Towards Seamless IoT Device-Edge-Cloud Continuum: Software Architecture Options of IoT Devices Revisited

Antero Taivalsaari[1,2], Tommi Mikkonen[3,4], Cesare Pautasso[5]

[1]Nokia Bell Labs, Tampere, Finland, [2]Tampere University, Tampere, Finland
[3]University of Helsinki, Helsinki, Finland, [4]University of Jyväskylä, Jyväskylä, Finland
[5]Università della Svizzera Italiana (USI), Lugano, Switzerland
`antero.taivalsaari@nokia-bell-labs.com`, `tommi.j.mikkonen@jyu.fi`, `cesare.pautasso@usi.ch`

Abstract. In this paper we revisit a taxonomy of client-side IoT software architectures that we presented a few years ago. We note that the emergence of inexpensive AI/ML hardware and new communication technologies are broadening the architectural options for IoT devices even further. These options can have a significant impact on the overall end-to-end architecture and topology of IoT systems, e.g., in determining how much computation can be performed on the edge of the network. We study the implications of the IoT device architecture choices in light of the new observations, as well as make some new predictions about future directions. Additionally, we make a case for isomorphic IoT systems in which development complexity is alleviated with consistent use of technologies across the entire stack, providing a seamless continuum from edge devices all the way to the cloud.

Keywords: Internet of Things, IoT, Embedded Devices, Programmable World, Software Architecture, Software Engineering, Edge Computing, Isomorphism, Isomorphic Software, Liquid Software

1 Introduction

The Internet of Things (IoT) represents the next significant step in the evolution of the Internet. The emergence of the Internet of Things will bring us connected devices that are an integral part of the physical world. We believe that this evolution will ultimately result in the creation of a *Programmable World* in which even the simplest and most ordinary everyday things and artifacts in our surroundings are connected to the Internet and can be accessed and programmed remotely. The possibility to connect, manage, configure and dynamically reprogram remote devices through local, edge and global cloud environments will open up a broad variety of new use cases, services, applications and device categories, and enable entirely new product and application ecosystems [1, 2].

From economic perspective, the Internet of Things represents one of the most significant growth opportunities for the IT industry in the coming years. According to Fortune Business Insights, the global Internet of Things market size stood at USD 250.72 billion in 2019, and is projected to reach USD 1463.19 billion (nearly \$1.5 trillion) by 2027, exhibiting a CAGR of 24.9% during the forecast period[1].

At the technical level, the Internet of Things is all about *turning physical objects and everyday things into digital data products and services* – bringing new value and intelligence to previously lifeless things. Effectively this means adding computing capabilities and cloud connectivity to hitherto unconnected devices, as well as adding backend services and web and/or mobile applications for viewing and analyzing data and controlling those devices in order to bring new value and convenience to the users. Given the integrated, connected nature of the devices, applications and cloud, IoT systems are *end-to-end* (E2E) systems in which the visible parts – the devices and the apps – are only a small part of the overall solution.

In our earlier work, we have pointed out that a common, generic end-to-end (E2E) architecture for IoT systems has already emerged. Furthermore, we have also identified relevant research topics and problems associated with IoT development [3–5]. In this paper we will revisit those topics, and analyze the software architecture options for IoT devices in view of new occurrences in the industry in the past four years. We study the implications of the IoT device architecture choices in light of the new occurrences, as well as make some new predictions about future directions. More specifically, we make a case for isomorphic IoT systems in which development complexity is alleviated with consistent use of technologies across the entire end-to-end system, spanning from IoT/mobile devices on the edge all the way to the cloud. The paper is based on the authors' experiences in a number of industrial and academic IoT development projects carried out in the past ten years, as well as countless discussions with our colleagues and acquaintances in the academia and in the industry.

The structure of this paper is as follows. In Section 2, we start the paper by discussing the generic end-to-end IoT system architecture that serves as the backdrop for the rest of the paper. In Section 3, we examine the basic software architecture options for IoT devices. In Section 4, we focus on the emergence of inexpensive AI/ML hardware, which is bringing significant changes in the overall IoT system architecture by enabling much more computationally intensive AI/ML capabilities at the edge of the IoT systems. In Section 5, we make a case for isomorphic IoT software technologies that will be crucial in driving the industry towards a more seamless device-edge-cloud technology continuum. We discuss the implications of these trends briefly in Section 6. Finally, in Section 7 we draw some conclusions.

[1] `https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307`

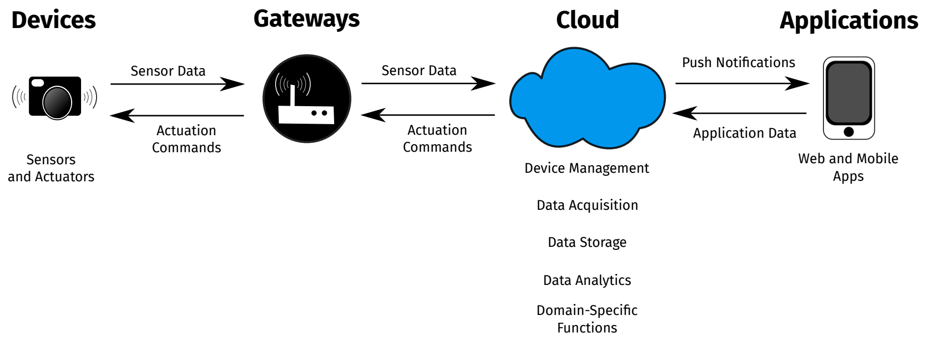

Fig. 1. Conventional Generic End-to-End (E2E) IoT Architecture.

