A Survey on IoT Programming Platforms: A Business-Domain Experts Perspective

Fatma-Zohra Hannou, Maxime Lefrançois, Pierre Jouvelot, Victor Charpenay, Antoine Zimmermann

A Survey on IoT Programming Platforms: A Business-Domain Experts Perspective

FATMA-ZOHRA HANNOU*, École des Mines Saint-Étienne, France
MAXIME LEFRANÇOIS, École des Mines Saint-Étienne, France
PIERRE JOUVELOT, Mines Paris, PSL University, France, France
VICTOR CHARPENAY, École des Mines Saint-Étienne, France
ANTOINE ZIMMERMANN, École des Mines Saint-Étienne, France

The vast growth and digitalization potential offered by the Internet of Things (IoT) is hindered by substantial barriers in accessibility, interoperability, and complexity, mainly affecting small organizations and non-technical entities. This survey paper provides a detailed overview of the landscape of IoT programming platforms, focusing specifically on the development support they offer for varying end-user profiles, ranging from technical developers with IoT expertise to business experts willing to take advantage of IoT solutions to automate their organization processes. To this aim, the survey examines a range of IoT platforms, classified according to their programming approach between general-purpose programming solutions, model-driven programming, mashups and end-user programming. Necessary IoT and programming backgrounds are described to empower non-technical readers with a comprehensive field summary. In addition to offering a study of the IoT programming solution landscape, the paper introduces a comprehensive table comparing the features of the most representative platforms and a discussion section identifying valuable decision insights and guidelines supporting end-users in selecting appropriate IoT platforms for their use cases. This work contributes to narrowing the knowledge gap between IoT specialists and end users, breaking accessibility barriers and further promoting the integration of IoT technologies in various domains.

Additional Key Words and Phrases: Internet of Things, IoT programming, IoT platforms, Smart Building, Smart Agriculture

ACM Reference Format:
Fatma-Zohra HANNOU, Maxime Lefrançois, Pierre Jouvelot, Victor Charpenay, and Antoine Zimmermann. 2023. A Survey on IoT Programming Platforms: A Business-Domain Experts Perspective. In . ACM, New York, NY, USA, 49 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The emergence of the Internet of Things has opened up a broader, deeper, and more realistic perception of the surrounding environment by transforming any "thing" into a quasi-continuously available data source or control lever. It has demonstrated an unlimited potential for boosting the digitalization of many aspects of everyone's daily life [76]. It also supports companies in optimizing process automation, increasing operational efficiency, and optimizing costs. The IoT market has witnessed tremendous growth (from 300 billion USD in 2021 to an expected 600 billion USD in 2026 [80]), offering many new device technologies, tools, architectures, and projects, either generic or tailored to specific use cases. While this diversity accelerates the adoption of IoT within multiple domains, such as building automation, healthcare, agriculture, or energy management, it raises interoperability and access challenges. The subsequent complexity might

Manuscript submitted to ACM

represent a significant barrier to the immediate IoT technologies use for small organizations (or non-technical companies) that cannot afford the cost of hiring IoT expert teams to handle complex architectures and deployment processes. IoT platforms combine hardware and software technologies to enable the building and deployment of IoT applications [4] through a common user interface, easing access and interoperability within the IoT ecosystem. It provides services and facilities to develop IoT solutions, including device integration, data storage and processing, user communication and development tools. Yet, the promise of easy-to-deploy, easy-to-configure IoT systems has not been entirely fulfilled. In most cases, IoT systems are developed on generic platforms offering predefined routines with limited customization options. Domain-oriented tools are often focused on fixed architectures, device ranges, and features that do not necessarily match the organization's needs, especially those covering overlapping vertical domains (e.g., building management and healthcare, agriculture and industry), requiring the combined automation of both domains' processes.

Somewhere between the average end user and the IoT specialist, the so-called "vertical domain expert" increasingly expresses the will to obtain the most out of the IoT technologies by going beyond the 'configuration routine" mode. To meet these expectations, a new generation of IoT platforms has widened the spectrum of their user profiles by creating simplified user interfaces (UIs) with more expressive power. These UIs, often rely on *domain-specific languages* [36] (DSLs), with the aim to reach larger end-user communities, including those with no or little technical background.

As an end-user, understanding the extent of what an IoT platform offers, its characteristics, and whether it makes a suitable choice for the use case at hand is a challenging matter. Committing to a specific IoT deployment requires a positive cost-benefit balance and compliance with additional requirements: interoperability, domain tasks, level of abstraction (following user skills), maintenance cost, or documentation availability. These factors, among others, are of variable importance depending on the application use cases. For example, the user/developer should consider interoperability guarantees when dealing with heterogeneous devices, protocols or sub-domains. In other cases, as within healthcare or building management fields, sensitive data and processes have to be protected against disclosure and unauthorized access risks, making security and privacy guarantees of utmost importance. Other requirements include reliability, latency, autonomy, and cost.

In this survey paper, we consider these key factors to describe the landscape of IoT programming platforms and exhibit some of their strengths and weaknesses. However, given the vast spectrum of available technologies, this survey focuses solely on the development side of IoT applications. Accordingly, particular attention is paid to the support the existing platforms offer to users aiming to develop applications. This technology segment is explored based on the most relevant parameters, such as the abstraction level of IoT-specific programming languages, the expressiveness or richness of the associated toolboxes, etc. Various programming languages are accessible through IoT platforms, from the lowest abstraction level (assembly) to high-level DSLs dedicated to vertical domains, with adapted representations and vocabularies.

Contributions and Audience. This survey aims at providing vertical domain experts striving to develop IoT applications with programming-related resources to understand the existing IoT landscape. IoT platforms are presented following different development approaches that match more or less target users' profiles. It is intended to be used as a resource by this particular developer community for making an informed decision regarding which platform presents the best characteristics for their expected application. As such, the contribution of this work is twofold. First, it provides background on the IoT field with a particular focus on programming languages. The most relevant characteristics are defined and serve as a basis for assessing platforms. Second, it produces an analysis of the IoT landscape driven by end-user/developer requirements, examining platforms' families and characteristics.

Scope. Note that several related topics fall outside the scope of this survey, such as IoT architectures, interoperability projects, software architectures, security, or DSL implementations. Some of these aspects are mentioned in the high-level description of the IoT platforms, and therefore introduced in the survey background to support reader understanding decision insights completing programming features. However, no systematic study covering these matters is conducted here. For more details about these topics, we refer the reader to the following works: IoT architectures [136], research toward interoperability [122, 134], IoT security and privacy challenges [91, 169, 172], and DSLs development [49].

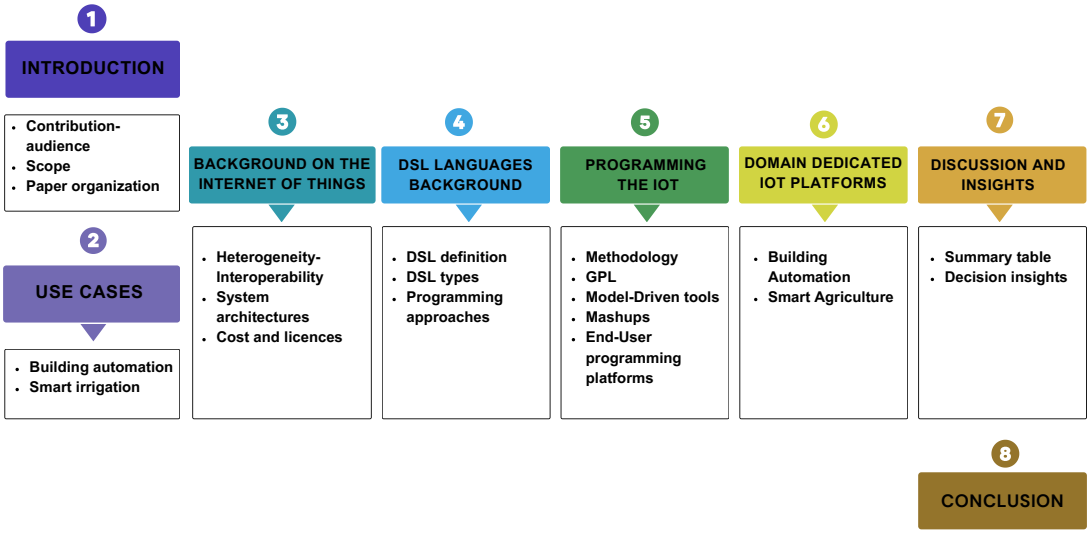


Fig. 1. Roadmap of the paper

Organization. The remainder of this paper is organized as follows.

- Section 2 introduces two running examples used along the paper to illustrate representative usage scenarios of IoT platforms. The first scenario deals with a building automation application for indoor air-quality monitoring, and the second considers precision agriculture through the case of a smart irrigation process.
- Section 3 provides an overview of the IoT field to introduce some key platform characteristics such as interoperability, software architecture, licences, or business domain.
- Section 4 provides background information about programming languages with a focus on domain-specific languages highly used within the IoT community.
- Section 5 presents an insight into generic IoT platforms grouped by programming approaches. It showcases how the technical characteristics of each tool can be exploited in an automation scenario and to what IoT task it contributes. A domain expert will thus be able to relate the proposed development support with the corresponding technical indicators.
- Section 6 covers specific agriculture and building automation IoT solutions. Both sections start by highlighting selection criteria that have been used to choose relevant platforms.
- Section 7 summarizes the outcomes through a comparative table highlighting discussed platform's properties and decision making insights.

A roadmap illustration of the survey paper organization is provided in Figure 1.

2 USE CASES

For this survey, we introduce two representative use cases to provide concrete examples of possible use of IoT programming technologies. The first scenario deals with a building automation application for indoor air-quality management, and the second considers precision agriculture through a smart irrigation task. The choice of these domains is first motivated by their respective large communities among IoT technologies consumers. Also, building and agriculture automation scenarios often rely on similar indicators/input data such as temperature, CO_2, and humidity but with contrasting semantics, purposes, and deployment architectures (indoor/outdoor), displaying an interesting spectrum of variations allowing to cover a broad scope of use cases.

2.1 Building Automation Use Case

Building-automation scenarios often aim at fulfilling energy-optimization strategies while guaranteeing users' health and comfort. Modern building management increasingly relies on sensor networks that collect data about environmental indicators, energy consumption, or room occupancy. Various actuators are deployed within the building for automatic window and door opening or triggering heating, ventilation, or lighting.

In the first use case, illustrated in Figure 2, sensors continuously monitor the indoor temperature and CO_2 levels to determine the air quality in a classroom. These data, enriched with room occupancy information (course schedule), can be used to determine the air quality and user preference for optimal temperature. If the temperature is too low and the CO_2 level are normal, heating can be triggered before or during users' attendance, depending on the room heating capacity. If the observed CO_2 level increases significantly, a window opening command is sent to the window actuator in order to ventilate the room.

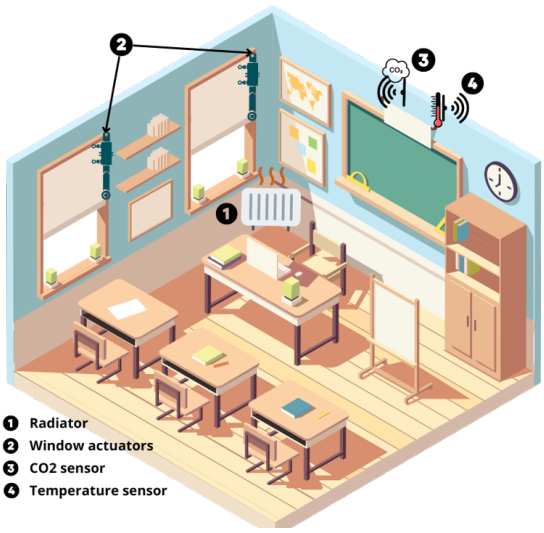


Fig. 2. Indoor air quality monitoring through automatic window opening

This IoT-based automation process performs air quality supervision while preserving resources. First, heating is only activated when actual or imminent presence is expected. Otherwise, automatic window closing can help maintain a suitable temperature within the room. Also, the ventilation of the room is naturally guaranteed through the outdoor air, under appropriate conditions, without requiring air conditioning.

2.2 Smart Irrigation Use Case

A farmer might want to use precision agriculture techniques to automate and optimize different crop-related processes. In this use case, the farmer deploys a set of sensors and actuators in the farm, covering the outdoor plots and an indoor greenhouse. Figure 5 illustrates a possible installation scenario. The objective is to collect timely data regarding climate,

soil and crop characteristics. Sensing data consist in (1) current indoor temperature, humidity, and soil moisture within the greenhouse, collected by sensors 1, 2, and 3 (in the figure), respectively, and (2) soil moisture of outdoor plots, current and forecast temperature and humidity, provided by the local weather station and outdoor sensors.

Data are assumed to be stored in a central system, enriched with the crop-development state, and inference (symbolic or statistical) mechanisms are applied to support the decision-making process. The goal is to properly supervise the irrigation process by automatically triggering or stopping crop watering. The decision-making process is twofold: when to start/stop the watering and what amount of water to deliver via water valves to ensure optimal crop development. The optimal strategy also considers budget, water-availability planning and productivity constraints. This system would provide multiple benefits: irrigate only when necessary; adapt the irrigation plan to exceptional situations such as hot days leading to abnormally low moisture levels; preserve resources and the environment from systematic watering.

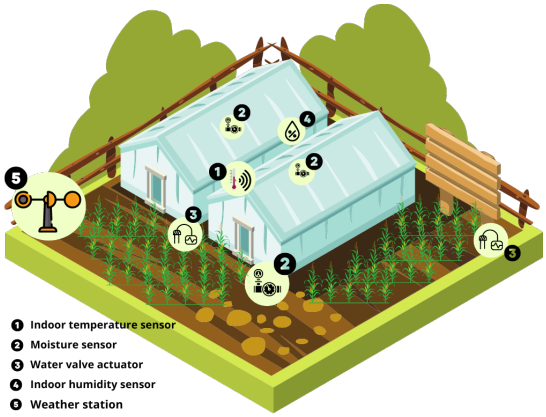


Fig. 3. Smart irrigation scenario through environment indicators processing

Of course, multiple challenges need to be considered when deploying such systems: the outdoor configuration on large plots, which exposes electronic equipment to possibly intense environmental phenomena, high cost of battery maintenance, etc. Choosing appropriate tools that enable the development and deployment of efficient IoT applications while respecting such constraints is necessary to make applications reliable and limit the maintenance effort and cost.

3 BACKGROUND ON THE INTERNET OF THINGS

There was a time when Internet access was restricted to humans via computers. In recent years, the Internet of Things has revolutionized this principle with a basic but strong assumption: all objects that physically exist can and, in many cases, should have a digital identity and gain access to the Internet, generating and exchanging various data regarding their state or their immediate environment.

The first usage of the term IoT can be traced back to 1999 when Kevin Ashton [7] associates the Internet of Things to networks of Radio-Frequency Identification (RFID) chips. The democratization of access to low-cost technologies and the adaptation of connectivity standards to constrained devices has enlarged the extent of accessible ojects in the IoT. The concept of the IoT gained complete sense in 2008 when the number of connected objects exceeded the number of humans accessing the Internet (Cisco Internet Business Solutions [42]).

There is a plethora of definitions for the IoT, with sometimes special focus on particular deployment cases or architectural considerations. The International Telecommunication Union[1] (ITU) gives the following definition [43]:

> *A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.*

[1] https://www.itu.int/fr/

The availability of Cloud-based application services [8] has further encouraged its use in an ever-growing number of applications at variable scales: industry [175], smart buildings [87], healthcare [81], agronomy [135], and more [147].

Objects in the IoT context can be any physical entity that can connect directly to the Internet, or indirectly (through sensors or wearable devices on humans or crops). They can be distinguished according to their computational capacities, from computers or boards with high capacities to very small constrained chips. The need to integrate "smartness" (i.e., somewhat advanced local data manipulation capabilities) and sensing possibilities into spatially restricted environments has required inventing small, inexpensive devices for easy and suitable deployment in indoor systems, for example. The underlying constraints are diverse: run on internal batteries and limited computing and internal memory, to name but a few.

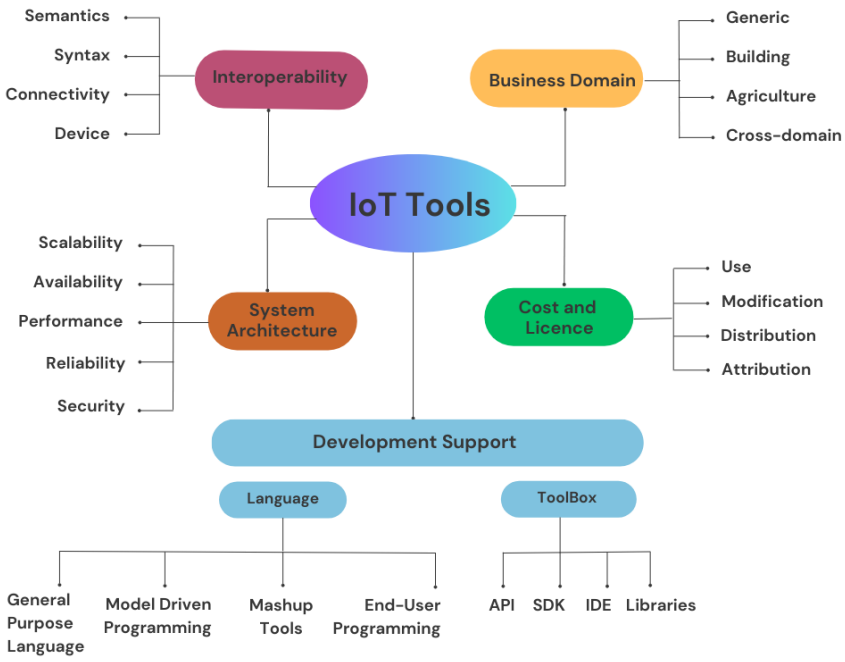


Fig. 4. Overview of IoT tools main characteristics

Figure 4 illustrates the main characteristics used in this survey to describe IoT platforms, grouped into five topics: interoperability, software architecture, licence and cost, business domain, and development support. This section provides insights on the first four dimensions, and how tools are evaluated according to their subsequent criteria. Given that the primary focus of this survey is the programming of IoT applications, the development support dimension is detailed in a dedicated section (Section 4).

3.1 Heterogeneity-Interoperability

Adapting to the high pace of technological advances, IoT tools and platforms constantly expand support to new devices, protocols, and services, increasing the heterogeneity of IoT systems and raising new interoperability challenges.

This section reviews common heterogeneity sources such as device, connectivity, syntax, and semantics (illustrated in Figure 5), and provides an overview of related interoperability issues. It is worth mentioning that IoT heterogeneity is not limited to these aspects but goes beyond. Indeed, the blooming market of platforms boosts cross-platform and cross-domain heterogeneity.

3.1.1 Device. The diversity of IoT technology providers led to a mixture of billion devices with heterogeneous hardware specifications coexisting within IoT systems. Within an IoT network architecture, nodes correspond to devices with computing capabilities. They are generally categorized following their resources, mainly RAM, storage, CPU, supported communication protocols, energy supply, transmission power, and the architectural layer they belong to. These characteristics are generally determined by the tier in which they are deployed (Cloud, Fog, Edge) [136]. Related taxonomies prosper and sometimes differ (see, e.g., [142, 149]), but a frequently used classification method builds upon three levels: low or constrained, middle, or powerful nodes.

