bandwidth, and latency [15,83,86]. A popularity study based on the number of results on the literature (from Google Scholar) can be found in Fig. 9, and while there is a considerable amount of results for most protocols, there is a clear tendency in using MQTT, HTTP, and CoAP. MQTT and CoAP are lightweight and simple-to-use protocols that were designed for IoT, thus their popularity. HTTP is one of the long-establish protocols, and its use is common when IoT devices have the necessary computational resources.

4.4. Architectural styles

The Internet-of-Things leverages well-known building blocks of several software architectures in a unified fashion, where smart objects and humans responsible for operating them (if needed) are capable of universally and ubiquitously communicating with each other. Like almost any other computational system, IoT ones have an organizational structure, viz. system architecture.

At a high-level view, the architecture of IoT systems has the aforementioned three-tiered organization, *i.e.*, edge, fog, and cloud (see Fig. 4). However, when considering the particularities of IoT systems, the architecture needs to guarantee flawless operation of its components and *fuse* the physical and virtual realms. For reaching this objective, the architecture needs to be dependable, adaptable, highly scalable, human-centric, and handle dynamic interactions [33].

Some architectures are more suitable for the requirements of several IoT applications, such as low-power consumption, lightweight data transmission, messaging mechanisms, and reliability.

Regarding the software modules, packages, or components of IoT systems, the architectural interpretation of its division depends on the abstraction layer being used. Most modern IoT systems of modest-to-high complexity now tend to follow a Service-Oriented Architecture (SOA). Notwithstanding there are several other architectures that are being used for designing these systems, which can be used both jointly or individually. A mapping survey by Muccini et al. identifies the following as the most common architectural styles in IoT while mentioning that most systems use an overlay of one or more styles in their architecture [87]. The following presented some styles resulting from their mapping study with some additional relevant styles.

Layered Systems are built by a combination of several, heterogeneous, parts that interact between them. The number of layers goes from 3 to 6 across implementations, where the 3-layer architecture has a *perception*, *processing and storage*, and *application* layers and the 6-layer one presents the additional *adaptation*, *network*, and *business* layers [87].

Service-Oriented Architecture in which services are provided to other components or subsystems by the use of a common communication channel. As such, it is suitable for the connectivity, interoperability, and integration needs of IoT systems, where sensing and actuating services are highly distributed by design [88–91].

Microservices Applications that follow a microservices architecture are structured as a set of services that are, as per Chris Richardson: (1) highly-maintainable and testable, (2) loosely coupled, (3) independently operable, and (4) organized around business capabilities [92]. In IoT context, this architecture fits the SoS paradigm, adding the ability to encompass the different components and system parts that can be spread in both logical and geographical terms, while making it possible to assure scalability, extensibility, fault-tolerance, and scatter of services among tiers [90,93–95].

Event-driven Event-driven Architectures (EDA) focuses on events — significant changes in the state of the system — and deals with the production, detection, consumption, and reaction to those events. IoT systems typically are built-on architectures that are already event-based, such as publish–subscribe (also known as *pub-sub*) — *architecture that states that producers publish events on an event bus (or message bus) and consumers subscribe to the events they want to receive from that bus* [96] — and Complex Event Processing — *architecture to combine events from diverse sources, looking for patterns in these event streams and then typically responding in real-time* — which provides aid in dealing with the ever-growing scale of IoT systems [51,88,97–99].

Broker and Mediator Broker architecture is commonly used for decoupling higher system tiers from the underlying tiers, *i.e.*, decoupling applications from the underlying sensing and actuating devices. A broker is a core component of most of the publish–subscribe architectures due to the dependency of a middle-man that receives messages from publishers and delivers them to different subscribers [100–102]. Mediator architectures, supported by gateways and implemented using bridges and adapters, assure the communication between the devices and applications and can have responsibilities such as collecting and pre-processing data from multiple devices, device management capabilities, and assuring security and privacy [103,104].

Client–Server One of the most common architectures on the Internet is that of a client and a server connected by some protocol such as Representational State Transfer (REST[10]) or Simple Object Access Protocol (SOAP) [33,105,106]. Some IoT systems also use this architecture, and due to the constraints of some devices, new protocols have been developed as a way to meet this constraint, making them able to use the client–server architecture such as CoAP [107,108].

In their work, Muccini et al. adds Cloud-based style to this list — which refers to the use of cloud computing as a core part of the system — and Information-Centric Networking (ICN) style — that focus on making device communication information-centric — however, these overlap the already presented styles [87]. Some other but less relevant architectures that also appear in this context are Peer-to-Peer, Space-based architecture, and Shared Nothing architecture, but their repeated usages in the literature are mostly unnoticeable [109,110].

Looking at the popularity of these architectures (Fig. 10), we see that one of the most common is the layered architecture. This is understandable, as the IoT systems typically have components dispersed among different tiers, which typically correlate with layers, and a layered architecture can co-exist with other architectures (*e.g.*, client–server). Further, systems have been long designed in

[10] Also known as RESTful, which is used to describe services that follow a REST architecture.

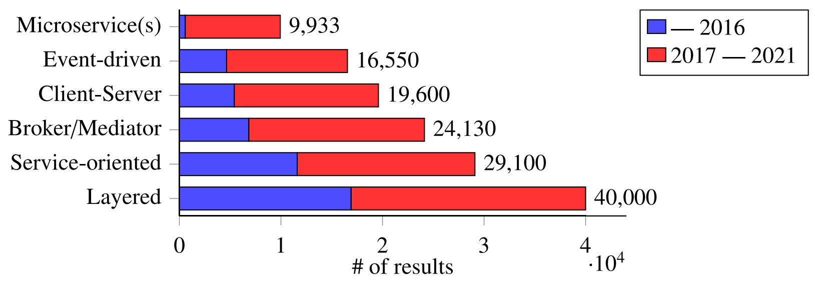

Fig. 10. Number of occurrence of each architectural style in the literature gathered using Google Scholar with the search query `‘‘<architecture_name> architecture’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<architecture_name>` is the name of the architecture. Data collected in November 2021.

Table 3
Reference architectures for IoT [112].

Initiative Name	Description
Reference Architecture Model Industrie 4.0 (RAMI 4.0)	A reference architecture for smart factories dedicated to IoT standards, which started in Germany and today is driven by all major companies and foundations in the relevant industry sectors.
Industrial Internet Reference Architecture (IIRA)	The Industrial Internet Consortium (founded by AT&T, Cisco, General Electric, IBM, and Intel) has delivered a reference architecture with a special focus on the industrial application of IoT.
Internet of Things-Architecture (IoT-A)	The IoT-A delivered a detailed architecture and model from the functional and information perspectives of the IoT system.
Standard for an Architectural Framework for IoT	The IEEE P2413 project has a working group on the IoT's architectural framework, highlighting protection, security, privacy, and safety issues.
Arrowhead Framework	Focus on enabling collaborative automation by open-networked embedded devices. It is a major EU project to deliver best practices for cooperative automation.
WSO2's Reference Architecture	A five-layer architecture with a special focus on modularity, which makes it suitable for a wide array of use cases. It also suggests a core set of communication protocols to be used.
IoT Architecture Reference Model (ARM)	Based on IoT-A, it focuses on presenting a reference organization of the basic functional components without detailing their interactions. From an information viewpoint, the core part is virtual entities (*i.e.*, digital twins).

layers according to the responsibilities of the components that are part of it (*e.g.*, business layer *vs.* application layer). This view also fits with some IoT systems' necessities.

Most of the remaining architectures can coexist; thus, their popularity is not an indicator of exclusivity — *i.e.*, a system can use several architectures in their different parts. Nonetheless, there is a trend towards the use of service-oriented architectures (SOA), as it is common to other kinds of software systems (*e.g.*, Web services). Additionally, broker-mediator architectures are highly used, which is correlated with the popularity of protocols such as MQTT and AMQP.

4.5. *Reference architectures*

Reference architectures and models give a bird's eye view of the whole underlying system; hence, their advantage over other architectures relies on providing a better and higher level of abstraction. They provide template solutions for an architecture for a particular domain (in this case, the IoT domain), hiding specific constraints and implementation details, aiming to help developers meet the development challenges of the domain [111].

Several entities are proposing and pushing into the market different reference architectures on how to develop systems that meet IoT characteristics and requirements. Weyrich et al. [112] survey circa 2016 refers that a fast-growing number of initiatives have been working towards standardized architectures for the IoT domain, aiming to facilitate interoperability between different solutions, simplify development, and ease implementation. Their work points out five major IoT reference architectures, namely: Reference Architecture Model Industrie 4.0 (RAMI 4.0), Industrial Internet Reference Architecture (IIRA), Internet of Things-Architecture (IoT-A), Standard for an Architectural Framework for the Internet of Things (IoT) and Arrowhead Framework. Also, they note the importance and relevance of initiatives to define machine-to-machine (M2M) communication standards and other initiatives related to the IoT domain. Saemaldahr et al. [113] survey circa 2020 found some other IoT-focused reference architectures, including the WSO2's Reference Architecture and IoT Architecture Reference Model (ARM). A summary of the hereby enumerated reference architectures for IoT is given in Table 3, detailing their focus and presenting some additional information [112,113]. An overview of the appearances of these reference architectures in the literature is given in Fig. 11.

Some practitioners also present their vendor-specific reference architectures which are tailored to their services and solutions, including IBM Internet of Things architecture [114], Azure IoT reference architecture [115], and AWS Architecture Best Practices for IoT [116].

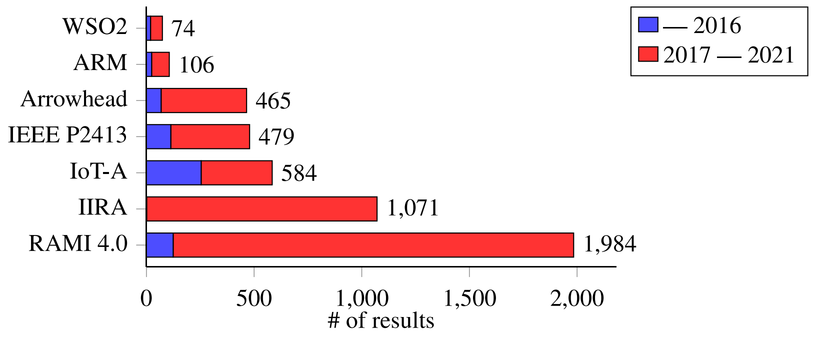

Fig. 11. Number of occurrence of each reference architecture in the literature gathered with the search query `‘‘<architecture_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<architecture_name>` is the name of the architecture. Data collected in November 2021.

Table 4
Overview of the IoT interoperability enabling models and APIs.

Name	Description
Web of Things (WoT) [118]	Common data model and API for the Internet-of-Things, with a focus on web-based interaction (*i.e.*, Web of Things).
IOTDB [119]	A semantic layer for the IoT, which includes definitions for all the data to provide formal definitions for all important items and unlimited expandability.
Web Thing Model [120]	A common model to describe the virtual counterpart of physical objects in the Web of Things.
OGC SensorThings API (SensorML) [121]	An open, geospatial-enabled and unified way to interconnect IoT devices, data, and applications over the Web.
SENML [122]	The Media Types for Sensor Markup Language is a standard for representing simple sensor measurements and device parameters.
LsDL [60]	The *Lemonbeat smart Device Language* is a XML-based smart devices encoding language that is read as Lemonbeat smart Device Language (LsDL).

4.6. *Interoperability*

Different entities have been working on different standards to ensure a semantic interoperable Internet-of-Things, establishing a common language that devices can speak between them and different applications, envisioning a reduction in IoT fragmentation. A summary of the most known initiatives is given in Table 4 [117], and the most active ones are analyzed in the following paragraphs.

Web of Things (WoT) [11] A standard data model and API for the Web of Things. The Web Thing item provides a vocabulary for describing physical devices connected to the World Wide Web in a machine-readable format with a default JSON encoding. The Web Thing REST API and Web Thing WebSocket API allow a web client to access the properties of devices, request the execution of actions, and subscribe to events representing a state change. Some fundamental Web Thing Types are provided, and additional types can be defined using semantic extensions with JSON-LD. Also, the proposal includes details on Web of Things Gateway Protocol Bindings which proposes non-normative bindings of the Web Thing API to various existing IoT protocols and a set of Web of Things Integration Patterns which contains advice on which design patterns can be used for integrating connected devices with the Web of Things, and where each pattern is most appropriate [118].

OGC SensorThings API by OGC An open, unified and geospatial-enabled way to interconnect the Internet-of-Things (IoT) devices, data, and applications over the Web purposed by the Open Geospatial Consortium (OGC). At a high level, the OGC SensorThings API provides two main functionalities, and a part handles each function. The two parts are the Sensing part and the Tasking part. The Sensing part provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems while the Tasking part provides a standard way for parameterizing (*i.e.*, *tasking*) taskable IoT devices [121].

IOTDB Things are described by semantically annotating the *data associated with the Thing*, being this item built from *composition* of *atomic* elements (that cannot be meaningfully subdivided further) and are *extensible* (allowing to add in elements from other Semantic ontologies). The Things can have many different *bands* of data associated with them (*e.g.*, the metadata, the actual state) [119].

[11] Part of the Mozilla WebThings initiative which is now an independent project.

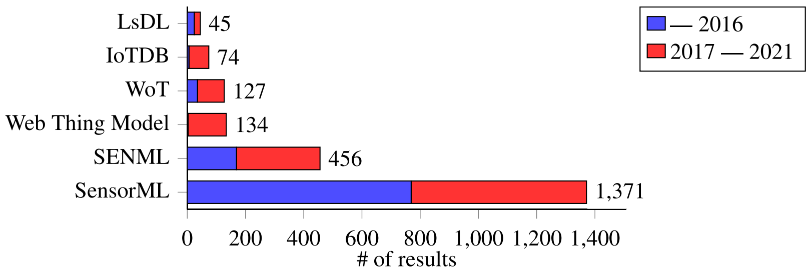

Fig. 12. Number of occurrence of each interoperability standard in the literature gathered using Google Scholar with the search query `''<standard_name>'' AND (''IoT'' OR ''Internet-of-Things'' OR ''Internet of Things'')`, where `<standard_name>` is the name of the interoperability standard. Data collected in November 2021.

There is, however, no consensus on what is the most suitable standard for a particular scenario nor a commonly accepted standard that assures semantic interoperability among IoT systems and applications [123]. Additionally, it is noticeable that little to no focus has been given to these questions from a literature standpoint, as can be seen by the few results displayed in Fig. 12.

The lack of consensus on interoperability standards is one of the most pressing issues that IoT faces, influencing all system stakeholders. From an end-user viewpoint, they do not know for how long the devices that they brought will be supported by the different system parts (*e.g.*, a light bulb that is no longer supported by the integration gateway). The same happens from the vendor point-of-view, since designing a system using the latest available standard proposal does not ensure that it will be adopted, or deprecated even before it reaches the market (which pressures vendors to define their in-house solutions contributing to today's fragmentation) [124].

Noura et al. [125] defines a taxonomy for interoperability in IoT systems, analyzed from different perspectives: (1) device, (2) network, (3) syntactic, (4) semantic, and (5) platform level. They conclude that most IoT interoperability proposals focus on a single, specific perspective rather than embracing different ones. They also propose the use of semantic web technologies along with inter-working APIs as a good foundation for providing cross-platform interoperability. From the analyzed standards we believe that the one that goes more accordant with such view is the Mozilla Web Thing proposal.

4.7. *Design patterns*

IoT is a relatively recent field for both academia and industry, resulting in large amounts of knowledge being created rapidly and disseminated in a variety of formats. The system's designers are tasked to pick and represent the best solutions that cope with the desired scenario mostly based on empirical evidence, usually captured as standard reference architectures and case studies. However, most information on IoT systems' design is *widespread, diffuse, hard to handle, redundant, ever-changing* and sometimes even *unorganized* as different authors chose different vocabulary to target the same things (ambiguity).

Christopher Alexander was faced with a similar challenge regarding civil architecture and came up with the idea of *patterns* — recurrent problems with recurrent solutions — to capture this widespread knowledge coherently. The software engineering community later borrowed this concept as a way to capture and share practical knowledge and experience [126–128].

The work by Washizaki et al. attempt to map the landscape of IoT patterns resulted in a total of 136 patterns scattered among abstraction levels and IoT layers, and the identified works are included in Table 5, but their work does not encompass several works due to the reduced number of scientific databases considered while encompassing others that does not focus on design patterns per se [20,144].

Thus, using as knowledge base the *software engineering pattern literature*,[12] we surveyed it for works specifically concerning IoT or for nearby fields (such as *cloud*) that we believe share a considerable subset of concerns with our domain. We only considered works that identify patterns as a primary outcome.

The summary of the analyzed publications is presented in Table 5, and the identified design patterns in the context of IoT systems are listed in Table 6. An overview of the literature shows that various works focusing on patterns for these systems and their life-cycle, primarily focus on the *edge* tier, *viz.* the *things* themselves. Even though we also found existing work tackling cloud computing, there is a (somewhat variable-length) gap concerning the IoT-specific *cloud* and *fog* tiers, and on how all of them come together.

This latter fact raises the hypothesis that there may exist a non-negligible amount of knowledge in other areas of software engineering that is not being researched for IoT. Being aware of this could probably help bootstrap well-known software development practices (*e.g.*, continuous integration, continuous development, isolation/containerization, fuzzing and fault injection) for our target systems, as they share similar, if not equal, characteristics, such as distributed computing, fault-tolerance, and large-scale systems orchestration.

[12] Patterns and pattern languages are usually published in the PLoP conference series and made available in the ACM-DL.

Table 5
The landscape of relevant literature about design patterns for the Internet-of-Things, and their relevance for each IoT architectural tier by ranking, from no relevance ( ) to most relevant (• • •).

		IoT tier		
		Cloud	Fog	Edge
IoT Focused	IoT Patterns for Device Bootstrapping and Registration [129]	•	•	• • •
	IoT Security Patterns [130]		•	• • •
	Internet of Things Patterns [131]		•	• • •
	Internet of Things Patterns for Devices [132]			• • •
	IoT design patterns: Design, build and engineer edge applications [133]	•	• •	• • •
	Design patterns for the industrial Internet of Thing [134]	• •	• •	• •
	Patterns for Devices: Powering, Operating, and Sensing [135]			• • •
	Fogxy — An Architectural Pattern for Fog Computing [51]	• • •	• • •	• • •
	Patterns for Things that Fail [136]	•	• •	• • •
	Testing and Deployment Patterns for the Internet-of-Things [137]	• • •	• • •	• • •
	A Pattern-Language for Self-Healing Internet-of-Things Systems [138]	• • •	• • •	• • •
Relevant for IoT	Cloud Computing Patterns [139]	• • •		
	A Pattern Language For Microservices [92]	• • •	•	
	Continuous Integration: Patterns and Anti-Patterns [140]	• • •		
	Patterns for Fault Tolerant Software [141]			• • •
	Designing Distributed Control Systems: A Pattern Language [142]	• •	• •	•
	Patterns for software orchestration on the cloud [143]	• • •		

Table 6
List of the IoT-focused design patterns grouped per primary concern.