2 Background

Given the connected nature of smart things and the need for a backend service, IoT systems are *end-to-end* (E2E) systems that consist of a number of high-level architectural elements that tend to be pretty much identical in all IoT solutions. In our 2018 IEEE Software article, we pointed out that a common, generic end-to-end (E2E) architecture for IoT systems has already emerged [3].

As depicted in Fig. 1, IoT systems generally consist of Devices, Gateways, Cloud and Applications. *Devices* are the physical hardware elements that collect sensor data and may perform actuation. *Gateways* collect, preprocess and transfer sensor data from devices, and may deliver actuation requests from the cloud to devices. *Cloud* has a number of important roles, including data acquisition, data storage and query support, real-time and/or offline data analytics, device management and device actuation control. *Applications* range from simple web-based data visualization dashboards to highly domain-specific web and mobile apps. Furthermore, some kind of an administrative web user interface is typically needed, e.g., for managing usage rights and permissions. Granted, IoT product offerings have their differentiating features and services as well, but the overall architecture typically follows the high-level model shown in Fig. 1.

Given the relatively uniform nature of the end-to-end IoT systems, it is not surprising that a large number of IoT platforms have emerged. According to IoT Analytics, in 2020 the number of known IoT platforms was 620[2]. In addition, there are a lot of company-specific IoT platform implementations that are less widely known.

Historically, IoT systems were very *cloud-centric* (Fig. 2) in the sense that nearly all the computation was performed in the cloud in a centralized fashion [6]. In contrast, the role of devices and gateways was limited mainly to sensor data aggregation, acquisition and actuation. However, as more computing power and storage has become available on the edge (devices and gateways), the more

[2] `https://iot-analytics.com/iot-platform-companies-landscape-2020/`

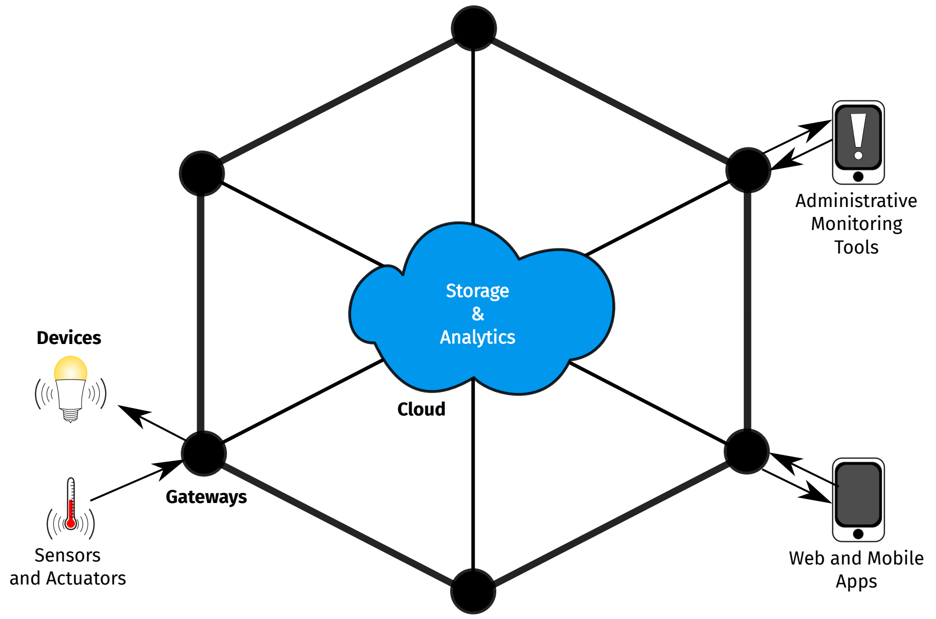

Fig. 2. Cloud-centric IoT Architecture.

realistic it has become to perform significant computation also in IoT devices and gateways.

The recent development of the Internet of Things paradigm is enabled primarily by advances in hardware development. Hardware evolution has led to widespread availability of increasingly capable and power-efficient low-cost computing chips and development boards. These boards easily match or even exceed the memory and processing power capabilities that mobile phones or even PCs had in the late 1990s, and far surpass the processing capabilities of the early 8-bit and 16-bit personal computers from the early 1980s.

Examples of low-cost IoT development boards include the 10€ Arduino Nano Every (`https://store.arduino.cc/arduino-nano-every`) and the $4 Raspberry Pi Pico (`https://www.raspberrypi.org/products/raspberry-pi-pico/`). In spite of its low price, the latter device has computing capabilities that exceed those of the once dominant Intel 80386 and 80486 based personal computers in the late 1980s and early 1990s. Some of the more recently introduced devices, such as the popular $89 *Raspberry Pi 4B* and the $109 all-in-one keyboard-integrated Raspberry Pi 400 device have a quad-core ARM Cortex-A72 processor running at 1.5 GHz and 1.8 GHz, respectively – and are thus more powerful than many laptop computers only some 15 years ago.

The widespread availability of inexpensive IoT chips has made it possible to embed intelligence and Internet communication capabilities in virtually all everyday devices. Projecting five to ten years ahead, everyday objects in our surroundings such as coffee brewing machines, refrigerators, sauna stoves and door locks might have more computing power, storage capacity and network bandwidth than computers that were used for running entire computing departments in the 1970s and 1980s.

Nowadays, 16-bit microcontroller-based IoT devices are less common, as the market is increasingly dominated by 32-bit solutions. In that sense the evolution of IoT devices has been even faster than the evolution of the PC and mobile phone markets a few decades earlier. Except for Narrowband IoT (NB-IoT) cellular connectivity (discussed later in the paper), network speeds supported by today's IoT devices are also significantly faster than those available for personal computers or mobile phones in their early history.