- *Powerful nodes*, such as Cloud infrastructure providing remote services. They have large computing, memory, and storage capacities and support Internet and Web communication protocols. They rely upon complex operating systems with extended functionalities, with a predominance for Linux server distributions [53].
- *Middle nodes*, commonly identified as gateways or local servers of fog architectures. They include single-board computers such as Raspberry Pi [162], Onion Omega Board, or Intel Gallileo (refer to [52] for a good overview). They provide sufficient computing and communication capabilities to join end devices to Cloud platforms while executing local data transformation programs.
- *Constrained nodes*, or end devices. They are the closest to the physical environment, on which they generally collect data and act. They integrate microcontroller units (MCUs) for storing and processing raw data. Edge MCU examples include ESP32, STM32 or Arduino Nano 33 BLE Sense [93]. They frequently communicate with low-power wireless protocols (lowPAN) like BLE and Zigbee, rely on internal batteries with low data-exchange rates, and use dedicated embedded OSs, with reduced services and efficient task processing [65, 142]. Some devices even operate without any OS (bare metal). Constrained nodes concentrate research efforts for the best efficiency footprint compromise to support edge-computing appliances.

3.1.2 Connectivity. One early challenge the IoT field faced was the strong requirement for continuous object connectivity. Building reliable smart applications relies on the quasi-timely availability of sensing and actuating capabilities. The device heterogeneity and the need for reliable data exchange channels require deployment strategies to leverage hybrid networking and messaging protocols.

At the infrastructure (network) level, an extensive set of communication networks are inherited from the Wireless Sensor Network field [177], providing a wide spectrum of constrained objects with their inherent power specifications and connectivity range. Power requirements are an intrinsic characteristics of these devices, often linked to specific supervision needs to avoid high maintenance costs. While some devices support multiple protocols, the connectivity range is defined according to the deployment and application scenario: indoor or outdoor installation, device mobility, available gateways, etc. The connectivity protocols at the lower layers (Physical-Data Link in the OSI model [150]) are commonly identified by their operating range [108]:

- Contact area: RFID and NFC,
- Wireless Personal Area Network (WPAN) protocols as ZigBEE and Bluetooth,
- Wireless Local Area Network (WLAN) as WiFi,

- Wireless Wide Area Network (WWAN), including cellular and Low Power Wide Area (LPWAN) networks, up to 100 km as LoRaWAN.

At the top (application) layer of the devices communications, the most popular protocols are [33] HyperText Transfer Protocol (HTTP), Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (COAP).

3.1.3 Syntax. Syntactic heterogeneity denotes the variability of formats and data structures used to exchange information between IoT system components (nodes and services).

As popular data formats, IoT systems rely upon the representations most frequently used on the Web: plain text, comma-separated values (CSV), JavaScript Object Notation (JSON), Extensible Markup Language (XML), or Resource Description Framework (RDF) (different serializations). The RDF formats and schemas may turn out to be unsuitable for the most constrained devices that generate and process increasing amounts of data. More lightweight formats with binary representations have been defined, such as Efficient XML Interchange format (EXI) by the W3C,[2] Constrained Binary Object Representation (CBOR) of the Internet Engineering Task Force (IETF)[3], or CBOR Linked Data (CBOR-LD) and Header, Dictionary, Triples (HDT) as RDF binary serializations for linked data. These formats intend to standardize data exchange, but they do not apply everywhere [122]. Syntactic heterogeneity implies additional interoperability efforts and transformation costs in middleware and high-level platforms to avoid data integrity loss. When two devices (or a device and service) use two different data schemas for the data generated at the producer's node level, the risk is to alter the data due to incorrect extraction and reading in other devices. The W3C Web of Things intends to unify these descriptions through a single data model and format, with however only modest success so far [122].

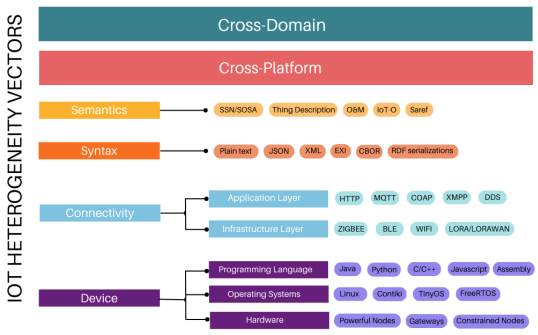


Fig. 5. Overview of IoT systems heterogeneity dimensions

3.1.4 Semantics. Semantic heterogeneity relates to data understanding. IoT systems communicate data between their different nodes to achieve data-centric services. When two entities exchange data, the meaning of the raw data is not necessarily *understood* the right way by the receiver. The message integrity is threatened, leading to unreliably processed outputs. Metadata can be used to describe data content and its production context to facilitate the interpretation and use of the data. Still, node software might use distinct schemas to generate the metadata, increasing interpretation mismatches. One way to homogenize the knowledge representation in a field of study are ontologies. An ontology is defined according to [153] as:

> "a formal, explicit specification of a shared conceptualisation."

This definitions underlines that ontologies provide machine-readable (explicit) abstraction of real-world phenomenon concepts, their relationships and constraints.

[2]World Wide Web Consortium, https://www.w3.org/
[3]https://www.ietf.org/

IoT ontologies as Semantic Sensor Network (SSN)[66] / Sensor, Observation, Sample, and Actuator (SOSA) [84], Thing Description (TD) [23], Smart Applications REFerence (SAREF) [31], IoT-Ontology (IOT-O) [146], have been defined to create a common data interpretation ground for IoT systems [105, 156]. They cover various aspects (e.g., objects, context, communications) to different extents. These ontologies, originating from different organizations, may overlap and do not necessarily share the same structure or terminology. Moreover, the granularity level of an ontology varies from generic cross-domain descriptions to domain-specific ontologies (energy, agriculture, etc.). Alignments are not systematically made explicit. Another challenging factor relates to constrained nodes, which lack the resources to produce semantically enriched data, where semantic annotation makes it possible to attach meaning through transformation pipelines.

3.1.5 Interoperability. As an answer to the high fragmentation of the IoT ecosystem, multiple standard development organizations (SDOs) and initiatives deliver tools, models and guidelines to improve interoperability. Interoperability denotes the ability of different systems to work together [173], and has been very quickly identified as a challenging topic for the IoT field development [9], due to the accumulation of heterogeneity factors. The quest for global interoperability is first motivated by efficiency. Recent studies claim that almost 60% of IoT-published data are poorly exploited due to a lack of interoperability. The ISO/IEC 19118 standard [83] defines interoperability as "*the capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units.*" This definition showcases the utmost importance of ensuring seamless user access with low effort, underlying low cost for system management, while taking into consideration scalability issues. Reaching global interoperability in the context of IoT enables earning reliability and lowering the time it takes to access (connectivity and communication), read (syntax and schemes), understand and transform data into valuable knowledge (semantics). Many research works survey interoperability in IoT [2, 4, 17, 122] and propose different classifications, frequently correlated with sources of heterogeneity.

For each heterogeneity source, standards have been introduced to overcome heterogeneity as for communication protocols (HTTP, CoAP), data formats (JSON, CBOR), or standard ontologies (SSN/SOSA, TD). To address multiple heterogeneity dimensions, platforms include a standardization layer at their lowest level, where several devices with variable communication and access protocols can connect to a unique interface. They envision the standardization of access to objects, data and exchanges between devices to simplify their exploitation with low effort and cost. At the application layer, application programming interfaces (APIs) and cloud services interfaces offer data storage and analysis services to enhance collaboration with other platforms and systems.

However, the multiplicity of IoT platforms, reaching 620 suppliers in 2019 [80], poses a new problem of heterogeneity for users who must combine more than one platform to answer their needs, considering that each often adopts its own standards (structural or semantic). Although platforms play the interoperability game by developing access interfaces such as APIs/software development kit (SDK) or gateways, the diversity of these interfaces complicates the cross-platform and cross-domain application definition task. In the same vein, several projects have been or are being developed by research entities and SDOs to offer a reference solution for IoT interoperability. Amongst these architectures, one can cite: IETF SenML [154], OGC SensorThings API [106], W3C Web of Things (WoT) [61, 62]. The WoT stands out as a promising solution toward global interoperability thanks to its adequacy with web standards and the coverage of syntactic and semantic standards (Thing description model). The plug-and-play access mechanism eases the device's integration and discovery and supports the solution's flexibility.

3.2 System Architectures

Software architecture refers to a software system's overall design and structure, including its components, their properties and the rules shaping their connections and interactions [13]. A system's architecture substantially constrains its non-functional requirements [22, 26]. IoT systems often have a complex and heterogeneous architecture, interconnecting multiple hardware and software components at varying abstraction levels. Comparing IoT programming platforms at the architectural level is therefore important.

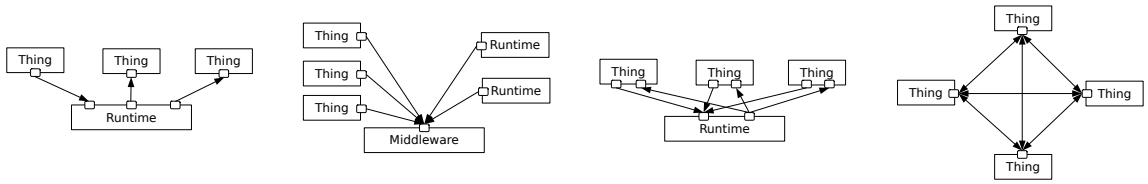
Standard ISO/IEC 25010:2011 [82] defines a taxonomy for software quality dimensions, such as efficiency, reliability and maintainability. Across literature [64, 118, 138], some qualities are frequently associated with the evaluation of ubiquitous systems, a class of systems that greatly overlaps with IoT systems: availability (the system remains accessible to users), scalability (the system can be extended to more components), reliability (the system is fault-tolerant) and performance efficiency (the system is functional with minimal resource usage). In the more specific context of IoT programming platforms, other quality attributes are particularly relevant: portability (any program can be executed by the system), usability (user interfaces of the system are easy to use) and evolvability (new components such as connected devices can easily be integrated into the system).

Information systems are built as a set of elementary components, each responsible for a collection of specific tasks, while connections between components take place via specific interfaces (e.g., a ZigBEE or MQTT network interface, or the API implemented by some software library). In an IoT system architecture, it is important to distinguish between thing components, capable of sensing and actuating in the physical world, and pure software components, only interacting physically via communication protocols (on the Internet or on restricted ZigBEE, BLE or LoRaWAN networks). Things tend to have low availability and low reliability and to offer no scalability (taken in isolation) and limited computing resources. In contrast, cloud-based IoT platforms are generally considered to offer a high quality of service (combining availability and reliability constraints), to scale on demand and to have virtually infinite resources. The quality of an IoT system combining things with a third-party software platform thus depends on the quality of the interactions between the things and the platform.

From a programming point of view, it also worth distinguishing between software components that are programmable and those that aren't. We will refer to the former as *application-runtime* components and to the latter as *middleware* components. In some cases, the application runtime is embedded in things themselves. Depending on the level of abstraction chosen to describe the system's architecture, a thing can be seen either as the combination of a sensing/actuation component and a runtime component, deployed on the same platform, or as a single component, whose interface to other components is at the physical layer. In the following, we consider the highest level of abstraction—with the least number of components–and only consider physical-layer connectors between components. A thing hosting an application runtime will therefore be considered as a single component.

Given the three types of architectural components of IoT systems (things, runtimes and middlware components), the following criteria are relevant for a comparison of IoT platforms.

- Number of components (of each type). Every IoT system architecture has at least 1 thing component but it may have 0 runtime and/or 0 middleware component. There may be n things, runtimes or middleware components.
- Number of distinct connectors per component. Each component has 1 or more interfaces to other components. Each interface is implemented by one connector (e.g., a library, linked statically or dynamically as an add-on to the component's main program). If connectors can be added dynamically to a component, it is assumed that a finite number of connectors can be added to the component. Some IoT platforms are used jointly with a software



(a) Kernel style (b) Blackboard style (c) Orchestrator style (d) Peer-to-peer style

Fig. 6. Common architectural styles for IoT platforms (: component, : connector, →: connection)

artifact repository from which add-ons may be downloaded (similar to Maven Central for Java or PyPI for Python, and to Linux package repositories). In this case, the number of connectors provided by the platform component equals the number of (communication-related) artifacts in the repository.

- Number of connections. Two interacting components must have compatible connectors to communicate (e.g., both components embed an MQTT client or an HTTP library). If the connectors of two components are compatible, a connection exists (at the architectural level) between the two components. For the sake of simplicity, broadcast and multicast communication is modeled as a set of one-to-one connections between components.
- Type of connections. A connection may be unidirectional or bidirectional. It is unidirectional if only one of the components can initiate communication (regardless of the number of subsequent messages being exchanged between the two components): an HTTP connection is directed from the client towards the server; an MQTT connection is directed from the client (taking either the role of a publisher or a subscriber) towards the hub.

Certain combinations of values for these 4 criteria can be factorized into *architectural styles*, as per usual terminology [158, p. 72]. Figure 6 shows some of the main architectural styles in use in the IoT literature: the *kernel*, *blackboard*, *orchestrator* and *peer-to-peer* styles. In the kernel style, the platform is a single runtime able to interact with a heterogeneous set of things via multiple connectors. It is analogous to an operating system kernel that exposes a high number of system calls and can handle a high number of I/O signals. A kernel platform is not naturally scalable, but it tends to display good performances and a fair level of reliability. A blackboard platform has a small number of connectors (usually 1 or 2, to read and write content) such that any thing with the suitable interface can write data (or read commands) and any application runtime can read data (and write commands). The blackboard is a passive middleware that can compensate low availability of things, but that may not match the performance of a kernel for asynchronous communication. To overcome this problem, the application runtime may be hosted on the same platform as the blackboard. One then obtains the orchestrator style (one runtime, few connectors). An orchestrator has similar qualities to a kernel. Finally, in the peer-to-peer style, there is no dedicated runtime and no middleware. Things drive the application in a fully decentralized manner, all things having a connection to all other things. The reliability of a peer-to-peer system is low in general, but its scalability may be high.

With respect to portability, usability and evolvability, architectural styles offer various guarantees. Portability of a program on a kernel is high, usability is high but evolvability generally depends on the richness of the artifact repositories associated with the kernel. On a blackboard, any application program can be executed on a blackboard, thanks to a uniform interface with all things, though it has the drawback that integrating a new thing requires developing connector code on the thing. The orchestrator has similar characteristics, except that portability depends on the expressivity of the orchestration language. The peer-to-peer style tends to have low usability, portability and evolvability.

Not surprisingly, a complex system is almost never restricted to one architectural style [50], and their combination is expected to represent an acceptable trade-off with respect to quality attributes targeted for a given application. Multiple studies have assessed the impact of implemented architectural styles on the quality attributes of the final application, sometimes with contradictory conclusions [118]. Some assessment frameworks hav e been designed for Edge Computing [64], but none specifically target IoT systems.

3.3 Cost and Licenses

A product license is a legal agreement detailing the terms and conditions governing the use of the product and its possible modification or distribution. It specifies in which context the product can be used and the underlying fees potentially charged. Software product licenses fall under five categories, with broad variations in the detailed terms:

- Open Source Licenses: A license where the product code source is open, free for use, modification and distribution.
- Proprietary Licenses: the product is provided to users under precise access conditions, often involving fees. Extending the product features is impossible since the code source is unavailable for modification or distribution.
- Free Licenses: This license is similar to an open source license, where the user can access the platform for free. In free licenses, however, the modification of the software code source and its distribution is reserved to its owner.
- Freemium Licenses: This license type relies on a cost model where a basic version of the product is available for free, while the full version, including more advanced services and features, is only provided by subscribing to a paying offer.
- Commercial Licences: include licenses where the use of the product requires paying fees. The modification and distribution of the product might be possible, but paying and restricted. Additional features, services and support options can be paid too.

Licensing is an important factor when selecting an appropriate product; users should carefully review the terms and constraints, mainly for financial reasons, but also to check the legal compliance with their intended use. Some solutions, although adapted to the requirements in terms of features, require an unaffordable budget, which blocks use. Consequently, licensing also has a significant role in adopting IoT technologies for organizations that can not afford proprietary solutions [18]. In agriculture or healthcare, for example, domain-oriented solutions are mainly proprietary. Business experts need to hold the necessary IoT background to adapt generic open-source tools to their needs, creating a real barrier to IoT solution deployment. Furthermore, some cost models restrict access to the product to a predefined number of devices/users/scenarios, preventing the scalability of the developed applications.

4 BACKGROUND ON DSL PROGRAMMING LANGUAGES

IoT systems are computational environments that need to be programmed to enable their services to be activated. A programming language denotes how humans interact with machines to define processes for solving problems belonging to a particular space [16]. Programming languages enable writing programs that express a set of computations to perform via instructions. Instructions encoding follows the language syntax (more or less high level) and their translation into binary machine code by semantic analysis, performed by the associated compilers or interpreters. The abstraction gap between machine code and user-defined code relates to the trade-off a language provides for usability as the readability of the syntax and behaviours understanding, over expressiveness defined as the scope of problems a language can address. While assembly language [75] was the first simplification of coding over machines, nowadays, programmers surf on thousands of high-level languages such as Java or Python. Beyond their syntactical variations,

programming languages may conform to several programming paradigms, i.e., approaches for problem-solving based on mathematical rules: object-oriented programming [137], functional programming [171], logic programming [27], etc. Van Roy and Haridi provide a good reference detailing programming paradigms [166] (see [165] for a recapitulated taxonomy). General-Purpose Languages (GPLs) refer to languages with usually completeness requirements providing powerful expressiveness for any computation. They enjoy wide popularity with strong communities, as for Java, C++, or Python. They are designed to fit any problem specification and enable domain-independent constructs. This powerful expressiveness comes with significant drawbacks on usability, learning curve, specific domain experts' adoption and execution costs. Nowadays, developers are exposed to an ever-growing wallet of GPLs with complex programming styles and multiple constructs, getting them often encumbered with unnecessary concepts for their specific usage.

4.1 DSL Definition

In contrast to GPLs, Domain-Specific Languages (DSLs) aim to address a small range of problems in a well-defined target domain, thus providing solutions that are often more adapted to the issue of IoT programming. In [164], a DSL is defined as

> *a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.*

A DSL is meant to offer a high programming-abstraction level by narrowing its scope to constructs capturing domain concepts [49]. From a domain-centric perspective, DSLs are regarded as an executable domain model [63] that :

> *embodies syntax and semantic that represents the concepts, attributes, operations, and relationships of a domain as an interpreted or compiled computer language.*

Higher abstraction level improves developers productivity and precision [99], reducing errors risks and enabling the automation of repetitive tasks, all by bringing user-friendly syntax (graphical and/or textual) and semantics. The perspective of restricting the application scope explains multiple mentions to DSLs as *small*, *tiny*, or *little* languages [14]. Beyond their basic global definition, DSLs involve a variety of characteristics rich enough to form taxonomies. The reader might refer to survey [123] for comparison dimensions, including their abstraction level, expressiveness, computation power, and domain experts' involvement and usage.