Concern	Pattern
Operation Mode	Always-On Device [135], Normally-Sleeping-Device [135]
Energy Supply	Mains-Powered Device [135], Lifetime Energy-Limited Device [135], Energy-Harvesting Device [135], Period Energy-Limited Device [135]
Bootstrapping	Factory Bootstrap [129], Medium-based Bootstrap [129], Remote Bootstrap [129]
Registration	Device Registry [129,136], Manual User-Driven [129], Automatic Client-Driven [129], Automatic Server-Driven [129]
Device Model	Device-Driven Model [129], Pre-Defined Device-Driven Model [129], Server-Driven Model [129]
Communications	Delta Update [132], Device Gateway [131], Device Shadow [131], Device Wakeup Trigger [131], Visible Light Communication [132], Remote Device Management [132], Simulation-based Testing [137], Testbed-based Testing [137]
Deploy	Update Delegation [137], Edge Code Deployment [133]
Security	Trusted Communication Partner [130], Outbound-Only Connection [130], Permission Control [130], Personal Zone Hub [130], Whitelist [130], Blacklist [130], Remote Lock and Wipe [131]
Architecture	Fogxy [51]
Processing	Rules Engine [131]
Sensing	Event-Based Sensing [135], Schedule-Based Sensing [135]
Fault-Tolerance	Device Raw Data Collector [136],Device Error Data Supervisor [136], Predictive Device Monitor [136], Action Audit [138], Suitable Conditions [138], Reasonable Values [138], Unimpaired Connectivity [138], Within Reach [138], Component Compliance [138], Coherent Readings [138], Internal Coherence [138], Stable Timing [138], Unsurprising Activity [138], Timeout [138], Conformant Values [138], Resource Monitor [138], Redundancy [138], Diversity [138], Runtime Adaptation [138], Balancing [138], Circumvent and Isolate [138], Debounce [138], Timebox [138], Reset [138], Checkpoint and Rollback [138], Rebuild Internal State [138], Flash [138], Compensate [138], Calibrate [138], Consensus Among Values [138]

The adoption of these existing patterns and practices in a new context should not be taken lightly. As an example, consider the case of continuous integration/delivery (CI/CD). The typical CI/CD approach assumes that low-friction tools for delivery exist (*e.g.*, a containerized pipeline) and that software is made to work in a well-defined set of hardware and/or platform configurations [145]. Although one can leverage existing tools and approaches, IoT poses special needs that make CI/CD more expensive and risky; Jim Ruehlin [146] identifies several *physical* needs and considerations beyond the *software-only* world needs: (1) Physical deployment of new sensors, (2) manual deployment of software (which implies the need for more resources and increased costs), (3) sensors or devices that are physically compromised, (4) weather conditions, (5) geographic considerations and constraints, and (6) devices and systems connectivity characteristics and issues.

These factors might hinder the feasibility of delivering new versions in a short regular schedule when compared to other software systems; which is unfortunate, since the criticality of fixing security vulnerabilities and bugs in IoT is increasingly paramount [147,148].

When taking into account the number of open technological challenges, despite the pattern mining effort done work by Reinfurt et al. [129,131,132,135] and others, the number of captured patterns is still residual. Existing patterns do not encompass the ecosystem as a whole and are tied to specific architectural tiers and specific hardware/software perspectives. Extensive work should be pursued on the systematization of existing solutions (in both academia and enterprises). We also believe that fields close to IoT

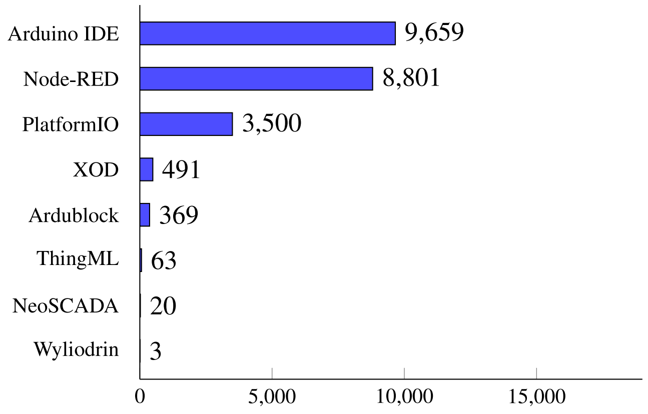

Fig. 13. Number of *stars* in the GitHub repositories of the *open-source* IoT-focused development tools (data collected in mid-2020).

should be studied as they might be of relevance when considering practices that can be adopted, *e.g.*, Microservices Patterns by Chris Richardson [92], Patterns for Fault Tolerant Software by Robert Hanmer [141], and Patterns for Software Orchestration on the Cloud by Boldt et al. [143].

Ultimately, the existence of enough IoT-specific design patterns and the discovery of relationships between them has the potential to produce a holistic *pattern language* or *pattern map* that can be shared and improved among practitioners. Pattern languages would help IoT developers by providing them with a proper vocabulary, abstractions, and insights into the solutions other developers have recurrently found.

5. Constructing IoT systems

While the design activity of the software development life-cycle focuses on reaching suitable and high-level solutions for a given problem, the construction (also known as implementation) activity of the life-cycle focuses on actually building the systems (*i.e.*, *create an executable version of the software* [29]). The construction of software systems may involve writing code in one or more high- or low-level programming languages (at different abstraction levels), or tailoring and adapting generic, *off-the-shelf* systems to meet the specific requirements of a given project, organization or scenario [29].

The heterogeneity and vastness of devices, platforms, and services that are part of IoT require development approaches that attend to this ecosystem particularities. Traditional programming — procedural computer programming — using code editors and integrated development solutions has been the go-to solution for developers and other technical individuals. However, as the heterogeneity of devices keeps increasing, as well as the number of application scenarios and environments, it became necessary to build abstractions of sensors, actuators, and whole devices as a way to reduce the complexity of developing and managing IoT systems. This need for abstractions leads to the birth of several, IoT-focused, model-driven development solutions as well as mashup-based development tools [149].

With the growing number of non-technical users that use IoT systems, even such solutions did not suffice, since they required a certain level of technical knowledge that most end-users do not possess. This lead to the birth of several low-code solutions in the field that leverage visual programming concepts, natural language processing tools, and voice assistants as a way for the users to configure (viz. program) their systems. One of the most common strategies used by these approaches is the use of *if-then* rules programming solutions (*e.g.*, IFTTT and Zapier [150])— also known as trigger-action programming (TAP) — where rules are defined as a sequence of trigger-action *flows*. More concretely, in the TAP paradigm, a trigger is followed by an action, in a mostly natural language fashion, *i.e.*, `if` *`trigger`* `then` *`action`*, *e.g.*, `if` *`I leave home`* `then` *`switch off the lights`* [151].

Fortino et al. present a similar review that we consider complementary to our study, as it focus more on the purpose of the solution (tools and platforms), encompassing characteristics such as interoperability, autonomy, scalability and smartness (*e.g.*, support of autonomic decision making), and less on the properties of the solution/language itself (*e.g.*, model-driven or visual programming) [10].

In the following paragraphs a total of 43 solutions are listed and summarized. These tools are split amongst several subsections accordingly to their most preeminent programming paradigm, which is commonly correlated with their abstraction level and easiness of use. An high level categorization of these tools is presented in Table 7, and an overview of the popularity of some of the tools typically used for IoT development is given in Fig. 13.

5.1. Traditional development

Most IoT systems result from a combination of already existing technologies and systems (viz. Systems-of-Systems). As a consequence, there has been for a long time a set of development facilities — *e.g.*, programming languages, Integrated Development

Table 7
A most comprehensive list of the available solutions to construct IoT systems. The first four columns describe which programming paradigms the solutions leverage: text-based, model-driven, visual programming, and mashup. Decentralized tab states if a solution can be used to program distributed systems transparently (*i.e.*, automatically allocating tasks to available resources). Source tab is used to identify if the solution has available code. Runtime tab identifies if the solution has any dependency on a specific runtime (*e.g.*, platform). Deployment tab specifies if the resulting software packages can be run directly on the device or have to be run in an external environment (*e.g.*, cloud) or on a specific mobile application. Lastly, the end-user tab identifies solutions focused on end-user, typically leveraging simple visual notations or other kinds of interaction (*e.g.*, conversational). N/A stands for information Not Available, • symbolizes a full support or use, and ∘ a partial/hybrid support or use.

	Text	Model	Visual	Mashup	Decentralized	Source	Runtime	Deployment	End-User
ArduBlock			•			•		Device	•
Arduino IDE	•					•		Device	
AT&T Flow Designer			•				•	External	
Atmel Studio	•							Device	
Blynk			•	•			•	External	•
Braines et al.						N/A	•	External	•
Chen et al.		•				N/A			
DDFlow			•	•	•		•	External	
DG Solution Builder			•	•			•	External	
Durmaz et al.		•	∘			N/A		Device	
Epidosite	•		•	•			•	Mobile App	•
ESPlorer	•					•		Device	
Fluidware	•	•					•		
FogFlow			•	•	•		•	External	
FRASAD	•	•				N/A		Device	
glue.things			•	•			•	External	
GraspIO			•				∘	Device	
IFTTT			•	•			•	External	•
IoT-MAP			•	•		N/A	•	Mobile App	
IoTMaaS	∘			•			•	External	
Jarvis						•	•	External	•
Kang et al.						N/A	•	External	•
Kodali et al.			•	•		N/A	•	Mobile App	∘
MDE4IoT		•	∘			•			
NeoSCADA	•	∘				•			
Node-RED			•	•		•	•	Hybrid	
Noodl			•	•			•	External	•
Platform.IO	•					•		Device	
Pymakr	•					•			
Rajalakshmi et al.			•	•		N/A	•	External	•
Silva et al.			•	•	•	•	•	Hybrid	
Siri, Alexa...							•	External	•
SysML4IoT		•	∘			•		Hybrid	
ThingML	•	•				•		Device	
UML4IoT		•	∘			•			
WComp	∘		∘	∘		N/A	•	External	
WoTFlow/DDF			•	•	•	•	•	External	
WoTKit			•	•		N/A	•	External	
WoX		•				N/A	N/A	N/A	
Wyliodrin	•		•	•			•	External	
XOD			•			•		Device	
Yao et al.			•			N/A	•	External	
Zenodys			•	•			•	External	

Environments (IDE), and other toolsets — that empower the construction of components for each tier of IoT systems (Fig. 4) and the systems as a whole. Some of these tools are vendor-specific and/or board-specific.

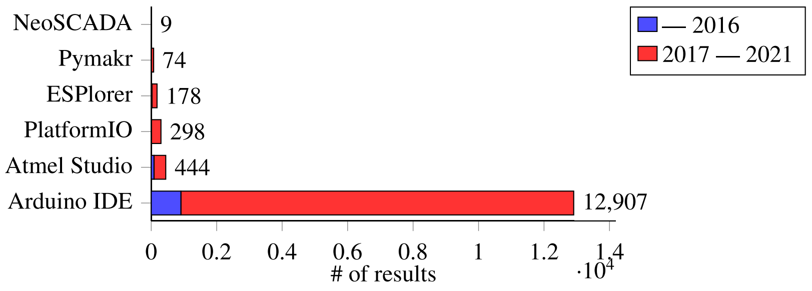

Fig. 14. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query `‘‘<tool_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<tool_name>` is the name of the development solution. Data collected in November 2021.

From a programming languages perspective, an IoT Developer Survey carried by the Eclipse IoT Working Group, AGILE IoT, IEEE, and the Open Mobile Alliance circa 2019 enumerates the most common languages for each IoT systems tier, namely (by order of relevance): (1) C, C++, Java and JavaScript for the *edge* tier, (2) Java, Python, C++ and C for the *fog* tier and (3) Java, JavaScript, Python, and PHP for the *cloud* tier [152].

Regarding development environments, it is noticeable that the most popular ones (*e.g.*, Eclipse IDE) can be extended and adapted (*e.g.*, via plugins) to encompass the reality of IoT systems development. However, new tools are developed to deal with the particularities of each tier (*i.e.*, adapted to the heterogeneity, scale, and specific concerns).

We can identify some tools as reference development environments for IoT development, such as the following ones:

PlatformIO An IDE for the Internet-of-Things with a particular focus on the programming of edge tier devices. It has support for more than 650 development boards, and comes with a built-in library manager and has a *C/C++ Intelligent Code Completion and Smart Code Linter* suitable for embedded devices. It also has debugging features (*PIO Unified Debugger*) alongside with testing features (*PIO Unit Testing*) and support for integration in local- or cloud-based CI/CD pipelines [153]. `PlatformIO` can also be used as a standalone Command Line tool or as a plugin in several code editors and IDE’s (*e.g.*, Visual Studio Code and Atom).

Arduino IDE An IDE focusing on the programming of microcontrollers based on the Arduino framework (not to be confused with the Arduino development boards based on Atmel AVR), commonly found on the edge tier. Includes support for C and C++ programming languages and is pre-loaded with several examples and libraries [154].

NeoSCADA A project from the Eclipse Foundation targeting SCADA (Supervisory Control and Data Acquisition) systems development. These kinds of systems are omnipresent in manufacturing and control operations and are now commonly Internet-connected. Eclipse NeoSCADA is not an out-of-the-box solution for SCADA systems but instead a set of development libraries, interface applications, mass configuration tools and other facilities that allow the construction of this type of systems [155].

There are also several micro-controller-specific development tools including, among others, (1) the ESPlorer, an Integrated Development Environment (IDE) for ESP8266-based boards using Lua or MicroPython languages [156], (2) Atmel Studio 7, a tool for developing and debugging AVR and SAM microcontroller applications using C/C++ or assembly code [157], (3) Pymakr, an IDE extension to code editors to aid the development of IoT edge devices that run microPython [158] and (4) Particle IDE, a development environment for Particle’s IoT boards [159].

An overview of the appearances of these tools in the literature is given in Fig. 14. While there is a considerable amount of references to the Arduino IDE — which is justifiable by the popularity of the Arduino development boards and compatible devices — it is noticeable that there has been a little-to-no focus on the study of the existent traditional (*e.g.*, text-based) development environments for embedded systems.

Additionally, several PaaS and SaaS solutions — both cloud-based and on-premises — have appeared in the last few years as a response to the needs of IoT systems development, in a fashion similar to that of other kinds of software systems. The survey by Mineraud et al. identifies 23 different solutions that fit this vision [160], such as the EvryThng, ThingWorx, and SensorCloud. The same survey points out several gaps in the existing solutions, mostly resulting from the non-existing standardization and lack of cross-platform support [160] (*i.e.*, fragmented ecosystem, no interoperability between different IoT solutions [161]). These solutions generally provide a mostly-complete programming environment with an out-of-the-box set of Web-based Application Programming Interfaces (APIs), Software Development Kits (SDKs), libraries for specific hardware boards (*i.e.*, devices) and dashboards for device and data management.

Most of these solutions, ranging from the PaaS to the board-agnostic IDEs are somewhat limited in scope, both in terms of the supported devices and/or frameworks and in terms of system tiers (*e.g.*, tools focused on developing software for micro-controllers disregard the developments towards other tiers completely).

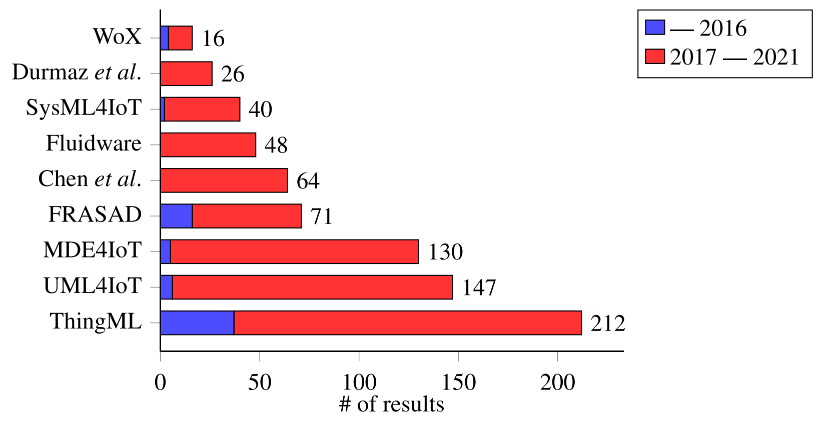

Fig. 15. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query `‘‘<tool_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<tool_name>` is the name of the development solution. For works that do not define a *name* the values presented correspond to the number of citations of the work. Data collected in November 2021.

5.2. Model-driven development

The growth of software complexity, together with the lack of an integrated view of the system under-development *often forces developers to implement sub-optimal solutions that unnecessarily duplicate code, violate key architectural principles, and complicate system evolution and quality assurance* [162]. Model-Driven Software Engineering (MDSE) appears as a software development discipline that tries to attain these issues by the use of (mostly graphical) domain-specific models, allowing the different stakeholders to understand the system better, providing transformation engines that automatically generate new (or update) models from existing models, thus facilitating the transfer of models between phases of the software development, while ensuring consistency between them [163].

Model-driven approaches are considered to have several advantages over traditional approaches to software development, including (1) shorter time-to-market, (2) increased reuse and fewer bugs, and (3) straightforward system and up-to-date documentation [164]. However, the same authors point out that code-generation does not solve all the problems; identifying as the main drawback the *delay between model change and model instance execution*. They continue arguing that *generating code from models, compiling the code, shutting down the existing system, installing and configuring the new system, and starting it up can take from minutes to hours*. Nonetheless, such a drawback is a common property of the typical deployment and evolution of almost all software systems.

Several works have studied the application of Model-Driven approaches like Domain-Specific Languages (DSL) and model transformations for providing IoT-focused low-code programming solutions [18]. These try to improve the interaction with IoT systems and make the production of platform-specific code more efficient by abstracting the platforms and allowing users to design systems at a high-level, most of them focusing on visual abstractions.

Some of the most known and comprehensive works that focus on model-driven development for IoT include the following (an overview of the popularity of these solutions can be seen in Fig. 15):

FRASAD FRAmework for Sensor Application Development is a framework for model-driven IoT system development, based on a *node-centric, multi-layered software architecture* to hide *low-level details and to raise the level of abstraction* of developing IoT systems. It leverages a rule-based programming model to define the system behavior and can generate code from the initial models through an automatic model transformation process. The evaluation done by the authors concludes that the framework improves the development process by *reducing the cost of dealing with the heterogeneity and complexity exhibited by sensor nodes and their operating systems* [165].

UML4IoT A model-driven UML-based approach for IoT system development. It defines a new UML profile, UML4IoT profile, which automates the generation of an IoT-compliant layer, IoTwrapper, that allows the full integration of different IoT components in an IoT system. This IoTwrapper is tailored for IoT manufacturing environments. However, the *UML4IoT profile contains basic key constructs independent of the application domain* so that it can be used in other application domains [166].

SysML4IoT It consists of framework to aid the modeling of IoT applications and verify their properties, more concretely, QoS. It consists of a SysML profile based on the IoT-A Reference Model and a model-to-text translator that converts the model and the specified properties to be run inside a model checker (NuSMV) [167]. Hussein et al. in their work adopt this language to define system functions and adaptations. Together with a “publish/subscribe paradigm to model environment information and its relationship with the system” and a code generation solution (SysML4IoT to Java) they developed a solution capable of programming adaptable IoT systems [168].

MDE4IoT An approach for modeling things as well as their Emergent Configurations[13] with a DSML. This approach exploits the use of high-level abstraction and separation of concerns to manage heterogeneity and complexity of things, enabling collaborative development, and enforcing reusability of design artefacts. It also provides automation mechanisms by providing model manipulation features enabling intelligence as runtime self-adaptation [170].

WoX: Web of Topics A conceptual model for IoT development based on the notion of *topics*, building block of the event-driven architecture followed by several IoT systems. In WoX, a *WoX Topic* is defined by two parts, namely: (1) *a discrete semantic feature of interest* (*e.g.*, temperature, humidity, air pressure), and (2) an Uniform Resource Identifier (URI) based location. A thing (IoT entity) role within a *Topic* is specified by its technological and collaborative dimensions, *i.e.*, *Topic-Role* [171].

Fluidware The Fluidware conceptual idea is to abstract the different system parts (including sensors, actuators and other system components) as *sources, digesters and target of distributed flows of contextualized events*, maintaining information about the data produced and manipulated over time. In this solution, developing new services and applications implies the definition of *funnel processes* to channel, elaborate, and redirect such flows in a fully-distributed way, *i.e.*, declaration of how these flows consume and produce events over space and time [172,173].

ThingML The Internet of Things Modeling Language is an approach for IoT systems' development that encompasses a modeling language (including state-charts, an imperative platform-independent action language and IoT-specific constructs), a set of tools (language editor, model transformations to diagrams, code generation framework), and a methodology to develop IoT systems and to extend ThingML itself. This solution is one of the most comprehensive and popular model-driven approaches and has a text-based description language. The primary focus is on the development of highly-distributed and heterogeneous systems by abstracting from the heterogeneous platforms and devices to model the desired IoT system's architecture [174].