Finally, IoT key design choices are often connected to energy consumption. In practice, one of the most significant differentiating feature driving or even dictating the selection of the software architecture in the majority of IoT devices is the *battery*. Battery-operated IoT devices typically have strict minimum operating time requirements. For instance, a smartwatch should usually be capable of operating a full day without recharging. A safety tracker bracelet for elderly people should ideally operate at least 4-7 days between charges. An air quality sensor installed in a remote forest might need to operate several months without recharging. Such requirements mean that the most energy-consuming components of the device such as the CPU, display and the radio modem will have to be chosen (and used) carefully to meet the power requirements. The form factor characteristics of the device (e.g., wearability and design aspects) play a significant role in determining the right tradeoffs, hence impacting also the type of software architecture that the device can support.

3 IoT Devices – Basic Software Architecture Options

Based on our experiences with various industrial IoT development efforts, the software architecture choices for IoT client devices can be classified as follows, ranging from simple to more complex architectures (Fig. 3 and Table 1):

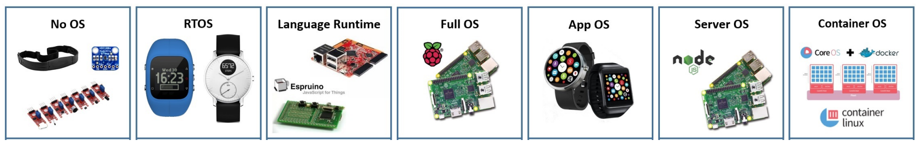

Fig. 3. Example IoT Devices for Each Architecture Choice.

1. *No OS architecture*: for simplest sensing devices that do not need any operating system at all. Software is written specifically for the device, and software development is typically carried in-house. Hence, there is no need for third-party developer support, and support for firmware updates may be limited

Table 1. Basic Hardware Configurations for IoT Devices

Architecture Choice	**Typical Device**	**Hardware**	**OS**	**RAM**	**Expected Battery Duration**
No OS	Simple sensor devices (e.g., heartbeat sensor)	Low-end microcontrollers	None; basic drivers only	up to 10 kB	weeks-months
RTOS	More advanced sensors (e.g., feature watches)	Higher-end microcontrollers	Real-time OS (e.g., FreeRTOS, Nucleus, QNX)	10KBs-1MB	days-weeks
Language Runtime	Generic sensing solutions, ”maker” devices	Off-the-shelf hardware	RTOS + VM supporting specific programming languages	100s kB to few MBs	days
Full OS	Generic sensing solutions, ”maker” devices	Off-the-shelf hardware	Linux	1/2 to few MBs	N/A or days
App OS	High-end smartwatches	Off-the-shelf or custom hardware	Android Wear or Apple Watch OS	From 512MB	up to 24 hours
Server OS	Solutions benefiting from portable Web server	Off-the-shelf hardware	Linux and Node.js	10s of MBs	up to 24 hours
Container OS	Solutions benefiting from fully isomorphic apps	Fully virtualized	Linux and Docker	GBs	N/A or hours

or non-existent. Given the fixed nature of software in these types of low-end devices, the amount of RAM and Flash memory can be kept minimal. In many cases, only a few kilobytes or tens of kilobytes of RAM will suffice.

2. *RTOS architecture*: for more capable IoT devices that benefit from a real-time operating system. Several off-the-shelf systems exist, both commercial and open source. Software development for RTOS-based IoT devices is usually carried out in-house, since such devices do not typically provide any third-party developer APIs or the ability to reprogram the device dynamically (apart from performing a full firmware update). Memory requirements of RTOS-based architectures are comparable to No OS architectures, often necessitating as little as a few tens of kilobytes of RAM and a few hundred kilobytes of Flash memory. However, in recent years even in this area the amounts of memory have increased significantly; modern RTOS solutions may have megabytes of storage memory.

3. *Language runtime architecture*: for devices that require dynamic programming capabilities offered by languages such as Java, JavaScript or Python. Compared to No OS or RTOS solutions, language runtime based IoT devices are significantly more capable in the sense that they can support third-party application development and dynamic changes, i.e., updating the device software (or parts thereof) dynamically without having to reflash the entire firmware. The dynamic language runtime serves as the portable execution layer that enables third-party application development and the creation of developer-friendly application interfaces. Such capabilities leverage the interactive nature of the dynamic languages, allowing flexible interpretation and execution of code on the fly without compromising the security of the underlying execution environment and device.
   Examples of IoT development boards that provide support for a *specific built-in language runtime or virtual machine (VM)* are: *Espruino* (`https://www.espruino.com/`) or *Tessel 2* (`https://tessel.io/`) which provide built-in support for JavaScript applications, while Pycom's *WiPy* boards (`https://pycom.io/development-boards`) enable Python development.
   The technical capabilities and memory requirements of devices based on language runtime architecture vary considerably based on the supported language(s). Virtual machines for minimalistic programming languages such as Forth might require only a few tens of kilobytes of dynamic memory, while Java, JavaScript, WebAssembly, or Python VMs require at least several hundreds of kilobytes or preferably multiple megabytes of RAM. The size and complexity of the virtual machines also varies considerably, and thus the minimum amount of Flash or ROM memory can also range from a few hundreds of kilobytes to several megabytes.
4. *Full OS architecture*: for devices that are capable enough to host a full operating system (typically some variant of Linux). The presence of a full operating system brings a lot of benefits, such as built-in support for secure file transfers, user accounts, device management capabilities, security updates, very mature development toolchains, possibility to run third-party applications, and numerous other features. Compared to low-end No OS or RTOS architectures, the memory and CPU requirements of Full OS stacks are significantly higher. For instance, the desire to run a Linux-based operating system in a device bumps the RAM requirements from a few tens or hundreds of kilobytes (for an RTOS-based solution) up to half a megabyte at the minimum. Also, the significantly higher energy consumption requirements make it difficult to use such devices in use cases that require battery operation.
5. *App OS architecture*: for devices that are designed specifically to support third party application development. Located at the current high end of the IoT device spectrum, wearable device platforms such as Android Wear (`https://www.android.com/wear/`) or Apple watchOS (`https://www.apple.com/watchos/`) are in many ways comparable to mobile phone application platforms from 5-10 years ago. These wearable device platforms provide very rich platform capabilities and third-party developer APIs, but they