DSL development witnesses growing popularity within the software engineering community. One can type the keyword "domain-specific language" in scientific libraries search bars or on Github to discover hundreds of projects. This trend is not to be confused with a short history of DSLs, the emergence of which goes back as far as the definition of GPLs. For instance, the Structured Query Language (SQL), introduced in the 1970s by IBM under the original name SEQUEL, is a widely known DSL. SQL allows querying and managing relational databases.

Domains for which DSLs are developed can be split into vertical domains [63], representing business areas (health, finance, agriculture, energy, etc.), and horizontal domains capturing subsystems or technical components. Examples of horizontal domains-specific languages are in web development, with HyperText Markup Language -HTML- (web-page structure), cascading style sheets -CSS- (web-page style), Hypertext Preprocessor -PHP- (script language for dynamic web page generation); text editing, such as LaTeX, graphs, with GraphIt[4] (or DOT[5]); scientific calculation with tools

[4]https://graphviz.org/doc/info/lang.html
[5]https://graphit-lang.org/

such as Matlab or Mathematica; databases, with SQL or ScalaQL; parser generators, with Lex, Yacc or ANother Tool for Language Recognition -ANTLR-[6].

Sometimes, the frontier between DSLs and GPLs is blurred. Many developers still claim that languages such as SQL, capable of expressing various programming patterns in areas other than relational databases, cannot be "restricted" to the sole DSL qualification. However, it is much more natural and user-friendly to use these languages as per their main design intent (e.g., queries over relational data) and where they provide optimized behaviors. A DSL is considered as good not because it may capture all possible programs but because it performs significantly well in its domain of relevance.

4.2 DSL Types

DSLs can be classified following multiple aspects underlying their design and development. In [63], the author provides a morphology of DSL types by examining languages according to their syntactic appearance (textual or graphical), origin (embedded in a host language or external), implementation (compilation, interpretation, macro), and considering domain coverage (horizontal, vertical or technical patterns).

In the literature, the most studied distinction relates to the dependence of the created language to an existing language (GPL). Two families emerge, namely embedded DSLs and standalone DSLs, with significant disparities in design and implementation.

- Standalone, or external, DSLs require the definition of a specific syntax, a semantic analyzer (parser), and a compiler or interpreter to translate programs into machine code. Everything has to be done/redone from scratch. The development of external DSLs is very time- and effort-demanding, but they offer greater adaptability to the domain requirements, given the full privilege of syntactic and semantic definition.
- Embedded, or internal, DSLs [73] are developed on top of a host language, upon which they depend and inherit syntax and semantics. Nevertheless, they introduce notations, functions, and operators transforming the language's basic syntax to describe the idioms of the domain they address. All these additions remain bounded by the syntax and semantics of the target language, since they share the same compiler/interpreter.

There is no predominant choice between embedded DSLs and external DSLs, and they are, moreover, almost equally represented in the literature [164]. The pros and cons of the two types are discussed more exhaustively in [152], which introduces specific cases where internal DSLs are preferable over external ones or not.

4.3 Programming Approaches

Emerging IoT platforms expose simplified user interfaces by deploying domain-specific languages that offer increasingly simplified solutions for application development. The "domain" in the DSL nomination goes beyond the horizontal domain IoT to also address vertical domains such as building automation or agriculture. In particular, they offer variable levels of abstraction adapted to the skills of these vertical domain experts and their familiarity with the IoT context.

In this survey, we distinguish four programming approaches covering the development interface types that IoT platforms offer. In addition to GPL, DSLs are further classified in three categories: Domain-Driven programming, mashups and end-user programming (EUP). Figure 7 ranks these categories following their expressivity and abstraction levels. It is worth noting that these classifications regard the general characteristics of each programming approach and are not valid for all underlying platforms. Platforms may allow the combination of different modes of programming.

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

Some platforms are identified as model-driven but, in fact, include coding features via GPLs to modify the generated code and configure additional technical characteristics that go beyond what is strictly allowed by the exclusive use of models.

4.3.1 Model-Driven Programming. Model-Driven Programming is a software programming approach concerned with addressing challenges related to application development for complex systems, such as cyber-physical systems or multilayered IoT architectures, using a simplified representation of systems' components through modelling. It is defined in [116] as:

> a discipline in software engineering that relies on models as first-class entities and that aims to develop, maintain and evolve software by performing model transformations.

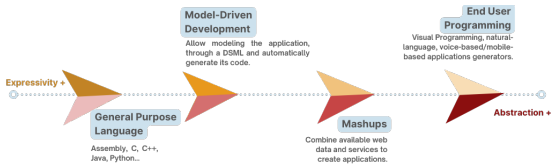


Fig. 7. Abstraction-Expressivity of IoT platforms developpement approaches

Models represent an abstraction of the system to be built. They are practical and understandable ways of structuring domain knowledge and a system's expected behaviour regarding a defined set of problems. The more the MDE approach is applied to specific problems and domains, avoiding generic formalizations that may be out of the scope of the system, the more likely the tool will be widely adopted [174].

The MDE approach relies on modelling the system to develop using languages like Unified Modeling Language (UML) [139] and code generator tools (Model-To-Text or Model-To-Code [116]) to create executable applications for task automation.

Domain-Specific Modeling Languages (DSML) leverage MDE principles to automatically generate fully operational application code based on the diagrams defined by users. While a DSL definition can be achieved informally, a DSML definition requires the formal characterization of:

- an abstract syntax metamodel, which is "*a model that defines the structure of a modeling language*" [30];
- a concrete syntax, i.e., notations in which the user specifies the model as a "*an abstraction of a system that helps to define and to give answers of the system under study without the need to consider it directly*" [30];
- a semantics that maps the concrete syntax (model) to the target language constructs.

Figure 8 illustrates the development process using MDE. The end user addresses the business-domain problem by designing a model through the DSML syntax. The DSML developer (platform engineer) defines the metamodel to which the defined models should conform. He also specifies how the user-defined model shall be transformed to cope with the platform's technical specifications. The output model is injected into a code-generation tool to get the final operational code. Code generation might be passive and thus require additional programming steps.

DSMLs offer users a high-level specification interface, manipulating a simplified syntax (often visual). The abstraction and automation of application generation allow a better understanding of the system, greater reuse possibilities, and better adaptability to evolving domain requirements. The time/cost factors are consequently optimized. In addition, DSML "*imposes domain-specific constraints and performs model checking that can detect and prevent many errors early in the life cycle*" [144]. Nevertheless, DSML applications are limited in terms of expressiveness to the semantics of core metamodel templates (capturing domain constraints).

4.3.2 Mashups. Mashup tools are applications combining the use of services and data available on the Web to offer a uniform user interface with programming functionalities. As for the MDE tools, the mashup tools aim to remove hindrances in access to development by simplifying the use of services and data interfaces. They go further by offering users the ability to compose services without editing models or lines of code.

Koschmider et al. [100] provide a good definition of a mashup:

> *a mashup is a web-based application that is created by combining and processing on-line third party resources, that contribute with data, presentation or functionality.*

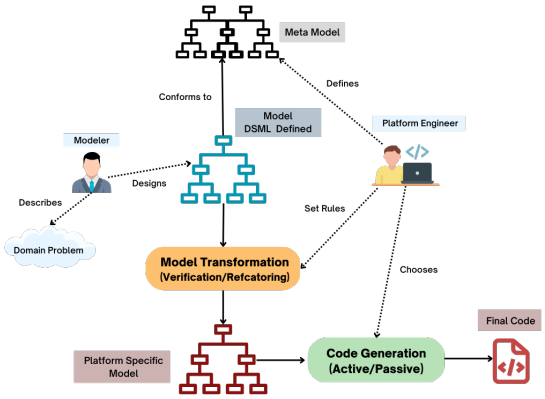


Fig. 8. Application creation in a model-driven development approach

The first mashup applications appeared with the internet emergence as Web Mashups. With the web considered as a big database, these tools extract and combine data on the web along with available services to offer querying, updating, and visualization features. Users can use and modify existing mashup scenarios, create their own applications by composing widgets, and suggest their products to other users. Examples include Yahoo! Pipes [44], DERI Pipes [103] a semantic web pipe (SWP) with RDF data support, or Microsoft Popfly [59], which offers visual formats.

4.3.3 End-User Programming. The End-User programming (EUP) approach appeared in the early 90s to open access to application development for users without a technical background. The rapid expansion of IoT witnesses renewed interest in EUP ([127, 159]) with the desire to enable end users to make use of their own data for daily-life tasks automation. Several platforms offer user-friendly interfaces for high-level programming, avoiding the need to access as many APIs as deployed physical architectures. The concept of end-user development has evolved with technological progress. While the first model-driven tools were already considered end-user development solutions enabling high-level abstractions over technical specifications, the democratization of the IoT is reaching increasingly large audiences. The end-user concept covering MDE and mashups, targeting domain experts holding a minimum technical background to define models or compose services, extends to profiles with no technological experience aiming at automating building management, for example.

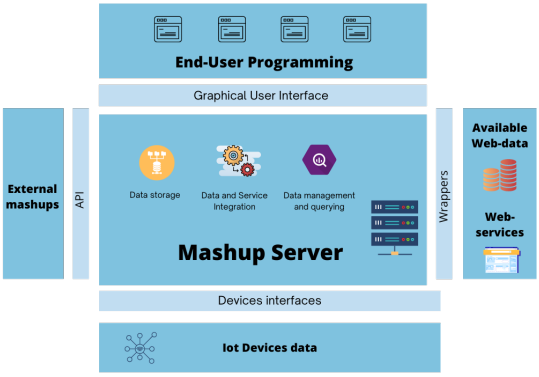


Fig. 9. IoT mashup platforms architecture overview

Paternò and Santoro [127] define end-user programming as a subset of EUP offering:

> *a set of techniques that empower end users to write programs by adopting special-purpose programming languages.*

The end-user programming approach includes multiple programming tools, with various abstraction levels to fit users with variable experiences and match specific use-case requirements [11, 126]: programming by example, natural-language techniques, spreadsheets, voice-based.

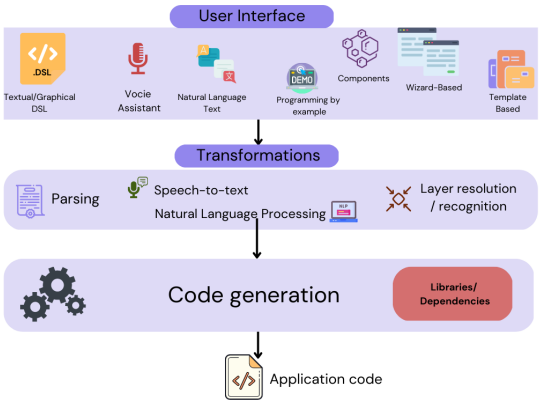


Fig. 10. Overview of end user programming techniques.

5 PROGRAMMING THE IOT

Creating applications for the IoT is of significant complexity, even for experienced domain developers. This is mainly due to considerable hardware and software heterogeneities, requiring advanced knowledge to achieve interoperability. As discussed in Section 3.1, device heterogeneity, due to hardware specification, communications, or resource limitations, makes programming IoT systems borrow a mix of different computational patterns with potentially different styles and programming languages with varying technical abstractions. In layered architecture deployments (cloud, fog, or edge), several programming tools are used for to increase technology coverage, widening the range of skills programmers must have.

An IoT-compatible program, either written using a GPL or a DSL, and depending on the role of the node running the program, should present some features for reliability and ease of use [10]:

- lightweight footprint and efficient resource use;
- fault tolerance to support connectivity and access instability;
- interoperability support through libraries and APIs covering various devices;
- scalability, managing access to masses of services, devices, and data (load-balancing) [179];
- concurrent access and coordination between computations performed in distributed parallel systems;
- availability of language tools and community via development environments, plugins, specific IoT libraries (e.g., General Purpose Input/Output -GPIO-, communication protocols).

Choosing a programming language requires defining the expected outcomes of the application, affordable cost, and time effort. The set of skills necessary for a programmer (end user of the language) to create a full end-to-end IoT system application is getting larger and larger, including knowledge of embedded devices, cloud systems, mobile and web development [157]. Developers use different levels of IoT software stacks depending on the aforementioned considerations but also their expected capabilities to fit the specificities of different deployments.

To make the potential of IoT accessible to end users regardless of their technical proficiency, a wide range of technologies and domain platforms have been defined to enable both developers and end users to design smart applications, breaking down the complexity curve and providing abstractions for low-level specifications. The IoT landscape offers hundreds of such platforms, and a selection methodology has been followed to present those relavant to this paper audience (Section 1).

5.1 Methodology

This survey follows a purposeful selection approach in identifying Internet of Things (IoT) platforms. The following selection criteria were adopted to ensure a holistic representation of available technologies.

- C1: Open Source Platforms.Preference is given to open-source platforms due to the benefits they provide for seamless access, collaboration and community extent. This aligns with the purpose of offering end-user/developer affordable and easy testing. This criterion has been relaxed for domain-specific platforms targeting the paper's use cases (building and agriculture), considering the predominance of commercial solutions and the high adaptability that might offer to these domain use cases.
- C2: Domain-Specific Language (DSL) Support. Platforms that offer a DSL designed for IoT applications are prioritized. The use of a DSL can simplify complex tasks, improve productivity, and enrich the community with additional programming resources.
- C3: Various Programming Approaches. The survey aims to present platforms adhering to different programming approaches. Some criteria might be relaxed to favour platforms that accommodate different programming paradigms.
- C4: Richness of the toolbox. Platforms offering different programming channels adapt better to larger user-profile communities. Modelling or mashup platforms might offer scripting APIs for GPL-based programming or leverage VPL to support non-technical end-users.
- C5: Interoperability.Platforms that enable interoperability are chosen to ensure the potential for seamless integration with different IoT devices, protocols, and systems. Given the heterogenous nature of IoT ecosystems, higher interoperability support enlarges platform applicability and impact.
- C6: Established User-Base. Preference is given to platforms with a significant user base over those in the early development phase. The user base supports future users to master the platform and adapt its features to their use cases.

It's important to note that this survey does not follow a Systematic Literature Review (SLR) methodology, since platforms come from both academia and industry, and the selection criteria for generic platforms differ from those considered for domain-specific tools. While this approach may not provide an exhaustive overview of every available platform, it aims to present a representative sample of the most relevant solutions relying on various programming approaches for end-user programming.

5.2 General Purpose Languages

When dealing with a small limited number of devices, IoT systems programming does not much differ from traditional web or mobile development. Interoperability and distribution concerns are less critical, and common general-purpose languages are convenient to use, assuming device resources can afford the requirements of the output programs. To this extent, one tends to take as the main criteria the device's capabilities and computational power, which explains why assembly language makes a strong comeback among the top ten languages, according to the TIOBE index for the most used programming languages [161]. By using assembly or C, developers trade productivity in code writing for performance and adequacy to a broader device range.

On larger dimensions, building an IoT system requires dealing with a system of systems [47], where handling low-level programming drawbacks is no longer reasonable. Many cloud-based solutions (Section 3.1) (Platform as a Service and Software As A Service) provide a "golden middle" solution, abstracting devices interoperability while

offering web-enabled programming environments through high-level programming languages SDKs (such as Java, python). Still, these solutions are tied to a subset of devices and low-end technologies.

Every year, the Eclipse IoT working group produces an IoT developer survey [38], tracking the most used technologies by IoT developers. Top programming languages are ordered by tier (refer to Figure 11 for an illustration):

- Cloud tier, with Java, Python, Javascript, and C++;
- Fog tier, with Java, Python, C, and C++;
- Constrained devices, with C, C++, Python, and Java.

The IoT-programming languages' ranking (last updated June 2022) correlates with the general programming trends, where python takes for the first time the top position, overpassing Java and C [161].

In addition to its extensive adoption for machine learning-based programs, the use of MicroPython (embedded python language) overcomes the lack of resources by providing complete OS and programming features. Constrained-device programming is evolving with advances to hardware technology, from one-shot configuration (plus multiple flashings) to dynamic programming capabilities that allow many computations with increasing high-level abstractions [157].

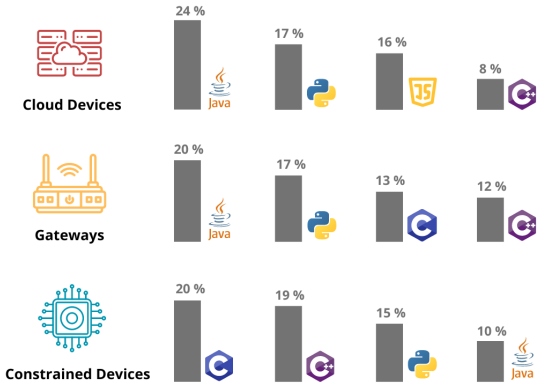


Fig. 11. Most used programming languages for IoT systems per tier, according to [38]

The use of general-purpose languages for IoT is supported by multiple libraries handling specific IoT features (General Purpose Input/Output -GPIO-, communication protocols, etc.), and IoT plugins extending development environments. The top three general IDEs following their adoption by the IoT community are [38]: Eclipse desktop, Visual Studio Code, and IntelliJ. Besides, other vendor-specific and board-specific tools carve a niche: ESPlorer (for ESP8266 programming with lua and python), Android Studio, Thonny IDE (for micropython programming), Eclipse Orion.

5.3 Model Driven Tools

The literature pays considerable attention to MDE works for the IoT domain. In [39, 113], authors survey these tools whether they are generic or domain-specific. In the following, we discuss the most significant IoT domain-specific modeling languages.

5.3.1 ThingML. [67] is an open-source project offering a DSML, along with a set of tools for cross-platform code generation through model transformation, plugins for the Eclipse IDE, a standalone text editor, and methodological/-documentation support. A first verison of the platform has been released in 2012. The cross-platform code generation framework is designed for use in distributed systems with heterogeneous nodes with support for C, C++, Java, JavaScript and Arduino. ThingML generated code can be run on devices with varying capabilities and operating systems: windows/linux servers, Rasberry Pi, ESP32... It is intended for a hub-based system but can be considered for cloud use in a client-server architecture. Unlike many DSMLs, the ThingML language has a textual descriptive syntax allowing users to model:

- Things, for the structure of the system components with asynchronous message interfaces;
- state machines, with UML state-charts-like configuration files describing the components (things) behaviors and their interactions. Things communicate in an asynchronous fashion ;
- Events, with an imperative action language (platform-independent) for event processing in an (Event-Condition-Action) fashion.

A multiplatform code-generation framework enables model transformation into target languages such as Java, Javascript, C, Arduino. The generated implementation contains the executable code and configuration files for different target devices. It offers wide support for devices and communication protocols, and allows for the definition of custom data types and structures that can be used to represent data in a variety of formats (XML, JSON,...). In addition, ThingML provides a rich toolbox, including APIs (Java API, Rest API) along with Eclipse IDE SDK allowing developpers to interact with the generated model-based generation code. While the ThingML DSL is purely textual, the Eclipse SDK provides a corresponding graphical modeling tool and a runtime environment for thingML-generated code.

The ThingML is built using the Eclipse Modeling framework, which allows flexibility in extending the system as in [119], where the authors present an extension of the language to support things with data analytics features. To support different IoT domains, ThingML can be considered as a generic tool, since it provides a set of built-in features and constructs that can be used to define the behavior of IoT components, regardless of the specific domain they are being used in. The tool is not restricted to fixed hardware or network specifications.

