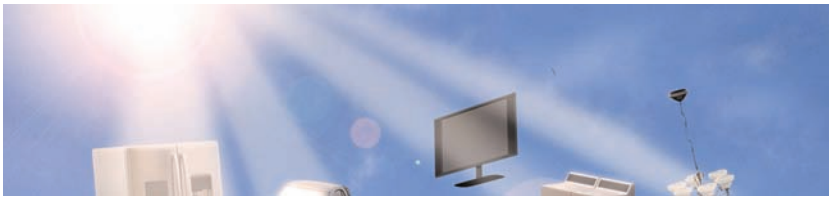
A Roadmap to the Programmable World

Software Challenges in the IoT Era

Antero Taivalsaari, Nokia Technologies

Tommi Mikkonen, Mozilla

// The emergence of millions of remotely programmable devices in our surroundings will pose significant challenges for software developers. A roadmap from today's cloud-centric, data-centric Internet of Things systems to the Programmable World highlights those challenges that haven't received enough attention. //



THE INTERNET OF Things (IoT) represents the next significant step in the Internet's evolution. In the 1970s and 1980s, the Internet (such as it was) was primarily about connecting computers. In the 1990s and 2000s, the Internet was all about connecting people. Now, the emphasis is shifting toward connecting everything (or literally every thing) to the Internet.

Most IoT research currently revolves around data acquisition, real-time and offline analytics, machine learning, data visualization, and other big data topics.[1] This focus isn't surprising, given the massive business potential arising from the ability to collect data from millions of sensing devices and then digest and combine that data to provide new insights into people's behavior and other real-world phenomena.

However, an equally important but subtler and more tranquil disruption is underway. Hardware advances and the general availability of powerful but inexpensive integrated chips will make it possible to embed connectivity and fully fledged virtual machines and dynamic language runtimes everywhere. So, our everyday things—light bulbs, door knobs, air-conditioning systems, lawn sprinklers, vacuum cleaners, toothbrushes, and the kitchen sink—will become connected and programmable dynamically.

Here, we present a roadmap from today's cloud-centric, data-centric IoT systems to a world where everyday objects are connected and the network's edge is truly programmable. The Programmable World—programmable in a literal sense—will pose new challenges for software developers. Today's development methods, languages, and tools—or at least those that are in widespread use—are poorly suited to the emergence of millions of programmable things in our surroundings. We highlight issues and technical challenges that deserve deeper study beyond those IoT topics that receive the most attention today.

Because this article is forward-looking, our roadmap is somewhat subjective. Our viewpoints stem from our own projects and collaborations in the IoT domain,[2–5] as well as from our experience predicting and partaking in mobile and web computing's evolution over the past 20 years. For instance, the emergence of virtual machines in mobile