Durmaz et al. This work presents a metamodel (*i.e.*, modeling language syntax) and a supporting graphical modeling environment tailored for the specifications of the Contiki IoT platform. As result, this work focus on abstracting the event-driven mechanism and protothread (*cf.* [175]) architecture of Contiki [176].

Chen et al. The work presents a runtime model (*models@run.time*) based approach to IoT application development. It focuses on two facets, namely: (1) *manageability of sensor devices is abstracted as runtime models that are automatically connected with the corresponding systems* and (2) *the customized model is constructed according to a personalized application scenario and the synchronization between the customized model and sensor device runtime models is ensured through model transformation*. Such an approach allows all the *application logic to be carried out by executing programs on this customized model* [177].

Some other works propose DSLs to tackle the shortcomings of the current solutions [178–180], but only present ideas or minimal proofs-of-concept, without validating their feasibility nor empirical validation with users. A more complete list of other relevant works can be found in [181].

Additionally, other less comprehensive approaches, have been developed as an answer to specific issues, such as the modeling of security and privacy concerns [182,183], and critical-mission systems [184]. Others have been trying to bring widely used architectures to the IoT domain by using models to abstract the devices and subsystems heterogeneity [185].

5.3. Visual programming

Diagrams, and other graphical logic and model representations, have been playing a role in software development since the appearance of modern digital computers in the 1940s. Visual programming makes use of an extensive set of icons and diagrams to convey information and to allow for multi-modal communication and interaction between humans and computers [27].

VPLs have been explored and used in several domains, including, but not limited to, educational activities (*e.g.*, learning to program), multimedia, video game development, system design and development, simulations, automation, data warehousing, and business analytics [60].

In favor of using visual programming, several researchers point out the dramatic reductions in time and cost when developing with this approach within industry scenarios [186]. Further, the achieved improvements in productivity and reliability were considerably noticeable [186].

However, visual programming, having several benefits and drawbacks, is not the *silver bullet* for software development; some authors even suggest that the solution for the adoption of visual notations lies in a hybrid visual–textual approach. Visual notations are more effective when dealing directly with an application domain (*e.g.*, IoT), and have several drawbacks when applied to general purposes, especially when dealing with complex control structures and recursion [187]. Further, the developer experience, when using such languages, depends significantly on its previous programming experience, how quickly they can create and debug programs, and how easy they can maintain applications over time [188].

There is a long-established relationship between models and visual programming, as is pointed out by the work of Lau-Kee et al. [189] circa 1991 that mentions the use of data flow as the computational model underlying the visual programming features of systems such as the LabVIEW. Such relationship between models and visual programming languages is visible as of today in

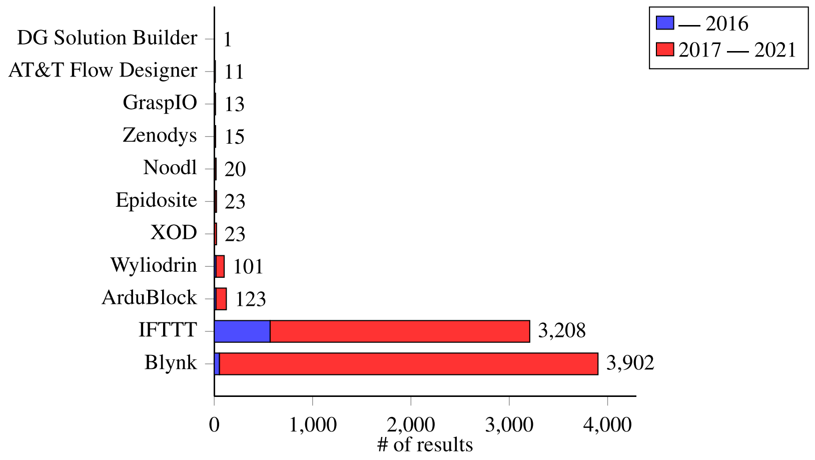

Fig. 16. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query `‘‘<tool_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<tool_name>` is the name of the development solution. Data collected in November 2021.

systems such as the Executable UML (xtUML or xUML), which are at the same time a visual notation and a visual programming language [190].

In the context of IoT systems development, several visual programming languages and ecosystems have been created with the main objective of easing the development process, leveraging different visual metaphors, and programming paradigms. These languages can be categorized as being (1) purely visual, (2) hybrid text-visual, (3) programming-by-example, (4) constraint-oriented and (5) form-based languages [191]. Some of the most common examples as per Fig. 16 are:

ArduBlock ArduBlock is purely visual programming, based on the Google and MIT Media Lab’s Blockly [192], focused on physical computing devices based on Arduino [193]. Several similar solutions are available such as S4 A, Snap4Arduino, Microblocks, and BlocklyDuino [194].

XOD XOD is a purely visual device programming platform (microcontrollers). It uses a visual language to program the devices and then generates native code for the target platform. In the language, a node is a block that represents either a physical device (*e.g.*, sensors, motors or a relay) or an operation (*e.g.*, addition, comparison, or a text concatenation). Each node has one or more typed inputs that accept values to be processed and outputs that return results. Creating a link from an output to an input builds a path for data, allowing one node to feed values into another. The development environment also allows the simulation of the programmed systems before deploying to real hardware [195].

GraspIO Graphical Smart Program for Inputs and Outputs is purely visual programming, based on the Google/MIT Media Lab’s Blockly [192], targeting the *Cloudio* hardware shield for Raspberry Pi SBCs. It is most suitable for educational purposes and offers the ability to build simple IoT and Robotics systems quickly [60].

Wyliodrin Wyliodrin is an online IDE for Linux-based embedded systems (Raspberry Pi, Intel Galileo) programming. The low-level GPIO-connected components are abstracted by the use of the *Libwyliodrin*, which offers an Arduino-like API, and can be programmed by the use of a hybrid text-visual programming language, in a *drag-and-drop* fashion. It also provides a set of dashboards to visualize the data being collected, and a mechanism to ease the communication between different devices [196].

Zenodys Zenodys is a hybrid text-visual programming environment, based on a *drag-and-drop* interface, that can construct both the logic and the user interfaces (UI) of IoT applications. It has built-in debugger features as well as text-based programming mechanisms, allowing fine-tuning of the solutions. Solutions designed in Zenodys can be deployed to both Linux and Windows environments [197].

Noodl Noodl is a hybrid text-visual programming environment that allows the creation of interfaces, and both logic and data flow. Although it does not focus on IoT, it also covers IoT system programming. It leverages the use of *nodes*, *connections*, and *hierarchies* to define the behavior of the system [60,198].

[13] Emergent Configurations is defined as the *set of things with their functionalities and services that connect and cooperate temporarily to achieve a goal.* [169]

DG Solution Builder DG Solution Builder is a purely visual IoT programming language, and environment, powered by a *drag-and-drop* interface. It also provides a dashboard feature capable of displaying the IoT systems data. It is tailored to be used with the Distributed Services Architecture (DSA) IoT middleware platform that abstracts the *inter-communication, logic, and applications at every layer of the Internet of Things infrastructure* [60,199].

AT&T Flow Designer AT&T IoT Platform Flow IDE is a cloud development environment for IoT systems. The visual language allows the creation of prototypes of IoT solutions, giving the ability to iterate and improve through multiple versions, then deploy the final solution. It gives the developer a set of pre-configured nodes that allow easy access to multiple data sources, cloud services, device profiles, and communication methods. Inspired by the Node-RED programming environment, it shares the same hybrid visual-text approach to development [60,200].

Epidosite A programming-by-demonstration system tailored for smartphones leveraging it as a hub for IoT systems automation. Users can express the wanted system logic by demonstrating the desired behaviors by directly manipulating the corresponding smartphone app for each IoT device. This programming environment also supports the creation of highly context-aware applications by taking into consideration the *smartphone app usage context and external web services as triggers and data for automation's* [201].

IFTTT IFTTT is a web and mobile application that leverages the use of a visual programming language to develop *if-this-do-that* rules in a form-based programming fashion. The language provides integration with several third parties. Despite not being focused on IoT solutions, it provides integration with several IoT products on the market, allowing the programming of their behavior [202,203]. Similar solutions include Microsoft Flow and Zapier [204].

Blynk Blynk is a mobile application that allows the control of Arduino and Raspberry Pi devices using a digital dashboard in a program-by-example fashion. It allows the *drag-and-drop* of widgets and then configures them using a form-like approach [205,206].

While there are other solutions for developing IoT systems employing a visual programming strategy (Node-RED, WoTKit, glue.things, IoT-MAP, IoTMaaS and the solution proposed by Yao et al. [207]), these tools also employ mashup strategies in order to build the system and are analyzed in detail in the following Section 5.4.

In the scope of Industrial IoT, there has been a considerable increase in the number of visual programming solutions (some with mashup characteristics). One can associate this trend to the historical usage of such visual notation for programming PLCs (Programmable Logic Controllers) using Ladder Diagrams (LD), Sequential Function Charts (SFC), and Function Block Diagram (FBD) [208]. Some of these solutions include Crosser [209], Critical Manufacturing Connect IoT [210], and Tomlein et al. [211], most of them using hybrid text-visual notations to construct data flow programs, similar to other solutions already presented; thus, we will not delve into them in detail. Nonetheless, most of the above presented solutions can be used in industrial settings.

These visual programming languages and environments are mostly tailored for IoT systems, with some targeting one specific tier (*cf.* Fig. 4) while others are capable of connecting different IoT devices, third-parties, and subsystems across tiers. Regarding the last, these are highly-inter-winded with their visual languages runtime and can be deployed on Linux-based systems, usually on the fog and cloud IoT tiers. They are also web-centric and encourage rapid development [212].

Several works highlight the issues that users have when configuring and understanding the trigger-action programs built using such tools that leverage a TAP-like mechanism (*e.g.*, IFTTT) [213,214]. Huang and Cakmak in their work identify that ambiguities between *trigger types* (states and events) and *action types* (instantaneous, extended, and sustained actions), lead users to misconstrue and misinterpret their rules (the authors state that "*people create different programs given the same prompt and are still in disagreement in their interpretations after having created programs themselves*") [213]. Ghiani et al. mention similar issues in their work and emphasize that different individuals understand the same *concept* or *metaphor* differently, which also increases the proneness to errors and the difficulty to understand the programmed rules [214]. Nonetheless, the quick development apps leveraging the trigger-action programming model, are the most popular examples of visual programming (*cf.* Fig. 16), *i.e.*, IFTTT.

5.4. *Mashup-based development*

The term mashup is vast and widespread. Using as definition the one purposed by Rümpel et al. [215], mashup tools are solutions that allow developers to construct applications and systems in a component-based fashion (*e.g.*, Widgets) or Web service composition (*e.g.*, REST APIs), *i.e.*, building applications by *mashing-up* existing components. Early examples of mashup tools are Microsoft Popfly [216] and Yahoo Pipes [217].

According to Maximilien et al. [218], mashup tools are characterized by providing data and process mediation (*i.e.*, connecting data sources, operators and consumers) facilities along with user interface customization tools (*e.g.*, data plots).

Typically mashup solutions are Web-based and provide a visual editor for composing the different services into an application or system. As such, several mashup tools provide a visual programming environment that allows the definition of the message flow between components, viz. *nodes* (*e.g.*, sensors and actuators, processing units, aggregations and external Web-based services). Due to this fact, most of the mashup-based solutions are also visual programming environments [149].

Mashup tools enable rapid prototyping by combining the use of rule-based mechanisms, graphical composing interfaces, and templates [219]. Some of the most relevant examples of mashup tools specific to IoT systems development are (Fig. 18):

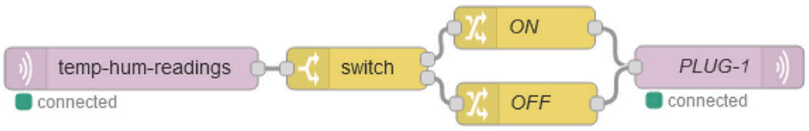

Fig. 17. Example of Node-RED *flow*, where the status of an electric plug (PLUG-1) changes (ON/OFF) accordingly to the current temperature value (SWITCH), provided by the temperature and humidity sensor (TEMP-HUM-READINGS).

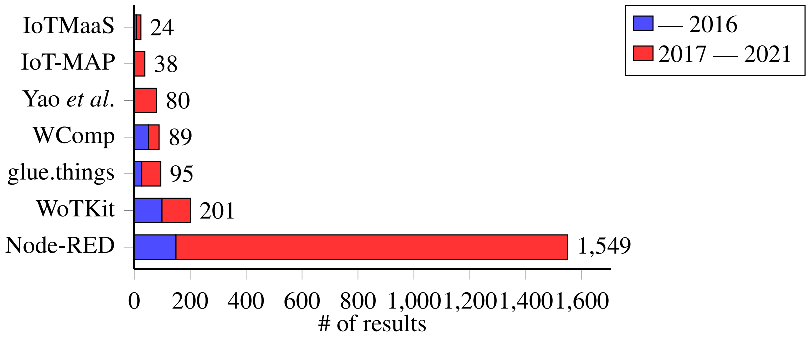

Fig. 18. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query ``<tool_name>'' AND (``IoT'' OR ``Internet-of-Things'' OR ``Internet of Things''), where <tool_name> is the name of the development solution. Data collected in November 2021.

Node-RED Node-RED is one of the most widespread visual programming solutions targeting IoT systems. It presents a flow-based development for wiring together hardware devices, APIs, and third-party services, in a hybrid text-visual programming approach, as it can be seen in Fig. 17 [220,221]. IBM originally developed it, and its runtime is built on JavaScript. The language can be extended by building new blocks using JavaScript snippets. The flows created can be exported in JSON format. Several entities have merged their solutions with Node-RED providing new services, such as the Sense Tecnic IoT Platform with provides FRED, a cloud-based Node-RED service [222]. Some work is also being carried on towards a distributed version of Node-RED, *i.e.*, Distributed Node-RED or D-NR [223].

WComp A three-part middleware designed for ubiquitous systems that provide (1) a foundational software infrastructure, (2) a service composition architecture, and (3) a composition-based adaptation mechanism designed with the dynamicity and heterogeneity of the underlying system in mind. Systems are built by composition of Web Services (which can correspond to devices, cloud services, or any other Web Service compliant entity). Their approach also introduces Aspect of Assembly (AA) as a way to define reactive behaviors to respond to changes in the system (*i.e.*, self-adaptation) using a form-based interface [224,225].

WoTKit A lightweight Java web application based on a message broker for programming IoT systems that allows the integration, visualization, processing of data from different devices and subsystems. The environment uses a hybrid text-visual language that allows the connection between different *modules* using *pipes*. Visualizations are based on the concept of *widgets* that are displayed in a dashboard, allowing rapid visualization of sensor data [212,226].

glue.things A Node-RED inspired mashup tool focused on the composition of Web services data streams and Web-enabled IoT devices (*i.e.*, Web-of-Things). A special focus is given on the delivery and management aspects of device data streams, consumer applications, and their integration within the ecosystem [227].

IoT-MAP Is a *thing* service composition platform that dynamically discovers devices, deploys drivers and provides them in the uniform interface to IoT-App. The IoT-App decouples the development of mobile applications from the heterogeneous devices specificity's by proving an *IoT-App* API that abstracts the functionalities of various things (*i.e.*, abstracted service objects) [228].

IoTMaaS IoT Mashup as a Service is a mashup platform that provides a (1) thing model, (2) a software model and (3) a computation resource model. During the mashup process, these three components can be tailored to the specificity's of the system under development (*e.g.*, select *things*, processing software and resource allocation, notification preferences), while the platform assures interoperability between devices and services (following the service-oriented architecture principles) [54].

Yao et al. This work presents a web-based solution for *connect, monitor, control, mashup, and visualize* devices in an IoT system. Their solution provides a layered framework for managing the data produced by the devices and delivering it to different consumers, includes a rule-based system to program the system logic in a context-aware fashion, and exploits the concept of avatars (*i.e.*, virtual representations of the physical things) [207].

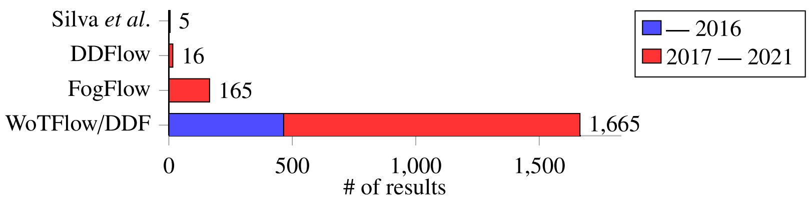

Fig. 19. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query `‘‘<tool_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<tool_name>` is the name of the development solution. Data collected in November 2021.

Although more popular (*c.f.* Figs. 13 and 18) when compared with model-based solutions, most of the mashup tools for IoT disregards the concerns about user interface definition and customization, focusing on the data and process mediation. Further, some of the mashup tools provide extra features such as simulation mechanisms and support for interoperability with other platforms.

Focusing on the popularity of the mashup solutions, Node-RED is the go-to solution. This can be justified by the setup simplicity, open-source nature, and an active community contributing with new nodes (addons part of the Node-RED Library). This Node-RED Library count with more than 3560 nodes, and more than 2030 example flows.[14]

5.5. Development of decentralized IoT systems

Several of the aforementioned solutions, specifically the visual programming and mashup-based ones (*i.e.*, low-code), provide a certain level of orchestration features — they connect services and devices, allowing their configuration, coordination, and management. But, in most cases, these solutions go further than that and also perform some level of computation, from data parsing to conditional statements, in a centralized fashion, thus becoming the single point of failure of the system.

Only a small fraction of those aims to offer a way for distributing computing tasks among devices and other computational resources while dealing with the challenges posed by the nature of these resources, especially the highly-dynamic topology of these networks. Thus, as a way to leverage the existing computational power in edge devices, several proposals have been drafted, mostly resulting from scientific research — thus being more proofs-of-concept and not full-fledged ready-to-use solutions. A non-extensive list of the ones found in the literature is summarized in the following paragraphs, and an overview of their popularity in Fig. 19 [229].

WoTFlow [230] and DDF [231] A set of extensions to Node-RED to make it more suitable for developing fog-based applications that are context-dependent on edge devices where they operate. DDF starts by implementing D-NR (Distributed Node-RED), which contains processes that can run across devices in local networks and servers in the cloud. The application, called *flow*, is built with a visual programming environment, running in a development server. All the other devices running D-NR subscribe to an MQTT topic that contains the status of the flow. When a flow is deployed, all devices running D-NR are notified and subsequently analyze the given flow. Based on a set of constraints, they decide which nodes they may need to deploy locally and which sub-flow (parts of a flow) must be shared with other devices. Subsequent work focused on support for the Smart Cities domain, including the deployment of multiple instances of devices running the same sub-flow and the support for more complex deployment constraints of the application flow [231]. Subsequent works [232,233] introduce the support for CPSCN (Cyber–Physical Social Computing and Networking), making it possible to facilitate the development of large-scale CPSCN applications.

Szydlo et al. [234] Work focused on the transformation and decomposition of data flow. Parts of the flow can be translated into executable parts, such as Lua. Their contribution includes data flow transformation concepts, a new portable runtime environment (`uFlow`) targeting resource-constrained embedded devices, and its integration with Node-RED. Their solution transforms a given data flow by allowing the developer to choose the computing operations run on the devices. These operations are implemented using `uFlow`. The communication between the devices requires a cloud layer, without support for peer-to-device communication. Later, the authors proposed `FogFlow` [234], which enables the decomposition into heterogeneous IoT environments according to a chosen decomposition schema. Several algorithms for flow decomposition are mentioned [235,236], but none were explored/provided results.

FogFlow by Cheng et al. [237,238] The authors extend the dataflow programming model with hints to facilitate the development of fog-ready applications by providing an environment that allows developers to streamline the construction of decentralized IoT systems. An application is first designed using the `FogFlow` Task Designer (a hybrid text and visual environment), which outputs an abstraction called *Service Template*. This abstraction contains details about the resources needed for each part of the system. Once the *Service Template* is defined and inserted, the framework determines how to instantiate it using the context data available. Each task is associated with an operator (a Docker image), and its assignment is based on (1) how many resources are available on each edge node, (2) the location of data sources, and (3) the prediction of workload. Edge nodes are autonomous since they can make their own decisions based on their local context without relying on the cloud.