also bump up the minimum hardware requirements considerably. For instance, already back in 2014, the minimum amount of RAM required by Android Wear was half a gigabyte (512 MB) – over 10,000 times more than the few tens of kilobytes of RAM required for simple IoT sensor devices. Furthermore, the processing power requirements of App OS devices are also dramatically higher than in simplest microcontroller-based IoT devices. Typically, an ARM Cortex-A class processor is mandated (for instance, an ARM A7 processor running at 1.2 GHz was stated as the minimum requirement for Android Wear back in 2014), limiting maximum battery duration to a few days, or only to a few hours in highly intensive use.

6. *Server OS architecture*: for devices that are capable enough to run a server-side operating system stack (typically Linux + Node.js). The Node.js ecosystem (https://nodejs.org/) has popularized the use of the JavaScript language also in server-side development, thus turning JavaScript into lingua franca for web development from client to cloud. By default, Node.js assumes the availability of at least 1.5 GB of RAM. However, Node.js can be configured to operate with considerably smaller amounts of memory, starting from a few tens of megabytes. In addition to (or instead of) Node.js, there are several other web server offerings that are more tailored to embedded environments.
7. *Container OS architecture*: for high-end IoT devices that are powerful to host a virtualized, container-based operating system stack such as Docker or CoreOS Rocket (rkt). Given the independence of the physical execution environment that containers can provide, containers are a very attractive concept also for IoT development, especially in light of the technical diversity of IoT devices. Thus, although container technologies add considerable overhead compared to traditional binary software, their use has already started also in the context of IoT devices. From a purely technical viewpoint container-based architectures are definitely a viable option for IoT devices if adequate memory and other resources are available [7]. At the minimum, the host environment must typically have several gigabytes of RAM available, thus making this approach unsuitable for the vast majority of IoT devices.

4 Emergence of Edge IoT AI/ML Capabilities

One of the things unforeseen by our original taxonomy was the rapid emergence of AI/ML capabilities on the edge. These capabilities have made it possible to perform tasks such as object recognition, voice recognition, gesture detection and gas detection in IoT devices themselves. Only some five years ago, such tasks would have required significant data transfer and computation in the cloud.

In this section we will take a look at edge IoT AI/ML capabilities that have already resulted in significant changes in the future software architecture of IoT devices and their broader end-to-end system architecture.

From the viewpoint of IoT devices, current edge IoT AI/ML capabilities can be divided broadly into two or three categories based on whether AI/ML support is provided with dedicated hardware or in the form of software libraries. Software libraries can be further divided into ”generic” libraries and such AI/ML libraries that have been provided to support specific sensors.

4.1 Dedicated AI/ML Hardware for the Edge

In the past few years, a broad variety of AI/ML enabled single board computers (SBCs) and modules have emerged, all providing remarkable edge processing capabilities at a reasonable price. Examples of such single board computers and modules include the following (in alphabetical order):

– *BeagleBone AI* (`https://beagleboard.org/ai`) – priced at about 115€– is BeagleBoard.org's open source single-board computer (SBC) that is meant for use in home automation, industry automation and other commercial use cases. Beaglebone AI is intended for bridging the gap between small SBCs and more powerful industrial computers. The hardware and software of the BeagleBoard are fully open source.

– *Coral* (`https://coral.ai/`) is Google's product family for local, on-device AI applications. Google Coral product family consists of a number of standalone development boards, PC enhancement solutions (in the form factor of USB sticks or M.2 or PCIe accelerator boards), as well as modules that can be embedded into custom hardware products. Coral development boards such as the Dev Board Mini come pre-flashed with Mendel Linux, so the setup process only requires connecting to the board's shell console, updating some software, and then running a TensorFlow Lite model on the board. In contrast, the PC enhancement boards (such as the Coral Accelerator USB stick) can simply be plugged in to a host computer. After installing some additional TensorFlow-related software, the user can run typical examples such as object detection, pose detection, and keyphrase detection.

– *Khadas VIM3* (`https://docs.khadas.com/vim3/`) is an NPU (Neural Processing Unit) enabled development board that can be used as a standalone computer, small-footprint server or robotics system driver. Just like the other single-board AI/ML computers, Khadas VIM3 comes with a number of camera- and image-oriented sample applications for performing tasks such as object recognition.