ThingML has been validated in multiple use cases with different platforms, but remains a fairly technical DSML intended for users with technical background and IoT architecture knowledge to fully make profit of the code generator framework.

5.3.2 Midgar. [54] [55] Midgar is an IoT platform offering several services, including a DSL and a graphical editor to interconnect web services and create applications for heterogeneous devices. Midgar aims to support interoperability by interconnecting objects through a Midgar server with REST services without getting to manage the technical sophistication specific to each object. The platform consists of 4 layers: Process definition, service generation, data processing, and object management. Only the process layer is intended for user interaction. The latter defines connection processes on objects via MOISL, the DSL developed with the HTML5 canvas and JavaScript generating serialized XML models. Service generation achieves model transformation into the target application language. The platform offers code generation into Java for desktop or mobile application and C for arduino/android devices. Things or objects implement the messaging interfaces with the Midgar server (REST service).

The platform evolved by offering several model-based programming languages. The primary contribution includes Midgar Object Interconnection Specific Language (MOISL), a language with graphical notations enabling the definition of heterogeneous and ubiquitous object interconnections. It allows the user to describe how each object behaves with its network and the Midgar server and considers security aspects. A textual DSL with the same purpose has been defined later, as Midgar Use Case Specification Language (MUCSL) [56], which provides development support of automation routines expression using a natural language (English-like) syntax. This layer does not support the automatic generation of IoT applications and requires the explicit definition of the logic of the objects by the users. In recent work [55], the same authors define Midgar Object Case Specification Language (MOCSL) with a graphical interface to facilitate the creation of smart objects and the generation of data-processing applications by users with a little technical background. A drag and drop interface allow to define communications betweem different conented objects.

Example 5.1. Consider the use case defined in Section 2.1. Three objects can be connected to the Midgar platform :

- A temperature sensor, registered under the ID tempSensorXXXX01 (numerical values),
- A carbon dioxide sensor, registered with the ID carbonDioxideYYYY01 (numerical values),
- An actuator for automatic window openning, registered under the ID windowAutoZZZZ01, (0 for close and 1 for open).

The application logic can be defined using the graphical notation of the platforms, or the textual DSL (MUCSL), where a conjunction of conditions can be expressed and verified (here, temperature above 25 degrees and CO2 levels above 350 ppm) to trigger an action on the connected device (window actuator).

Listing 1. Example of MUCSL rule definition

```
when [the] tempSensorxxx01 [is]
greater than 25 or
CarbonDIoxydeSensoryyyy01 [is]
greater than 350 Then
[the] windowAutozzzz01 to 1
```

5.3.3 FRASAD. FRAmework for Sensor Application Development [121] is a development framework allowing users to create IoT applications by defining a multi-layered node-centric model. It focuses on in-board sensor computation providing a simple alternative to program sensor events reading, alerts and actions. Communication within a sensor network is handled through two communication protocol support: unicast and broadcast protocols.

The platform enables developing applications while hiding hardware technical specifications related to the nodes. This abstraction is possible thanks to the definition of two layers: the application abstraction layer (APL) and the operating-system abstraction layer (OAL). Accordingly, the platform uses model-driven architecture principles at three distinct layers, supported by a DSL definition. At the application layer, the visual model operates on rule-based programming patterns to describe the sensor nodes and their local behaviours independently from their technical specifications. The model is stored as an XML file. A code generation tool automatically transforms the user-defined models to generate platform-independent applications at the OAL. The latter is then mapped and compiled by OS-specific C compiler into binary code. The rule DSL language allows for a three parts rule definition: Select-Clause, Processing Clause and ActionClause.

The platform leverages simplified text and visual notations to cope with non-technical programmers' capabilities, which results in bounded expressiveness. As code generation is a two-step process, device-specific code might deviate from the user-defined model. This gap can be explained by the pipeline of mappings applied to get a platform-specific code. Additionally, the platform focuses on a fixed number of devices and operating systems (Contiki OS, TinyOS) and protocols, limiting the applicability. To the best of our knowledge, the platform development has not been perused, and no code repository has been maintained.

5.3.4 WoX. Web of Topics (WoX) is a multi-layer IoT platform first introduced in 2015, that aims at simplifying the design of user-centric applications by adopting a model-driven approach. It extends the web of things WOT by focusing on the functional aspects of the system and this despite the object's hardware complexities and their diverse communication protocols.

To overcome the technical complexity of the Web of Things, WOX introduces an abstraction layer hiding heterogeneous technical details of things in a conceptual model. The WoX model is built around the "topic" concept: a topic describes the feature values (perceivable, measurable characteristics) of entities of interest. It is a representation of

a physical or virtual sensor/actuator. The topic is identified by a URI designating it and its location. An IoT entity is defined by the couple (topic, role), given that a role expresses the entity's technological and collaborative status.

Unlike many IoT tools addressing the development within a single IoT architecture layer, WoX defines three variants, one for each layer: WoX cloud, WoX local (L-WoX) for mobile use, and eMbedded WoX (M-WOX), for constrained objects embedded. Corresponding adapted APIs are presented: Java/Python/.NET APIs, Android/iOS or C/C++ for constrained objects.

To homogenize modelling, WoX middleware (Hardware Abstraction Layer) exhibits an exchange format based on the publish-subscribe model and provides a set of adapters for physical and virtual communication protocols. Besides, the tool is structured to manage the flow of events issued by physical or virtual environments through adapters connected to a REST service to several web technologies and a Linked. Open Data (LOD) API for the semantic transformation of topics into a semantic data model.

The platform has been demonstrated via prototypes such as the airport short-stay parking service or City4Age, an application for the e-monitoring of elderly people's behaviors [46].

In a recent work [112], WOX authors introduce WOX+, extending the original model by integrating machine learning for automatic automation rules discovery. Rules are created by mining users' habits in smart environments.

5.3.5 UML-Based Tools. Other works adapt the use of the UML language to build visual interfaces enabling users to express the structure and behavior of IoT systems. Eterovic et al. [41] propose a visual programming modeling language based on UML. The base elements of the model refer to "things" (sensors/actuators, physical or virtual), and their communication interfaces describe data flows. The things compose hierarchical subsystems with input/output notations. Even if the language is designed for non-technical users, the authors propose an extension using advanced UML notations for developers to achieve more complex tasks. UML4iot [160] borrows some UML constructs to create customized profiles that address IoT systems' challenges, namely heterogeneity and distribution. The profile is used within a wrapper to define an IoT-compliant layer, providing cyber-physical manufacturing-system components with capabilities to integrate IoT architectures and thus take advantage of IoT protocols (LWM2M).

Other tasks support. The model-driven development approach is used to support different aspects of IoT systems. Some tools aim to generate executable applications automatically, as in the examples discussed earlier. However, other proposals follow the MDE approach to perform tasks such as simulation [19], design [129], or specification.

5.4 Mashup Tools

In the IoT era, the need to compose services and expose them in a uniform user-centric fashion has arisen, and the mashup principle has gained extensive attention. New challenges appeared regarding mashups' use for the IoT, given the heterogeneity of devices and resources to integrate.

The Web of Things (discussed in Section 3.1) tackles these challenges by allowing users to create applications based on a uniform semantic data model for heterogeneous sources (things or resources) integration and combine the use of web services and standard technologies (connecting functional islands). Such a platform is possible thanks to the availability of communication protocols and REST APIs. Mashup tools are helpful to speed-up application prototyping and are suitable for non-technical users or domain experts to quickly create and deploy new IoT applications. They often leverage visual interfaces to encode rule-based and flow-based programming patterns, describing message flows between components as resources producing data or web services. A significant drawback of mashups comes from

their high abstraction level and restriction to a flow-based programming paradigm, limiting the scope of the application out of the system behaviour on other functional aspects or descriptions [131].

Some examples of widespread IoT mashup tools are discussed in the following.

5.4.1 Node-RED. Node-RED [125] is a mashup tool created by IBM for IoT applications' development with the aim to wire together IoT components. In 2016, Node-RED was transformed into an open-source JS Foundation project. The tool offers a browser-based drag-and-drop graphical user interface for building and editing event flows through nodes, reducing the size of code writing for users. A Node.js runtime supports the application deployment of node-centric event-driven programs, taking advantage of the built-in event model and native support for JavaScript. A node characterizes a hardware device, a software API, or a service. The network of nodes is managed by a node package manager (npm) with an extensible set of components, where each node behavior is described in JavaScript and its structure in HTML (browser UI).

The extensible palette of nodes includes function nodes (trigger, execute), network nodes (MQTT, HTTP, etc.), parser nodes (for CSV, XML, HTML, and so on), etc. The interaction between nodes is constructed through links, capturing events transformation, and each process is a black box, autonomously and asynchronously transforming data from the input nodes to some output result for the target node. The created flows can be stored for reuse in JSON format. When a user deploys a Node-RED flow, the runtime generates and executes the underlying JavaScript code based on the connections and configurations explicited in the visual flowchart.

Node-RED is platform-agnostic tool written in Javascript; it can be deployed on small devices due to the lightweight nature of Node.js, supporting edge-computing scenarios as well as cloud devices, which guarantees a full-IoT system-development process. Another advantage of Node-red is the possibility to integrate different technology libraries (databases, MQTT communication protocol, etc.), strengthening interoperability and extensibility. When a user deploys a Node-RED flow, the runtime generates and executes the underlying JavaScript code based on the connections and configurations made in the visual flowchart. One drawback of the open possibilities in flow definition (unique or multiple flows with different node settings) is the difficulty of fault detection and system behavior fixing.

Many works extend Node-RED with additional features such as voice command interface [133] or adapt it to specific uses cases [45, 148, 176]. Glue.things [96] is built on top of Node-RED, providing user-friendly predefined trigger and action nodes. In glue.things, a flow editor is associated with a master device that controls all the nodes in a local network. The master device can be deployed on the cloud allowing service compositing at this level. Node-RED is built to run on a single device. Distributed Node-RED [15, 57] proposes a distributed version to run in fog-based architectures.

Ji et al. [86] propose an extension of Node-RED with a module running as a WoT servient. It combines REST APIs and IoT devices as web things. The Thingweb node-wot [48] is used as a runtime environment on the Scripting API. A top layer implementing WOT Thing Description (TD) allows Web things (sensors/actuators) to interoperate, exposing their properties, actions, and events. Sensing data are streamed in a chat channel, and activated triggers are reported to users as WoT events. Authors claim that Node-RED enriched with specific WoT semantic description and servient-behavior integration represent a good candidate for a standard WoT implementation.

Example 5.2. Node-red can be used for both automation scenarios in building and agriculture, presented in section 2. To showcase an automation program for smart irrigation, we assume a minimalist hardware setting with a moisture sensor and a water valve, both connected to an Arduino microcontroller. An Arduino-node can be installed into the palette manager :

Listing 2. npm arduino node installation

```
npm install node-red-node-arduino
```

This node enables communicating with the Arduino through the serial-in node for data reading and the serial-out node for command sending. Moisture data received is processed (through function nodes) to check whether the moisture level is below crop watering recommendations.

The flow can also integrate available web services for weather data, such as the OpenWeather API, accessible via HTTP requests. To this aim, an *inject* node allows triggering API requests at a certain defined frequency, while the *HTTP* node is set to GET weather data. A function node filters data (JSON) for rainfall forecasts. The data obtained from the API can be combined with moisture sensor data to avoid valve actuation if rainfall is expected.

5.4.2 Dynamic Dashboard. Vanden Hautte et al. [167] provide a dash-boarding tool enabling the visualization of RESTful web things (sensors) and available data-aggregations services, abstracting sensor settings. The tool uses semantic reasoning on things metadata to suggest a suitable visual interface composed of single-service widgets. The dashboard interface is dynamically customized given the events picked by the user. The dynamic dashboard provides real-time data and is supported by three core services:

- It subscribes to a data streamer (Kafka Stream) to enable access to data and apply filters to get relevant event-related values. The output is then transmitted to widgets for visualization.
- It interacts with a broker, which stores and provides the user interface state. It hosts the semantic reasoner in charge of suggesting appropriate widgets given things semantic metadata.
- WoT compliant gateway API discovers things, and the semantic annotations explain things capabilities.

5.4.3 WoTKit. : is a web-centric lightweight Java platform for managing IoT things and their real-time events. Things can be grouped into systems (and subsystems), identified as modules that communicate through wires in a flow-based fashion. The combination of modules and wires creates pipes inspired by Yahoo! Pipes [44]. WoTKit web application has both visual and textual user interfaces. The main visual user interface is a javascript dashboard combining widgets that enable pipe creation, start, stop and editing features, as well as navigating through different user pipes. Besides, users can extend the built-in features by defining new modules' scripts in Python to integrate into pipes. A central sensor gallery allows reusing created sensors or discovering other public users' shared sensors added to the system. The gallery also stores meta-data describing the sensors, their location, and data output. WoTKit serves as a sensor-data aggregator for processing, with the generation of control messages to actuators and visualization features. Additional sensors can be registered to the platform through gateways definition. New sensor information are posted through the REST API by providing a description file in supported formats (JSON, CSV, KML, HTML).

An advantage of the WotKit architecture is the separation between wires and modules; modules can have many input wires enriching program patterns. The processor can handle many flows for a single user.

Other similar tools include ClickScript [111] is a Firefox plugin fully written in JavaScript, on top of an Ajax library, enabling access to REST APIs to create visual mashup applications. A set of resources (such as websites, sensors) can be used to define javascript programs using basic conditions/loops.IoTMaaS (IoT Mashup as a Service) [78] is a cloud-based IoT solution offering a mashup service to overcome IoT system-component heterogeneity. It leverages the model-driven development principles and composes three models: thing, software, and computation resources. The end user operates on models through a browser-based UI by selecting things, the software required for a program,

and the computation resources to allocate at run time. Open.sense is a mashup tool supporting the social Internet of Everything [74], connecting humans, services and things through social networks and APIs.

5.4.4 A-Mage. A-Mage, an Atomic Mashup Generator [98], is a tool enabling Atomic mashup generation based on Thing Descriptions files and a possible set of constraints and filtering conditions expressed by an end-user. All candidate atomic mashups are automatically generated, enabling user evaluation through a sequence diagram. The user picks the best fit atomic mashup (or adjusts constraints to get a new one), achieving the desired system behavior, and generates an executable code based on the WoT Scripting API. The end user constraints are expressed in natural language and then processed and used to generate semantic filters restricting the number of things involved in the WoT system, their types, contexts, TD annotation, or annotation availability. The tool has been tested against agriculture, industrial and smart home scenarios, with reasonable generation time and good response to user's constraints.

5.5 End-User Development Tools

The following sections overview some examples of widespread EUP platforms. Detailed comparative studies of IoT end-user development tools are reported in [104, 126, 143].

5.5.1 Trigger-Action Based Approaches. This programming approach uses event-condition-action (ECA) rules [25], with visual user interfaces. The user specifies an expected system behavior through if-then statements, to automate actions following some trigger occurence.

IFTTT. IF This Then That [77] is a widely-used IoT platform launched in 2011, and represents the most frequently used tool of event-action programming [117, 163]. IFTTT presents a web-based interface (and a mobile app), enabling users to define programs through automation scripts called *appelets* or *recipes* that connect services, websites, and physical devices.

Each applet encodes the automation logic, i.e., a trigger-action rule "on trigger do action", such as turning off the light when someone leaves the room or sharing stats data on Twitter following a smartwatch notification. A trigger is an event produced by a connected service (ingredients data) as the sensing data output, while actions can be executed by physical devices or web applications.

IFTTT enjoys a large community thanks to its user-friendly interface and support for trending use cases such as home automation [117]. This led to many research works and products extending the platform with support for voice assistant commands [102], home automation extension [170], or secure authentication mechanisms [12]. The platform owes its popularity to the simple means for creating automation rules, which in contrast makes for a significant expressiveness limitation. Tasks only include rules with a single action per trigger, preventing the composition of rules and, therefore, the expression of complex scenarios. IFTTT also provides a platform called "Platform API" that allows developers to create more complex integrations. The Platform API is based on GraphQL and allows developers to query and mutate data in the IFTTT ecosystem.

The platform supports a wide range of device integrations but do not allow the use of virtual devices.

A recent development of IFTTT supports the composition of actions through a "Maker" platform enabling developers to add multiple triggered actions. Similar platforms such as Zapier [178], Microsoft Power Automate [120], or recently Mozart [101] address this issue by enabling broader rule automation patterns.

Example 5.3. A popular use scenario of IFTTT is home automation. Consider the air quality monitoring scenario defined in Section 2.1. A minimalist hardware setting includes an ESP32 microcontroller hosting CO2 sensor and

a window actuator. An IFTTT applet defines a *trigger* on any available MQTT service that can get data from the microcontroller. Filter code option enable defining custom data transformation to check whether the CO2 level is above normal. A window openning command through MQTT service constitutes the *action* statement. To combine a trigger recorded by a web service, such as temperature on weather API, a *query* statement must be defined (only avalable for IFTTT Pro users).

Zapier. : [178] This is a web-based workflow automation platform primarily interested in business project management. The zaps (equivalent to recipes) allow more composition rules than IFTTT [132] since one trigger can lead to a sequence of actions. The subsequent tasks are more expressive since they include filtering and data transformation on trigger data. Zapier also provides developers with support for customized service integration.Zapier exposes a REST API enabling developpers to create zaps custom templates.

Microsoft Power Automate. : previously called, Microsoft Flow[120], provides advanced conditions and looping constructs for more expressivity. It is a workflow-management platform with web and mobile tools, offering an interface for connecting two or more cloud services to create business workflows, such as automating file synchronization, alerting, and data organization. It is oriented towards supporting Microsoft business tools integration for business task-automation scenarios.

Trigger-action-based programming tools offering compositions over rules appear to be complex while expressing advanced scenarios. The user might need to compose numerous constructs (textual or visual) to encode a program, and quickly be confused by an overwhelming flow of events and triggers. This setting leads to a high error risk and a growing complexity.

5.5.2 *Programming by Demonstration.*

Epidosite. Enabling Programming of IoT Devices On Smartphone Interfaces for The End-users (Epidosite) [104]A mobile platform for home automation leveraging a programming-by-demonstration style. A user performs operations on a real use case example to define expected system behavior. Epidosite reasons about the underlying logic to infer a generalization program pattern that can be applied to new scenarios and features later. The programming is performed through mobile, using the Android accessibility API to record user behavior when using IoT applications. The recorded scenario is then used to generate a script for further use automatically. Therefore, the created automation tasks are sequential action-triggered rules and do not handle trigger or action composition. The tool also supports additional web services integration thanks to a REST API allowing to connect to IFTTT. Epidosite scripts can be used in IFTTT as triggers or actions.

Improv. This system [24] exploits the programming-by-example approach, enabling users to compose software operating across different devices to define a common interoperable system-interaction behavior.

The user mimics the interaction desired for the set of devices, and the tool transforms these parameters to extend the input-device application by connecting it to additional devices.