 0740-7459/17/$33.00 © 2017 IEEE

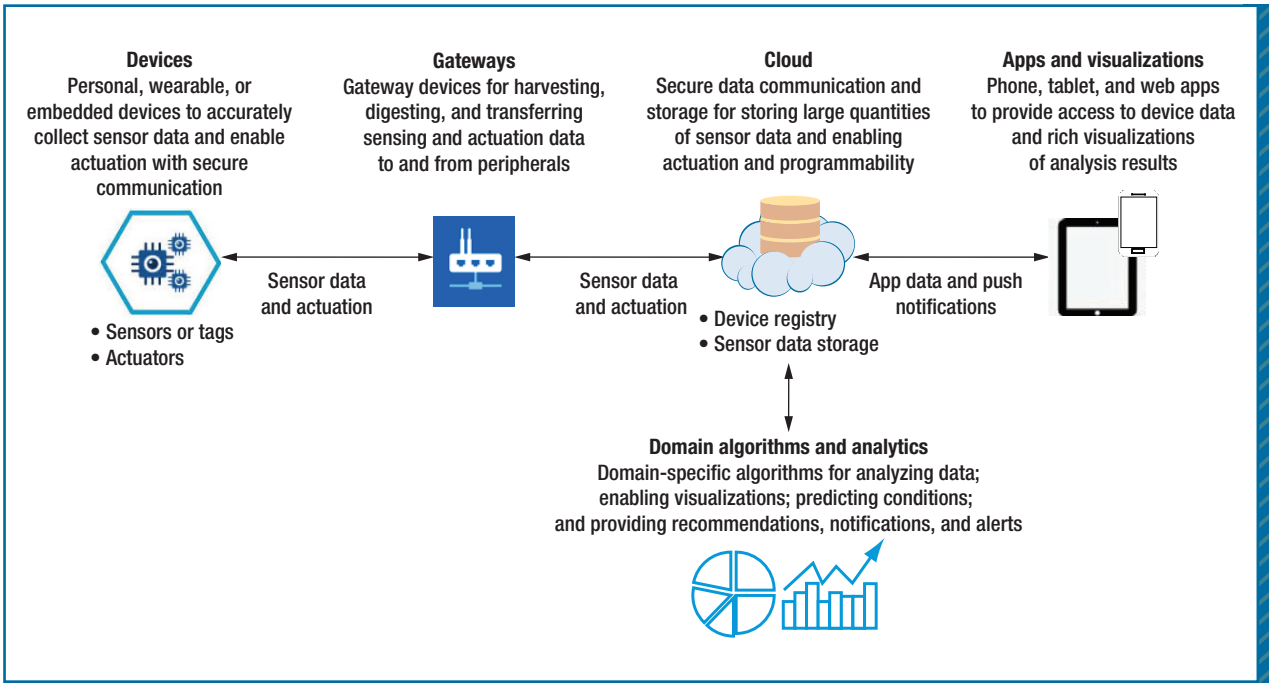


FIGURE 1. The emerging common end-to-end Internet of Things architecture. Despite the apparent diversity and huge number of vendors targeting the IoT market, a number of elements of this architecture are pretty much the same in all the systems.

phones in the late 1990s wasn't a dramatic technical achievement per se. However, it opened up mobile phones for the vast masses of developers, creating today's multibillion-dollar mobile-app industry. Although history rarely repeats itself, it often rhymes. In this case, parallels with the past seem especially obvious, because we're just reaching the point at which we'll be able to embed dynamic programming capabilities virtually everywhere.

The Emerging Common End-to-End IoT Architecture

The term "Internet of Things" isn't new. Over 20 years ago, MIT professors described a world in which things (devices or sensors) are connected and can share data.[6] At the beginning, IoT concepts emerged around enabling technologies such as RFID and wireless sensor networks.[7] However, recently, the concepts' use has spread rapidly to various domains, including healthcare, transportation, energy, retail, building management, and process automation. Hundreds of startup companies have been created in this area, and most major corporations have announced IoT platform and component offerings. A recent study listed 115 IoT cloud platforms.[8] We're currently witnessing a classic "crossing the chasm" period in which the ultimate winners haven't been determined yet.[9] Most likely, by 2025 only a handful of dominant IoT platforms and platform vendors will remain.

What's interesting about these IoT offerings is their conceptual similarity. Despite the apparent diversity and huge number of vendors targeting the IoT market, a common end-to-end architecture is emerging for IoT solutions, with a number of elements that are pretty much the same in all the systems.[1,10,11] Figure 1 depicts the basic concepts, some of which we summarize next.

Devices

Devices (or peripherals) are the hardware elements that collect sensor data or perform actuation, with built-in communication capabilities to submit the collected data to the broader IoT ecosystem. Functionally, devices are either sensors or actuators.

Sensors provide information about the physical entity they monitor. This information can range from the physical entity's identity to measurable qualities such as temperature, humidity, pressure, luminosity, sound level, fluid flow, vibration, and abrasion. Sensors whose sole purpose is to facilitate an identification process are called tags.

Actuators take energy, usually transported by air, electricity, or liquid, and convert it to a state change, thus modifying one or more physical entities.

Gateways

Gateways (or hubs) collect, preprocess, and transfer data from IoT devices and their sensors, employing different (usually wireless) communication protocols such as Wi-Fi or Bluetooth Smart. Gateways provide secure protocol translation between devices and the cloud, and might support tasks such as intermediate sensor data storage and preprocessing, service discovery, geolocalization, verification, and billing. Gateways also deliver the actuation requests from the cloud to devices.

In some systems, the devices themselves can upload sensing data directly to the cloud and receive actuation requests directly from the cloud—for example, through Wi-Fi, 3G or 4G, or (soon) NB-IoT (NarrowBand IoT) and 5G networks. Such solutions don't require dedicated gateways. Furthermore, devices themselves commonly act as gateways to other devices, possibly forming peer-to-peer or mesh topologies through local connectivity.