– *NVIDIA Jetson* product family (`https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/`). NVIDIA's Jetson product family provides a number of different options ranging from $59 Jetson Nano and $399 Jetson TX2 ”supercomputer-on-a-module” all the way to Jetson AGX Xavier priced at $649 USD apiece. Each model is a complete System-on-Module (SOM) with CPU, GPU, PMIC (integrated power management), DRAM and flash storage. Jetson boards can run multiple neural networks in parallel for applications such as image classification, object detection, segmentation, and speech processing. Support for different AI/ML frameworks is provided. Even the smallest NVIDIA Jetson Nano model is rather powerful, featuring a quad core ARM Cortex A57 processor, 2GB LPDDR4 memory, 16 GB eMMC Flash and NVIDIA Maxwell GPU with 128 cores. However, for more advanced video processing tasks, higher-level Jetson devices are

recommended. The more powerful Jetson TX2 model features on-board AI support with an NVIDIA Pascal GPU, up to 8 GB of LPDDR4 memory, 59.7 GB/s of memory bandwidth, and a wide range of hardware interfaces. These devices are fit for various use cases, including also those that involve 'do-it-yourself' characteristics [8]. For software development, quite sophisticated tools are provided, including support for container-based deployment (`https://docs.nvidia.com/egx/egx-introduction/`).
- *ROCK Pi N10* (`https://wiki.radxa.com/RockpiN10`) – developed by Radxa – is an inexpensive NPU-equipped single-board computer that is part of the ROCK Pi product family (`https://rockpi.org/`). It is being offered in three variants: ROCK Pi N10 Model A, ROCK Pi N10 Model B and ROCK Pi N10 Model C. The only differences between these variants are the price, RAM and Storage capacities. The base variant of ROCK Pi N10 is available at $99, while its range topping variant comes in at $169. All the variants are equipped with an NPU that offers up to 3 TOPS (Tera Operations Per Second) of performance.

For a summary of single-board computers for AI/ML applications, refer to: `https://itsfoss.com/best-sbc-for-ai/`

4.2 Software-Based AI/ML Solutions for the Edge

In addition to custom hardware, AI/ML capabilities can be added to edge devices also in the form of software libraries. As already indicated by the summary of emerging AI/ML hardware technologies above, AI/ML software is being redesigned to support intelligent operations away from the cloud. This involves the ability to feed local sensor data to pre-trained models running on the sensor devices themselves, or even to use such data to locally train or improve the training of such models, thus paving the way towards offline and disconnected operation of self-calibrating sensors.

Software-based AI/ML solutions can be divided into (a) generic solutions and (b) those that have been customized to support specific devices or sensors; we will discuss both categories below.

Generic solutions. Popular generic AI/ML solutions include *TinyML* [9] which allows the integration of ML capabilities with microcontroller units for analytics applications which require an extremely low power (typically in the mW range and below). In addition, existing ML tools and libraries such as TensorFlow are being adapted so they can be used to perform ML inferences on such devices. Popular systems include *TensorFlow Lite* and *TensorFlow Lite for Microcontrollers*. Moreover, customized tools are needed for converting ML models trained on high-powered devices so that these models can be simplified to fit in low-power devices. Other libraries, e.g., the Artificial Intelligence for Embedded Systems (AIfES) library developed at Fraunhofer IMS, have been designed with the consideration of the limitations of small powered devices from the start. For example, they can ensure that pre-allocated, static data structures are used to store the weights and the training data of a neural network [10].

As an example of an inexpensive microcontroller that provides support for software-based AI/ML capabilities, we mention the *Arduino Nano 33 BLE Sense* (`https://docs.arduino.cc/hardware/nano-33-ble-sense`). This device is intended for developers who are becoming familiar with embedded machine learning; it combines multiple sensors for inertial measurements, digital microphone, temperature, humidity and barometric pressure, proximity, light and gesture recognition with the ability to run MicroPython code invoking the TensorFlow Lite libraries.

Custom solutions for specific devices/sensors. The main arguments in favor of using AI/ML capabilities on the edge include both the abundance of incoming data which can be locally classified or filtered as well as privacy concerns [11]. In the past year or so, new technologies have been announced that will provide such processing capabilities at the *very* edge – as part of the sensors that can be embedded in the IoT devices. A good example of such technology is the AI-enabled Bosch BME688 gas sensor (`https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme688/`) that can be trained to detect different gas compositions. The sensor itself is very small – only 3mm by 3mm by 1mm, and it costs under 10€ apiece. In practice, the AI/ML capabilities of this sensor are provided in the form of a custom library that needs to be compiled into the firmware of the microcontroller that is hosting the sensor.

5 Towards Seamless Device-Edge-Cloud Continuum

Historically, IoT systems were very cloud-centric, with the majority of computation taking place centrally in the cloud. However, given the rapidly increasing computing and storage capacities of IoT devices, it is clear that in the future IoT systems it can be very beneficial to balance and seamlessly transfer intelligence between the cloud and the edge. Such capabilities are important, since the ability to preprocess data in IoT devices allows for lower latencies and can also significantly reduce unnecessary data traffic between the devices and cloud. In general, in recent years there has been a noticeable trend towards *edge computing*, i.e., cloud computing systems that perform a significant part of their data processing at the edge of the network, near the source of the data [12]. In IoT systems supporting edge computing, devices and gateways play a key role in filtering and preprocessing the data, thus reducing the need to upload all the collected data to the cloud for further processing.

Intelligence at the Edge. The edge of a modern computing system consists of a broad range of devices such as base stations, smart phones, tablets, measurement and sensing devices, gateways, and so on. In contrast with the cloud, the computing capacity, memory and storage of edge devices are often limited. However, these devices are conveniently located near the endpoint, thus offering low latency. Due to limitations in connectivity and the increasing capacity of edge devices, intelligence in a modern end-to-end computing system is moving towards the edge, first to gateways and then to devices. This includes both generic software functions, and – more importantly – time critical AI/ML features for processing data available in the edge, referred to as Edge AI. It has been envisioned that evolution of telecom infrastructures beyond 5G will consider highly distributed AI, moving the intelligence from the central cloud to edge computing resources [13]. Furthermore, edge intelligence is a necessity for a world where intelligent autonomous systems are commonplace.