5.5.3 *Visual Mobile Applications.*

Puzzle. is a desktop/mobile platform enabling the development of smartphone IoT applications. It uses the jigsaw puzzle metaphor, where each system component is perceived as a puzzle piece exhibiting some functionality, and connections provide necessary data inputs/outputs. The puzzle diagram corresponds to a sequential automation logic,

and Looping on particular triggers or actions is possible. The color code and the shape of the pieces suggest to the user what puzzles can be connected based on data types and transformation for functionality composition. The tool allows the integration of multiple hardware things and query web services, but its expressivity of the tool is limited to a predefined puzzle set.

5.5.4 Tools using Natural Language . In natural language programming, the user expresses the requirements a system should fulfill or enumerates automation rules using natural language. The constructs can sometimes be constrained with templates, and UIs are either textual or voice-based. Recently, advances in natural language processing (NLP) opened up avenues to a less constrained user-interaction mode, especially when it comes to transforming the output of voice assistants.

AppsGate. AppsGate [28] is a rule-based web platform for home automation using a pseudo-natural language syntax for user interaction.

A dependency graph and timeline depict home devices and associated services, enabling users to specify control patterns in a syntax editor with simple predefined constructs. Visual notations and color codes assist the user in identifying allowed syntax, checking rule completeness extent, and understanding the system's state. The rules define how automation actions are associated with the environment state or occurring events. The simplified syntax does not allow for rule composition or operating changes on the graph devices or the timeline of events.

5.5.5 Voice Assistants. Voice-activated devices are increasingly integrated into various automation fields, thanks to their growing hardware capabilities and the advances in artificial intelligence. Voice assistants are increasingly smaller, cheaper, and equipped with broader skills. Leading manufacturers are marketing powerful assistants: Alexa, Siri, Cortana, Google Assistant.

Beyond acting as high-level interfaces for getting weather-like data, the progress of natural language processing and the richness of human-interaction logs offer a learning base for programming control behaviors through IoT systems. Most of the applications of this programming style have been applied to smart home scenarios, which remains one of the most end-user-centric IoT applications, with a central need to simplify access to programming for non-expert users.

Early uses of voice assistants extend visual programming platforms such as Node-RED or IFTTT with voice user interaction, enabling humans to get natural-language answers. Rajalakshmi and Shahnasser [133] combine the use of Alexa and Node-red to offer complex automation-rules definition for home-environment control through the platform and the use of Alexa to simplify the user awareness of the house state. However, in the cases mentioned above, the voice-activated device does not operate to trigger or configure rules but is limited to query answering.

Other propositions allow users to trigger predefined rules in IFTTT via voice [92, 95]. The tool expressivity is limited to the IFTTT applets, and the voice can be used only to activate the rules. A similar prototype has been defined by [97], which integrates Google Assistant to an Android application using MQTT, Node-RED, IFTTT and Mongoose OS. The event-action rules are first defined in Node-RED and can be triggered through Google Assistant. TAREME (Trigger-Action Rule Editing, Monitoring) [114] allows triggering and defining rules through Alexa.

Almond [21] is an open-source platform [151] that defines a textual domain-specific language named *Thing Talk*, for connecting IoT devices, abstracting their technical specifications and enabling a pseudo-natural language for rules definition. A knowledge-base *Thingpedia* includes natural language interfaces and open APIs for IoT devices and services used for system building. The Almond Virtual Assistant is a Java/JavaScript-based mobile application running a web service for voice-to-text transformation enabling users to formulate Thing Talk constructs vocally.

6 DOMAIN DEDICATED IOT PLATFORMS

In this section, we discuss IoT platform providing domain-specific functionnalities: building automation (Subsection 6.1) and smart agriculture (Subsection 6.2).

6.1 Building Automation

The earliest form of building automation goes back to the 1960s [34] with the use of digital computers to automate heating and ventilation in large commercial buildings. With the evolution of control systems to handle additional automation aspects such as lighting, security, or electrical equipment, the concept of Building Automation Systems (BAS) has emerged. A BAS is a distributed system integrating all building automation services, aiming for energy optimization and user comfort. The rise of IoT technology has accelerated the development of BAS by enriching the lower perception layer with new sensors, actuators, and connectivity protocols, enabling the definition of autonomously triggered routines and opening up the spectrum of use cases.

IoT platforms in smart buildings support the centralization of management systems and offer multiple features:

- Data collection and visualization, e.g., sensors, connected devices, web-available data, and third-party databases, in order to create more accurate data analyses;
- Data analytics oriented towards managing and optimizing energy and tangible resources use, e.g., heating, air conditioning, lighting, occupancy status, predictive maintenance;
- Third-party systems and web services integration for strengthening the interoperability and reinforcing the automation routines definition.

There exist a large panel of home automation platforms with varying technical characteristics [51], from device and protocol support to user interfaces and the application layer. Some building automation IoT platforms offer development tools, including APIs, SDKs, plugins and test environments, that allow users to exploit system data to create applications with custom features. The following subsections illustrate open-source and commercial platforms dedicated to home automation.

6.1.1 OpenHAB. Open Home Automation Bus is an open-source home automation platform designed to integrate a large panel of home devices and different systems services. The project was launched in 2010 and became an official Eclipse Foundation project in 2013. OpenHab is a cloud-agnostic platform written in Java that operates on various operating systems and can be deployed on either classical computers or embedded boards such as Rasberry Pi, Beaglebone Black, Intel Gallilo, or any device running Java Virtual Machine (JVM). Compared to other open-source solutions, it has been tested on a large number of ARM-based single board computers. The platform is built as a modular software architecture following the OSGi (Open Service Gateway Initiative) framework [37], where modules called "bundles" are implemented separately and communicate via a publish/subscribe architecture.

The physical layer of the OpenHAB system, named Things, manages over 400 bindings (plugins), integrating more than 3000 entities (things). A "thing" corresponds to a physical device identified and described by a thing-configuration file detailing its items and channels. The latter represents the thing's exposed capabilities. An "item" is the virtual representation of a thing whose state can be modified by the application commands. There exist 14 possible items that can be linked to a range of channels for event handling. The set of supported things is extensible: device or system integrations can be achieved via binding definition, extending the thing handler Java library.

In addition to physical things, a set of add-on bundles increase the core services with external systems or services. The cloud connector enables connecting the local OpenHab runtime to a cloud instance to use advanced features, including secure remote access, push notifications, or access to web services requiring Open Authorization (OAuth[7]), such as IFTTT or Google Assistant. Additional system-integration add-ons cover HomeKit, Philips Hue, NEEO, ImperiHome. The platform also integrates a REST API and enables event monitoring and historical data management through bindings integrating InfluxDB and Grafana open-source time-series databases and visualization tools.

Different user interfaces are available on both mobile and web for programming automation routines. OpenHAB defines a rule DSL, based on Xtend, that allows using a simplified textual syntax for rule definition. Since rules follow an event-condition-action (ECA) programming style, actions are automatically triggered by time or sensor-based events. DSL-based rules can be edited via a graphical rule editor and are ultimately parsed into actuators' commands. Technically trained users can access full configuration and automation routines' options thanks to the supported scripting languages: Python, Javascript, Groovy, and Ruby.

Version 3 of OpenHAB, published in 2020, brings significant changes to the platform. A semantic model enriches automation rules with semantic actions definition. The model relies on a modular ontology (location, equipment, point, property) that users can employ for modelling their physical environment. In addition, the user interface is reorganized with page definitions, including user-defined widgets. Finally, a blockly [58] rule engine has been integrated, easing rule definition for non-technical programmers through a fully visual UI. Note that the first milestone of the upcoming Version 4 has been released in March 2023.

OpenHAB enjoys increasing popularity and has gathered an active community [124], which supports its adoption along the different versions. Users benefit from an extensive documentation adapted to beginning as well as advanced use scenarios. The richness of the interoperability features and supported interaction interfaces increases the system's complexity, especially for users aiming to define advanced automation rules or integrate new devices. We provide below an example of use of OpenHAB.

Example 6.1. Consider the window-opening automation scenario mentioned as a use case in Section 2.1. OpenHAB rule DSL can be used to define an automation rule whose action is a command to a window, triggered if the carbon dioxide sensor records a high level (above the threshold).

Listing 3. Example of OpenHAB rule definition

```
rule "Open_window_when_CO2level_is_high"
when
  Item CarbonDioxideLevelSensor changed
then
  val co2Level =
    CarbonDioxideLevelSensor.state as N
    // CO2 level read from the sensor

  val co2Threshold = 1000
    // CO2 limit, in parts per million
```

[7] https://oauth.net/2/

```
  if (co2Level > co2Threshold) {
    logInfo("co2.rules",
            'CO2 level too
             high: opening window')
    Window.sendCommand(OPEN)
  }
end
```

Open-source platforms similar to OpenHAB have been developed. We survey some of them below.

- Home Assistant [68] is a home-automation platform developed in Python that enjoys worldwide popularity; it supports interoperability with a wide range of plugins allowing the integration of various devices and protocols (currently over 2200). Home Assistant is a free software backed by a strong community, assisting users with multiple topics and providing extensive documentation. The platform makes privacy and user control a priority; it does not depend on cloud services to operate. However, cloud-service integration with reinforced authentification is provided to ensure remote access and support voice assistant interfaces such as Alexa. As for OpenHAB, the core component of the architecture is an event bus handling event-based components' interactions. End-users program automation routines with ECA rules via a graphical dashboard generating YAML files. More sophisticated developers can create advanced services using Python scripting integration or REST and WebSocket APIs. The configuration of the dashboard, automation rules, devices and concepts is mainly achieved through YAML configuration files, which may turn out to be somewhat heavy for beginner end users.
- Jeedom [85] is a home-automation platform developed in 2014, written in PHP and operating on various Linux distributions. The platform is standalone, providing a user-friendly and flexible interface thanks to numerous customization options via widgets. The platform supports several device integrations, protocols and services via official plugins or third-party plugins. The Jeedom market counts around 600 plugins, but some are not free. The definition of new plugins is possible in PHP, based on provided templates and JSON configuration files. In addition to the visual dashboard, automation rules can be expressed via scripted plugins written on Python, PHP, Shell or Ruby. The documentation of the platform is provided in different languages. One drawback hindering the platform's expansion is its focus on French users, since the community forums are mainly written in French.
- IOBrocker [79] is a node.js-based home-automation platform freely available and running on multiple operating systems. The platform has a modular architecture and currently provides 450 adapters implementing devices, protocols and services integration. IOBroker Pro is a paying cloud service proposed by the platform to support remote access or interfacing voice assistant services such as Alexa or Google Home. Developers can create custom javascript code for home automation using VSCode[8] or Webstrom [9] IDEs.
- DomoticZ [35] is a lightweight home-automation system first released in 2012, written in C++. It runs on different operating systems and is adapted to low-resource devices. All things connected to the platform interact through MQTT. The ECA rules can be defined using Blockly or through scripts. Supported languages are Lua, Python, shell and a DomoticZ DSL called DzVents. DomoticZ has an E-vehicle framework enabling the integration of

[8] https://code.visualstudio.com/
[9] https://www.jetbrains.com/fr-fr/webstorm/

electrical vehicle systems from, e.g., Tesla, Mercedes or KIA to read sensor data and automate in-car temperature or door locks.

Open-source automation platforms exhibit different architectures and characteristics. Home Assistant and OpenHAB are the most popular solutions with strong interoperability guarantees and active communities. DomoticZ can be a top choice for developpers interested in constrained deployement of an automation solution on the edge. A full comparison of these home-automation platforms is discussed in [145].

6.1.2 Hubitat Home Hub. Hubitat [72] is an automation system introduced by the Hubitat Elevation company in 2018. It is designed with a strong focus on data privacy and security by promoting user control of data and devices. It runs locally without requiring any data transfer and uses industry-compliant encryption protocols to secure communication between home devices and Hubitat Hub.

Hubitat applications are written in the Groovy scripting language, running on top of the JVM. Javascript support is available, enabling custom automation and device integration. Event-driven automation is achieved through a rule machine, a powerful built-in tool offering complex automation scenarios. Events handling considers events issued by devices, hubs, networks or that are time-related. Creating new rules is available for advanced use through the rule machine API or HTTP requests. The rule machine enables complex rule definitions, combining multiple conditions and triggering complex device actions. Beginner end users can set simple actions using the simple automation rule, which provides an user-friendly interface with predefined rules that users can easily configure. On the physical layer, the list of supported devices includes z-wave- and ZigBee-enabled and LAN-based connected devices, and virtual or cloud devices. It does not provide integration for Bluetooth-connected devices, and the hub might be limited in terms of simultaneous integration. In addition to traditional home-automation tasks for lighting, electrical equipment control or sound-based monitoring, the platform dedicates an app for safety monitoring providing intrusion alerts and warnings.

Web and mobile applications provide a grid-based dashboard that users can customize according to the apps they use and the tasks they achieve. The design of pages for UI relies on bundles, which are developed modules containing drivers, apps and libraries for a specific task.

Other proprietary solutions similar to Hubitat exist. We survey the most important ones below.

- HomeSeer [71] is a home-automation platform marketed by Home Seer Technologies since 1999. The company also sells controllers and hubs compatible with its software solution, with variable capabilities and prices, which allow automation routines to run across multiple devices. In addition to its hardware offer, including sensors and actuators, the platform is popular for its extensive panel of plugins (currently around 400), free or not, allowing interoperability with several cloud systems and services and third-party devices and protocols. The Home Seer products are z-wave-plus certified, focusing on lighting automation, door-locking systems, and water management through valves and controllers. In addition, Home Seer stands out for its integration of video surveillance systems and anti-intrusion functionality.
  The platform works locally and autonomously, although it also provides a cloud solution to interface the use of voice assistants and IFTTT services. Pricing plans vary with the number of devices and plugins required for the use case.
- Apple HomeKit [70] is a home-automation system that allows users to automate their homes by controlling compatible devices, virtually defined as accessories. The system is available through an application called Home, running exclusively on Apple operating systems (IOS, iPadOS, macOS). Accessories can be integrated into the smart-home instance by automatically discovering network-connected devices (Wifi, Bluetooth). In addition to

single-device control, the platform allows the definition of complex automation rules, defined as scenes where combined actions on multiple accessories can be triggered if a specific event condition is fulfilled. A set of accessories can be declared as physically belonging to the same room in the home layout. Events are either time-based or sensor-output data. The platform provides a multi-control option where multiple users can share the automation of the home, defining distinct preferences. User's geolocation data are then used to adapt home scenes according to the resident's presence. To guarantee remote access, an Apple device (Apple TV, home pod, iPad,..) must be configured as a homekit hub. The hub also enables integrating additional accessories such as matter-compatible devices [5] or cloud-based services such as the Siri assistant for voice commands.
To enlarge the range of accessories compatible with the platform, the open-source project homebridge [69] is a node.js-based solution that emulates the iOS HomeKit API. It defines a set of plugins to interface third-party devices' APIs with the HomeKit platform.
Developers aiming to integrate new accessories or communicate with HomeKit in their applications might consider the Homekit Accessory Development Kit (ADK) framework [6], using the supported programming languages Objective-C and Swift.

6.2 Smart Agriculture

Agriculture represents one of the application fields where the integration of IoT technologies records the most significant growth. According to [130], the IoT market in agriculture has been valued at USD 12.5 billion in 2021 and is expected to reach 28.56 billion by 2030.

The evolution of the world population demands an increase in agricultural production while paying attention to the ongoing climate and energy crises. These challenges require optimization strategies for improving crop yield and monitoring resources such as water and energy. IoT provides effective solutions to support the automation of several agriculture tasks [94]:

- Precision agriculture for, e.g., irrigation, fertilizer products, greenhouses;
- Environmental management for, e.g., pollution, water reserves, weather conditions;
- Crop monitoring for, e.g., animal management, product quality, soil health;
- Horticulture via soil control or machinery.

The introduction of dedicated IoT platforms for agriculture is still recent [90] compared to other application domains such as smart cities. Emerging projects aim to handle the complexity of the outdoor configuration, deliver (multi) task-oriented decision support tools where farmers can define automation routines, collect data and get analytics to make informed decisions towards performance improvement. We briefly describe below the most relevant current propositions.

6.2.1 Proprietary solutions. CropX is an agronomic farm management platform that leverages mashup principles to enable farm data analytics and decision making [29].

CropX includes a web UI and a mobile application to allow farmers access to a personalized dashboard. The dashboard displays field and environment data recorded by CropX sensors or online cloud services. Data insights cover underground indicators on the soil state, such as temperature or moisture, and additional data analytics on satellite images, topological maps, or meteorological forecasts. Further farm management operational statuses are integrated thanks to machinery APIs, or farmer's feedback.

In addition to data visualization, the farmer can apply machine learning models to her data to obtain custom recommendations about irrigation planning, crop protection against diseases, effluence irrigation, and fertilization task. The pre-trained models support a large set of crops and are designed to increase the crop yield while ensuring water and energy saving. While supporting various crops and soil types, the platform does not allow end users to customize model definitions or service integration.

As the platform is proprietary, its use leads to various underlying costs, the first of which comes from acquiring the hardware set compatible with the software solution. The sensors the company markets transmit real-time soil indicators, including soil moisture, temperature, and electrical conductivity, at variable depths. To target farms with an existing hardware installation and interoperate with additional connected devices as weather stations or rain gauges, CropX sells a telemetry device to provide connectivity to the platform. In addition, access to cloud services and ML models requires a monthly or annual subscription, and additional fees can be charged for in-person technical support services.

Other although similar proprietary solutions exist in the market. We mention the most relevant ones below.

- Azure Farmbeats [168] is a Microsoft system designed to provide precision agriculture features by aggregating agricultural datasets to generate actionable insights. The platform uses intensive ML models to create farm-specific recommendations. Farmbeats is a cloud-based platform with a three-layered architecture: Iot base station, gateway and Azure as a cloud platform. The IoT base station at the system's edge connects and integrates physical device data that interface modules into the gateway. FarmBeats gateway collects data for local computing tasks and some offline services. The cloud part reuses Azure services such as IoT Hub for data-centric engineering.
- John Deere Operations Center[10] is a cloud-based platform provided by the John Deere company, which specialises in the manufacturing of agriculture equipment. The platform offers a set of tools to support farm management, including data collection and integration, data analysis and visualization, and decision support through recommendations and notifications. The platform is also available through a mobile application enabling remote management on several tasks, primarily machine-based: machine operating and maintenance status, data analysis and recommendations for yield improvement, and collaboration features with agronomists or support teams. The platform does not allow for automatic control of machinery or field actuators, except for remote settings updates on the John Deere machines. Although the platform can integrate other brands' equipment, most features are limited to the John Deere ecosystem, restricting its interoperability.
- AgriWebb [3] is a cloud-based platform with web and mobile interfaces dedicated to farm management with a primary focus on livestock management, including cattle and sheep. Livestock management covers animal activity, health and insurance records, chemical and feed inventory (animal treatment, water stocks), and grazing activity management. The platform enables activity planning based on financial forecasts by integrating financial market data to ensure profitability and efficiency. AgriWebb has a GraphQL open API and supports multiple data import and export options and formats. Sensing activity is limited to vendor-specific items through Bluetooth and wifi. The solution is designed for livestock management and offers limited features regarding crop management (only grazing activity).

6.2.2 Open-source solutions. The Smart Water Management Platform (SWAMP) project [155] develops an IoT-based solution for smart irrigation to improve water saving [32, 89]. It pays particular attention to security challenges raised around farmers' data management. The project has four pilots in three countries (Italy, Spain, and Brazil) sharing the

[10]John Deere Operations Center. https://operationscenter.deere.com/, accessed 2023-04-14.

same technical architecture but covering different crops and soil characteristics, with the purpose of reinforcing the irrigation and water distribution models and reaching higher genericity.