[14] Data collected from https://flows.nodered.org/ in November 2021.

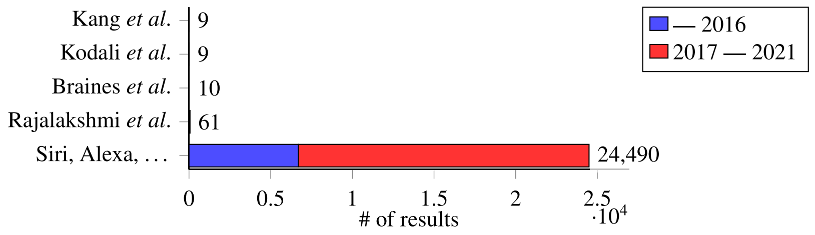

Fig. 20. Number of occurrence of each development solution in the literature gather using Google Scholar with the search query `‘‘<tool_name>’’ AND (‘‘IoT’’ OR ‘‘Internet-of-Things’’ OR ‘‘Internet of Things’’)`, where `<tool_name>` is the name of the development solution. For works that do not define a *name* the values presented correspond to the number of citations of the work. Data collected in November 2021.

DDFlow [231] Presents another distributed approach by extending Node-RED with a system runtime that supports dynamic scaling and adaption of application deployments. The distributed system coordinator maintains the state and assigns tasks to available devices — each device has a set of capabilities and a list of services that correspond to an implementation of a *Node*. Dataflow notions of *node* and *wire* are expanded, with a *node* in DDFlow representing an instantiation of a task deployed in a device, receiving inputs and generating outputs. *Nodes* can be constrained in their assignment by optional parameters, *Device*, and *Region*, inserted by the developer, and are automatically orchestrated by a coordinator that maps tasks to devices by minimizing the task graph's end-to-end latency of the longest path.

Silva et al. [239] introduces a decentralized computation environment as an extension to Node-RED that supports automatic decomposition and partitioning of the computational flows into tasks that can run in edge devices. The edge devices themselves run a custom firmware which is responsible for exposing each device's capabilities and running custom MicroPython scripts on demand. Each computational task has *Predicates*, constraints that cannot be violated, and *Priorities*, requests that can be violated when necessary. The orchestrator (*i.e.*, Node-RED extension) greedy algorithm for the assignment of nodes to devices, based on the devices' capabilities and the priorities and predicates of each node.

Most of these approaches are only proof-of-concept and not widely adopted, even if most of them are based on Node-RED which is one of the most common IoT-focused development environments and runtimes. Only a few of the approaches — FogFlow and uFlow — does in fact leverage the available resources in the edge tier of the system. Some other works have been suggesting the use of serverless[15] in IoT scope, as a way of distributing computational tasks on-the-fly amongst available resources that meet the resource demands of them [240–243].

5.6. End-user development

There is a considerable amount of literature that focuses on enhancing the interactions between users and smart spaces [244–247], taking advantage of several approaches, such as introducing artificial intelligence-based solutions (personal assistants) and exploring natural language processing (NLP). These solutions try to improve the end-user experience on configuring their own IoT systems, by providing easy-to-use mechanisms to program their systems that require little to no technical skills. Some of these solutions that focus on providing a graphical interface for the user to interact with the system have already been presented in previous sections, thus in the following paragraphs, the focus will be on the approaches that leverage other ways of interaction with the IoT system.

Some of the identified approaches that either use NLP or conversational assistants in the context of IoT systems are the following:

Siri, Alexa, Cortana, and Google Assistant There exist a plethora of conversational assistants in the market (see [248,249] for a comparison of these tools) which are capable of answering natural language questions. Recently, these assistants have gained the ability to interact with IoT devices, with Ammari et al. identifying IoT as the third most common use case of voice assistants [247]. However, these are general conversational assistants and do not focus on providing IoT-focused features. Further, they are limited in the number of device manufacturers and ecosystems they support (relying on the open-source community for supporting other devices and ecosystems).

Jarvis Introduced as an enhancement/extension to Google Assistant, Jarvis provides mechanisms to manage IoT spaces in a conversational way [250,251]. The authors also introduce the concept of "causality queries" which enables users to better grasp why something happened in their smart space. A feasibility experiment was carried with mostly non-technical participants, showcasing that the developed solution was intuitive enough to be used by common end-users, with participants showcasing an overall preference by conversational assistants over visual low-code solutions.

[15] Some authors have been calling this *deviceless* computing [240–242].

Kodali et al. An approach for a home automation system to "*increase the comfort and quality of life*", by developing an Android app that can control and monitor home appliances using MQTT, Node-RED, IFTTT, Mongoose OS, and Google Assistant. Their limitations lie in that the *flows* must have been created first in Node-RED, and the conversational interface is used to trigger them, ignoring all the management and configuration activities [252].

Braines et al. An approach based on Controlled Natural Language (CNL) — natural language using only a restricted set of grammar rules and vocabulary — to control a smart home. Their solution supports (1) *direct question/answer exchanges*, (2) *questions that require a rationale as response* such as "*Why is the room cold?*" and (3) *explicit requests to change a particular state*. The most novel part of their solution is in trying to answer *questions that require a rational response*; however, they depend on a pre-defined smart home model that maps all the possible causes to effects [253].

Rajalakshmi et al. A solution using both Node-RED and Alexa to interact with IoT systems, which they claim to simplify the interaction between users and IoT systems, but also manage complex systems. The system proposed by the authors uses Node-RED to create rules and link devices with each other, allowing the user to create complex rules while at the same time taking advantage of Alexa, providing a simple way for users to control their smart devices. This system provides simple interaction with smart homes through Alexa, and the complexity needed for some use cases through Node-RED. However, there is no link between the voice control and the visual programming solution, which means they cannot create these rules using voice commands (being Alexa, the only way to trigger the Node-RED flows) [254].

Kang et al. In their work explores the use of multi-modal interaction within IoT systems — combining voice and gesture interactions — as a way of addressing the scalability and expressiveness supported by existing IoT-vendors mobile applications and voice assistants. Although most of the participants who took part in the study responded positively to many interaction techniques, one of the identified pitfalls was the lack of robustness of the voice assistant that failed to understand the user commands [255].

Several other works [256–258] combine the use of voice assistants with IFTTT, using the latter to define the system rules. While the primary control mechanism over the IoT system is voice-based, it is mostly used to trigger the IFTTT specified rules, depending on the rules' definition in a form-based visual interaction. Thus, it is also limited by them.

Although commercially-grade voice assistants are more common, as can be seen in Fig. 20, as they nowadays come as part of the operating systems, there is a considerable number of works that attempt to improve these voice assistants, either by adding new functionalities or by mixing them with other development approaches.

6. Analysis and discussion

In this section, we provide a summary of the current state of the practice of designing and constructing IoT systems, enumerating the observed gaps, limitations, and challenges. We also point out new research directions the scientific community believes should be tackled during the following years by analyzing shortcomings and limitations of current practices, tools, and methods.

There has been a steadily increasing focus in the last five years on improving the design and construction approaches by academia and industry alike; still, there are several pending issues and research challenges to be tackled before achieving the full potential of IoT.

Tables 8 and 9 summarize some of the trends, open issues, and research challenges that are discussed in the following paragraphs.

6.1. Designing the Internet-of-Things

The following paragraphs attempt to match the findings presented in Section 4 with published works that explore the same topics.

6.1.1. Physical platforms and devices

Despite the ongoing discussion about the concrete definition[16] of the *Internet of Things*, and consequently, what devices comprise its ecosystem, the total count of connected devices is growing fast.[17] The plethora of devices and other physical platforms, (re–)designed and adapted to different application domains (*cf.* Table 1 and Fig. 3), coupled with the fact that they are being produced and commercialized by different vendors, have resulted in a highly-fragmented market [260].

Regarding the *cloud* tier (*i.e.*, servers and other platforms), IoT solutions are being provided as Everything-as-a-Service (XaaS) [261],[18] where out-of-the-box, service-oriented architectures are offered as a panacea for any application domain. However, the dependability of this approach can be severely compromised given that centralizing all the systems' core logic in the cloud

[16] Defining IoT is *fuzzy because of breathless hype, including attempts to anticipate demand for devices that have yet to be invented or commercialized* [259].

[17] Amy Nordrum states that the exact total number of devices will be *somewhere between Gartner's estimate of 6.4 thousand million (excluding smartphones, tablets, and computers), and IHS Markit's estimate of 17.6 thousand million (with all such devices included)* by the year of 2020 [259].

[18] Also known as Anything-as-a-Service (*aaS) depending on the authors [262].

Table 8
Summary of the trends, open issues, and research challenges on the design of IoT systems.

Aspect	Trends (T), Open Issues (I) & Research Challenges (C)
Physical Platforms and Devices	• (I) Fragmented ecosystem of different boards, frameworks, and embedded operating systems. • (I) Common dependency on always-on Internet (cloud), raising several dependability and privacy concerns. • (T) Software-defined approaches gaining relevance, and hardware becoming more generic. • (T, C) Energy harvesting and energy-aware operating modes becoming crucial (*e.g.*, battery-dependency). • (C) Development of mechanisms for over-the-air updates that focus on minimal manual intervention while ensuring recovering in case of deployment failure. • (T, C) Several entities have been working on domain-specific languages — and adequate development tools — that abstract the complexity and heterogeneity of devices leveraging advancements such as WebAssembly on devices (*e.g.*, Wasm3).
Communication Protocols	• (I) Competing protocols created and maintained by different vendors with no standard in sight leading to an *untamed* fragmentation. • (T) Focus on lightweight protocols that are suitable for highly-constrained devices. • (T) Protocols with mesh operation support gaining relevance to meet coverage needs. • (C) Runtime adaptation of communication modes to meet operational demands (*e.g.*, battery levels). • (T) Wide adoption of publish–subscribe protocols (*cf.* event-driven architectures). • (I, C) Use of simple or no encryption at all in communication due to devices' constraints. • (T, C) Renewed focus on *ad-hoc* networks to reduce the dependency on pre-existing infrastructure.
Architectural Styles	• (T, I) Well-known architectural styles are widely adopted, mostly disregarding IoT particularities. • (T) Multi-layer architectures being common, from 3 to 6 or more layers. • (T) Client–server architectures continue to be one of the most common go-to solutions (*i.e.*, directly connecting devices to the cloud). • (C) Distributed architectures such as peer-to-peer are mostly unnoticeable in the literature and market. • (C) The increase popularity of mesh networks can, potentially, open doors to a large adoption of architectures suitable for distributed computation. • (T) Event-based architectures gaining traction due to their suitability to deal with high data volume data streams and complex event processing.
Reference Architectures	• (I) On-going efforts by several governments and non-government entities towards the definition of a common reference architecture but no common standard at this point. • (I, C) Most architectures focus on specific application domains and their intricacies factor, which hinders its role in the multi-domain nature of IoT applications. • (T) Vendor-specific reference architectures becoming the go-to solution for new IoT projects. • (I) A common agreement between academia and industry is the key to addressing the issue of competing reference architectures; however, it is not expected that an *one-size-fits-all* standard will appear.
Interoperability	• (I) Lack of standards raises the question of how long the different system parts will support a given device. • (I) Continuous creation, modification, and deprecation of interoperability standards make it hard to pick one to use from the available ones. • (T) Several authors refer to web standards as the way to ensure the interoperability between different IoT systems as it became the solution in other kinds of systems. • (T) Zero-configuration solutions (*e.g.*, DPWS and multicast DNS) make interoperability across vendors possible are gaining traction.
Design Patterns	• (I, C) Lack of a reference set of *best practices* on addressing common problems in IoT development (*i.e.*, lack of a pattern-language for IoT). • (I, C) Most patterns are dispersed among different functional aspects, architectural tiers, and concrete application domains. • (I, C) Considerable gap of IoT-specific cloud and fog tiers patterns and best practices. • (T) Knowledge from other fields of software engineering being slowly adopted in IoT field, with several practices (*e.g.*, CI/CD) being in its early adoption stages. • (C) Systematization of the existing knowledge in IoT solutions becomes crucial to a better IoT systems development and can improve the ecosystem as a whole. • (T) Recent focus on autonomic computing strategies to meet the IoT operational needs in an autonomous fashion.

makes it a single point of failure and dependent on an always-on Internet connection [263]. Additionally, several authors raised concerns regarding the security and privacy of such cloud-centric approaches [263,264].

Another research focus is the concept of Software-defined Everything (SDE), which has been expanding the importance and impact of software in a previous hardware-only world, becoming a vital part of the IoT ecosystem [265]. The dissemination of Software-defined networking (SDN), Software-defined storage (SDS), and software-defined data centers (SDDC) as core components of Cloud-tier are strong indicators of the Software-defined tendency [265]. However, this also impacts the lower-tiers of IoT systems, reducing the need to physically upgrade and change the hardware to meet new requirements or correct existing flaws.

It is in the lower-tiers of IoT systems (*fog* and *edge*) where the peak of heterogeneity is observed, resulting in a vast mix of Single-board Computers (SBC), networking devices, and embedded/constrained devices. These devices, tailored to the different application domains and working environments, are built with various central processing units (CPUs), communication radios, and power consumption needs. They also provide different end-user interactions and include several sensing and actuating capabilities (*cf.* Section 4.1).

Table 9
Summary of the trends, open issues, and research challenges on the construction of IoT systems.

Aspect	Trends (T), Open Issues (I) & Research Challenges (C)
Traditional Development	• (I) Embedded systems (edge tier devices), commonly developed in C/C++, privilege vendor-specific development toolchains (*e.g.*, Atmel Studio). • (T) Some alternatives to C/C++ have been created and typically leverage language/framework-specific development tools (*e.g.*, ESPlorer). • (I, C) Most development solutions lack mechanisms for verification and validation, thus hindering the adoption of CI/CD practices. • (T) PlatformIO IDE has become a reference in IoT development due to multi-vendor, multi-board, and multi-framework support. It is also one of the firsts to support CI/CD practices and provides a built-in package/library manager. • (T) SCADA systems are still largely used in industrial settings, being supported by a set of development tools. • (I) Traditional development solutions typically focus on edge tier devices and are not suited for serving as an *all-in-one* development tool for IoT systems development. • (T, C) Several authors argue that a *new generation of development environments to bring software engineering practices up-to-date with other domains* is required, with some pointing to cloud-based development solutions.
Model-driven Development	• (I) Most approaches employ *leaky abstractions*, *i.e.*, device-specific code is always required to program certain device features. • (I) These solutions have similar shortcomings to traditional development in software engineering practices but, typically, have built-in verification and validation mechanisms. • (I) Most solutions have scale limitations, either in terms of devices or the number of system's moving parts. • (T) The use of *models@run.time* is repeatedly mentioned in the literature but remains without adoption. • (T) ThingML appears as the more comprehensive model-based solution encompassing a methodology, a modeling language, and a set of development supporting tools. • (I, C) Model-driven development remains mostly not adopted in IoT development at this stage.
Visual Programming	• (T) Visual notations have been long used to program embedded devices in industrial settings (*e.g.*, PLCs). • (I) Lack of mechanisms to deal with the complexity of the system logic as it increases given that visual notations typically hinder reasoning about the system in place. • (I, C) Lack of debugging capabilities and verification/validation mechanisms is the norm. • (T, I) While there are a few solutions that target edge tier devices, most of them target fog/cloud tiers, with some being cloud-based solutions. • (T, I) Most solutions require an external runtime to execute the developed programs (*e.g.*, data flows). • (I, C) Most solutions cannot deal with dynamic networks (*i.e.*, when devices join and leave the network).
Mashup Development	• (T) Mashups are among the most popular strategies to develop fog/cloud tier software in IoT systems. • (T) Most solutions leverage high-level abstractions that ease the connection between the system moving parts. • (T) Commonly use visual notations (*i.e.*, visual programming), thus sharing its limitations and drawbacks. • (I) Most solutions lack a way to debug and verify the programmed logic leading to failures during runtime. • (C) Some authors suggest that mashup development could adopt some model-based development concepts.
Development of Decentralized IoT	• (T) As the number of computing resources available in the network increases, developing systems that use these resources (opportunistic computing) brings several advantages (*e.g.*, complete complex tasks by distributing smaller tasks across devices). • (T, C) Some authors have been working on making visual data-flows (*e.g.*, Node-RED) distribute-able across devices, instead of running in the development solution runtime. • (I) Most solutions are still in their infancy, depending on concrete communication protocols and firmware. • (I) Most solutions have several issues dealing with highly dynamic networks. • (C) Several authors point out that the decentralization of IoT is crucial for the dependability of these systems, allowing edge tier devices to continue operating even when system coordinators (*e.g.*, rule-engines) fail.
End-user Development	• (T) Voice assistants (*e.g.*, Google Assistant) and TAP (*e.g.*, IFTTT) have been the go-to solution for IoT system development. • (T, C) Some authors have been researching multi-modal interactions to improve end-user development (*e.g.*, combining visual programming with voice assistants). • (T, C) A few authors have been researching the use of virtual and augmented reality as a way to improve the end-user understanding of the system, its mechanisms, and how to program it.

Current research efforts are focused on the development of ever-smaller boards with lower power needs and built-in power-efficiency strategies[19] (*e.g.*, on/off cycles), low-cost and highly-reliable data transceivers, improved sensing capabilities, and overall more efficient processing units [267]. Few, if any of these, would help the de-fragmentation of the Internet-of-Things.

6.1.2. Communication

Networking is a mandatory requirement of IoT devices. Every sensor and actuator device should be connected anytime, anywhere. However, these devices are limited by design concerning processing power, energy consumption and management, and available connectivity [15]. This constant connectivity is one of the critical building blocks of IoT that suffers more from the fragmentation mentioned earlier. In a similar way to what happens in terms of physical platforms and devices, different vendors are trying to

[19] Including the usage of energy-harvesting mechanisms [266].

develop their communication radios, protocols, and ecosystems leading to an ever-growing heterogeneous network ecosystem, thus increasing its accidental complexity. Ray et al. in their work states that such a scenario *enhances the complexity among various types of devices through different communication technologies showing the rude behavior of the network to be fraudulent, delayed, and non-standardized* [21].

The Wireless Sensor Networks (WSN) research field[20] has long been developing networking solutions that satisfy the highly stringent constraints due to the operational domains of sensor/actuator-based systems. These factors include considerations about fault tolerance, scalability, cost, hardware, mutable topology, environmental conditions, and power consumption and efficiency [267]. For example, Gubbi et al. points out in their work that it is common for a constrained device to stop working, thus requiring the network to be self-adapting and allow for multi-path routing [268].

The SDE trend also provides communication solutions such as Software-defined Radios (SDR) and Software-defined Networks (SDN). This is another evidence that components traditionally implemented in hardware are now being supplanted by others implemented in software, mainly due to their inherent agility (*e.g.*, changing between different communication frequencies or protocols happens without the need of changing the hardware itself) [269,270]. This brings advantages such as (1) addressing heterogeneity better, (2) being (more) future-proof, and (3) allowing updates for devices that are situated in inaccessible locations.

From a network services perspective (*cf.* Section 4.6), there has not been enough investment towards defining a future standard Service Description Language (SDL) [271]. This effort would improve service development, deployment, and resource integration among IoT solutions. Besides this much-needed standard, the development and usage of powerful service discovery mechanisms and object naming services are also crucial to transforming the *resources of physical objects into value-added services* [21].

Despite the full range of wireless technologies available, each one fulfills different requirements, and each one has a set of advantages and disadvantages (*e.g.*, communication range, power, and bandwidth needs). However, from a system designer's point of view, it is hard to conclude which is the most suitable for a particular scenario or to satisfy a given constraint. This led to the age-old question [11,125] of *which technology is the best one for my application?* Recently, some practitioners have tried to partially answer this question by unifying some of the low-power personal area network protocols — including ZigBee and Thread — under a novel protocol called Matter [64], thus easing the process of selecting a protocol, in this specific case, for low-range communications (*i.e.*, smart home).