AI/ML can be a new reason for using heterogeneous technologies across the different types of devices, assuming that we accept each technology domain to remain separate, as indicated in the taxonomy presented in Section 3. In contrast, the emergence of AI/ML capabilities at the edge can also provide a motivation for technologies that blur the line between the cloud and the edge with approaches that scale AI/ML operations down to small devices (e.g., TensorFlow Lite, AIfES, TinyML) or provide seamless flexibility for the migration of components that host the AI/ML features (e.g., isomorphic and liquid software).

Isomorphic IoT systems. One of the key challenges for IoT system development is development complexity. As we have pointed out in our previous papers (see, e.g., [14]), the software technologies required for the development of key elements of an end-to-end IoT system – devices, gateways and cloud backend – tend to be very diverse. For the development of cloud components, developers need to be familiar with technologies such as Docker, Kubernetes, NGINX, Grafana, Kibana, Node.js, and numerous Node.js libraries and modules. Gateways are commonly built upon native or containerized Linux or – in case of consumer solutions – on top of mobile operating systems such as Android or iOS. In contrast, IoT device development still requires the mastery of "classic" embedded software development skills and low-level programming languages such as C. In addition, IoT systems commonly include web-based user interfaces that are developed with frontend web technologies such as React.js or Angular. The palette of development technologies covered by the entire end-to-end system is so wide that hardly any developer can master the full spectrum.

Earlier in this paper, we noted that software containers and virtualization technologies are becoming available also in IoT devices. We predict that within the next five to ten years, this may lead the industry to *isomorphic IoT system architectures* [14] – in analogy to isomorphic web applications [15] – in which the devices, gateways and the cloud will have the ability to run *exactly the same software components and services*, allowing flexible migration of code between any element in the overall system. In an isomorphic system architecture, there does not have to be any significant technical differences between software that runs in the backend or in the edge of the network. Rather, when necessary, software can freely "roam" between the cloud and the edge in a seamless, liquid fashion. However, aiming at this goal means new research for lighter-weight containers [16] as well as new solutions for runtime migration [17].

Granted, there will still be various technical differences in the components that are intended for different elements in the end-to-end system. Still, it should be easier to constrain where isomorphic software can or cannot be deployed as opposed to rewriting it completely when the need for a new deployment arises. For instance, those components that are intended for interfacing with specific sensors or radio protocols in IoT devices do not necessarily have to run in end-user web or mobile applications. Conversely, end user UI components are not expected to run in IoT devices that do not have a display at all. However, the key point here is the reduced need to learn completely different development technologies and paradigms. This is important if we wish to reduce the intellectual burden and lower the steep learning curve that hampers end-to-end IoT systems development today.

In many ways, isomorphic architectures can be seen as the missing link in IoT development. Instead of having to learn and use many different incompatible ways of software development, in an isomorphic system architecture a small number of base technologies will suffice and will be able to cover different aspects of end-to-end development. At this point it is still difficult to predict which technologies will become dominant in this area. The earlier mentioned software container technologies such as *Docker* and *CoreOS rkt* are viable guesses, even though their memory and computing power requirements may seem ludicrous from the viewpoint of today's IoT devices. Amazon's *Greengrass* system (`https://aws.amazon.com/greengrass/`) also points out to a model in which the same programming model can be used both in the cloud and in IoT devices; in Greengrass, the programming platform is Amazon's Lambda. In the smaller end of the spectrum, the *Toit* system (`https://toit.io/`) developed by people from Google's original V8 JavaScript VM team seems very promising.

In our recent IEEE Computer paper [14], we predicted that isomorphic IoT systems would most likely form around two primary base technologies: (1) *JavaScript/ECMAScript* [18] and (2) *WebAssembly* [19]. The former is the *de facto* language for web applications both for the web browser and the cloud backend (Node.js); it is currently the most viable option for implementing static isomorphism, i.e., to allow the use of the same programming language throughout the end-to-end system. The latter is a binary instruction format to be executed on a stack-based virtual machine that can leverage contemporary hardware [20, 21]; we see WebAssembly as the best option for providing support for dynamic isomorphism, i.e., the ability to use of common runtime that is powerful but small enough to fit also in low-end IoT devices. Note that these options are not mutually exclusive, i.e., it would be possible to implement an architecture in which WebAssembly is used as the unifying runtime but in which JavaScript is used as the programming language throughout the end-to-end system.

Liquid software. Liquid software [22], also known as cross-device experience roaming [23], is a concept where software can dynamically flow between different computers, basically allowing execution of code and associated user experiences to be transferred dynamically and seamlessly from one computational element to another. While the majority of the work associated with liquid software has focused on the UI layer (e.g., [24–26]), the concept is applicable to any situation in which software can be dynamically redeployed and adapted to take full advantage of the storage and computational resources provided by different devices that are shared by one or multiple collaborating users.

In essence, building liquid applications needs two facilities. One is the ability to relocate code flexibly across different computing entities, which is an elementary expectation and principle also for the isomorphism of software. The second facility is the ability to synchronize the state of the application and its UI across all devices running the code. This has been implemented by Apple in their Continuity/Handoff framework [27], which today is the most advanced industrial implementation, as well as by many academic agent frameworks (e.g., [28]) and web development frameworks (e.g., [29, 30]).

Cellular IoT and mesh networking technologies will increase the role of edge computing. Another area in which there has been a lot of development after the creation of our initial software architecture taxonomy are radio technologies and communication protocols. These emerging technologies can have a significant impact on the overall IoT system architecture, thus also impacting the device-edge-cloud continuum. We focus especially on two categories: (1) LPWAN (Low-Power Wide Area Network) technologies and (2) local mesh networking connectivity.