The platform is designed as a modular framework, including several components interacting according to the microservice architectural style. The modules are organized vertically following the IoT Computing Continuum [181] in five layers, each corresponding to a processing step. The bottom layer consists of devices' integration and ensuring communications, while the top layer hosts application services, providing two use scenarios accessible through mobile and web applications. Farmers benefit from data insights to make informed decisions regarding their irrigation system, while water distribution actors get recommendations on distribution planning following real-time and forecasted farmers' needs. All soil indicators such as moisture or temperature can be visualized in real time, and farmers can trigger irrigation by controlling probes through the mobile application.

The SWAMP framework is open-source (full source code available [40]) and is built around the FIWARE framework [1], from which it integrates the data-management layer's services. It also reuses the IoT FIWARE security module within its data-acquisition layer, ensuring safe data transmission and storage with reinforced authentication and data-encryption protocols. All processing modules expose RESTful APIs for communication interfacing with the application layer. SWAMP pays particular attention to semantic interoperability. It deploys within the data management layer an RDF triplestore and a SPARQL Event Processing Architecture (SEPA) [141], enabling users to store and query knowledge graphs efficiently. The graphs are defined on top of a SWAMP ontology defined to fit the platform's use cases: irrigation and water distribution.

The platform stands out by offering the ability to request drone flight missions using the mobile application and visualize the path in real time. Drone missions are autonomously launched by the platform.

In addition to the wide range of devices and protocols supported by the platform, its modular architecture with multiple possible combinations of services increases its use complexity and requires considerable effort if the end user intends to extend or modify some features, in particular for users with a limited technical background. The platform is dedicated to irrigation-related tasks, which prevents other use scenarios such as crop-disease control.

Not only deployed in the SWAMP architecture, FIWARE is also the base platform of many other smart agriculture solutions, thanks to its open-source license and the diversity of the supported open standards and tools it offers for IoT application development. The FIWARE project is funded by the European Commission under Horizon 2020 program and considers interoperability within IoT applications as a primary focus. Frequently compared to W3C WoT, a WOT-FIWARE connector has been proposed and illustrated on smart agriculture scenarios in [180]. In [140], a review of smart agriculture solutions built around FIWARE is provided. The following list is enriched with recent literature contributions.

- Farm Management System (FMS) [88] is a cloud-based platform built around FIWARE General Enablers in a multilayer architecture. A local FMS collects data from sensors and machinery to integrate environment and soil indicators and operating status. The collected data are transferred to a cloud FMS for management, processing and application ends, including reporting and visualization. A case study on greenhouse management has been implemented to showcase the collaboration and use scenario. The platform targets multiple end-user groups allowing for collaboration among agriculture-related business domains (farmers, agriculturists, agronomists, ICT experts).
- Cropinfra [128] is a project carried out by MTT Agrifood Research Finland with the purpose of supporting farmers in the management of demanding farm operations. The platform allows end users to interconnect data

and services of fields, machinery and buildings to create a custom operation-management environment. The resulting application enables farmers to get real-time insights on their farming activities and receive warnings of potential risks.

- Testbed [115] presents an implementation of FIWARE in a laboratory to simulate a use scenario of a FIWARE-based platform for precision agriculture. The purpose is to evaluate the ability of the framework to scale to large deployments similar to those needed in the agriculture context. The testbed considers a distributed setting with devices deployed in different farms. Several nodes measure environmental factors such as temperature, humidity, and soil moisture. Results showcase that FIWARE identified relevant agricultural modules, can handle large payloads and provide real-time analysis.
- SME Widhoc [107] is a software solution built on top of FIWARE to store normalized agronomics data as a data warehouse. This common repository provides farmers with generic data to make precise decisions. It focuses on the optimization of irrigation systems to achieve water management efficiency.
- Agricolus [60] is a FIWARE-based IoT platform that enables farmers to increase their crop yield and reduce their environmental impact. It allows for in-soil sensor-based and remote sensing (satellite images) input data to analyze soils, crops and fruit trees' productivity and moisture states, allowing for efficient water management strategies. FIWARE Machine Learning GE analyzes satellite images to detect water-stress symptoms. Agricolus provides irrigation-scheduling recommendations and can be connected to actuators for automatic irrigation triggering.

7 DISCUSSION AND INSIGHTS

In this section, we summarize, in a table, the main elements covered in the previous sections, and discuss some of the insights that one can draw from this survey.

7.1 Summary Table

The presentation of the previously discussed IoT platforms showcases the diversity of their technical characteristics and use modalities. Table 3 provides a summary of IoT platforms characteristics as identified and defined in Section 3. The set of columns displays are defined as follows.

- **Programming-DSL**. It names the exposed DSL language, if proposed by the platform (Section 4). Other platforms relying on Visual Programming Interfaces (VPIs) do not expose a DSL and their corresponding DSL column indicates - (not to be confused with **NA**).
- **Programming-UI**. This indicates whether the user interface leverages textual (**T**) or graphical (**G**) notations. Platforms with graphical interfaces including scripting APIs or textual scripts are identified with as a hybrid **H** UI.
- **Programming-Toolbox**. This column enumerates the types of tools created by the platform to support developers' activities, including libraries, IDEs, plugins and APIs.
- **Programming-Dev Approch**. This refers to the classification of the platform based on its programming approach (Section 4.3), including General Purpose Programming, model-driven programming, Mashups, or end-user programming.

- **Programming-Supp Lang**. This includes the programming languages developers can access to develop applications, extend platforms functionalities and set interoperability options. If the platform uses an internal DSL, this column indicates the target language for code generation.
- **Interoperability**: This feature is evaluated according to 4 dimensions: **D**, for device interoperability, **C**, for connectivity protocols support, **Sx**, for syntactic interoperability, and **S**, for semantic interoperability. Multiple levels of interoperability are being discussed below. The table only indicates whether the platforms leverage tools to widen the spectrum of supported devices, protocols, exchange formats or data models by integrating third-party elements, or if it sticks with a core offer without extra technologies support. To this scope, a dimension interoperability is considered fulfilled if a platform extension is defined and documented.
- **Architectural Patterns**. This lists the paradigms adopted for designing the system and components interaction, as defined in Section 3.2. Options are: Kernel, Blackboard, Orchestrator and peer-to-peer.
- **License**. It reuses the elements of Section 3.3 regarding the defined licensing options to characterise the license category of the platform.
- **Domain**. This last facet distinguishes generic platforms from domain-oriented platforms. Task-oriented platforms are identified by a combination of domain-task mentions.

7.2 Decision Insights

7.2.1 K1: Expressivity and Technical Features. The first decision factor to consider when selecting any software solution is its adaptation to the envisaged use-case requirements. For IoT platforms, adaptability relates to the capacity of the platform to support the tasks of the vertical application domain, the use case at hand and the compatibility with the subsequent technical configuration, if already set by the client organization. Exploring the platform's features enables one to identify to what extent it can be exploited or what adaptation degree is necessary to maximize prequirements coverage. In the smart irrigation use case, platforms exclusively dedicated to building automation are likely outside the scope. Further, agriculture-oriented platforms might sometimes focus on specific tasks such as livestock management with Agriwebb; more generic solutions such as NodeRed present a better fit, considering the extensive extensibility features, documentation and support easing the development of custom automation routines.

Similarly, data-management strategies are relevant to evaluate what insights or automation routines can be provided by the platform. Data-analysis frequency and decision-making process pace do not always conform to the end user's expectation nor the device's accuracy and data transmission rate. At first glance, the decision to irrigate soil can be produced on a daily basis, which differs from the decision about ventilating a room recording unhealthy CO2 peaks, which requires a faster system reactivity.

Technical deployment constraints are an additional aspect impacting the platform choice. Agricultural use cases generally have outdoor deployments covering large areas, thus requiring wide-range communication protocols (WPLAN). Outdoor systems expose devices to harsh environmental conditions, such as wind and snow, leading to unstable connectivity and possible lack of data availability. Automation accuracy in these settings relies on the platform's fault tolerance capacity, including data synchronization mechanisms. In the building automation domain, indoor deployment presents different challenges resulting from the density and complexity of the sensor network deployed in a narrow area, requiring robust scalability.

7.2.2 K2: Learning Curve. From a user-developer perspective, the learning curve of a programming tool indicates the rate at which she acquires proficiency in writing programs over time. Thus, time to proficiency impacts IoT platform

choice. Several factors acting on this curve are related to the platform itself: abstraction level and toolbox, frequency of updates, documentation availability, customization degree and community strength (described later).

- Abstraction Level and Toolbox. As discussed in the background Section 4, classifying a platform as belonging to a programming approach family is hard. Table 3 showcases that most IoT programming platforms provide scripting APIs to enable low-level operations control, which increases expressivity. Depending on use cases, the use of scripting APIs and, consequently, general-purpose languages remain optional if the platform's main programming interface fully covers the user's requirements. This is also valid for plugin definition APIs, service integration, templating or configuration file editing, which might require more familiarization effort.
- System Architecture. The learning curve is generally lower for centric systems with one main programming interface such as kernel systems. Peer-to-peer systems for example exhibit more heterogeneities to consider in the learning time.
- Customization and extensibility: Highly customizable platforms usually correlate with higher expressivity and complexity, leading to steep learning curves. Identifying settings matching use-case requirements demands more effort to understand possible scenarios, and the provided documentation does not cover all customizations. Examples of similar platforms are NodeRed, for the extensive number of nodes in palette and flow-composition options, Home Assistant, relying on YAML-like configuration files, or ThingML, where the DSML supports high customization degrees and relies on multi-language code generation.
- Frequency of updates: Whether the platform is proprietary or open-source, new versions are regularly released, providing new features and enhancements addressing identified issues. This enrichment often affects users' familiarization with the platform services and sometimes involves changes in the programming operation; well-managed platforms carefully review new releases to ensure maximal stability and/or upward compatibility.
- Documentation and Tutorials: The availability of comprehensive learning resources is crucial for lowering the learning curve. High-quality documentation is well structured, introducing platform features and characteristics and supporting users in getting started creating automation programs. Tutorials consist of step-by-step programming examples covering the most popular applications. Many platforms rely on GitHub to host the source code of application examples guiding users through their learning experience.

Other factors impacting the learning curve derive from the complexity of the programs to write. Automation routines involving trivial logical rules for turning off lights in an empty room can be implemented quite quickly. However, scenarios such as window opening may present higher complexity due to the presence of several triggers to be considered and the management of concurrent actions originated by parallel triggering events, duration control, or user-preference integration. Furthermore, automation scenarios where objects communicate with different protocols in an outdoor/indoor system, such as irrigation, require homogenization and data integration mechanisms with query handling. Additional web-available data, such as crop-standard recommendations or timely weather forecasts should be considered in the automation rules for irrigation. The learning curve is expected to follow as the complexity of use case programs increases.

The final factor impacting the learning curve is the human factor; it relates to the user's prerequisites, her skills, and the rate at which she might master the platform's programming interfaces. Naturally, the background impact is more significant for general-purpose languages or DSMLs. The user profile is of utmost importance when formulating platform recommendations.

7.2.3 K3: Community Strength. Multiple metrics can be used to evaluate the strength of the community, with a difference between proprietary and open-source systems. Proprietary platforms communicate about their active user base, such as IFTTT, which indicates 20 million consumers (end of 2021). Open-source platforms usually make their projects available on collaborative software-development platforms like Github. GitHub provides indicators about the project's popularity by displaying the number of stars, forks, watchers or contributing users. Table 1 gives statistics (at the time of writing, mid-2023) about some open-source projects on GitHub.

The number of mobile application downloads might be relevant, especially for end-user programming platforms where mobile applications provide simplified automation routines. The IFTTT Android application has been downloaded over 5 million times. Some systems lead to the development of mobile applications as a secondary UI that does not provide all system features or might be created for remote access, visualization, and alerting purposes. Some of them are fee-based, which may explain their low popularity compared to the platform's success, as for the NodeRed mobile application, namely RedMobile (1000 downloads).

The community strength is an important factor for time-to-master evaluation and the richness of the open-source system features. IoT platforms with kernel architectures rely on multiple plugins (add-ons) to interface with heterogeneous devices and services, generally widely supported and developed by the community.

7.2.4 K4: Interoperability. Interoperability is a crucial aspect when choosing an IoT platform for programming automation applications. First, compliance with the technical constraints of the use case and the expected deployment are decisive factors for the project's viability. The system might also be regularly enriched with new devices operating via different communication protocols. Furthermore, IoT applications are frequently associated and meant to interoperate with other domain-related software solutions such as physical-security management systems, room-reservation or energy-management solutions in building automation, or inventory management and accounting solutions in agriculture.

Platform	Repository	Stars	Forks
Home Assistant	*/home-assistant/core	61 000	24 000
Node-Red	*/node-red/node-red	17 000	4 000
ThingML	*/TelluIoT/ThingML	100	30
Openhab	*/openhab/openhab-core	800	400
Jeedom	*/jeedom/core	374	312

Table 1. Overview of some IoT platforms popularity metrics on Gitub (*/ = https://github.com/)

Model-Driven programming platforms usually generate executable codes in general-purpose languages that can be executed on various environments and, in a general fashion, guarantee easier interoperability through protocols or devices code injection, either at the higher level, using modelling notations, or by directly acting on code generation. Mashups or EUP Platforms often come with a primary core of targeted hardware or software specifications and generally include a set of plugins or gateways to expand the extent of supported devices and protocols or integrate external web services to enrich delivered functionalities. Proprietary platforms sometimes require contacting the company for additional service integration or plugin definition, as for the John Deere operation centre or cropX. Apple Homekit supports interoperability through a proprietary communication protocol, namely HomeKit Accessory Protocol (HAP), which enables making devices compatible with the software solution by acquiring and implementing the protocol. In the case of open-source platforms, the developer community enriches these software solutions using provided templates and tools. Examples of such platforms include Node-Red, which provides Javascript templates to support custom node creation. The availability of this support is an indicator of interoperability concerns, enabling autonomous platform

adaptation to other technologies. However, the number of plugins or gateways is only partially relevant, considering that several plugins can address the same type of device or protocol, leading to redundancy and impacting support coverage.

Another aspect of interoperability involves the existence of interfaces allowing the integration of web services such as IFTTT or Zapier, which define templates for new service integrations through service APIs. However, these last two examples are intended for the service provider, so that this solution is integrated into the platform. Some platforms natively include external IoT systems integrations; Hubitat includes IFTTT or Google Home services, Epidosite API for IFTTT integration,

On the other hand, scripting APIs allow users to program scripts adapting the platform's functionalities or extending them through GPL programming languages, usually Java (ThingML, OpenHab), Python (Home assistant, Wox, Wotkit) or Javascript (NodeRed, IFTTT), but also C/C++ (FRASAD, MIDGAR), Swift (Apple homekit) or Groovy (Hubitat). See Table 2 for some examples.

7.2.5 K5: Cost. The selection of an IoT platform is impacted by its associated costs. Upfront costs of acquiring the platform significantly vary from one vendor to another, depending on their pricing models. In addition, the use of some solutions is constrained by the acquisition of platform-compatible devices. John Deere Operation Center designs its agriculture solutions to fit the John Deere machines, and although interoperability features are proposed, they considerably restrict the scope of applications. The required hardware settings are also often related to hubs acquisition as for Hubitat elevation, which entirely relies on local computations for privacy. For home automation, the Apple HomeKit platform is free for users owning an Apple device that can be employed as a hub (Ipad, homePad or Apple TV), but limited to compatible IoT devices. Devices with the label "works with Apple HomeKit" are considerably more expensive.

Platform	Plugins/ Integrations	Connect API	Extension support
IFTTT	700	Connect API (Javascript)	OpenAPI definition
NodeRed	4400	Module API (Javascript)	Editor API (Nodes) Library store Hooks
OpenHAB	400	REST API (Java)	Core bindings API Scripting APIs (Java, Jython, Groovy)
HomeKit	900	Module API (Swift)	Home Bridge project

Table 2. Number of plugins and APIs of some IoT platforms.

The financial aspects extend to the ongoing operating, support or maintenance costs. These include the costs for extending the platform use cases to additional devices or services. Hub-based platforms are limited to a number of devices, 100 for Apple HomeKit, for example. Scaling up to larger settings could require additional expenses. Freemium platforms offer users a set of basic functionalities and request them to subscribe to monthly or annual billing plans for additional services supporting complex automation, such as IFTTT. As discussed for interoperability, another aspect that might contribute to ongoing fees relates to purchasing plugins or add-ons to enlarge the extent of supported devices and protocols. JEEDOM open-source platform offers a plugin marketplace that provides free and chargeable components. Many hub-based platforms offer cloud-access options for data storage, web-based service integration or remote access, which leads to additional costs. Examples include the Home Assistant cloud. Maintenance and support costs, essential for maintaining platform efficiency and resolving technical issues, are potential ongoing costs. Most

IoT platforms, whether open-source or commercial, provide extensive documentation and publicly available tutorials. However, some offer paid training sessions or certification programs such as Azure Farmbeats or CropX.

A comprehensive cost analysis should be conducted on a case-by-case basis to ensure that the selected IoT platform aligns with the organization's financial capabilities.

Table 3. Summary of IoT programming platforms characteristics.

Tool	Programming					Interoperability	Architecture	Licence	Domain
	DSL	UI	Dev Approach	Toolbox	Supp Lang				
ThingML	UML-like	T	MDD	Eclipse-based IDE Network and Serialization plugins Port, Messages, Things APIs Eclipse SDK	Java Javascript C / C++	D-C-Sx	Peer-to-peer	Open Source	Generic
MIDGAR	MOISL, MOCSL, MUCSL DSLs	G T	MDD	APIs	Java C / C++	D-C-Sx	Orchestrator	Open source	Generic
FRASAD	Rule-DSL	G	MDD	Eclipse IDE plugins	C	D-C	Peer-to-peer	*NA*	Generic
WOX	–	T	MDD	REST APIs Linked Open Data API	Java/Python Android or C/C++	D-C-SX-S	Blackboard	Open Source	Generic
NodeRed	-	G	Mashup	Web visual editor Flow library, APIs Node.JS SDKs	Javascript	D-C-Sx-S	Kernel	Open-Source	Generic
WoTKIT	-	G T	Mashup	REST APIs Sensor gateways	Javascript, Python	D-C-Sx	Blackboard	Proprietary	Generic
IFTTT	-	G T	EUP	integration API	Javascript	D-C-Sx	Orchestrator	Freemium	Generic
Epidosite	–	G	EUP	REST API (IFTTT)	Java	D-C	Kernel	Open Source	Generic
Zapier	-	G	EUP	REST API	Javascript, Python	D-C-Sx	Orchestrator	Proprietary	Generic
Puzzle	-	G	EUP	–	Javascript	D-C	Blackboard	*NA*	Generic
Almond	Thing Talk	G V	EUP	API	Java Javascript	D-C-Sx-S	Kernel	Open Source	Generic
Appsgate	Appsgate DSL	T	EUP	Device adapters P-Openhab middleware	Java	D-C	Blackboard	Open Source	Home Automation
OpenHAB	Rule DSL	G T	EUP	REST APIs Bindings	Java Python Groovy	D-C-Sx-S	Kernel	Open Source	Home automation
Home Assistant	YAML-Based Scription	G T	EUP	Rest APIs Add-ons Web services integrations Webhooks	Python Javascript	D-C-Sx	Kernel	Open Source	Home automation
Hubitat	–	G	EUP	Rule machine API Hub variable API Driver/Library/App bundles Web services integrations	Groovy	D-C-Sx	Kernel	Proprietary	Home automation
CropX	-	G	Mashups	*NA*	*NA*	D-C-Sx	Kernel	Freemium	Agriculture
SWAMP	-	G	EUP	Rest APIs	Javascript	D-C-SX-S	Kernel	Open Source	Agriculture Irrigation

Programming: DSL: the exposed DSL language (see Sec. 4), or only a Visual Programming Interface (-); **UI**: textual (**T**) or graphical (**G**) notations; **Dev Approch**: the programming approach (see Sec. 4.3). model-driven programming (MDD), mashups, or end-user programming (EUP). **Toolbox**: types of tools that support developers' activities; **Supp Lang**: the programming languages developers can access to develop applications, extend platforms functionalities and set interoperability options. If the platform uses an internal DSL, indicates the target language for code generation. **Interoperability**: if extension point is defined and documented for supporting more devices (D), protocols (C), exchange formats (Sx) or data models (S); **Architectural Patterns**: system and components interaction paradigm (see Sec. 3.2); **License**: license of the platform (see Sec. 3.3; **Domain**: Distinguishes generic, domain-oriented, and task-oriented platforms.