The Cloud

Cloud computing and cloud-based storage and analytics solutions are central to most IoT platforms. In the generic end-to-end IoT architecture, the cloud plays three main roles.

The first is data acquisition, storage, and access. A fundamental functionality in IoT systems is sensor data collection and storage. IoT devices with sensing capabilities typically collect a large amount of data that must be harvested and stored in the cloud for further processing and analysis. Solutions in this area range from simple on-premises databases to massive replicated, fault-tolerant, scalable storage clusters. Query and notification APIs for accessing collected data are also provided.

The second role is data analytics. This refers to examining, cross-connecting, and transforming acquired sensor data to discover and present useful information—for example, to support remote information sharing and decision making. Real-time analytics refers to analysis functions that are run immediately after the data has been received. Offline analytics refers to batch-style operations that occur after large datasets have already been accumulated. Machine-learning and data-mining technologies and algorithms are important in this area.

The third role is actuation support. IoT system dataflows aren't just unidirectional. Besides sensor data collection from devices to the cloud, secure device actuation from the cloud to devices is important. Actuation capabilities are a fundamental enabler for remote device programmability.

In addition, a cloud solution supporting the IoT typically includes administrative functions such as device management, user account management, usage logging, server status monitoring, and reporting capabilities.

The Roadmap

We believe the IoT's future lies in the ability to orchestrate and program large, complex topologies of IoT devices remotely. As Bill Wasik argued, once devices are connected to public or private clouds, with sensor data flowing in and actuation capabilities being widely available, the focus will shift from sensor data collection and analytics to application-programming capabilities for manipulating complex real-world systems.[12] The actuation capabilities offered by IoT devices form the foundation for all this. These capabilities enable us to command and control everyday objects in our surroundings (and potentially all over the planet) from the comfort of a programming environment or an app in front of us.

As we mentioned before, hardware advances will drive the transition toward the Programmable World. IoT hardware capabilities and price points are rapidly reaching an inflection point at which we'll be able to run fully fledged OSs such as Linux or virtualized software stacks or dynamic language runtimes on almost any type of device. The low-cost chips that have recently become available already match or exceed the memory and processing power of mobile phones in the late 1990s, just before the emergence of the Java 2 Platform, Micro Edition[13] launched the mobile-app revolution. We assume that similar cross-manufacturer programming capabilities will soon find their way to our everyday things.

Another major trend driving the industry toward the Programmable World is *edge computing*. Classic cloud computing systems are highly centralized. Although centralized computing has significant benefits, it can be costly in terms of communication and power consumption. Consider an IoT environment consisting of a large collection of devices that are near each other. It might be inefficient to transmit the data from those devices to a remote datacenter for processing and then transmit actuation requests back from the datacenter to the devices. In an IoT deployment with tight latency

TABLE 1

The anticipated evolution of Internet of Things (IoT) systems.

Year	Data viewpoint	Programmability viewpoint
2015	• Data acquisition on a massive scale • Monitoring, tracking, routing, command and control, and mining for trends and behaviors • Cloud-centric data analytics, including both real-time and offline analytics • A focus on data visualization and simple If This Then That (IFTTT) alerts • Open source technologies available and widely used for implementing data acquisition and analytics features	• Most serious computation performed in the cloud • Device support for basic actuation only; actuation implemented mostly natively using device- or manufacturer-specific, proprietary APIs and apps • Device- or manufacturer-specific device control applications available in app stores • Visual notations (for example, Node-RED) emerging for implementing device control applications more portably • Standards emerging (for example, OMA Lightweight M2M, and IPSO Smart Objects) but not yet widely adopted
2020 (Edge IoT Era)	• Edge computing APIs and mechanisms leveraged extensively in data processing and analytics; more intelligent filtering and denoising of uploaded data • Increasingly autonomous operation of data acquisition and analytics systems based on direct machine-to-machine communication, using local connectivity • Adapting, enhancing, and extending; automatic determination and selection of computing and analytics resources (cloud versus edge)	• Edge computing capabilities and APIs available for provisioning computing flexibly between the cloud and edge devices • More advanced actuation capabilities; standardized actuation APIs • Domain-specific device control applications available—for example, for controlling lighting systems or home security equipment from different manufacturers • Virtual machines commonly available in IoT devices, enabling cross-manufacturer IoT application development and flexible migration of computation between the cloud and edge devices
2025 (Universal IoT Era)	• Fully automated, context-aware data acquisition, analytics, and decision optimization based on pervasive use of AI techniques such as machine learning • Universal machine-to-machine collaboration enabled by common cross-manufacturer, cross-industry APIs; social use of data among machines	• A universal, containerized application-deployment-and-execution model supported across multiple manufacturers and industries • Industry-wide, cross-manufacturer programming APIs allowing generalized device discovery, data acquisition, remote device programming, and device management • Universal-remote-control applications and universal device consoles possible • Dynamic remote programming and reprogramming of devices widely supported, enabled by the widespread use of virtual machines, containerization, and common developer APIs

requirements, latency overhead alone can make such solutions impractical.