Low-Power Wide Area Network technologies. Low-power wide area network (LPWAN) technologies make it possible for IoT devices to communicate with the cloud directly from a distance – without the need for gateway devices in the middle. Prominent LPWAN technologies include Cellular IoT technologies such as NarrowBand-IoT and LTE-M, as well as more proprietary technologies such as LoRa (`https://lora-alliance.org/`) and SIGFOX (`https://www.sigfox.com/`). We do especially wish to highlight the 3GPP Cellular IoT radio technologies – NB-IoT and LTE-M – which make it possible to connect virtually any artifact directly to the Internet at low cost and minimal battery consumption. Cellular IoT technologies can eliminate (or at least dramatically reduce the need for) gateways in IoT systems, allowing IoT devices to communicate with the cloud directly.

Standardization of 3GPP Cellular IoT technologies was completed in 2016, and these technologies have already been widely deployed onto existing commercial cellular networks. In fact, nationwide Cellular IoT coverage for IoT devices is already available in numerous countries, although these capabilities are still in relatively low use. Chipsets and hardware modules supporting Cellular IoT technologies are available from various vendors, including Gemalto, Nordic Semiconductor, Quectel, Sierra Wireless and u-Blox.

Local mesh networking connectivity. Another area that is likely to have a significant impact on the overall topology of IoT systems is peer-to-peer (P2P) connectivity between IoT devices. New technologies such as *Bluetooth Mesh* (`https://www.bluetooth.com/specifications/mesh-specifications`) are making it feasible for IoT devices to exchange information with each other efficiently with minimal latencies – thus further reducing the need for more expensive communication with the cloud. As opposed to current cloud-centric IoT systems, P2P and edge computing are fundamental characteristics of systems in which low latency is required. An interesting broader question is whether there will still be need for "constrained" protocols such as CoAP or MQTT, or will the landscape be dominated by broader *de facto* standard solutions such as REST/HTTPS. At the time of this writing, MQTT (`https://mqtt.org/`) seems to be the dominant IoT system communication protocol, although there are emerging standards such as Matter (`https://buildwithmatter.com/`) that may replace it in the longer run.

Even though the dominant LPWAN and mesh networking protocols have not been fully established yet, together the emergence of mesh networking and LPWAN technologies can be expected to *lead to a drastically increased role of edge computing*, as well as to a significantly reduced role of gateways in the overall IoT system architecture. As the role of the gateways withers down, IoT devices themselves will take a more active role in the overall E2E architecture.

Table 2. High-Level Comparison of Software Architecture Options

Feature	**No OS / RTOS**	**Language VM**	**Full OS**	**App OS**	**Server OS**	**Container OS**
Typical development language	C or assembly	Java, JavaScript, Python	C or C++	Java, Objective-C, Swift	JavaScript	Various
Libraries	None or System-specific	Language-specific generic libraries	OS libraries, generic UI libraries	Platform libraries	Node.js NPM modules	Various
Dynamic SW updates	Firmware updates only (Reflashing)	Yes	Yes	Yes (App Stores)	Yes	Yes (Image Snapshots)
Third-party apps supported	No	Yes	Yes	Yes (Rich APIs)	Yes	Yes
AI/ML at the edge	Emerging	Yes	Yes	Yes	Yes	Yes
Isomorphic apps possible	No	Yes	Only if the same OS/HW	Yes	Yes	Yes

6 Wrapping Things Up

In summary, there exists a broad range of software architecture options and stacks for IoT devices, depending on the expected usage, power budget, and memory requirements (see Table 1 earlier in the paper) and the need to support dynamic software deployment and/or third-party development as well as intelligent decision making on the device. Table 2 provides a condensed summary of the software architecture options for IoT devices, focusing on the broader architectural implications in the device-edge-cloud continuum. It should be noted that the options summarized in the table are by no means exclusive. For instance, as already mentioned above, devices based on the language runtime architecture commonly have an RTOS underneath. Likewise, in Full OS platforms, it is obviously possible to run various types of language runtimes and virtual machines as long as an adequate amount of memory is available to host those runtime(s). In general, the more capable the underlying execution environment is, the more feasible it is to run various types of software architectures, platforms and applications on it.

7 Conclusions

In this paper we have revisited a taxonomy of software architecture options for IoT devices, starting from the most limited sensing devices to high-end devices featuring full-fledged operating systems and developer frameworks. After examining each of the basic options, we presented a comparison and some broader observations, followed by relevant emerging trends and future directions. In particular, we noted that the emergence of inexpensive AI/ML hardware – unforeseen by our original taxonomy – is increasing the role of the edge in IoT systems. Later in the paper, we additionally predicted that new communication technologies such as Cellular IoT and mesh networking will alter the overall topology of IoT systems quite considerably, e.g., leading to a reduced role of gateways in the overall architecture.

Although the vast majority of IoT devices today have fairly simple software stacks, the overall software stack complexity can be expected to increase due to hardware evolution and the general desire to support edge computing, AI/ML technologies and software containers. In light of these observations, we made a case for isomorphic IoT systems in which development complexity is alleviated with consistent use of technologies across the entire end-to-end system, providing a more seamless technology continuum from IoT devices on the edge all the way to the cloud. In such systems, different subsystems and computational entities can be programmed with a consistent set of technologies. Although fully isomorphic IoT systems are still some years away, their arrival may ultimately dilute or even dissolve the boundaries between the cloud and its edge, allowing computations to be performed in those elements that provide the optimal tradeoff between performance, storage, network speed, latency and energy efficiency. We hope that this paper, for its part, encourages people to investigate these exciting new directions in more detail.