8 CONCLUSION

As for other complex systems,building, using, and maintaining IoT systems faces increasing challenges, primarily due to the technological diversity and complexity of system components. On the other hand, the use scenarios involving the Internet of Things and supporting task automation have never been this broad.

Several high-level tools are striving to offer simplified user-interaction interfaces for seamless access to these technologies, preventing users from being overwhelmed with technical stacks wrapping edge, cloud, mobile, or constrained device management, communicating via multiple adapted protocols. On another dimension, the popularity and success of IoT applications also transformed users' profiles, who have recently evolved from simple consumers to developers creating applications through user-friendly interfaces. IoT platforms nowadays attempt to expose interfaces with new languages (Domain-Specific Languages), thought and designed to bridge the gap between the physical layers (cloud infrastructures and protocols, mobile-constrained objects, etc.) and the high-level services supporting the communication between networked objects.

When dealing with IoT systems, general-purpose languages can be too sophisticated, requiring skilled users and much coding time, and include patterns that might be unnecessary for some given domain requirements. They are still used across various platforms but are generally wrapped into intermediate layers or exposed for device-programming utilities. Tools tending to enlarge their user profiles prefer simplified interaction modes supported by domain-specific languages.

In the realm of IoT domain-centric tools, three primary paradigms—model-driven, mashup tools, and end-user programming—mirror established programming models. Model-driven languages primarily cater to business experts possessing field knowledge, facilitating the creation of applications via UML-like models or other VSDML visual languages. These languages are beneficial for IoT systems that intertwine technical system building with vertical domains. The use of simplified modelling tools encourages the involvement of domain experts, enhancing the system's alignment with domain requirements and facilitating autonomy for domain experts who prefer modelling for encoding system behaviour and components.

Mashup tools, on the other hand, harness the plethora of available IoT web services to enable data and service composition. Unlike MDD tools, which focus on systems components, mashup tools are designed to focus on flow messages between components. While the preexistence of services eases system development, it also limits its applicability to available features. The tools enable the composition of various web-available services at the user-interface level, but the extent to which users act on logical behaviours is restricted.

End-user programming as a concept is common to the programming community beyond IoT. It has evolved with technological advances to facilitate higher abstraction levels, leveraging increasingly broader techniques. Historically, model-driven programming and mashup tools were considered EUP tools as they encapsulate technical specifications in high-level component representations. Recently, natural-language processing, programming by example, programming through games or voice-based interactions are employed through various platforms, including computers and mobiles, for simplified automation encoding.

The programming approach adopted by the platform usually affects its expressivity and the expected learning curve for new users. The IoT platform selection process is thus multifaceted and depends on various factors, including systems technical characteristics.

An extensive interest has been dedicated to interoperability within IoT systems in the last few years. Making complex systems interconnect and operate together becomes more challenging with each proposed standard, tool, device, or

language. Interoperability support is tooled with plugins, adapters, gateways and APIs, enabling effective collaboration and broader hardware support. As for complex systems, the architectural patterns also affect the deployment of IoT solutions and their non-functional requirements, such as scalability, performance efficiency or security. The number of system components and the type of interactions provide valuable insights about the system's operating mode, robustness, security or scalability. Platforms are increasingly putting effort into documentation and user communities to increase their popularity and ensure comprehensive and effective use of their platforms.

In summary, the fast-evolving landscape of IoT technologies, in its vast diversity and complexity, calls for an ongoing effort to empower end-users with the knowledge and selection criteria that support them to make informed decisions, thereby making steps towards democratizing technological progress in every domain. We believe this paper is a step in this direction, providing a broad survey of all these issues in a manner accessible to any technically inclined professional interested in entering the IoT field from a business-domain perspective. We provide up to date knowledge and references that should help fast-track the necessary selection, design and implementation steps that IoT deployment requires.

ACKNOWLEDGMENTS

This work was supported by the French national research agency (*Agence nationale de la recherche*) under grant ANR-19-CE23-0012.

REFERENCES

[1] FIWARE Foundation. 2022. FIWARE project. https://www.fiware.org/. Accessed: 2023-04-11.
[2] Charu C. Aggarwal, Naveen Ashish, and Amit Sheth. 2013. The Internet of Things: A Survey from the Data-Centric Perspective. In *Managing and mining sensor data*. Springer, 383–428.
[3] AgriWebb. 2023. Agriwebb. https://www.agriwebb.com/. Accessed: 2023-04-14.
[4] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE communications surveys & tutorials* 17, 4 (2015), 2347–2376.
[5] Connectivity Standards Alliance. 2011. *Matter Specification Version 1.0.* Technical Report. Connectivity Standards Alliance, San Jose, CA, USA. 11 pages. https://csa-iot.org/wp-content/uploads/2022/11/22-27349-001_Matter-1.0-Core-Specification.pdf
[6] Apple. 2023. HomeKit Accessory Development Kit Github. https://github.com/apple/HomeKitADK. Accessed: 2023-04-30.
[7] Kevin Ashton. 2009. That 'Internet of Things' thing. *RFID journal* 22, 7 (2009), 97–114.
[8] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.
[9] Debasis Bandyopadhyay and Jaydip Sen. 2011. Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless personal communications* 58, 1 (2011), 49–69.
[10] Sourav Banerjee, Chinmay Chakraborty, and Sudipta Paul. 2019. Programming Paradigm and the Internet of Things. In *Handbook of IoT and Big Data*. CRC Press, 145–163.
[11] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software* 149 (2019), 101–137.
[12] Barnana Baruah and Subhasish Dhal. 2018. A two-factor authentication scheme against FDM attack in IFTTT based Smart Home System. *Computers & Security* 77 (2018), 21–35.
[13] Len Bass, Paul Clements, and Rick Kazman. 2003. *Software architecture in practice*. Addison-Wesley Professional.
[14] Jon Bentley. 1986. Programming pearls: little languages. *Commun. ACM* 29, 8 (1986), 711–721.
[15] Michael Blackstock and Rodger Lea. 2014. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED)). In *Proceedings of the 5th International Workshop on Web of Things*. 34–39.
[16] Alan F Blackwell. 2002. What is programming?. In *PPIG*. Citeseer, 20.
[17] Arne Bröring, Stefan Schmid, Corina-Kim Schindhelm, Abdelmajid Khelil, Sebastian Käbisch, Denis Kramer, Danh Le Phuoc, Jelena Mitic, Darko Anicic, and Ernest Teniente. 2017. Enabling IoT ecosystems through platform interoperability. *IEEE software* 34, 1 (2017), 54–61.
[18] Paul Brous, Marijn Janssen, and Paulien Herder. 2020. The dual effects of the Internet of Things (IoT): A systematic review of the benefits and risks of IoT adoption by organizations. *International Journal of Information Management* 51 (2020), 101952.
[19] Mihal Brumbulli and Emmanuel Gaudin. 2016. Towards model-driven simulation of the internet of things. In *Complex Systems Design & Management Asia*. Springer, 17–29.

[20] Adriana Caione, Alessandro Fiore, Luca Mainetti, Luigi Manco, and Roberto Vergallo. 2017. WoX: Model-Driven Development of Web of Things Applications. In *Managing the Web of Things*. Elsevier, 357–387.

[21] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S. Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web*. 341–350.

[22] Humberto Cervantes and Rick Kazman. 2016. *Designing software architectures: a practical approach*. Addison-Wesley Professional.

[23] Victor Charpenay, Maxime Lefrançois, María Poveda-Villalón, and Sebastian Käbisch. 2023. *Thing Description (TD) ontology*. W3C internal document. World Wide Web Consortium. https://www.w3.org/2019/wot/td

[24] Xiang 'Anthony' Chen and Yang Li. 2017. Improv: an input framework for improvising cross-device interaction by demonstration. *ACM Transactions on Computer-Human Interaction (TOCHI)* 24, 2 (2017), 1–21.

[25] Bo Cheng, Da Zhu, Shuai Zhao, and Junliang Chen. 2016. Situation-aware IoT service coordination using the event-driven SOA paradigm. *IEEE Transactions on Network and Service Management* 13, 2 (2016), 349–361.

[26] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. 2009. On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos* (2009), 363–379.

[27] Alain Colmerauer. 1990. An introduction to Prolog III. In *Computational Logic*. Springer, 37–79.

[28] Joëlle Coutaz and James L. Crowley. 2016. A first-person experience with end-user development for smart homes. *IEEE Pervasive Computing* 15, 2 (2016).

[29] CropX inc. 2023. CropX. https://cropx.com/. Accessed: 2023-03-28.

[30] Alberto Rodrigues Da Silva. 2015. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures* 43 (2015), 139–155.

[31] Laura Daniele, Raúl García-Castro, Maxime Lefrançois, and María Poveda-Villalón. 2020. *SAREF: the Smart Applications REFerence ontology*. Technical Report TS 103 264, v3.1.1. ETSI. https://saref.etsi.org/core/v3.1.1/

[32] Ramide Augusto Sales Dantas, Milton Vasconcelos da Gama Neto, Ivan Dimitry Zyrianoff, and Carlos Alberto Kamienski. 2020. The swamp farmer app for iot-based smart water status monitoring and irrigation control. In *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. IEEE, 109–113.

[33] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2019. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–29.

[34] Pedro Domingues, Paulo Carreira, Renato Vieira, and Wolfgang Kastner. 2016. Building automation systems: Concepts and technology review. *Computer Standards & Interfaces* 45 (2016), 1–12.

[35] Domoticz. 2023. DomoticZ. https://www.domoticz.com/. Accessed: 2023-04-24.

[36] David L. Donoho. 2006. Compressed Sensing. *IEEE Transactions on Information Theory* 52, 4 (April 2006), 1289–1306.

[37] Eclipse Foundation. 2023. Ecplise Equinox. http://www.eclipse.org/equinox/. Accessed: 2023-04-18.

[38] Inc Eclipse Foundation. 2019. *IoT Developer Survey 2019*. Technical Report. Eclipse Foundation, Inc.

[39] Fatima Essaadi, Yann Ben Maissa, and Mohammed Dahchour. 2016. MDE-based languages for wireless sensor networks modeling: A systematic mapping study. In *International Symposium on Ubiquitous Networking*. Springer, 331–346.

[40] SWAMP Essentials. 2020. SWAMP IOT platform Github Repository. https://git.rnp.br/swamp-essentials/iot-platform.

[41] Teo Eterovic, Enio Kaljic, Dzenana Donko, Adnan Salihbegovic, and Samir Ribic. 2015. An Internet of Things visual domain specific modeling language based on UML. In *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*. IEEE, 1–5.

[42] Dave Evans. 2011. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. Technical Report. Cisco Internet Business Solutions Group (IBSG), Cisco Systems, Inc., San Jose, CA, USA. http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[43] Dave Evans. 2012. *Overview of the Internet of Things*. Technical Report. International Telecommunication Union. 22 pages. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.2060-201206-I!!PDF-E&type=items

[44] Jody Condit Fagan. 2007. Mashing up Multiple Web Feeds Using Yahoo! Pipes. *Computers in Libraries* 27, 10 (2007), 10–17.

[45] Katalin Ferencz and József Domokos. 2019. Using Node-RED platform in an industrial environment. *XXXV. Jubileumi Kandó Konferencia, Budapest* (2019), 52–63.

[46] Alessandro Fiore, Adriana Caione, Luca Mainetti, Luigi Manco, and Roberto Vergallo. 2018. Top-Down Delivery of IoT-based Applications for Seniors Behavior Change Capturing Exploiting a Model-Driven Approach. *Journal of Communications Software and Systems* 14, 1 (2018), 60–67.

[47] Giancarlo Fortino, Claudio Savaglio, Giandomenico Spezzano, and MengChu Zhou. 2020. Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51, 1 (2020), 223–236.

[48] Eclipse Foundation. 2018. Eclipse Thingweb node-wot Repository. https://github.com/eclipse/thingweb.node-wot.

[49] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.

[50] Martin Fowler. 2012. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley.

[51] Iván Froiz-Míguez, Tiago M Fernández-Caramés, Paula Fraga-Lamas, and Luis Castedo. 2018. Design, implementation and practical evaluation of an IoT home automation system for fog computing applications based on MQTT and ZigBee-WiFi sensor nodes. *Sensors* 18, 8 (2018), 2660.

[52] Pavlo Galkin, Lydmila Golovkina, and Igor Klyuchnyk. 2018. Analysis of single-board computers for IoT and IIoT solutions in embedded control systems. In *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*. IEEE, 297–302.

[53] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 3–18.

[54] Cristian González García, B Cristina Pelayo G-Bustelo, Jordán Pascual Espada, and Guillermo Cueva-Fernandez. 2014. Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios. *Computer Networks* 64 (2014), 143–158.

[55] Cristian González García, Daniel Meana-Llorián, Vicente García-Díaz, Andrés Camilo Jiménez, and John Petearson Anzola. 2020. Midgar: Creation of a Graphic Domain-Specific Language to Generate Smart Objects for Internet of Things Scenarios Using Model-Driven Engineering. *IEEE Access* 8 (2020), 141872–141894.

[56] Cristian González García, Liping Zhao, and Vicente García-Díaz. 2019. A user-oriented language for specifying interconnections between heterogeneous objects in the Internet of Things. *IEEE Internet of Things Journal* 6, 2 (2019), 3806–3819.

[57] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor C.M. Leung. 2015. Developing IoT Applications in the Fog: a Distributed Dataflow Approach. In *2015 5th International Conference on the Internet of Things (IOT)*. IEEE, 155–162.

[58] Google. 2023. Google Blockly. https://developers.google.com/blockly?hl=fr. Accessed: 2023-04-25.

[59] Eric Griffin. 2008. *Foundations of Popfly: rapid mashup development*. Apress.

[60] Diego Guidotti, Susanna Marchi, Sara Antognelli, and Andrea Cruciani. 2019. Water management: agricolus tools integration. In *2019 Global IoT Summit (GIoTS)*. IEEE, 1–5.

[61] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. 2011. From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices. Springer, 97–129.

[62] Dominique Guinard, Vlad Trifa, and Erik Wilde. 2010. A resource oriented architecture for the Web of Things. In *2010 Internet of Things (IOT)*. IEEE, 1–8.

[63] Sebastian Günther. 2011. Development of internal domain-specific languages: design principles and design patterns. In *Proceedings of the 18th Conference on Pattern Languages of Programs*. 1–25.

[64] Sandeep Gupta. 2022. Non-functional requirements elicitation for edge computing. *Internet of Things* 18 (2022), 100503.

[65] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. 2015. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal* 3, 5 (2015), 720–734.

[66] Armin Haller, Krzysztof Janowicz, Simon Cox, Danh Le Phuoc, Jamie Taylor, and Maxime Lefrançois. 2017. *Semantic Sensor Network Ontology, W3C Recommendation*. W3C Recommendation. World Wide Web Consortium. https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/

[67] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. ThingML: a language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems*. 125–135.

[68] Home Assistant. 2023. Home Assistant. https://www.home-assistant.io/. Accessed: 2023-04-24.

[69] Home bridge. 2023. Home Bridge open-source project. https://github.com/homebridge/homebridge. Accessed: 2023-04-30.

[70] HomeKit. 2023. Apple HomeKit Home Automation. https://developer.apple.com/apple-home/. Accessed: 2023-04-24.

[71] HomeSeer. 2023. HomeSeer Home Automation. https://homeseer.com/. Accessed: 2023-04-24.

[72] Hubitat Elevation. 2023. Hubitat Home Automation. https://hubitat.com/. Accessed: 2023-04-24.

[73] Paul Hudak. 1996. Building domain-specific embedded languages. *ACM Computing Surveys (CSUR)* 28, 4es (1996), 196–es.

[74] Fatima Hussain. 2017. Internet of Everything. In *Internet of things*. Springer, 1–11.

[75] Randall Hyde. 2003. *The art of assembly language*. Vol. 75. No Starch Press San Francisco, CA, USA.

[76] Jorge E Ibarra-Esquer, Félix F González-Navarro, Brenda L Flores-Rios, Larysa Burtseva, and María A Astorga-Vargas. 2017. Tracking the evolution of the Internet of Things concept across different application domains. *Sensors* 17, 6 (2017), 1379.

[77] IFTTT [n. d.]. IFTTT: Every thing works better together. https://ifttt.com/. Accessed: 2022-07-13.

[78] Janggwan Im, Seonghoon Kim, and Daeyoung Kim. 2013. IoT mashup as a service: cloud-based mashup service for the Internet of things. In *2013 IEEE international conference on services computing*. IEEE, 462–469.

[79] IOBrocker. 2023. IOBrocker. https://www.iobroker.net/. Accessed: 2023-04-24.

[80] IoT Analytics. 2020. IoT Platforms Landscape. https://iot-analytics.com/product/iot-platforms-landscape-database-2020. Accessed: 2022-06-09.

[81] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. 2015. The internet of things for health care: a comprehensive survey. *IEEE access* 3 (2015), 678–708.

[82] ISO/IEC 25010:2011 2011. *Systems and software Quality Requirements and Evaluation (SQuaRE)* . Standard. International Organization for Standardization, Geneva, CH.

[83] ISO/TC 211. 2011. *Geographic information — Encoding*. Standard. International Organization for Standardization, Geneva, CH.

[84] Krzysztof Janowicz, Armin Haller, Simon JD Cox, Danh Le Phuoc, and Maxime Lefrançois. 2019. SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics* 56 (2019), 1–10.

[85] JEEDOM SAS. 2023. Jeedom website. https://www.jeedom.com/fr/. Accessed: 2023-04-18.

[86] Youngmin Ji, Kisu Ok, and Woo Suk Choi. 2018. Web of things based IoT standard interworking test case: Demo abstract. In *Proceedings of the 5th Conference on Systems for Built Environments*. 182–183.