The term "edge computing" was coined around 2002 and was originally associated with deploying applications over content delivery networks (CDNs). The main objective was to benefit from the proximity of CDN edge servers to improve scalability and lower latency. Edge computing IoT solutions harness the network's edge (usually gateway devices such as routers or base stations) for computation—for example, to preprocess sensor data and trigger alerts and actuation requests locally on the basis of predefined criteria. Such systems might leverage local connectivity technologies (for example, Bluetooth Smart or Wi-Fi Direct) to enable direct, more efficient, and decentralized communication between sensor devices. Virtualization technologies also play a central role in enabling migration of computation between entities.

Table 1 presents our roadmap predicting how IoT systems will evolve over the next 10 years, from both the data and programming viewpoints. The table is based on

- observed trends in industry and academia, reflecting recent theses, academic papers, and books;[1,5,7,10,11,14,15]
- expert views building on practical product and prototype development experience in industry and academia;[2,3]
- personal experience and past observations from the mobile-app revolution as mobile phones evolved from closed, voice-centric devices to application-rich smartphones;[13] and
- personal experience and past observations from the web's evolution from a simple document distribution environment to a platform that supports rich

application development and instant worldwide deployment.[16]

Because our focus is primarily the programming aspects, in the rest of the article we dive deeper into only the programmability-related challenges.

Basically, we expect IoT programming capabilities to evolve from simple actuation features, vendor-specific apps, device APIs, and cloud-centric data acquisition and processing, to systems that extensively leverage edge computing, virtualization, and containers. Such systems will support portable, cross-manufacturer, and cross-industry application development. They'll also allow—when required—flexible migration of computation and data between the cloud and heterogeneous edge devices. We call the forthcoming phases of IoT system evolution the Edge IoT Era and Universal IoT Era.

We call the forthcoming phases of IoT system evolution the Edge IoT Era and the Universal IoT Era.

It's debatable whether one API will ever cover IoT devices from entirely different domains and vertical markets. However, it's safe to predict that in 5 to 10 years, IoT devices and their APIs will have converged significantly. It's simply impractical for people to use a large number of vendor-specific apps to control all the devices in their surroundings. It's also likely that the necessary infrastructure will grow around the existing IP networking and web infrastructure, enhanced with local connectivity technologies to support edge computing.

What Makes IoT Development Different

IoT development differs from mainstream mobile-app and client-side web application development in seven ways. These differences are key reasons for the implications and technical challenges we present later.

First, IoT devices are almost always part of a larger system of devices—for example, an installation of interoperating devices in a home, an office, a factory, a shopping mall, or an airplane. Furthermore, IoT devices are usually just a small part of the end-to-end architecture we discussed earlier. Granted, PCs and smartphones also have major dependencies on cloud-based services. However, from the application developer's perspective, they're still primarily standalone devices, with the developer targeting a single computer or smartphone.

Second, IoT systems never sleep. PCs, smartphones, and other standalone computing devices can be viewed as "rebootables"—systems that can and will be rebooted when things go awry. In contrast, IoT systems usually shouldn't or can't be shut down in their entirety. Although individual devices might be shut down, the entire system must be resilient to device and network outages.

Third, IoT systems are more like cattle than pets. The number of computing units (devices or CPUs) in IoT systems is often dramatically larger than in traditional computing environments, potentially reaching hundreds, thousands, or even millions. Unlike PCs and smartphones, which users think of almost as pets, IoT devices in a large system are more like cattle that must be managed en masse instead receiving personal attention and care.

Fourth, IoT devices are often embedded in our surroundings such that they're physically invisible and unreachable. Devices might be permanently buried deep underground or physically embedded in various materials (for example, vibration sensors in mining equipment). It might be impossible to attach physical cables to those devices or replace hardware or embedded software when problems arise.