More open challenges in the communications field include the need for new wireless *ad hoc*[21] solutions. Finally, most of the communication protocols currently being used suffer from limitations and flaws regarding privacy and security, pointing out that there is a need for more research on such sensible topics [273].

6.1.3. Architectures

Internet-of-Things systems are based on well-known architectures such as service-oriented, client–server, peer-to-peer, and microservices. However, these usually require some adaptations to fit the typical IoT scenarios better, as they are not enough to solve the difficulties that arise due to stringent constraints regarding scalability, flexibility, interoperability, diverse QoS support, and security/privacy [274]. These requirements are core to fulfill the goals of IoT, and several architectural variants have been proposed to address them, such as the *broker*- and *mediator*-based architectures, which are different forms of the *publish–subscribe* approach (*c.f.* Section 4.4).

Yaqoob et al. [274] enumerates three challenges to be addressed by the next generation of IoT architectures, including (1) resource control, *i.e.*, devices should be accessible and configurable in a remote manner, (2) energy awareness, *e.g.*, automatically lowering processing power when the system load is low, and (3) interference management, *i.e.*, with the number of devices communicating by different manners, mostly of them radio-based, interference can become a real problem.

Sharing the same concerns, several authors have been pushing the concept of Autonomic Computing (*i.e.*, self-managed systems) in IoT [275–278]. Several aspects characterize Autonomic Computing, namely: (1) self-configuration, the ability of a system to automatically and seamlessly configure itself by following a set of high-level policies, (2) self-optimization, the ability of a system to improve its performance without human intervention (*e.g.*, load-balancing), (3) self-protection, the ability of a system to protect itself from malicious attacks, and (4) self-healing, the ability of a system to detect, diagnose, and repair automatically, system defects at both hardware and software levels [279]. Using IoT architectures that follow these "self-*" principles becomes paramount to coping with the complexity of IoT systems, reducing the dependency on constant human monitoring and manual system tuning [138,275,276].

Several authors are also invested in creating reference architectures, such as the nine different initiatives presented by Weyrich et al. [112]. European Union projects, such as SENSEI [280], Internet of Things-Architecture (IoT-A), and FIWARE [100], have been addressing these challenges from a WSN viewpoint and have been successful in defining some common grounds [268].

But Weyrich et al. also conclude — a recurrent topic in this domain — that the scientific communities and industry leaders need to agree on standards to avoid systems that are crippled due to the lack of interoperability; since not even a *de facto* reference architecture encompassing most kinds of IoT systems across different application domains was reached so far.

[20] Nowadays it is hard to separate the WSN and IoT research fields due to their overlap in several research directions.

[21] *Ad hoc* networks do not rely on pre-existing infrastructures (*e.g.*, routers or access points) [272].

6.1.4. Interoperability standards

Interoperability allows systems to communicate and integrate seamlessly with systems and devices from different vendors and with different operating scopes. This concern is paramount in reducing current fragmentation and pushing towards the *systems-of-systems* goal. However, while the different vendors refuse to sit idle, as standards that ensure proper interoperability are developed, in the long run, academia, industry, and other institutions must come together in defining a standard; if one wishes to achieve the Internet-of-Everything successfully [125,281].

The nonexistence of interoperability standards is one of IoT's most pressing issues, influencing all system stakeholders. From an end-user viewpoint, they do not know how long the different system parts will support the devices they buy (*e.g.*, a light bulb no longer supported by the integration gateway). The same happens from the vendor point-of-view, since designing a system using the latest available standard proposal does not ensure its adoption or that the standard will be deprecated even before the system reaches the market (which pressures vendors to define their in-house solutions contributing to today's fragmentation) [124].

Works such as the one by Noura et al. [125] focus on this matter, mainly focusing on IoT's current fragmentation has been directly impacting the emergence of a common interoperability standard. They also advocate the community to embrace web technologies (*i.e.*, Web Services/APIs) to define a common ground for IoT. Some proposals such as the Web Things [118] and Web Thing Model [120] already share this vision, and some network protocols already embrace this vision as a way of having zero-configuration devices, *i.e.*, DPWS, the evolution of UPnP [85].

The future seems gloomy, though, and most authors believe that it is not likely that a common set of standards will emerge shortly and be universally accepted among academia, industry and standardization bodies such as ISO.

6.1.5. Patterns and best practices

Patterns have been used for a long time to systematize knowledge in the form of problem-solution pairs. Although such systematization is no silver bullet for sharing knowledge, it is a common go-to solution for finding a common way of solving a given problem in a certain context [282].

While there have been efforts on systematizing some of the available knowledge [129,131,132,135–138], the number of captured patterns is still residual. Additionally, and more crucial, there is still no common catalog of patterns for IoT, somewhat similar to the one that exists for cloud computing [139].

Ultimately, the existence of enough IoT-specific design patterns and the discovery of relationships between them has the potential to produce a *pattern language* that can be shared and improved among practitioners. Pattern languages would help IoT developers by providing them with a proper vocabulary, abstractions, and insights into the solutions other developers have recurrently found.

Although one can leverage existing tools, approaches, and patterns from a best-practices viewpoint, IoT poses special needs that make the development processes more expensive and complex. These needs mainly result from the hardware aspects of IoT systems that differentiate them from *software-only* systems (*e.g.*, cloud computing) [146]. Practices such as CI/CD are limited, requiring physical testbeds or virtualized environments to be automated, and system verification and validation mechanisms are limited due to similar dependencies [283].

Further, in contrast to cloud computing, mechanisms to orchestrate a large fleet of devices (with the necessary safeguards such as rollbacks) are limited. This can be easily correlated with the lack of approaches to develop distributed IoT systems that leverage the computational resources of the devices in the network.

The lack of best practices for IoT development, in a similar fashion to the best practices that one can use to develop *software-only* systems, limit the full potential of IoT and directly influence the approaches used to create this kind of system, as further discussed in the following paragraphs. From another perspective, this lack of practices results in further fragmentation of the IoT ecosystem, with both academia and enterprise using their in-house practices, impacting the development processes (*e.g.*, making it harder to deploy new versions of software/firmware) and the on-boarding of new human resources (that have to learn a mostly-unique set of practices).

6.2. Constructing the Internet-of-Things

The next paragraphs revisit Section 5, analyzing the challenges and issues and supporting them with relevant literature.

6.2.1. Traditional development approaches

Most typical solutions used for IoT system development are tightly coupled to a specific tier. For example, PlatformIO [153] is arguably one of the most comprehensive solutions for embedded systems development. However, it fails to cover the full spectrum of devices, languages, and vendor-specific technologies that build up IoT systems.

Another aspect that limits the development solution to specific devices is their programming language, thus the development tool becomes specific to certain languages. As an example, in the case of edge tier devices, the language is typically C/C++, with some supporting certain frameworks or real-time operating systems, thus the development solutions are designed with these languages in mind. Recently both academia and industry have been working on domain-specific languages — and adequate development tools — that abstract the complexity and heterogeneity of devices leveraging advancements such as WebAssembly (WASM) [284].

Other tools, platforms, and approaches have been developed in the last few years to find new approaches for developing IoT systems. However, gap analysis on 39 IoT platforms circa 2016 (from PaaS to on-premises solutions) [160] revealed issues regarding (1) the absence of support for heterogeneous devices, (2) handling different formats and models of device data streams, (3) lack of proper documentation and tooling, (4) ability to simultaneously target different tiers (edge, fog, cloud), (5) lack of semantic-based service discovery, (6) on-intuitive construction mechanisms, and (7) privacy and security.

While most of the traditional development solutions for software-only systems have built-in — or easy integration — with verification and validation mechanisms, this is not common in IoT development solutions that typically consist of independent and vendor-specific tools, already analyzed and discussed by existing literature [283,285].

Several authors conclude that there is a need to evolve towards a *new generation of development environments* to bring software engineering practices up-to-date [7,286] with other domains: "*software developers [need] to realize that IoT development indeed differs from mobile-app and client-side web application development*" and, as such, appropriate tooling is needed to tackle this reality [6]. Larrucea et al. [7] also suggest that there is a need to shift from traditional development environments to cloud-based ones: "*development environments in the cloud — not for the cloud, but in the cloud — to enable the massively scalable verification and validation techniques (including simulation) that will be needed for most large mission-critical systems in the IoT*".

This idea has been followed by some practitioners that already offer cloud-based IDEs with built-in validation and verification mechanisms [158,159]. Some of these cloud-based IDEs have been empowered by the WebUSB API support by multiple web browsers that allow the program of IoT devices connected over USB directly from the browser [287], while others have been adopting over-the-air update strategies as a way to program devices.

6.2.2. Model, visual and mashup development

Several development approaches have been suggested as an alternative to traditional development for IoT systems due to these systems' specific constraints and necessities. More specifically, several works focus on improving the abstractions and reducing the technical knowledge necessary to develop them, as the complexity of these systems keeps increasing [18]. Most of the presented solutions are not mutually exclusive, being combined to improve the development, evolution, and maintenance processes.

Visual programming. Visual programming aims to abstract low-level concepts and details into high-level logic through the use of visual metaphors, *e.g.*, Programmed Logic Controllers (PLCs) are typically programmed using visual notations such as Ladder Diagrams [208].

When analyzing the range of visual development solutions available, we can notice that while most of them target the *fog tier*, they can also be used in the *cloud tier* (as most of them target non-constrained Linux-based systems common to both the *fog* and *cloud* tiers) [18,60]. However, it is noticeable that those that target physical devices are particular to the *edge* tier and more coupled to specific hardware (primarily due to the direct interaction with its physical capabilities).

Other typical shortcomings of visual development solutions relate to (1) testing and verification procedures, (2) debugging capabilities, (3) scalability, both in terms of components and in terms of the visual metaphors used, (4) maintainability (*e.g.*, version control), and (5) real-time feedback to the developer (*e.g.*, most of the time the feedback occurs after deployment) [18,60].

Development of decentralized IoT. Some research has been focusing on visual programming approaches to leverage the decentralized nature of IoT, using the computing resources available in the network. However, most existing solutions have limitations either in their functioning or in the devices they support. `DFF` assumes that all devices run Node-RED, limiting the types of devices used to computing units capable of running some type of operating system. `FogFlow`, `uFlow`, and Silva et al. are the only ones that specify how they truly leverage constrained devices with the transformation of sub-flows into Lua code. `DDFlow` assumes that all devices have a list of specific services they can provide that should match the node assigned to them [229].

Regarding the method used to decompose and assign computations to the available devices, `DDFlow` describes the process using the longest path algorithm focused on reducing end-to-end latency between devices. `FogFlow` and `uFlow` mention several algorithms that could be used but do not specify which one was implemented. Silva et al. solution leverages a greedy algorithm that shows limitations when the network is too dynamic. Both `DDF` and `FogFlow` do not specify the algorithm used besides some constraints but are the only ones with accessible source code and an open-source license. All the surveyed approaches claim to have some runtime adaptation mechanisms to deal with changes in the system, such as device failures [229].

Mashup-based and model-based. Both model-based and mashup-based software development approaches have been pursued as a way to improve the development of IoT systems (*cf.* Sections 5.2 and 5.4). These allow the abstraction of low-level programming details and processes into higher-level representations and constructions that can be used to manipulate IoT systems [288]. However, these approaches commonly have limitations and unaddressed challenges, such as not capturing the full software development life-cycle [288], having large-scale limitations [288], and suffering from leaky abstractions [289].

Prehofer et al. verify this in their work, suggesting that model-based approaches by themselves fall short in attaining the development requirements of IoT systems. This can partially justify their unpopularity when compared with other development approaches, *cf.* Fig. 15. The authors also noticed a need for better tooling to use such approaches in real-world development and that concepts such as models@run.time are not explored (which would ease the process of dealing with large-scale systems that are very dynamic). They also point out that mashup-based development tools would benefit from using model-based concepts. Nonetheless, both approaches have a common shortcoming: *if systems are very dynamic, modeling and mashup tools cannot accurately represent the system graphically* [219].

By analyzing the selected tools and their roles, both in academia and the market, we observed that two of them share wide popularity (*cf.* Fig. 13): (1) Node-RED as a mashup-based tool that also leverages visual programming, and (2) ThingML as a model-based solution highly inspired in UML. We consider these two approaches as reference implementations of these two development models (mashup- and model-based) and will analyze and compare them in detail in the following paragraphs.

Node-RED's approach to IoT development centers itself in a visual data-flow-based programming model. It does not leverage the use of models as a way of abstracting components (*e.g.*, devices), thus increasing the complexity of developing new *nodes* — requiring us to deal with the essential complexity of the *node* (*e.g.*, a new algorithm) and in the accidental complexity of communicating with the already existing *nodes*. An observable side-effect is that when exporting existing flows, the exported format is barely understandable by humans, thus limiting possible modifications outside the Node-RED environment [221].

Node-RED also has several shortcomings, including the lack of a proper way to debug and test *flows*, which becomes essential given that IoT systems are typically large-scale and complex by nature, being easy to end up with highly-complex *flows* that are visually hard to understand and reason. Some works have been trying to improve some of the development issues of Node-RED, either by enriching the visual notations or by adding development mechanisms common to other development solutions, *i.e.*, linting, debugging, and testing[22] [291,292].

The shortcomings of mashup-based development solutions, which are shared with other visual programming approaches, pose a considerable problem regarding scalability — at least from a developing process perspective — due to limitations in the usage of visual metaphors. A study by Petre [293] acknowledged that developers think that *simply repackaging massive textual information into a massive graphical representation is not helpful*, raising the need of adding *means of reasoning about artifacts too enormous to encompass fully in one view*.

Several authors have been trying to improve Node-RED. As an example, Blackstock et al. [230] and Margarida et al. [239] works include modifications that enable Node-RED to orchestrate the parts of a given flow in a *distributed fashion*, *i.e.*, orchestrating tasks between servers, gateways, and devices collaborating and coordinating actions defined in the same flow.

ThingML as a model-based development environment does fully leverage models as a way to develop IoT systems [174,294]. However, to use some of the device's features, it is required to add device-specific code (*i.e.*, leaky abstractions). When comparing to UML, one key difference is that ThingML's primary syntax is lexical and not graphical despite visual notations being the most common approach for Model-Driven Engineering [295]. It lacks in covering the full development life-cycle due to software deployment and updates limitations. Also, the ability to share computational resources and devices among IoT applications in a reliable and foreseeable fashion is not covered.

From the analysis of these two tools, several conclusions can be made that show that both miss some essential features. Node-RED has a simple to use drag-n-drop visual programming interface that can be used as a reference in the domain of IoT but needs to be extended to embrace the necessities of real-time feedback (*i.e.*, development feedback-loop) [185,296]. As per the current state of practice, we agree with Prehofer et al. [219] when they state that a model-based approach can be a suitable abstraction for IoT, as it is already proposed by approaches such as ThingML. However, it must contemplate other aspects, including the use of features common to mashup-based approaches.

From Fig. 13, and taking into account the observations mentioned earlier, we consider that there is a more significant focus on model-driven solutions, but typically these kinds of tools are unpopular and are at a more experimental development phase than mashup-based solutions.

6.2.3. *End-user development*

Popular end-user development environments are based on hybrid (visual–textual) rule-based programming (*if-this-do-that* or *when-trigger-then-action*). However, these kinds of solutions are not enough to express more sophisticated intents, *i.e.*, the process of expressing rules based on time, space, and fuzzy conditions make it too complicated for the end-user to attain their preferences [297].

There has been an effort towards conversational assistants and other NLP-based solutions for users with limited to no technical knowledge to ease their interaction with IoT systems. However, most of the solutions are limited in some aspects: (1) the lack of standards (and the disregard by the existing ones by practitioners), most of the solutions are limited to certain devices or ecosystems, (2) the simplicity offered by some solutions limits the users' ability to configure more complex[23] scenarios and behaviors [298], (3) several limitations in what regards test and analysis of the resulting configuration (users have to manually trigger events to check if their program matches their expectations), thus making the system more opaque and hard to understand "why" some events happen — causality — and (4) some existing solutions depend on a first setup by technicians [18,251].

Although there is a considerable amount of research challenges associated with the points mentioned above, there are still issues and challenges that are not being tackled effectively either by practitioners or the research community. This is a bigger problem since it is hard to draw a line between the simplicity of end-user development and the ability to configure and manage complex scenarios without requiring a certain degree of expertise (*e.g.*, power-users).

7. Threats to validity

Secondary studies, like the one presented in this work, are vulnerable to threats to validity. Although this has not been a systematic literate review, the same type of threats still apply [299]. In the following paragraphs we present some of them, justify their existence and how they were addressed:

[22] As of version 2.0, Node-RED added support for linting the visual flows and introduced some new debugging mechanisms [290].

[23] As an example, the complexity of managing devices schedules which rises with the number of devices and the shared conflicting preferences of household members.

- **Non-specification of the review setting.** The replicability of this study is limited due to the lack of a registry of the precise queries performed as they were defined on a case-by-case basis. Although we are aware of this threat, having a well-defined set of queries and keywords would impact the coverage of this work;
- **Incorrect or incomplete search terms.** Search terms used in automated queries can have a significant impact on the amount, quality, and relevance of retrieved works, especially as contextual ambiguity can cause noise in the results. For that reason, queries were tuned on a case-by-case basis, and query results were carefully vetted. As we only used one digital library, we did not have to adapt search terms between different ones. Additionally, when required, additional keywords were used to provide more reliable results. As an example, when searching for `HTTP` we were forced to change to `HTTP protocol` due to the high number of unrelated results of the first keyword. We are aware that this can result in an erroneous number of results being returned, thus we analyzed the first twenty results to validate each search term.
- **Inappropriate search method.** A deep analysis of highly-cited papers can result in a bias where papers cited by those highly-cited papers have a higher chance of being included in the survey (*e.g.*, self-citations). By not resorting solely to snowballing techniques, we partially avoided this problem.
- **Non-exhaustive venues or databases.** Some relevant works can pass unnoticed given the indexation limits and search capabilities of *Google Scholar*. As aforementioned, the first queries were run across several indexing services, which yield minimal differences when compared to *Google Scholar*. Nonetheless, we attempt to mitigate this by processing a large number of results for each one of the queries made and by experimenting with different keywords for each one of the specific topics addressed in this work;
- **Inappropriate inclusion & exclusion criteria.** The criteria used to include and exclude works from a literature review is of paramount importance as it will impact which works are considered. As we wanted to do a comprehensive study that includes both scientific and grey literature, we opted to use very flexible criteria to consider all relevant works. This is inherently a threat to the validity of this work, given that results that just mention some of the considered search keywords could be considered, even if they fall out of the scope. Nonetheless, efforts were carried to only select works that were indeed relevant to the construction and design of IoT systems from software and hardware perspectives. The only exception to this was the popularity studies that included any matching result.
- **Bias in Study Selection.** Authors have their subjective views on several aspects of the field and this can impact the choices done when selecting which works to include or exclude from this work. Every decision was double-checked to ensure that inclusion and exclusion criteria were correctly applied.

Although these threats were considered from the start of this work and mitigated by the means available, we are aware that they could influence the results of this survey. Given this, we advise the reader to consider these when analyzing this work.

8. Conclusions

Software engineering has been striving to design, develop, maintain, and operate software systems for a long time. The current state of software engineering for IoT systems is far behind the best practices used when developing other kinds of software systems (*e.g.*, cloud-based software solutions). Due to this current state of affairs, practitioners are having difficulties finding workers with the right skill-set and expertise to develop IoT systems [1].

The essential complexity that characterizes the IoT demands a considerable shift in the processes, methodologies, tools and practices for creating and maintaining this kind of system. This essential complexity results from the reunion of factors such as the large-scale systems, their heterogeneity, a demand for high-data throughput, human-in-the-loop considerations, and real-time needs. Several initiatives have been pushing towards standards and reference architectures for designing and developing IoT systems. Despite these initiatives, they are not having a noticeable impact on how developers tackle IoT, and, as a result, fragmentation of IoT solutions (vertical-silos and vendor-locks) is *standard*. As Chander et al. point out, IoT is *still in the hunt for its very own pattern, shape, and architectural structure* [300].