[87] Mengda Jia, Ali Komeily, Yueren Wang, and Ravi S Srinivasan. 2019. Adopting Internet of Things for the development of smart buildings: A review of enabling technologies and applications. *Automation in Construction* 101 (2019), 111–126.

[88] Alexandros Kaloxylos, Aggelos Groumas, Vassilis Sarris, Lampros Katsikas, Panagis Magdalinos, Eleni Antoniou, Zoi Politopoulou, Sjaak Wolfert, Christopher Brewster, Robert Eigenmann, et al. 2014. A cloud-based Farm Management System: Architecture and implementation. *Computers and electronics in agriculture* 100 (2014), 168–179.

[89] Carlos Kamienski, Juha-Pekka Soininen, Markus Taumberger, Ramide Dantas, Attilio Toscano, Tullio Salmon Cinotti, Rodrigo Filev Maia, and André Torre Neto. 2019. Smart water management platform: IoT-based precision irrigation for agriculture. *Sensors* 19, 2 (2019), 276.

[90] Andreas Kamilaris, Feng Gao, Francesc X Prenafeta-Boldu, and Muhammad Intizar Ali. 2016. Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE, 442–447.

[91] Ashwin Karale. 2021. The challenges of IoT addressing security, ethics, privacy, and laws. *Internet of Things* 15 (2021), 100420.

[92] M. Karthikeyan, T.S. Subashini, and M.S. Prashanth. 2020. Implementation of Home Automation Using Voice Commands. In *Data Engineering and Communication Technology*. Springer, 155–162.

[93] Ala' Khalifeh, Felix Mazunga, Action Nechibvute, and Benny Munyaradzi Nyambo. 2022. Microcontroller Unit-Based Wireless Sensor Network Nodes: A Review. *Sensors* 22, 22 (2022), 8937.

[94] Sabrine Khriji, Dhouha El Houssaini, Ines Kammoun, and Olfa Kanoun. 2021. Precision irrigation: an IoT-enabled wireless sensor network for smart irrigation systems. In *Women in precision agriculture*. Springer, 107–129.

[95] Tae-Kook Kim. 2020. Short research on voice control system based on artificial intelligence assistant. In *2020 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 1–2.

[96] Robert Kleinfeld, Stephan Steglich, Lukasz Radziwonowicz, and Charalampos Doukas. 2014. glue.things: a Mashup Platform for wiring the Internet of Things with the Internet of Services. In *Proceedings of the 5th International Workshop on Web of Things*. 16–21.

[97] Ravi Kishore Kodali, Sasweth C Rajanarayanan, Lakshmi Boppana, Samradh Sharma, and Ankit Kumar. 2019. Low cost smart home automation system using smart phone. In *2019 IEEE R10 Humanitarian Technology Conference (R10-HTC)(47129)*. IEEE, 120–125.

[98] Ege Korkan, Fady Salama, Sebastian Kaebisch, and Sebastian Steinhorst. 2021. A-MaGe: Atomic Mashup Generator for the Web of Things. In *Web Engineering - 21st International Conference, ICWE 2021, Biarritz, France, May 18-21, 2021, Proceedings*. Springer, 320–327.

[99] Tomaž Kosar, Marjan Mernik, and Jeffrey C. Carver. 2012. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical software engineering* 17, 3 (2012), 276–304.

[100] Agnes Koschmider, Victoria Torres, and Vicente Pelechano. 2009. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW*. 1–9.

[101] Ajay Krishna, Michel Le Pallec, Alejandro Martinez, Radu Mateescu, and Gwen Salaün. 2020. MOZART: design and deployment of advanced IoT applications. In *Companion proceedings of the web conference 2020*. 163–166.

[102] S.V. Aswin Kumer, P. Kanakaraja, A. Punya Teja, T. Harini Sree, and T. Tejaswni. 2021. Smart home automation using IFTTT and google assistant. *Materials Today: Proceedings* 46 (2021), 4070–4076.

[103] Danh Le-Phuoc, Axel Polleres, Giovanni Tummarello, and Christian Morbidoni. 2008. DERI Pipes: visual tool for wiring Web data sources. *Book DERI pipes: visual tool for wiring web data sources* (2008).

[104] Toby Jia-Jun Li, Yuanchun Li, Fanglin Chen, and Brad A. Myers. 2017. Programming IoT devices by demonstration using mobile apps. In *International Symposium on End User Development*. Springer, 3–17.

[105] Wenbin Li, Giuseppe Tropea, Ahmed Abid, Andrea Detti, and Franck Le Gall. 2019. Review of Standard Ontologies for the Web of Things. In *2019 Global IoT Summit (GIoTS)*. IEEE, 1–6.

[106] Steve Liang, Chih-Yuan Huang, and Tania Khalafbeigi. 2016. *OGC SensorThings API Part 1: Sensing, Version 1.0*. Standard OGC 15-078r6. Open Geospatial Consortium. http://dx.doi.org/10.25607/OBP-455

[107] JA López-Riquelme, N Pavón-Pulido, H Navarro-Hellín, F Soto-Valles, and R Torres-Sánchez. 2017. A software architecture based on FIWARE cloud for Precision Agriculture. *Agricultural water management* 183 (2017), 123–135.

[108] Mahmoud Shuker Mahmoud and Auday AH Mohamad. 2016. A study of efficient power consumption wireless communication techniques/modules for internet of things (IoT) applications. *Advances in Internet of Things* 6, 2 (2016), 19–29.

[109] Luca Mainetti, Luigi Manco, Luigi Patrono, Andrea Secco, Ilaria Sergi, and Roberto Vergallo. 2016. An ambient assisted living system for elderly assistance applications. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications*. IEEE, 1–6.

[110] Luca Mainetti, Luigi Manco, Luigi Patrono, Ilaria Sergi, and Roberto Vergallo. 2015. Web of Topics: An IoT-aware model-driven designing approach. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 46–51.

[111] Luca Mainetti, Vincenzo Mighali, Luigi Patrono, Piercosimo Rametta, and Silvio Lucio Oliva. 2013. A novel architecture enabling the visual implementation of Web of Things applications. In *2013 21st International Conference on Software, Telecommunications and Computer Networks-(SoftCOM 2013)*. IEEE, 1–7.

[112] Luca Mainetti, Paolo Panarese, and Roberto Vergallo. 2022. WoX+: A Meta-Model-Driven Approach to Mine User Habits and Provide Continuous Authentication in the Smart City. *Sensors* 22, 18 (2022), 6980.

[113] Ivano Malavolta and Henry Muccini. 2014. A study on MDE approaches for engineering wireless sensor networks. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 149–157.

[114] Marco Manca, Parvaneh Parvin, Fabio Paternò, and Carmen Santoro. 2020. Integrating Alexa in a Rule-based Personalization Platform. In *Proceedings of the 6th EAI International Conference on Smart Objects and Technologies for Social Good*. 108–113.

[115] Ramón Martínez, Juan Ángel Pastor, Bárbara Álvarez, and Andrés Iborra. 2016. A testbed to evaluate the fiware-based IoT platform in the domain of precision agriculture. *Sensors* 16, 11 (2016), 1979.

[116] Tom Mens and Pieter Van Gorp. 2006. A taxonomy of model transformation. *Electronic notes in theoretical computer science* 152 (2006), 125–142.

[117] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An empirical characterization of IFTTT: ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*. 398–404.

[118] José P Miguel, David Mauricio, and Glen Rodríguez. 2014. A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977* (2014).

[119] Armin Moin, Stephan Rössler, Marouane Sayih, and Stephan Günnemann. 2020. From things' modeling language (ThingML) to things' machine learning (ThingML2). In *Proceedings of the 23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion Proc.* 1–2.

[120] MSFlow [n. d.]. Microsoft Flow. https://flow.microsoft.com/en-us/. Accessed: 2022-07-13.

[121] Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki, and Kurt Geihs. 2015. FRASAD: A framework for model-driven IoT Application Development. In *2015 IEEE 2nd world forum on internet of things (WF-IoT)*. IEEE, 387–392.

[122] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. 2019. Interoperability in Internet of Things: Taxonomies and Open Challenges. *Mobile networks and applications* 24, 3 (2019), 796–809.

[123] Nuno Oliveira, Maria João Pereira, Pedro Rangel Henriques, and Daniela Cruz. 2009. Domain specific languages: A theoretical survey. *INForum'09-Simpósio de Informática* (2009).

[124] OpenHAB. 2023. OpenHAB community. https://community.openhab.org/t/. Accessed: 2023-04-18.

[125] OpenJS Foundation. [n. d.]. Node-RED: Low-code Programming for Event-driven Applications. https://nodered.org/. Accessed: 2022-07-12.

[126] Fabio Paternò and Carmen Santoro. 2017. A Design Space for End User Development in the Time of the Internet of Things. In *New perspectives in end-user development*. Springer, 43–59.

[127] Fabio Paternò and Carmen Santoro. 2019. End-user development for personalizing applications, things, and robots. *International Journal of Human-Computer Studies* 131 (2019), 120–130.

[128] Liisa A Pesonen, Frederick K-W Teye, Ari K Ronkainen, Markku O Koistinen, Jere J Kaivosoja, Pasi F Suomi, and Raimo O Linkolehto. 2014. Cropinfra–An Internet-based service infrastructure to support crop production in future farms. *Biosystems engineering* 120 (2014), 92–101.

[129] Antonio Pintus, Davide Carboni, and Andrea Piras. 2012. Paraimpu: A platform for a social Web of Things. In *Proceedings of the 21st International Conference on World Wide Web*. 401–404.

[130] Precedence Research. 2022. Internet of Things (IoT) in Agriculture Market. https://www.precedenceresearch.com/iot-in-agriculture-market. Accessed: 2023-03-28.

[131] Christian Prehofer and Luca Chiarabini. 2013. From IoT mashups to model-based IoT. In *W3C Workshop on the Web of Things*. 62.

[132] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2017. IFTTT vs. Zapier: A comparative study of trigger-action programming frameworks. *arXiv preprint arXiv:1709.02788* (2017).

[133] Anoja Rajalakshmi and Hamid Shahnasser. 2017. Internet of Things using Node-Red and Alexa. In *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*. IEEE, 1–4.

[134] Bharti Rana, Yashwant Singh, and Pradeep Kumar Singh. 2021. A systematic survey on internet of things: Energy efficiency and interoperability perspective. *Transactions on Emerging Telecommunications Technologies* 32, 8 (2021), e4166.

[135] Partha Pratim Ray. 2017. Internet of things for smart agriculture: Technologies, practices and future direction. *Journal of Ambient Intelligence and Smart Environments* 9, 4 (2017), 395–420.

[136] Partha Pratim Ray. 2018. A survey on Internet of Things architectures. *J. of King Saud U.-Computer and Information Sciences* 30, 3 (2018), 291–319.

[137] Tim Rentsch. 1982. Object oriented programming. *ACM Sigplan Notices* 17, 9 (1982), 51–57.

[138] Larissa C Rocha, Rossana MC Andrade, Andreia L Sampaio, and Valéria Lelli. 2017. Heuristics to evaluate the usability of ubiquitous systems. In *Distributed, Ambient and Pervasive Interactions: 5th International Conference, DAPI 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings 5*. Springer, 120–141.

[139] A. Wendell O. Rodrigues, Frédéric Guyomarc'h, and Jean-Luc Dekeyser. 2012. An MDE Approach for Automatic Code Generation from UML/MARTE to OpenCL. *Computing in Science & Engineering* 15, 1 (2012), 46–55.

[140] Maria Angeles Rodriguez, Llanos Cuenca, and Angel Ortiz. 2018. FIWARE open source standard platform in smart farming-a review. In *Collaborative Networks of Cognitive Systems: 19th IFIP WG 5.5 Working Conf.on Virtual Enterprises, Cardiff, UK, September 17-19, 2018, Proc. 19*. Springer, 581–589.

[141] Luca Roffia, Paolo Azzoni, Cristiano Aguzzi, Fabio Viola, Francesco Antoniazzi, and Tullio Salmon Cinotti. 2018. Dynamic linked data: A SPARQL event processing architecture. *Future Internet* 10, 4 (2018), 36.

[142] Challouf Sabri, Lobna Kriaa, and Saidane Leila Azzouz. 2017. Comparison of IoT constrained devices operating systems: A survey. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 369–375.

[143] Audrey Sanctorum, Suzanne Kieffer, and Beat Signer. 2020. User-driven Design Guidelines for the Authoring of Cross-Device and Internet of Things Applications. In *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*. 1–12.

[144] Douglas C. Schmidt. 2006. Guest editor's introduction: Model-driven engineering. *Computer* 39, 02 (2006), 25–31.

[145] Brian Setz, Sebastian Graef, Desislava Ivanova, Alexander Tiessen, and Marco Aiello. 2021. A comparison of open-source home automation systems. *IEEE Access* 9 (2021), 167332–167352.

[146] Nicolas Seydoux, Khalil Drira, Nathalie Hernandez, and Thierry Monteil. 2016. IoT-O, a core-domain IoT ontology to represent connected devices networks. In *Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings 20*. Springer, 561–576.

[147] Sajjad Hussain Shah and Ilyas Yaqoob. 2016. A survey: Internet of Things (IOT) technologies, applications and challenges. In *2016 IEEE Smart Energy Grid Engineering (SEGE)*. IEEE, 381–385.

[148] Sabrina Sicari, Alessandra Rizzardi, and Alberto Coen-Porisini. 2019. Smart transport and logistics: A Node-RED implementation. *Internet Technology Letters* 2, 2 (2019), e88.

[149] Kiran Jot Singh and Divneet Singh Kapoor. 2017. Create your own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine* 6, 2 (2017), 57–68.

[150] William Stallings. 1987. *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards*. Macmillan Publishing Co., Inc.

[151] Stanford. 2023. Almond Github Repository. https://github.com/stanford-oval/almond-gnome. Accessed: 2023-04-30.

[152] Mark Strembeck and Uwe Zdun. 2009. An approach for the systematic development of domain-specific languages. *Software: Practice and Experience* 39, 15 (2009), 1253–1292.

[153] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. 1998. Knowledge Engineering: Principles and Methods. *Data and Knowledge Eng.* 25, 1-2 (1998), 161–197.

[154] Xiang Su, Hao Zhang, Jukka Riekki, Ari Keränen, Jukka K. Nurminen, and Libin Du. 2014. Connecting IoT sensors to knowledge-based systems by transforming SenML to RDF. *Procedia Computer Science* 32 (2014), 215–222.

[155] Swamp Project. 2023. SMART WATER MANAGEMENT PLATFORM. http://swamp-project.org/. Accessed: 2023-04-11.

[156] Ioan Szilagyi and Patrice Wira. 2016. Ontologies and Semantic Web for the Internet of Things - a survey. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 6949–6954.

[157] Antero Taivalsaari and Tommi Mikkonen. 2018. On the development of IoT systems. In *2018 Third Int. Conf. on Fog and Mobile Edge Computing (FMEC)*. IEEE, 13–19.

[158] Richard N. Taylor, Nenad Medvidovic, and Dashofy Eric M. 2010. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, Inc.

[159] Daniel Tetteroo, Panos Markopoulos, Stefano Valtolina, Fabio Paternò, Volkmar Pipek, and Margaret Burnett. 2015. End-User Development in the Internet of Things Era. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. 2405–2408.

[160] Kleanthis Thramboulidis and Foivos Christoulakis. 2016. UML4IoT—A UML-based approach to exploit IoT in cyber-physical manufacturing systems. *Computers in Industry* 82 (2016), 259–272.

[161] TIOBE [n. d.]. TIOBE Programming Languages Index. https://www.tiobe.com/tiobe-index/. Accessed: 2022-06-24.

[162] Eben Upton and Gareth Halfacree. 2014. *Raspberry Pi user guide*. John Wiley & Sons.

[163] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. 2016. Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 3227–3231.

[164] Arie Van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices* 35, 6 (2000).

[165] Peter Van Roy. 2009. Programming Paradigms for Dummies: What Every Programmer Should Know. In *New computational paradigms for computer music*. IRCAM/Delatour France, 9–47.

[166] Peter Van Roy and Seif Haridi. 2004. *Concepts, techniques, and models of computer programming*. MIT press.

[167] Sander Vanden Hautte, Pieter Moens, Joachim Van Herwegen, Dieter De Paepe, Bram Steenwinckel, Stijn Verstichel, Femke Ongenae, and Sofie Van Hoecke. 2020. A dynamic dashboarding application for fleet monitoring using semantic web of things technologies. *Sensors* 20, 4 (2020), 1152.

[168] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta N Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. 2017. Farmbeats: An IoT platform for data-driven agriculture.. In *NSDI*, Vol. 17. 515–529.

[169] Molugu Surya Virat, SM Bindu, B Aishwarya, BN Dhanush, and Maniunath R Kounte. 2018. Security and privacy challenges in internet of things. In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 454–460.

[170] Supachai Vorapojpisut. 2015. A Lightweight Framework of Home Automation Systems Based on the IFTTT Model. *J. Softw.* 10, 12 (2015).

[171] Philip Wadler. 1992. The essence of functional programming. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1–14.

[172] Zhibo Wang, Defang Liu, Yunan Sun, Xiaoyi Pang, Peng Sun, Feng Lin, John CS Lui, and Kui Ren. 2022. A Survey on IoT-enabled Home Automation Systems: Attacks and Defenses. *IEEE Communications Surveys & Tutorials* (2022).

[173] Peter Wegner. 1996. Interoperability. *ACM Computing Surveys (CSUR)* 28, 1 (1996), 285–287.

[174] Jon Whittle, Tony Clark, and Thomas Kühne. 2011. Model driven engineering languages and systems. In *14th Int. Conf., models*. Springer, 16–21.

[175] Hansong Xu, Wei Yu, David Griffith, and Nada Golmie. 2018. A survey on industrial Internet of Things: A cyber-physical systems perspective. *Ieee access* 6 (2018), 78238–78259.

[176] Hikmat Yar, Ali Shariq Imran, Zulfiqar Ahmad Khan, Muhammad Sajjad, and Zenun Kastrati. 2021. Towards smart home automation using IoT-enabled edge-computing paradigm. *Sensors* 21, 14 (2021), 4932.
[177] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. 2008. Wireless sensor network survey. *Computer networks* 52, 12 (2008), 2292–2330.
[178] Zapier [n. d.]. Zapier. https://zapier.com/. Accessed: 2022-07-13.
[179] Wei-Zhe Zhang, Ibrahim A. Elgendy, Mohamed Hammad, Abdullah M. Iliyasu, Xiaojiang Du, Mohsen Guizani, and Ahmed A. Abd El-Latif. 2020. Secure and optimized load balancing for multitier IoT and edge-cloud computing systems. *IEEE Internet of Things Journal* 8, 10 (2020), 8119–8132.
[180] Ivan Zyrianoff, Alexandre Heideker, Luca Sciullo, Carlos Kamienski, and Marco Di Felice. 2021. Interoperability in open IoT platforms: WoT-FIWARE comparison and integration. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 169–174.
[181] Ivan Zyrianoff, Alexandre Heideker, Dener Silva, João Kleinschmidt, Juha-Pekka Soininen, Tullio Salmon Cinotti, and Carlos Kamienski. 2019. Architecting and deploying IoT smart applications: A performance–oriented approach. *Sensors* 20, 1 (2019), 84.

Received 12 July 2023