Fifth, IoT systems are highly heterogeneous. Their computing units can have dramatically varying computing power, storage capabilities, network bandwidth, and energy requirements. The I/O mechanisms, sensory capabilities, and supported input modalities can also vary considerably. Some devices have physical buttons and displays, whereas many devices have no visible user interface.

Sixth, IoT systems tend to have weak connectivity, with intermittent and often unreliable network connections. From the software developer's viewpoint, such an environment imposes the requirement of constantly preparing for failure. Applications with insufficient error handling will likely stall during a device or network outage, infinitely waiting for an answer packet or excessively consuming memory, power, or other resources.

Finally, IoT system topologies can be highly dynamic and ephemeral. For instance, factory environments might have a multitude of constantly moving pieces of equipment with

sensing capabilities. Or, they might produce products (or parts thereof) that stay within the factory perimeter only transiently. Situations such as these, combined with a large number of devices, will require programming and deployment technologies that can cope with dynamically changing "swarms" of devices.

Besides those basic differences, many additional issues arise from the IoT field's relative immaturity. Here are just two of them.

First, today's IoT systems have incompatible, immature development APIs. Common industry-wide APIs are still missing. Unlike in mobile-app and web application development, in which significant convergence has already occurred, IoT APIs still tend to be vendor- and hardware-specific. This severely hinders the creation of software that works across IoT devices and systems from different manufacturers and vendors. A number of standardization efforts are underway—for example, by the Industrial Internet Consortium, IPSO Alliance, Open Connectivity Foundation (formerly the Open Interconnect Consortium), and Open Mobile Alliance. However, standardization efforts will still take several years to reach consensus and maturity.

Finally, today's IoT systems are cloud-centric in that nearly all the data is collected in the cloud. Typically, most computation or actions on the collected data also take place on the cloud side. However, as IoT devices and gateways become more capable, computation can occur in various places—devices, gateways, or the cloud. Optimal IoT system behavior relies on the ability to migrate computation and data flexibly to those devices on which computations make the most sense at the time.

Implications and Challenges for Software Development

To summarize the differences we just presented, IoT developers must consider several dimensions that are unfamiliar to most mobile-app and client-side web application developers, including

- multidevice programming;
- the reactive, always-on nature of the system;
- heterogeneity and diversity;
- the distributed, highly dynamic, and potentially migratory nature of software; and
- the general need to write software in a fault-tolerant and defensive manner.

A typical IoT application is continuous and reactive. On the basis of observed sensor readings, computations get triggered (or retriggered) and eventually result in various actionable events. The programs are essentially asynchronous, parallel, and distributed.

Strictly speaking, these characteristics are by no means new in software development. Any developer who has built software for distributed or mission-critical systems is at least somewhat familiar with the challenges arising from these characteristics. The developers of cloud back-end server cluster software commonly face these challenges too.

Fallacies of Distributed Computing

Nevertheless, on the basis of our experience, we believe the average mobile-app or client-side web application developer isn't well equipped to cope with the challenges of IoT system development. As L. Peter Deutsch and James Gosling aptly summarized in their Fallacies of Distributed Computing, programmers invariably make eight false assumptions when writing software for distributed systems and applications for the first time:[17]

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.

Failure to take into account these false assumptions will result in various kinds of errors—for example, open sockets listening to devices that are no longer present, assuming immediate responses or infinitely waiting for reply packets, or consuming memory and batteries unnecessarily. As Leslie Lamport famously wrote, in a distributed system "the failure of a computer you didn't even know existed can render your own computer unusable."[18]

That said, the danger also exists that developers will have to write too much code to cope with any potential error and exception. This could bury the actual program logic under thousands of lines of error-handling boilerplate code, thus making programs much more difficult to understand and maintain. So, a major challenge in IoT development is to reach the proper balance between application logic and error handling. Ideally, the development languages and tools should facilitate this so that error-handling code doesn't clutter the program logic.

In general, the hidden costs of building and maintaining software for distributed systems are almost always underestimated. According to studies, verification and validation activities and checks might amount

to up to 75 percent of the development costs.[19]

Inadequate Languages and Tools

The programming languages and tools people use for IoT development are largely the same ones used for mainstream mobile-app and client-side web application development. For instance, the software development toolkits for today's popular IoT development boards—such as Arduino, Espruino, Intel's Edison and Galileo, and Tessel—provide the choice of C, C#, Java, JavaScript, or Python development.