From a system design perspective, a reasonable number of advances have been done both at the physical and logical architectural levels. These advances include several design principles, architectural styles, and reference architectures, which have been proposed to meet IoT requirements. As an example, efforts have been made to define a pattern language for IoT systems that can help design them. However, the majority of these solutions are still in their infancy in terms of maturity. However, areas with particularities similar to those of IoT systems (*e.g.*, distributed systems, critical systems, industrial control systems) have an extensive body of knowledge that can be adapted and used to design IoT solutions.

Regarding the construction of IoT systems, the most complete examples of both model-driven development and mashup-based development have several limitations. On the one hand, the level of abstraction achieved by Node-RED as a model-based development environment is low, although it has a simple to use *drag-n-drop* visual programming interface that can be used as a reference in the domain of IoT. On the other hand, ThingML fully leverages models as a way to develop IoT systems, but it does not embrace the use of visual notations as it is text-based and is substantially unpopular when compared to traditional development approaches.

Nonetheless, some of the most significant problems and barriers towards an IoT ecosystem are fragmentation and the disregard of interoperability between systems, architectures and *vendors* [3], issues that limit the wide adoption of common practices and tools since each device or system depends on device- and vendor-specific toolchains and protocols. Thus, we identify the creation and adoption of common standards (including protocols and reference architectures) by the industry as the cornerstone of future IoT systems.

From a more holistic viewpoint, and sharing the same concerns as Nordrum et al. [259], as the number of connected devices is fast approaching ten thousand million, the industry faces novel and formidable challenges from different perspectives, such as securing and powering all these devices. Also, governmental organizations will need to start working on how to handle all the resulting *e-waste* and how to tackle new privacy, security, and safety realities, including the definition of new regulations and certification processes.

We consider that the next steps to this work include a systematized comparison study of the different solutions, tools, and approaches described. This could complement this work by providing supporting evidence for any interested individual that wants to understand what is the most suitable solutions, tools, or approaches for their specific use case, application domain, or problem.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Microsoft, IoT Signals – Summary of Research Learnings, Tech. Rep., Microsoft, 2019.
[2] B. Fitzgerald, Software crisis 2.0, Computer 45 (4) (2012) 89–91.
[3] M. Aly, F. Khomh, Y. Guéhéneuc, H. Washizaki, S. Yacout, Is fragmentation a threat to the success of the internet of things? IEEE Internet Things J. 6 (1) (2019) 472–487.
[4] A. Kevin, That 'internet of things' thing, RFID J. (2009) URL https://www.rfidjournal.com/articles/view?4986.
[5] E. Fritsch, I. Shklovski, R. Douglas-Jones, Calling for a revolution: An analysis of IoT manifestos, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, in: CHI '18, ACM, New York, NY, USA, 2018, pp. 302:1–302:13.
[6] A. Taivalsaari, T. Mikkonen, A roadmap to the programmable world: Software challenges in the IoT era, IEEE Softw. 34 (1) (2017) 72–80.
[7] X. Larrucea, A. Combelles, J. Favaro, K. Taneja, Software engineering for the internet of things, IEEE Softw. 34 (1) (2017) 24–28.
[8] F. Zambonelli, Key abstractions for IoT-oriented software engineering, IEEE Softw. 34 (1) (2017) 38–45.
[9] D. Soares, J.P. Dias, A. Restivo, H.S. Ferreira, Programming IoT-spaces: A user-survey on home automation rules, in: Proceedings of the 21st International Conference on Computational Science, ICCS, Springer, 2021.
[10] G. Fortino, C. Savaglio, G. Spezzano, M. Zhou, Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools, IEEE Trans. Syst. Man Cybern. Syst. 51 (1) (2020) 223–236.
[11] S. Al-Sarawi, M. Anbar, K. Alieyan, M. Alzubaidi, Internet of things (IoT) communication protocols: Review, in: 2017 8th Int. Conference on Information Technology (ICIT), 2017, pp. 685–690.
[12] M.A. Khan, K. Salah, IoT Security: Review, blockchain solutions, and open challenges, Future Gener. Comput. Syst. 82 (2018) 395–411.
[13] M. Abomhara, G.M. Køien, Security and privacy in the internet of things: Current status and open issues, in: 2014 Int. Conference on Privacy and Security in Mobile Systems, PRISMS, IEEE, 2014, pp. 1–8, http://dx.doi.org/10.1109/PRISMS.2014.6970594.
[14] D. Glaroudis, A. Iossifides, P. Chatzimisios, Survey, comparison and research challenges of IoT application protocols for smart farming, Comput. Netw. 168 (2020) 107037.
[15] M. Centenaro, L. Vangelista, A. Zanella, M. Zorzi, Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios, IEEE Wirel. Commun. 23 (5) (2016) 60–67.
[16] M.J. Kaur, P. Maheshwari, Building smart cities applications using IoT and cloud-based architectures, in: 2016 Int. Conference on Industrial Informatics and Computer Systems, CIICS, IEEE, 2016, pp. 1–5.
[17] H. Jaidka, N. Sharma, R. Singh, Evolution of iot to iiot: Applications & challenges, in: Proceedings of the International Conference on Innovative Computing & Communications, ICICC, 2020.
[18] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, A. Pierantonio, Low-code engineering for internet of things: A state of research, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, in: MODELS '20, Association for Computing Machinery, New York, NY, USA, 2020.
[19] P.P. Ray, A survey of IoT cloud platforms, Future Comput. Inf. J. 1 (1) (2016) 35–46.
[20] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E.B. Fernandez, N. Yoshioka, Landscape of architecture and design patterns for IoT systems, IEEE Internet Things J. 7 (10) (2020) 10091–10101, http://dx.doi.org/10.1109/JIOT.2020.3003528.
[21] P. Ray, A survey on internet of things architectures, J. King Saud Univ. Comput. Inf. Sci. 30 (3) (2018) 291–319.
[22] L. Cablová, R. Pates, M. Miovskỳ, J. Noel, How to write a systematic review article and meta-analysis, in: Addiction Science: A Guide for the Perplexed, 2017, p. 173.
[23] C. Wang, N. Kato, Inaugural editorial, IEEE Internet Things J. 1 (1) (2014) 1–2, http://dx.doi.org/10.1109/JIOT.2014.2318793.
[24] SERP4IoT '19: Proceedings of the 1st International Workshop on Software Engineering Research & Practices for the Internet of Things, IEEE Press, 2019.
[25] I.C. Society, P. Bourque, R.E. Fairley, Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0, third ed., IEEE Computer Society Press, Los Alamitos, CA, USA, 2014.
[26] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer, 2007.
[27] S. Chang, Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co. 2002.
[28] J. Mishra, Software Engineering, Pearson Education India, 2011.
[29] I. Sommerville, Software Engineering, ninth ed., Addison-Wesley Publishing Company, USA, 2010.
[30] ISO/IEC JTC 1, Internet of Things (IoT) - Preliminary Report, Tech. Rep, ISO, 2014.
[31] H. Tankovska, Number of Internet of Things (IoT) Connected Devices Worldwide In 2018, 2025 and 2030, Tech. Rep., Strategy Analytics, 2018, Data by Statista URL https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/.
[32] F. Alkhabbas, R. Spalazzese, P. Davidsson, IoT-Based systems of systems, in: Proceedings of the 2nd Edition of Swedish Workshop on the Engineering of Systems of Systems, SWESOS 2016, Gothenburg University, 2016.
[33] R. Buyya, A.V. Dastjerdi, Internet of Things: Principles and Paradigms, Elsevier, 2016.
[34] Y. Liao, E.D.F.R. Loures, F. Deschamps, Industrial internet of things: A systematic literature review and insights, IEEE Internet Things J. 5 (6) (2018) 4515–4525.
[35] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelffle, Vision and Challenges for Realising the Internet of Things, Tech. Rep., European Commission, 2010.
[36] P. Scully, The top 10 IoT segments in 2018 – based on 1,600 real IoT projects, 2018, IoT Analytics GmbH, Online URL https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/. (Accessed 15 May 2018).

[37] S. Prescher, A.K. Bourke, F. Koehler, A. Martins, H. Sereno Ferreira, T. Boldt Sousa, R.N. Castro, A. Santos, M. Torrent, S. Gomis, M. Hospedales, J. Nelson, Ubiquitous ambient assisted living solution to promote safer independent living in older adults suffering from co-morbidity, in: 2012 Annual Int. Conference of the IEEE Engineering in Medicine and Biology Society, 2012, pp. 5118–5121.
[38] Z. Ji, Q. Anwen, The application of internet of things(IOT) in emergency management system in China, in: 2010 IEEE Int. Conference on Technologies for Homeland Security, HST, 2010, pp. 139–142.
[39] A.K. Sharma, M.F.R. Ansari, M.F. Siddiqui, M.A. Baig, IOT Enabled forest fire detection and online monitoring system, Int. J. Curr. Trends Eng. Res. (IJCTER) 3 (5) (2017) 50–54.
[40] K. Lyytinen, Y. Yoo, Ubiquitous computing, Commun. ACM 45 (12) (2002) 63–96.
[41] J. Krumm, Ubiquitous Computing Fundamentals, Chapman and Hall/CRC, 2016.
[42] H. Aksu, L. Babun, M. Conti, G. Tolomei, A.S. Uluagac, Advertising in the IoT era: Vision and challenges, IEEE Commun. Mag. 56 (11) (2018) 138–144.
[43] J.G. Andrews, S. Buzzi, W. Choi, S.V. Hanly, A. Lozano, A.C.K. Soong, J.C. Zhang, What will 5G Be? IEEE J. Sel. Areas Commun. 32 (6) (2014) 1065–1082.
[44] S. Munir, J.A. Stankovic, C.-J.M. Liang, S. Lin, Cyber physical system challenges for human-in-the-loop control, in: Presented As Part of the 8th Int. Workshop on Feedback Computing, USENIX, San Jose, CA, 2013, pp. 777–780.
[45] J.A. Stankovic, Research directions for the internet of things, IEEE Internet Things J. 1 (1) (2014) 3–9.
[46] I.S. Udoh, G. Kotonya, Developing IoT applications: challenges and frameworks, IET Cyber-Phys. Syst. Theory Appl. 3 (2) (2018) 65–72.
[47] W. Zhao, J. Liu, H. Guo, T. Hara, Etc-iot: Edge-node-assisted transmitting for the cloud-centric internet of things, IEEE Netw. 32 (3) (2018) 101–107.
[48] D.R. Vasconcelos, R.M. Andrade, V. Severino, J.N. De Souza, Cloud, fog, or mist in IoT? That is the qestion, ACM Trans. Internet Technol. 19 (2) (2019) http://dx.doi.org/10.1145/3309709.
[49] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, J. Henry, IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things, Cisco Press, 2017.
[50] M. Iorga, L. Feldman, R. Barton, M.J. Martin, N.S. Goren, C. Mahmoudi, Fog Computing Conceptual Model, Tech. Rep., National Institute of Standards and Technology (NIST), 2018.
[51] A. Seitz, F. Thiele, B. Bruegge, Fogxy: An architectural pattern for fog computing, in: Proceedings of the 23rd European Conference on Pattern Languages of Programs, Vol. 1, ACM, 2018, p. 33.
[52] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications, IEEE Internet Things J. 4 (5) (2017) 1125–1142.
[53] R. Toal, Software Architecture, Loyola Marymount University, 2018, URL https://cs.lmu.edu/∼ray/notes/architecture/.
[54] J. Im, S. Kim, D. Kim, IoT Mashup as a service: cloud-based mashup service for the internet of things, in: 2013 IEEE Int. Conference on Services Computing, IEEE, 2013, pp. 462–469.
[55] T. Pflanzner, A. Kertész, A survey of IoT cloud providers, in: 2016 39th Int. Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, IEEE, 2016, pp. 730–735.
[56] K.J. Singh, D.S. Kapoor, Create your own internet of things: A survey of IoT platforms, IEEE Consum. Electr. Mag. 6 (2) (2017) 57–68.
[57] M. Alioto, M. Shahghasemi, The internet of things on its edge: Trends toward its tipping point, IEEE Consum. Electr. Mag. 7 (1) (2017) 77–87.
[58] M. Vazquez-Briseno, F.I. Hirata, J. de Dios Sanchez-Lopez, E. Jimenez-Garcia, C. Navarro-Cota, J.I. Nieto-Hipolito, Using RFID/NFC and QR-code in mobile phones to link the physical and the digital world, in: Interactive Multimedia, IntechOpen, 2012.
[59] K.E. Jeon, J. She, P. Soonsawad, P.C. Ng, BLE Beacons for internet of things applications: Survey, challenges, and opportunities, IEEE Internet Things J. 5 (2) (2018) 811–828.
[60] P.P. Ray, A survey on visual programming languages in internet of things, Sci. Program. 2017 (2017) 1–6.
[61] S.K. Goudos, P.I. Dallas, S. Chatziefthymiou, S. Kyriazakos, A survey of IoT key enabling and future technologies: 5G, mobile IoT, sematic web and applications, in: Wireless Personal Communications, Vol. 97, Springer US, 2017, pp. 1645–1675.
[62] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, T. Watteyne, Understanding the limits of LoRaWAN, IEEE Commun. Mag. 55 (9) (2017) 34–40.
[63] U. Mehboob, Q. Zaib, C. Usama, Survey of IoT Communication Protocols: Techniques, Applications, and Issues, Tech. Rep., xFlow Research Inc, 2016, URL http://xflowresearch.com/wp-content/uploads/2016/02/Survey-of-IoT-Communication-Protocols.pdf.
[64] Connectivity Standards Alliance (formerly Zigbee Alliance), Matter (formerly project CHIP), 2020, URL https://buildwithmatter.com/. (Last Access: 2021).
[65] D. Lan, Z. Pang, C. Fischione, Y. Liu, A. Taherkordi, F. Eliassen, Latency analysis of wireless networks for proximity services in smart home and building automation: The case of thread, IEEE Access 7 (2019) 4856–4867.
[66] P. Andrés-Maldonado, et al., NB-IoT M2M Communications in 5G Cellular Networks (Ph.D. thesis), Universidad de Granada, 2019.
[67] M. Gast, 802.11 Wireless Networks: The Definitive Guide, " O'Reilly Media, Inc." 2005.
[68] C. Rey-Moreno, Z. Roro, W.D. Tucker, M.J. Siya, N.J. Bidwell, J. Simo-Reigadas, Experiences, challenges and lessons from rolling out a rural WiFi mesh network, in: Proceedings of the 3rd ACM Symposium on Computing for Development, 2013, pp. 1–10.
[69] S. Banerji, R.S. Chowdhury, Wi-Fi & WiMAX: A comparative study, 2013, arXiv preprint arXiv:1302.2247.
[70] Y. Toor, P. Muhlethaler, A. Laouiti, A.D. La Fortelle, Vehicle ad hoc networks: applications and related technical issues, IEEE Commun. Surv. Tutor. 10 (3) (2008) 74–88.
[71] W.S. Jeon, H.S. Oh, D.G. Jeong, Decision of ranging interval for IEEE 802.15. 4z UWB ranging devices, IEEE Internet Things J. (2021).
[72] D. Tsonev, S. Videv, H. Haas, Light fidelity (Li-Fi): towards all-optical networking, in: Broadband Access Communication Technologies VIII, Vol. 9007, International Society for Optics and Photonics, 2014, 900702.
[73] L.P.J. Rani, M.K. Kumar, K. Naresh, S. Vignesh, Dynamic traffic management system using infrared (IR) and internet of things (IoT), in: 2017 Third International Conference on Science Technology Engineering & Management, ICONSTEM, IEEE, 2017, pp. 353–357.
[74] A. Singh, P. Aggarwal, R. Arora, IoT Based waste collection system using infrared sensors, in: 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), ICRITO, IEEE, 2016, pp. 505–509.
[75] J. Dizdarevi&#x00107;, F. Carpio, A. Jukan, X. Masip-Bruin, A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration, ACM Comput. Surv. 51 (6) (2019) 116:1–116:29.
[76] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, J. Alonso-Zarate, A survey on application layer protocols for the internet of things, Trans. IoT Cloud Comput. 3 (1) (2015) 11–17, URL http://icas-pub.org/ojs/index.php/ticc/article/view/47.
[77] J. Iyengar, M. Thomson, QUIC: A UDP-Based multiplexed and secure transport, 2019, Internet Engineering Task Force IETF URL https://tools.ietf.org/html/draft-ietf-quic-transport-23.
[78] M. Bishop, Hypertext transfer protocol version 3 (HTTP/3), 2019, Internet Engineering Task Force IETF URL https://tools.ietf.org/html/draft-ietf-quic-http-23.
[79] D. Soni, A. Makwana, A survey on MQTT: a protocol of internet of things (IoT), in: Int. Conference on Telecommunication, Power Analysis and Computing Techniques, ICTPACT-2017, 2017.
[80] A. Stanford-Clark, H.L. Truong, Mqtt for sensor networks (mqtt-sn) protocol specification, Int. Bus. Mach. (IBM) Corp. Version 1 (2013).
[81] D. Dell'Aglio, D. Le Phuoc, A. Lê Tuán, M.I. Ali, J.-P. Calbimonte, On a web of data streams, in: DeSemWeb ISWC, 2017.
[82] P. Kumar, B. Dezfouli, Implementation and analysis of QUIC for MQTT, 2018, CoRR, arXiv:1810.07730.