References

1. Wasik, B.: In the Programmable World, All Our Objects Will Act as One. Wired (May 2013)
2. Munjin, D., Morin, J.H.: Toward Internet of Things Application Markets. In: 2012 IEEE International Conference on Green Computing and Communications (GreenCom), IEEE (2012) 156–162
3. Taivalsaari, A., Mikkonen, T.: Roadmap to the Programmable World: Software Challenges in the IoT Era. IEEE Software, Jan/Feb 2017 **34**(1) (2017) 72–80
4. Taivalsaari, A., Mikkonen, T.: Beyond the Next 700 IoT Platforms. In: Proceedings of 2017 IEEE International Conference on Systems, Man and Cybernetics (SMC'2017, Banff, Canada, Oct 5-8). (2017) 3529–3534
5. Taivalsaari, A., Mikkonen, T.: On the Development of IoT Systems. In: Proceedings of the 3rd IEEE International Conference on Fog and Mobile Edge Computing (FMEC'2018, Barcelona, Spain, April 23-26, 2018). (2018)
6. Botta, A., De Donato, W., Persico, V., Pescapé, A.: Integration of Cloud Computing and Internet of Things: a Survey. Future Generation Computer Systems **56** (2016) 684–700
7. Celesti, A., Mulfari, D., Fazio, M., Villari, M., Puliafito, A.: Exploring Container Virtualization in IoT Clouds. In: 2016 IEEE International Conference on Smart Computing, IEEE (2016) 1–6
8. Cass, S.: Nvidia Makes It Easy to Embed AI: The Jetson Nano Packs a Lot of Machine-Learning Power into DIY Projects. IEEE Spectrum **57**(7) (2020) 14–16
9. Sanchez-Iborra, R., Skarmeta, A.F.: TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. IEEE Circuits and Systems Magazine **20**(3) (2020) 4–18
10. Gembaczka, P., Heidemann, B., Bennertz, B., Groeting, W., Norgall, T., Seidl, K.: Combination of Sensor-Embedded and Secure Server-Distributed Artificial Intelligence for Healthcare Applications. Current Directions in Biomedical Engineering **5**(1) (2019) 29–32
11. Greengard, S.: AI on Edge. Communications of the ACM **63**(9) (2020) 18–20
12. Shi, W., Dustdar, S.: The Promise of Edge Computing. IEEE Computer **49**(5) (2016) 78–81
13. Peltonen, E., Bennis, M., Capobianco, M., Debbah, M., Ding, A., Gil-Castiñeira, F., Jurmu, M., Karvonen, T., Kelanti, M., Kliks, A., et al.: 6G White Paper on Edge Intelligence. arXiv preprint arXiv:2004.14850 (2020)
14. Mikkonen, T., Pautasso, C., Taivalsaari, A.: Isomorphic Internet of Things Architectures With Web Technologies. Computer **54** (July 2021) 69–78
15. Strimpel, J., Najim, M.: Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions. O'Reilly Media, Inc. (2016)
16. Park, M., Bhardwaj, K., Gavrilovska, A.: Toward Lighter Containers for the Edge. In: 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20). (2020)
17. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering **24**(5) (1998) 342–361
18. ECMA International: Standard ECMA-262: ECMAScript 2020 Language Specification. (June 2020, available online: https://www.ecma-international.org/publications/standards/Ecma-262.htm)
19. World Wide Web Consortium: WebAssembly Core Specification (2019) https://webassembly.github.io/ spec/core/_download/WebAssembly.pdf.
20. Bryant, D.: WebAssembly Outside the Browser: A New Foundation for Pervasive Computing. In: Keynote at ICWE'20, June 9-12, 2020, Helsinki, Finland. (2020)

21. Jacobsson, M., Willén, J.: Virtual Machine Execution for Wearables Based on WebAssembly. In: EAI International Conference on Body Area Networks, Springer (2018) 381–389
22. Taivalsaari, A., Mikkonen, T., Systä, K.: Liquid Software Manifesto: The Era of Multiple Device Ownership and Its Implications for Software Architecture. In: 38th Annual IEEE Computer Software and Applications Conference (COMPSAC), IEEE (2014) 338–343
23. Brudy, F., Holz, C., Rädle, R., Wu, C.J., Houben, S., Klokmose, C.N., Marquardt, N.: Cross-Device Taxonomy: Survey, Opportunities and Challenges of Interactions Spanning Across Multiple Devices. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. (2019) 1–28
24. Voutilainen, J.P., Mikkonen, T., Systä, K.: Synchronizing Application State Using Virtual DOM Trees. In: International Conference on Web Engineering, Springer (2016) 142–154
25. Gallidabino, A., Pautasso, C.: The Liquid.js Framework for Migrating and Cloning Stateful Web Components Across Multiple Devices. In: Proceedings of the 25th International Conference Companion on World Wide Web. (2016) 183–186
26. Husmann, M., Rossi, N.M., Norrie, M.C.: Usage Analysis of Cross-Device Web Applications. In: Proceedings of the 5th ACM International Symposium on Pervasive Displays. (2016) 212–219
27. Gruman, G.: Apple's Handoff: What Works, and What Doesn't. InfoWorld (2014)
28. Systä, K., Mikkonen, T., Järvenpää, L.: HTML5 Agents: Mobile Agents for the Web. In: Web Information Systems and Technologies - 9th International Conference, WEBIST 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers. (2013) 53–67
29. Gallidabino, A., Pautasso, C.: Decentralized Computation Offloading on the Edge with Liquid WebWorkers. In: International Conference on Web Engineering, Springer (2018) 145–161
30. Gallidabino, A., Pautasso, C.: Multi-Device Complementary View Adaptation with Liquid Media Queries. Journal of Web Engineering (2020) 761–800