Against all odds, JavaScript and Node.js (the server-side version of JavaScript) are becoming central tools for IoT development because of their popularity in web development. This is unfortunate because JavaScript wasn't designed for writing asynchronous, distributed applications or for programming-in-the-large. Recently, the expression "callback hell" has become popular in the context of JavaScript to characterize situations in which the application logic becomes impossible to follow because of asynchronous function calls and the separate event-handling functions used for writing success- and error-handling routines (see, for example, callbackhell.com). The **Promise** mechanism added in the ECMAScript 6 standard alleviates this problem. Nevertheless, it's fair to say that JavaScript is hardly optimal for developing any distributed software system.

The currently popular IoT development languages don't really address the programming-in-the-large aspects, either. That is, they provide neither facilities for orchestrating systems consisting of thousands of devices nor mechanisms that would let code migrate flexibly between the cloud, gateways, and devices.

The Dynamic Nature of IoT Systems

Software development has generally become much more agile and dynamic in the past 10 to 15 years. Most applications are connected to back-end services. Dynamic programming languages such as JavaScript and Python have gained popularity. The World Wide Web's emergence has enabled instant worldwide software deployment. Developers expect to be able to push updates to their applications at a dramatically faster pace than 10 years ago—often even several times an hour. The DevOps development-and-deployment model[20] has largely replaced earlier practices in automating software delivery and infrastructure.

Although these advances have benefited the software industry, the extremely dynamic nature of IoT systems poses additional challenges. For instance, debugging and testing IoT systems can be challenging because of the large number of devices, dynamic topologies, unreliable connectivity, and device heterogeneity and (sometimes) invisibility.

Although individual IoT devices might have limited data collection functionality and thus be reasonably easy to test, the testing of an entire system consisting of hundreds or thousands of devices deployed in complex real-world environments (factories, malls, greenhouses, ships, and so on) can be very challenging. Debugging and testing become even more complicated if the system can self-adjust and balance (trade off) computation speed, network latency, and device battery consumption by migrating computation dynamically between the cloud, gateways, and devices. In this case, system behavior might not ever be fully repeatable for testing and debugging.

Again, for the developers of distributed or mission-critical software or process automation systems, these challenges aren't necessarily new. However, the vast majority of mobile-app and client-side web application developers haven't faced such challenges and thus are likely to underestimate the effort and potential problems associated with debugging and testing.

Looking Ahead—While Learning from the Past

The previous observations and challenges reveal that the emergence of the Programmable World will require much more than just new hardware development, new communication protocols and technologies, or new techniques to digest and analyze massive datasets. To harness the Programmable World's full power, we'll need new software engineering and development technologies, processes, methodologies, abstractions, and tools. Past experience in and technologies for developing distributed systems and mission-critical software play an important role in avoiding duplicated work and reinventing the wheel. Largely, the challenges need to be addressed by educating software developers to realize that IoT development truly differs from mobile-app and client-side web application development.

Security

A major challenge to realization of the Programmable World is security. The remote management of complex installations of IoT devices in environments such as factories, power plants, or oil rigs obviously requires the utmost attention to security. Remote actuation and programming capabilities can pose high security risks. Cryptographic protocols for

transport layer security, security certificates, physical isolation, and other established industry practices play a critical role in this area, but various interesting technical challenges remain. However, perhaps the most significant practical security risk is that thousands of IoT devices still have their standard security settings, often with the default admin password. In Finland, incidents have already occurred in which the heating systems of entire apartment blocks were turned off remotely because their control systems were left exposed to Internet-based attacks.

Discussion

At the moment, IoT systems' programming features mainly take the form of custom apps from different manufacturers to control equipment specific to their ecosystems. For instance, Philips lightbulbs can be controlled with the Philips Hue app, whereas GE and Cree provide separate apps for their lighting systems. Correspondingly, air-conditioning systems, security systems, home-media control systems, and car remote-control applications from different manufacturers all typically require separate apps. The "separate app for every thing" approach doesn't scale well and is bound to be only a temporary solution until industry-wide standards become available.