[83] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP, in: 2017 IEEE Int. Systems Engineering Symposium, ISSE, 2017, pp. 1–7.
[84] A. Al-Fuqaha, S. Member, M. Guizani, M. Mohammadi, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376.
[85] S. Reichhard, UPnP and DPWS, Tech. Rep., Technical University of Vienna, 2013.
[86] Y. Chen, T. Kunz, Performance evaluation of IoT protocols under a constrained wireless access network, in: 2016 Int. Conference on Selected Topics in Mobile Wireless Networking, MoWNeT, 2016, pp. 1–7.
[87] H. Muccini, M.T. Moghaddam, IoT Architectural Styles: A Systematic Mapping Study, Springer International Publishing, 2018, pp. 68–85.
[88] Y. Zhang, L. Duan, J.L. Chen, Event-driven SOA for IoT services, in: 2014 IEEE Int. Conference on Services Computing, 2014, pp. 629–636.
[89] Y. Zhang, L. Duan, J.L. Chen, Event-driven soa for iot services, in: 2014 IEEE Int. Conference on Services Computing, IEEE, 2014, pp. 629–636.
[90] O. Uviase, G. Kotonya, IoT Architectural framework: Connection and integration framework for IoT systems, Electr. Proc. Theor. Comput. Sci. 264 (2018) 1–17.
[91] T. Teixeira, S. Hachem, V. Issarny, N. Georgantas, Service oriented middleware for the internet of things: a perspective, in: European Conference on a Service-Based Internet, Springer, 2011, pp. 220–229.
[92] C. Richardson, Microservices Patterns: With Examples in Java, Manning, 2018.
[93] A. Krylovskiy, M. Jahn, E. Patti, Designing a smart city internet of things platform with microservice architecture, in: 2015 3rd Int. Conference on Future Internet of Things and Cloud, 2015, pp. 25–30.
[94] L. Sun, Y. Li, R.A. Memon, An open IoT framework based on microservices architecture, China Commun. 14 (2) (2017) 154–162.
[95] D. Lu, D. Huang, A. Walenstein, D. Medhi, A secure microservice framework for IoT, in: 2017 IEEE Symposium on Service-Oriented System Engineering, SOSE, 2017, pp. 9–18.
[96] P.T. Eugster, P.A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2) (2003) 114–131.
[97] C.Y. Chen, J.H. Fu, T. Sung, P.-F. Wang, E. Jou, M.-W. Feng, Complex event processing for the internet of things and its applications, in: 2014 IEEE Int. Conference on Automation Science and Engineering, CASE, IEEE, 2014, pp. 1144–1149.
[98] S. Alam, M.M.R. Chowdhury, J. Noll, SenaaS: An event-driven sensor virtualization approach for internet of things cloud, in: 2010 IEEE Int. Conf. on Networked Embedded Systems for Enterprise Applications, 2010, pp. 1–6.
[99] B. Butzin, F. Golatowski, D. Timmermann, Microservices approach for the internet of things, in: 2016 IEEE 21st Int. Conference on Emerging Technologies and Factory Automation, ETFA, 2016, pp. 1–6.
[100] A. Preventis, K. Stravoskoufos, S. Sotiriadis, E.G. Petrakis, IoT-A And FIWARE: Bridging the barriers between the cloud and IoT systems design and implementation, in: CLOSER (2), 2016, pp. 146–153.
[101] M. Blackstock, N. Kaviani, R. Lea, A. Friday, MAGIC Broker 2: An open and extensible platform for the internet of things, in: 2010 Internet of Things, IOT, IEEE, 2010, pp. 1–8.
[102] M. Collina, G.E. Corazza, A. Vanelli-Coralli, Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST, in: 2012 IEEE 23rd Int. Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, IEEE, 2012, pp. 36–41.
[103] K. Misura, M. Zagar, Internet of things cloud mediator platform, in: 2014 37th Int. Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO, 2014, pp. 1052–1056.
[104] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, B. Amos, Privacy mediators: Helping IoT cross the chasm, in: Proceedings of the 17th Int. Workshop on Mobile Computing Systems and Applications, ACM, 2016, pp. 39–44.
[105] D. Guinard, I. Ion, S. Mayer, In search of an internet of things service architecture: REST or WS-∗? A developers' perspective, in: Int. Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, Springer, 2011, pp. 326–337.
[106] D. Guinard, V. Trifa, E. Wilde, et al., A resource oriented architecture for the web of things, in: IoT, 2010, pp. 1–8.
[107] A.P. Castellani, M. Gheda, N. Bui, M. Rossi, M. Zorzi, Web services for the internet of things through CoAP and EXI, in: 2011 IEEE Int. Conference on Communications Workshops, ICC, IEEE, 2011, pp. 1–6.
[108] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP, in: 2017 IEEE Int. Systems Engineering Symposium, ISSE, IEEE, 2017, pp. 1–7.
[109] P.R. Chelliah, H. Subramanian, A. Murali, Architectural Patterns: Uncover Essential Patterns in the Most Indispensable Realm of Enterprise Architecture, Packt Publishing Ltd, 2017.
[110] C. Becker, C. Julien, P. Lalanda, F. Zambonelli, Pervasive computing middleware: current trends and emerging challenges, CCF Trans. Pervasive Comput. Interact. (2019) 1–14.
[111] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, M. Bone, The concept of reference architectures, Syst. Eng. 14 (3) (2009).
[112] M. Weyrich, C. Ebert, Reference architectures for the internet of things, IEEE Softw. 33 (1) (2016) 112–116.
[113] R. Saemaldahr, B. Thapa, K. Maikoo, E.B. Fernandez, Reference architectures for the IoT: A survey, in: F. Saeed, F. Mohammed, A. Al-Nahari (Eds.), Innovative Systems for Intelligent Health Informatics, Springer International Publishing, Cham, 2021, pp. 635–646.
[114] IBM, Internet of Things Architecture, 2021, IBM URL https://www.ibm.com/cloud/architecture/architectures/iotArchitecture/reference-architecture/. (Last Access 2021).
[115] Azure, Azure IoT reference architecture, 2021, Microsoft URL https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot. (Last Access 2021).
[116] AWS, Architecture best practices for IoT, 2021, Amazon URL https://aws.amazon.com/architecture/iot/. (Last Access 2021).
[117] L. Gaur, R. Ramakrishnan, Internet of Things: Approach and Applicability in Manufacturing, CRC Press, 2019, http://dx.doi.org/10.1201/9780429486593.
[118] B. Francis, Web Thing API, Tech. Rep., Mozilla, 2017.
[119] D. Janes, IOTDB, Tech. Rep., IOTDB.org, 2017.
[120] V. Trifa, D. Guinard, D. Carrera, Web Thing Model, Tech. Rep., 2017, EVRYTHNG.
[121] S. Liang, C.-Y. Huang, T. Khalafbeigi, OGC Sensorthings API, 2019, Open Geospatial Consortium.
[122] C. Jennings, Z. Shelby, Cisco, Sensinode, Ericsson, J. Arkko, Media types for sensor markup language (SENML), 2013, Online URL https://tools.ietf.org/html/draft-jennings-senml-10. (Accessed 19 May 2018).
[123] D. Thaler, H. Tschofenig, J. Jimenez, Report from the Internet of Things (IoT) Semantic Interoperability (IOTSI) Workshop 2016, Tech. Rep., Internet Engineering Task Force (IETF), 2018.
[124] A. Bujari, M. Furini, F. Mandreoli, R. Martoglia, M. Montangero, D. Ronzani, Standards, security and business models: Key challenges for the IoT scenario, Mob. Netw. Appl. 23 (1) (2018) 147–154.
[125] M. Noura, M. Atiquzzaman, M. Gaedke, Interoperability in internet of things: Taxonomies and open challenges, Mob. Netw. Appl. 24 (3) (2019) 796–809.
[126] G. Meszaros, J. Doble, Pattern Languages of Program Design, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997, pp. 529–574.
[127] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995, p. 334.
[128] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional, 2004.
[129] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg, Internet of things patterns for device bootstrapping and registration, in: 22nd European Conference on Pattern Languages of Programs, ACM, 2017, p. 15.

[130] D.A. Lukas Reinfurt, U. Breitenbücher, M. Falkenthal, P. Fremantle, F. Leymann, Internet of things security patterns, in: Proceedings of the 24rd Conference Pattern Languages of Programs, ACM, The Hillside Group, 2017.

[131] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg, Internet of things patterns, in: 21st European Conference on Pattern Languages of Programs, 2016, pp. 1–21.

[132] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg, Internet of things patterns for devices, in: Ninth Int. Conferences on Pervasive Patterns and Applications (PATTERNS) 2017, in: EuroPlop16, Xpert Publishing Services (XPS), New York, NY, USA, 2017, pp. 117–126.

[133] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, S. Dustdar, IoT Design patterns: Computational constructs to design, build and engineer edge applications, in: IEEE First Int. Conference on Internet-of-Things Design and Implementation (IoTDI), 2016, IEEE, 2016, pp. 277–282.

[134] G. Bloom, B. Alsulami, E. Nwafor, I.C. Bertolotti, Design patterns for the industrial internet of things, in: 2018 14th IEEE Int. Workshop on Factory Communication Systems, WFCS, IEEE, 2018, pp. 1–10.

[135] L. Reinfurt, U. Breitenbucher, M. Falkenthal, F. Leymann, A. Riegg, Internet of things patterns for devices: Powering, operating, and sensing, Int. J. Adv. Internet Technol. (2017) 106–123.

[136] A. Ramadas, G. Domingues, J.P. Dias, A. Aguiar, H.S. Ferreira, Patterns for things that fail, in: Proceedings of the 24th Conference on Pattern Languages of Programs, in: PLoP '17, ACM - Association for Computing Machinery, 2017, pp. 7:1–7:10.

[137] J.P. Dias, T. Boltd, H.F. Sereno, Testing and deployment patterns for the internet-of-things, in: Proceedings of the 24th European Conference on Pattern Languages of Programs, ACM, 2019.

[138] J.P. Dias, T.B. Sousa, A. Restivo, H.S. Ferreira, A pattern-language for self-healing internet-of-things systems, in: Proceedings of the 25th European Conference on Pattern Languages of Programs, in: EuroPLop '20, Association for Computing Machinery, New York, NY, USA, 2020, http://dx.doi.org/10.1145/3361149.3361165.

[139] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications, Springer, 2014, http://dx.doi.org/10.1007/978-3-7091-1568-8.

[140] P.M. Duvall, A. Glover, S. Matyas, Continuous Integration, Addison-Wesley Professional, 2007.

[141] R. Hanmer, Patterns for Fault Tolerant Software, Wiley Publishing, 2007.

[142] V.-P. Eloranta, J. Koskinen, M. Leppnen, V. Reijonen, Designing Distributed Control Systems: A Pattern Language Approach, Wiley Publishing, 2014.

[143] T.B. Sousa, F.F. Correia, H.S. Ferreira, Patterns for software orchestration on the cloud, in: Proceedings of the 22nd Conference on Pattern Languages of Programs, in: PLoP '15, ACM, The Hillside Group, USA, 2015, p. 17.

[144] H. Washizaki, N. Yoshioka, A. Hazeyama, T. Kato, H. Kaiya, S. Ogata, T. Okubo, E.B. Fernandez, Landscape of iot patterns, in: Proceedings - 2019 IEEE/ACM 1st International Workshop on Software Engineering Research and Practices for the Internet of Things, SERP4IoT 2019, IEEE, 2019, pp. 57–60.

[145] M. Shahin, M. Ali Babar, L. Zhu, Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices, IEEE Access 5 (Ci) (2017) 3909–3943.

[146] J. Ruehlin, Continuous delivery and iterative development for internet of things projects, 2018, URL https://www.ibm.com/cloud/garage/content/deliver/practice_continuous_delivery_iot/.

[147] H. Lin, N.W. Bergmann, IoT Privacy and security challenges for smart home environments, Information 7 (3) (2016) 44.

[148] N. Beijer, Continuous delivery in IoT environments, in: A. Alliance (Ed.), Int. Conference on Agile Software Development, Agile Alliance, 2018, pp. 1–7.

[149] C. Prehofer, L. Chiarabini, From IoT mashups to model-based IoT, in: W3C Workshop on the Web of Things, 2013.

[150] M.B. Hoy, If this then that: An introduction to automated task services, Med. Ref. Serv. Q. 34 (1) (2015) 98–103, http://dx.doi.org/10.1080/02763869.2015.986796, PMID: 25611444.

[151] A. Rahmati, E. Fernandes, J. Jung, A. Prakash, IFTTT Vs. Zapier: A comparative study of trigger-action programming frameworks, 2017, ArXiv, arXiv:1709.02788.

[152] I. Eclipse Foundation, IoT Developer Survey 2019, Eclipse Foundation, Inc. 2019.

[153] PlatformIO, PlatformIO: An open source ecosystem for IoT development, 2019, PlatformIO URL https://platformio.org/.

[154] Arduino, Arduino, 2019, Arduino Open Source Hardware URL https://www.arduino.cc/.

[155] I.S. GmbH, Eclipse NeoSCADA, 2019, IBH SYSTEMS GmbH URL https://www.eclipse.org/eclipsescada/index.html.

[156] esp8266.ru, ESPlorer — Integrated development environment (IDE) for ESP8266, 2019, esp8266.ru URL https://esp8266.ru/esplorer/.

[157] Microchip Technology Inc., Atmel studio 7 | microchip technology, 2019, Microchip Technology Inc. URL https://www.microchip.com/mplab/avr-support/atmel-studio-7.

[158] Pycom, Pymakr — IDE plugin for MicroPython, 2021, Pycom URL https://pycom.io/products/supported-networks/pymakr/. (Last Access 2021).

[159] I. Particle Industries, Particle IDE, 2021, Particle Industries, Inc. URL https://www.particle.io/developer-tools/. (Last Access 2021).

[160] J. Mineraud, O. Mazhelis, X. Su, S. Tarkoma, A gap analysis of internet-of-things platforms, Comput. Commun. 89–90 (2016) 5–16.

[161] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, E. Teniente, Enabling IoT ecosystems through platform interoperability, IEEE Softw. 34 (1) (2017) 54–61.

[162] D.C. Schmidt, Model-driven engineering, IEEE Comput. 39 (2) (2006) 25–31.

[163] M. van Amstel, M. van den Brand, Z. Protic, T. Verhoeff, Model-driven software engineering, in: Automation in Warehouse Development, Springer, London, 2012, pp. 45–58.

[164] D. Riehle, S. Fraleigh, D. Bucka-Lassen, N. Omorogbe, The architecture of a UML virtual machine, in: Int. Conference on Object Oriented Programming Systems Languages and Applications, OOSPLA, 2001, pp. 327–341.

[165] X.T. Nguyen, H.T. Tran, H. Baraki, K. Geihs, FRASAD: A Framework for model-driven IoT application development, in: 2015 IEEE 2nd World Forum on Internet of Things, WF-IoT, 2015, pp. 387–392.

[166] K. Thramboulidis, F. Christoulakis, UML4IoT–A UML-Based approach to exploit IoT in cyber-physical manufacturing systems, Comput. Ind. 82 (2016) 259–272.

[167] B. Costa, P.F. Pires, F.C. Delicato, W. Li, A.Y. Zomaya, Design and analysis of IoT applications: A model-driven approach, in: 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech, 2016, pp. 392–399.

[168] M. Hussein, S. Li, A. Radermacher, Model-driven development of adaptive IoT systems, in: MODELS (Satellite Events), 2017, pp. 17–23.

[169] F. Alkhabbas, R. Spalazzese, P. Davidsson, Architecting emergent configurations in the internet of things, in: 2017 IEEE Int. Conference on Software Architecture, ICSA, 2017, pp. 221–224.

[170] F. Ciccozzi, R. Spalazzese, MDE4IoT: SUpporting the internet of things with model-driven engineering, in: C. Badica, A. El Fallah Seghrouchni, A. Beynier, D. Camacho, C. Herpson, K. Hindriks, P. Novais (Eds.), Intelligent Distributed Computing X, Springer Int. Publishing, Cham, 2017, pp. 67–76.

[171] L. Mainetti, L. Manco, L. Patrono, I. Sergi, R. Vergallo, Web of topics: An IoT-aware model-driven designing approach, in: 2015 IEEE 2nd World Forum on Internet of Things, WF-IoT, 2015, pp. 46–51.

[172] F. Zambonelli, M. Viroli, G. Fortino, B. Re, Towards adaptive flow programming for the IoT: The fluidware approach, in: 2019 IEEE Int. Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019, IEEE, 2019, pp. 549–554.

[173] G. Fortino, B. Re, M. Viroli, F. Zambonelli, Fluidware: An approach towards adaptive and scalable programming of the IoT, in: Models, Languages, and Tools for Concurrent and Distributed Programming, Vol. 11665, Springer Int. Publishing, 2019, pp. 411–427.
[174] N. Harrand, F. Fleurey, B. Morin, K.E. Husa, ThingML: A language and code generation framework for heterogeneous targets, in: Proceedings of the ACM/IEEE 19th Int. Conference on Model Driven Engineering Languages and Systems, in: MODELS '16, ACM, New York, NY, USA, 2016, pp. 125–135.
[175] A. Dunkels, O. Schmidt, T. Voigt, M. Ali, Protothreads: Simplifying event-driven programming of memory-constrained embedded systems, in: Proceedings of the 4th Int. Conference on Embedded Networked Sensor Systems, in: SenSys '06, ACM, New York, NY, USA, 2006, pp. 29–42.
[176] C. Durmaz, M. Challenger, O. Dagdeviren, G. Kardas, Modelling contiki-based IoT systems, in: R. Queirós, M. Pinto, A.S. oes, J.P. Leal, M.J. Varanda (Eds.), 6th Symposium on Languages, Applications and Technologies, SLATE 2017, in: OpenAccess Series in Informatics (OASIcs), vol. 56, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017, pp. 5:1–5:13.
[177] X. Chen, A. Li, X. Zeng, W. Guo, G. Huang, Runtime model based approach to IoT application development, Front. Comput. Sci. 9 (4) (2015) 540–553.
[178] M. Sneps-Sneppe, D. Namiot, On web-based domain-specific language for internet of things, in: 2015 7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT, 2015, pp. 287–292, http://dx.doi.org/10.1109/ICUMT.2015.7382444.
[179] T. Gomes, P. Lopes, J. Alves, P. Mestre, J. Cabral, J.L. Monteiro, A. Tavares, A modeling domain-specific language for IoT-enabled operating systems, in: IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society, IEEE, 2017, pp. 3945–3950.
[180] D. Cacciagrano, R. Culmone, IRON: REliable domain specific language for programming IoT devices, Internet Things 9 (2020) 100020, URL https://www.sciencedirect.com/science/article/pii/S2542660518300623.
[181] A.J. Salman, M. Al-Jawad, W. Al Tameemi, Domain-specific languages for IoT: Challenges and opportunities, IOP Conf. Ser. Mater. Sci. Eng. (2021) 012133.
[182] R. Neisse, I.N. Fovino, G. Baldini, V. Stavroulaki, P. Vlacheas, R. Giaffreda, A model-based security toolkit for the internet of things, in: 2014 Ninth Int. Conference on Availability, Reliability and Security, IEEE, 2014, pp. 78–87.
[183] X. Yang, Z. Li, Z. Geng, H. Zhang, A multi-layer security model for internet of things, in: Internet of Things, Springer, 2012, pp. 388–393.
[184] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione, R. Spalazzese, Model-driven engineering for mission-critical iot systems, IEEE Softw. 34 (1) (2017) 46–53.
[185] J.P. Dias, J.P. Faria, H.S. Ferreira, A reactive and model-based approach for developing internet-of-things systems, in: 2018 11th Int. Conference on the Quality of Information and Communications Technology, QUATIC, 2018, pp. 276–281.
[186] J.J. Marciniak, Encyclopedia of Software Engineering, Wiley-Interscience, New York, NY, USA, 1994.
[187] M. Erwig, B. Meyer, Heterogeneous visual languages-integrating visual and textual programming, in: Proceedings of Symposium on Visual Languages, 1995, pp. 318–325.
[188] T.F. Masterson, R.M. Meyer, Sivil : a true visual programming language for students, J. Comput. Small Coll. 4 (May 2001) (2001) 74–86.
[189] D. Lau-Kee, A. Billyard, R. Faichney, Y. Kozato, P. Otto, M. Smith, I. Wilkinson, VPL: An active, declarative visual programming system, in: Proceedings 1991 IEEE Workshop on Visual Languages, 1991, pp. 40–46.
[190] S.J. Mellor, M. Balcer, I. Foreword By-Jacoboson, Executable UML: A Foundation for Model-Driven Architectures, Addison-Wesley Longman Publishing Co., Inc. 2002.
[191] M. Boshernitsan, M.S. Downes, Visual Programming Languages: a Survey, Tech. Rep., (December) Computer Science Division, University of California, Berkeley, 2004.
[192] N. Fraser, et al., Blockly: A visual programming editor, 2013, Google and the MIT Media Lab URL https://developers.google.com/blockly/.
[193] ArduBlock, Ardublock: A graphical programming language for arduino, 2019, ArduBlock URL http://ardublock.com/.
[194] W. Xi, E.W. Patton, A blocks-based approach to internet of things in MIT app inventor, in: BLOCKS+ 2018 Workshop, Part of the SPLASH 2018, ACM, 2018.
[195] X. Inc., XOD: A Visual Programming Language for Microcontrollers, XOD Inc. 2019, URL https://xod.io/.
[196] R. Matei, A. Radovici, Remote management system for embedded devices: Wyliodrin, in: 2016 15th RoEduNet Conference: Networking in Education and Research, 2016, pp. 1–5.
[197] Z. B.V.", Zenodys: Internet of things development platform, 2019, Zenodys B.V. URL https://www.zenodys.com/.
[198] T. AB", Noodl reference documentation, 2019, Topplab AB URL http://www.dglogik.com/.
[199] I. "DGLogik, IoT Application platform dglogik, 2019, DGLogik, Inc. URL http://www.dglogik.com/.
[200] A.I. Property", AT&T flow – design and build solutions for the internet of things, 2019, AT&T Intellectual Property URL https://flow.att.com/.
[201] T.J.-J. Li, Y. Li, F. Chen, B.A. Myers, Programming IoT devices by demonstration using mobile apps, in: S. Barbosa, P. Markopoulos, F. Paternò, S. Stumpf, S. Valtolina (Eds.), End-User Development, Springer Int. Publishing, Cham, 2017, pp. 3–17.
[202] "IFTTT", IFTTT Helps your apps and devices work together, 2019, IFTTT URL https://ifttt.com.
[203] F. Corno, L. De Russis, A.M. Roffarello, A high-level approach towards end user development in the IoT, in: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, ACM, 2017, pp. 1546–1552.
[204] I. Bastys, M. Balliu, A. Sabelfeld, If this then what?: Controlling flows in IoT apps, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018, pp. 1102–1119.
[205] B. Inc.", How blynk works, 2019, Blynk Inc. URL http://docs.blynk.cc/.
[206] B. Bohora, S. Maharjan, B.R. Shrestha, IoT Based smart home using blynk framework, Zerone Scholar 1 (1) (2016) 26–30.
[207] L. Yao, Q.Z. Sheng, S. Dustdar, Web-based management of the internet of things, IEEE Internet Comput. 19 (4) (2015) 60–67.
[208] M.B. Younis, G. Frey, Formalization of existing PLC programs: A survey, 2003, CESA 2003, Lille (France), Paper No. S2-R –00–0239.
[209] C. Technologies", Crosser: Low code stream analytics & automation, 2021, Crosser IIoT Crosser Technologies URL https://crosser.io/.
[210] A. Rodrigues, J.P. Silva, J.P. Dias, H.S. Ferreira, Multi-approach debugging of industrial IoT workflows, 2020, arXiv preprint arXiv:2009.05828, URL https://www.criticalmanufacturing.com/mes-for-industry-4-0/connect-iot/.
[211] M. Tomlein, K. Grønbæk, A visual programming approach based on domain ontologies for configuring industrial IoT installations, in: Proceedings of the Seventh International Conference on the Internet of Things, in: IoT '17, Association for Computing Machinery, New York, NY, USA, 2017, http://dx.doi.org/10.1145/3131542.3131552.
[212] M. Blackstock, R. Lea, IoT Mashups with the WoTKit, in: 2012 3rd IEEE Int. Conference on the Internet of Things, 2012, pp. 159–166.
[213] J. Huang, M. Cakmak, Supporting mental model accuracy in trigger-action programming, in: UbiComp 2015 - Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, 2015, pp. 215–225.
[214] G. Ghiani, M. Manca, F. Paternò, C. Santoro, Personalization of context-dependent applications through trigger-action rules, ACM Trans. Comput. Hum. Interact. 24 (2) (2017) 1–33, URL https://dl.acm.org/doi/10.1145/3057861.
[215] A. Rümpel, K. Meißner, Requirements-driven quality modeling and evaluation in web mashups, in: 2012 Eighth Int. Conference on the Quality of Information and Communications Technology, IEEE, 2012, pp. 319–322.
[216] T. Loton, Introduction to Microsoft Popfly, No Programming Required, Lotontech Limited, 2008.
[217] M. Pruett, Yahoo! Pipes, O'Reilly, 2007.
[218] E.M. Maximilien, H. Wilkinson, N. Desai, S. Tai, A domain-specific language for web apis and services mashups, in: Int. Conference on Service-Oriented Computing, Springer, 2007, pp. 13–26.