We foresee IoT development changing over the coming years. At the moment, no universal, interoperable software development environments exist that will let developers effortlessly write one IoT application that runs on all types of devices, let alone orchestrate and manage large, complex topologies and heterogeneous installations of such devices. The lack of such tools reflects the IoT market's immaturity and fragmentation, as well as the devices' technical limitations—for example, constrained CPU power or the inability to update software without attaching physical cables to the devices. Virtual machines, dynamic language runtimes, and "liquid" software techniques will play a fundamental role in enabling flexible transfer of computation and data. (Liquid software operates seamlessly on multiple devices owned by one or more users.) Traditional binary software and hardware-specific IoT development kits are at a significant disadvantage if the same code must be executable (or adaptable to execution) on a variety of devices.

The Programmable World presents exciting opportunities and challenges. We hope that this article—and particularly our roadmap—inspires and encourages people to work hard in this area.

ABOUT THE AUTHORS

ANTERO TAIVALSAARI is a Fellow at Nokia. His research interests include software development for the Internet of Things, digital health, and liquid multidevice software. Taivalsaari received a doctorate in information technology from the University of Jyväskylä. Contact him at antero.taivalsaari@nokia.com.

TOMMI MIKKONEN is a professor of software engineering at the University of Helsinki and a technologist at Mozilla Connected Devices. His research focuses on software architectures, agile methodologies, web technologies, and connected devices. Mikkonen received his doctorate in information technology from the Tampere University of Technology. Contact him at tommi@mozilla.com.

Acknowledgments

The Academy of Finland (project 295913), Mozilla, and Nokia Technologies have supported this research.

References

1. R. Stackowiak et al., *Big Data and the Internet of Things: Enterprise Information Architecture for a New Age*, Apress, 2015.
2. P. Selonen and A. Taivalsaari, "Kiuas: IoT Cloud Environment for Enabling the Programmable World," *Proc. 42nd Euromicro Conf. Software Eng. and Advanced Applications* (SEAA 16), 2016, pp. 250–257.
3. F. Ahmadighohandizi and K. Systä, "Application Development and Deployment for IoT Devices," to be published in *Proc. 4th Int'l Workshop Cloud for IoT* (CLIoT 16), 2016.
4. A. Gallidabino et al., "On the Architecture of Liquid Software: Technology Alternatives and Design Space," *Proc. 13th Working IEEE/IFIP Conf. Software Architecture* (WICSA

16), 2016; ieeexplore.ieee.org/document/7516819.
5. J. Miranda et al., "From the Internet of Things to the Internet of People," *IEEE Internet Computing*, vol. 19, no. 2, 2015, pp. 40–47.
6. "Internet of Things History," Postscapes, 2014; postscapes.com/internet-of-things-history.
7. A. Al-Fuqaha et al., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, 2015, pp. 2347–2376.
8. "IoT Cloud Platform Landscape," Postscapes, 2016; www.postscapes.com/internet-of-things-platforms.
9. G. Moore, *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers*, 3rd ed., HarperBusiness, 2014.
10. O. Said and M. Masud, "Towards Internet of Things: Survey and Future Vision," *Int'l J. Computer Networks*, vol. 5, no. 1, 2013, pp. 1–17.
11. J. Gubbi et al., "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, vol. 29, no. 7, 2013, pp. 1645–1660.
12. B. Wasik, "In the Programmable World, All Our Objects Will Act as One," *Wired*, 14 May 2013; www.wired.com/gadgetlab/2013/05/internet-of-things/all.
13. R. Riggs, A. Taivalsaari, and M. VandenBrink, *Programming Wireless Devices with the Java 2 Platform, Micro Edition*, Addison-Wesley, 2001.
14. S. Greengard, *The Internet of Things*, MIT Press, 2015.
15. N. Balani, *Enterprise IoT: A Definitive Handbook*, CreateSpace, 2015.
16. D. Ingalls et al., "The Lively Kernel: A Self-Supporting System on a Web Page," *Self-Sustaining Systems*, LNCS 5146, Springer, 2008, pp. 31–50.
17. A. Rotem-Gal-Oz, "Fallacies of Distributed Computing Explained"; www.rgoarchitects.com/Files/fallacies.pdf.
18. L. Lamport, email, 28 May, 1987; research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt.
19. J.-C. Laprie, "Dependable Computing: Concepts, Limits, Challenges," *Proc. 25th IEEE Int'l Symp. Fault-Tolerant Computing*, 1995, pp. 42–54.
20. P. Debois, "DevOps: A Software Revolution in the Making?," *Cutter Business Technology J.*, 30 Aug. 2011; www.cutter.com/article/devops-software-revolution-making-416511.