[219] C. Prehofer, L. Chiarabini, From internet of things mashups to model-based development, in: Proceedings - Int. Computer Software and Applications Conference, Vol. 3, 2015, pp. 499–504.

[220] T. Szydlo, R. Brzoza-Woch, J. Sendorek, M. Windak, C. Gniady, Flow-based programming for IoT leveraging fog computing, in: 2017 IEEE 26th Int. Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, 2017, pp. 74–79.

[221] J. Foundation, Node-RED, flow-based programming for the internet of things, 2019, JS Foundation URL https://nodered.org/.

[222] M. Blackstock, R. Lea, FRED: A Hosted data flow platform for the IoT, in: Proceedings of the 1st Int. Workshop on Mashups of Things and APIs, in: MOTA '16, ACM, New York, NY, USA, 2016, pp. 2:1–2:5.

[223] N.K. Giang, M. Blackstock, R. Lea, V.C.M. Leung, Developing IoT applications in the fog: A distributed dataflow approach, in: 2015 5th Int. Conference on the Internet of Things, IOT, 2015, pp. 155–162.

[224] J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, M. Riveill, WComp middleware for ubiquitous computing: Aspects and composite event-based web services, Ann. Telecommun. Ann. Télécommunications 64 (3) (2009) 197–214.

[225] N. Ferry, V. Hourdin, S. Lavirotte, G. Rey, M. Riveill, J.-Y. Tigli, Wcomp, middleware for ubiquitous computing and system focused adaptation, 2012.

[226] M. Blackstock, R. Lea, WoTKit: A Lightweight toolkit for the web of things, in: Proceedings of the Third Int. Workshop on the Web of Things, in: WOT '12, ACM, New York, NY, USA, 2012, pp. 3:1–3:6.

[227] R. Kleinfeld, S. Steglich, L. Radziwonowicz, C. Doukas, Glue.things: A mashup platform for wiring the internet of things with the internet of services, in: Proceedings of the 5th Int. Workshop on Web of Things, in: WoT '14, ACM, New York, NY, USA, 2014, pp. 16–21.

[228] S. Heo, S. Woo, J. Im, D. Kim, IoT-MAP: IoT mashup application platform for the flexible IoT ecosystem, in: 2015 5th Int. Conference on the Internet of Things, IOT, 2015, pp. 163–170.

[229] M. Silva, J.P. Dias, A. Restivo, H.S. Ferreira, A review on visual programming for distributed computation in IoT, in: Proceedings of the 21st International Conference on Computational Science, ICCS, Springer, 2021.

[230] M. Blackstock, R. Lea, Toward a distributed data flow platform for the web of things (distributed node-RED), in: Proceedings of the 5th Int. Workshop on Web of Things - WoT '14, 2014, pp. 34–39.

[231] J. Noor, H.Y. Tseng, L. Garcia, M. Srivastava, DDFlow: Visualized declarative programming for heterogeneous IoT networks, in: IoTDI 2019 - Proceedings of the 2019 Internet of Things Design and Implementation, in: IoTDI '19, ACM, New York, NY, USA, 2019, pp. 172–177, URL http://doi.acm.org/10.1145/3302505.3310079.

[232] N.K. Giang, M. Blackstock, R. Lea, V.C. Leung, Developing iot applications in the fog: A distributed dataflow approach, in: 2015 5th Int. Conference on the Internet of Things, IOT, IEEE, 2015, pp. 155–162.

[233] N.K. Giang, R. Lea, V.C.M. Leung, Exogenous coordination for building fog-based cyber physical social computing and networking systems, IEEE Access 6 (2018) 31740–31749.

[234] J. Sendorek, T. Szydlo, M. Windak, R. Brzoza-Woch, FogFlow - Computation organization for heterogeneous fog computing environments, in: Computational Science – ICCS 2019, Springer International Publishing, Cham, 2019, pp. 634–647.

[235] M.I. NAAS, L. Lemarchand, J. Boukhobza, P. Raipin, A graph partitioning-based heuristic for runtime IoT data placement strategies in a fog infrastructure, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, in: SAC '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 767–774, http://dx.doi.org/10.1145/3167132.3167217.

[236] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Softw. - Pract. Exp. 47 (9) (2017) 1275–1296.

[237] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, FogFlow: EAsy programming of IoT services over cloud and edges for smart cities, IEEE Internet Things J. (2017) 1.

[238] B. Cheng, E. Kovacs, A. Kitazawa, K. Terasawa, T. Hada, M. Takeuchi, FogFlow: ORchestrating IoT services over cloud and edges, NEC Tech. J. 13 (2018) 48–53.

[239] M. Silva, J.P. Dias, A. Restivo, H.S. Ferreira, Visually-defined real-time orchestration of IoT systems, in: Proceedings of the 17th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, in: MOBIQUITOUS 2020, Association for Computing Machinery, New York, NY, USA, 2020.

[240] A. Glikson, S. Nastic, S. Dustdar, Deviceless edge computing: extending serverless computing to the edge of the network, in: Proceedings of the 10th ACM International Systems and Storage Conference, 2017, p. 1.

[241] S. Nastic, S. Dustdar, Towards deviceless edge computing: Challenges, design aspects, and models for serverless paradigm at the edge, in: The Essence of Software Engineering, Springer, Cham, 2018, pp. 121–136.

[242] M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Dustdar, O. Scekic, T. Rausch, S. Nastic, S. Ristov, T. Fahringer, A deviceless edge computing approach for streaming IoT applications, IEEE Internet Comput. 23 (1) (2019) 37–45.

[243] D. Pinto, J.P. Dias, H. Sereno Ferreira, Dynamic allocation of serverless functions in IoT environments, in: 2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing, EUC, 2018, pp. 1–8, http://dx.doi.org/10.1109/EUC.2018.00008.

[244] X. Mi, F. Qian, Y. Zhang, X. Wang, An empirical characterization of IFTTT: ecosystem, usage, and performance, in: Proceedings of the 2017 Internet Measurement Conference, 2017, pp. 398–404.

[245] J. Austerjost, M. Porr, N. Riedel, D. Geier, T. Becker, T. Scheper, D. Marquard, P. Lindner, S. Beutel, Introducing a virtual assistant to the lab: A voice user interface for the intuitive control of laboratory instruments, SLAS Technol. Transl. Life Sci. Innov. 23 (5) (2018) 476–482.

[246] D. Priest, T. Dyson, T. Martin, Every alexa command to give your amazon echo smart speaker or display, 2019, Online URL https://www.cnet.com/how-to/every-alexa-command-to-give-your-amazon-echo-smart-speaker-or-display/. (Accessed January 2020).

[247] T. Ammari, J. Kaye, J.Y. Tsai, F. Bentley, Music, search, and IoT: How people (really) use voice assistants, ACM Trans. Comput. Hum. Interact. 26 (3) (2019).

[248] M. Mitrevski, Conversational interface challenges, in: Developing Conversational Interfaces for IOS, Springer, 2018, pp. 217–228.

[249] G. López, L. Quesada, L.A. Guerrero, Alexa vs. Siri vs. Cortana vs. Google assistant: A comparison of speech-based natural user interfaces, in: I.L. Nunes (Ed.), Advances in Human Factors and Systems Interaction, Springer International Publishing, Cham, 2018, pp. 241–250.

[250] J.P. Dias, A. Lago, H.S. Ferreira, Conversational interface for managing non-trivial internet-of-things systems, in: Proceedings of the 20th International Conference on Computational Science, ICCS, Springer, 2020, pp. 27–36.

[251] A.S. Lago, J.P. Dias, H.S. Ferreira, Managing non-trivial internet-of-things systems with conversational assistants: A prototype and a feasibility experiment, J. Comput. Sci. 51 (2021) 101324, URL https://www.sciencedirect.com/science/article/pii/S1877750321000223.

[252] R. Kishore Kodali, S.C. Rajanarayanan, L. Boppana, S. Sharma, A. Kumar, Low cost smart home automation system using smart phone, in: 2019 IEEE R10 Humanitarian Technology Conference (R10-HTC)(47129), 2019, pp. 120–125.

[253] D. Braines, N. O'Leary, A. Thomas, D. Harborne, A.D. Preece, W.M. Webberley, Conversational homes: a uniform natural language approach for collaboration among humans and devices, Int. J. Adv. Intell. Syst. 10 (3/4) (2017) 223–237.

[254] A. Rajalakshmi, H. Shahnasser, Internet of things using node-red and alexa, in: 2017 17th International Symposium on Communications and Information Technologies, ISCIT, 2017, pp. 1–4.

[255] R. Kang, A. Guo, G. Laput, Y. Li, X.A. Chen, Minuet: Multimodal interaction with an internet of things, in: Symposium on Spatial User Interaction, in: SUI '19, Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3357251.3357581.

[256] D.S.K. Bheesetti, V.N. Bhogadi, S.K. Kintali, M. Zia Ur Rahman, A complete home automation strategy using internet of things, in: A. Kumar, S. Mozar (Eds.), ICCCE 2020, Springer Singapore, Singapore, 2021, pp. 363–373.
[257] T. Kim, Short research on voice control system based on artificial intelligence assistant, in: 2020 International Conference on Electronics, Information, and Communication, ICEIC, 2020, pp. 1–2, http://dx.doi.org/10.1109/ICEIC49074.2020.9051160.
[258] M. Karthikeyan, T.S. Subashini, M.S. Prashanth, Implementation of home automation using voice commands, in: K.S. Raju, R. Senkerik, S.P. Lanka, V. Rajagopal (Eds.), Data Engineering and Communication Technology, Springer Singapore, Singapore, 2020, pp. 155–162.
[259] A. Nordrum, The internet of fewer things [news], IEEE Spectr. 53 (10) (2016) 12–13.
[260] D. Guinard, The IoT needs a defrag, 2016, O'Reilly Media, Online URL https://www.oreilly.com/ideas/the-iot-needs-a-defrag. (Accessed 10 Mar 2018).
[261] Y. Duan, G. Fu, N. Zhou, X. Sun, N.C. Narendra, B. Hu, Everything as a service (xaas) on the cloud: Origins, current and future trends, in: 2015 IEEE 8th Int. Conference on Cloud Computing, 2015, pp. 621–628.
[262] Z. Maamar, T. Baker, N. Faci, E. Ugljanin, M.A. Khafajiy, V. Burégio, Towards a seamless coordination of cloud and fog: illustration through the internet-of-things, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, ACM, 2019, pp. 2008–2015.
[263] M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan, Local-first software: You own your data, in spite of the cloud, in: Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, in: Onward! 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 154–178.
[264] E. Shaikh, I. Mohiuddin, A. Manzoor, Internet of things (IoT): Security and privacy threats, in: 2019 2nd International Conference on Computer Applications Information Security, ICCAIS, 2019, pp. 1–6.
[265] R. Bawa, R. Clarck, Software-Defined Everything, Tech. Rep., Deloitte Consulting LLP, 2015.
[266] N. Garg, R. Garg, Energy harvesting in IoT devices: A survey, in: 2017 International Conference on Intelligent Sustainable Systems, ICISS, 2017, pp. 127–131.
[267] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Comput. Netw. 38 (4) (2002) 393–422.
[268] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.
[269] S. Bera, S. Misra, A.V. Vasilakos, Software-defined networking for internet of things: A survey, IEEE Internet Things J. 4 (6) (2017) 1994–2008.
[270] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, N. Venkatasubramanian, A software defined networking architecture for the internet-of-things, in: 2014 IEEE Network Operations and Management Symposium, NOMS, IEEE, 2014, pp. 1–9.
[271] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, K. Geihs, A resource oriented integration architecture for the internet of things: A business process perspective, Pervasive Mob. Comput. 20 (2015) 145–159.
[272] M. Mohammadi Zanjireh, H. Larijani, A survey on centralised and distributed clustering routing algorithms for WSNs, in: IEEE Vehicular Technology Conference, Vol. 2015, 2015, pp. 1–6.
[273] L.D. Xu, W. He, S. Li, Internet of things in industries: A survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243.
[274] I. Yaqoob, E. Ahmed, I.A.T. Hashem, A.I.A. Ahmed, A. Gani, M. Imran, M. Guizani, Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges, IEEE Wirel. Commun. 24 (3) (2017) 10–16.
[275] C. Savaglio, G. Fortino, Autonomic and cognitive architectures for the internet of things, in: G. Di Fatta, G. Fortino, W. Li, M. Pathan, F. Stahl, A. Guerrieri (Eds.), Internet and Distributed Computing Systems, Springer Int. Publishing, Cham, 2015, pp. 39–47.
[276] R. Angarita, Responsible objects: Towards self-healing internet of things applications, in: Proceedings - IEEE Int. Conference on Autonomic Computing, ICAC 2015, IEEE, 2015, pp. 307–312.
[277] J.P. Dias, B. Lima, J.P. Faria, A. Restivo, H.S. Ferreira, Visual self-healing modelling for reliable internet-of-things systems, in: Proceedings of the 20th International Conference on Computational Science, ICCS, Springer, 2020, pp. 27–36.
[278] J.P. Dias, A. Restivo, H.S. Ferreira, Empowering visual internet-of-things mashups with self-healing capabilities, in: 2021 IEEE/ACM 3rd International Workshop on Software Engineering Research Practices for the Internet of Things, SERP4IoT, 2021.
[279] R. Murch, Autonomic Computing, IBM Press, 2004.
[280] M. Presser, P.M. Barnaghi, M. Eurich, C. Villalonga, The SENSEI project: Integrating the physical world with the digital world of the network of the future, IEEE Commun. Mag. 47 (4) (2009) 1–4.
[281] S.A. Al-Qaseemi, H.A. Almulhim, M.F. Almulhim, S.R. Chaudhry, IoT Architecture challenges and issues: Lack of standardization, in: 2016 Future Technologies Conference, FTC, 2016, pp. 731–738.
[282] R.P. Gabriel, Patterns of Software, Vol. 62, Oxford University Press, 1996.
[283] J.P. Dias, F. Couto, A.C.R. Paiva, H.S. Ferreira, A brief overview of existing tools for testing the internet-of-things, in: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, 2018, pp. 104–109, http://dx.doi.org/10.1109/ICSTW.2018.00035.
[284] N. Mäkitalo, T. Mikkonen, C. Pautasso, V. Bankowski, P. Daubaris, R. Mikkola, O. Beletski, WebAssembly Modules as lightweight containers for liquid IoT applications, in: International Conference on Web Engineering, Springer, 2021, pp. 328–336.
[285] M. Bures, M. Klima, V. Rechtberger, X. Bellekens, C. Tachtatzis, R. Atkinson, B.S. Ahmed, Interoperability and integration testing methods for IoT systems: A systematic mapping study, in: International Conference on Software Engineering and Formal Methods, Springer, 2020, pp. 93–112.
[286] R.M. Andrade, B.R. Aragão, P.A.M. Oliveira, M.E. Maia, W. Viana, T.P. Nogueira, Multifaceted infrastructure for self-adaptive IoT systems, Inf. Softw. Technol. 132 (December 2020) (2021) 106505.
[287] R. Grant, K. Rockot, O. Ruiz-Henríquez, WebUSB API, 2022, Web Platform Incubator Community Group URL https://wicg.github.io/webusb/.
[288] B. Morin, N. Harrand, F. Fleurey, Model-based software engineering to tame the IoT jungle, IEEE Softw. 34 (1) (2017) 30–36.
[289] J. Spolsky, The law of leaky abstractions, in: Joel on Software, Springer, 2004, pp. 197–202.
[290] N. O'Leary, What's next with node-RED? 2020, OpenJS Foundation URL https://nodered.org/blog/2020/10/13/future-plans. (Last Access 2021).
[291] D. Clerissi, M. Leotta, G. Reggio, F. Ricca, Towards an approach for developing and testing node-RED IoT systems, in: Proceedings of the 1st ACM SIGSOFT International Workshop on Ensemble-Based Software Engineering, in: EnSEmble 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–8.
[292] D. Torres, J.P. Dias, A. Restivo, H.S. Ferreira, Real-time feedback in node-RED for IoT development: An empirical study, in: 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications, DS-RT, 2020, pp. 1–8.
[293] M. Petre, Mental imagery, visualisation tools and team work, in: Proceedings of the Second Program Visualization Workshop, 2002, pp. 3–14, URL http://oro.open.ac.uk/4090/.
[294] F. Fleurey, B. Morin, ThingML: A generative approach to engineer heterogeneous and distributed systems, in: 2017 IEEE Int. Conference on Software Architecture Workshops, ICSAW, 2017, pp. 185–188.
[295] S. Sendall, W. Kozaczynski, Model transformation: the heart and soul of model-driven software development, IEEE Softw. 20 (5) (2003) 42–45.
[296] A. Aguiar, A. Restivo, F.F.C. Correia, H.S. Ferreira, J.P. Dias, Live software development — Tightening the feedback loops, in: Proceedings of the 5th Edition of the Programming Experience Workshop, 2019, pp. 22:1–22:6.
[297] B.R. Barricelli, S. Valtolina, Designing for end-user development in the internet of things, in: End-User Development: 5th Int. Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, Springer Int. Publishing, Cham, 2015, pp. 9–24.
[298] W. He, J. Martinez, R. Padhi, L. Zhang, B. Ur, When smart devices are stupid: Negative experiences using home smart devices, in: 2019 IEEE Security and Privacy Workshops, SPW, 2019, pp. 150–155.

[299] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: 2016 23rd Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2016, pp. 153–160.

[300] B. Chander, G. Kumaravelan, Internet of things: Foundation, in: Principles of Internet of Things (IoT) Ecosystem: Insight Paradigm, Springer Int. Publishing, Cham, 2020, pp. 3–33.