Roadmap to the Programmable World: Software Challenges in the IoT Era

Antero Taivalsaari
Nokia Technologies
FI-33100 Tampere, Finland
Email: antero.taivalsaari@nokia.com

Tommi Mikkonen
Tampere University of Technology
FI-33720 Tampere, Finland
Email: tommi.mikkonen@tut.fi

February 8, 2017

Abstract

The Internet of Things (IoT) represents the next significant step in the evolution of the Internet and software development. Although the majority of current work in the IoT area focuses on data acquisition, analytics and visualization, there is a more subtle but equally important transition underway. Advances in hardware development are making it possible to embed full-fledged virtual machines and dynamic language runtimes virtually everywhere, thus leading us to a *Programmable World* in which all the everyday things around us will become connected and programmable dynamically.

The emergence of millions of remotely programmable devices in our surroundings will pose significant challenges for software development. In this paper we present a roadmap from today's cloud-centric, data-centric IoT systems to the Programmable World, highlighting technical challenges that deserve developer education and deeper investigation above and beyond those topics that receive the most attention in the IoT area today.

1 Introduction

The Internet of Things (IoT) represents the next significant step in the evolution of the Internet. In the early days in the 1970's and 1980's, the Internet was primarily about *connecting computers*. In the 1990's and 2000's, the Internet was all about *connecting people*. In contrast, in the 2010's and 2020's the focus shifts towards *connecting everything* (or literally every *thing*) to the Internet.

Today, the majority of research work in the IoT area revolves around data acquisition, real-time and offline analytics, machine learning, data visualization and other big data topics [1]. The focus on these topics is not surprising, given the massive business potential arising from the ability to collect data from millions of sensing devices, and then digest and combine the data to provide new insights into people's behavior and other real-world phenomena.

However, there is an *equally important but more subtle and tranquil disruption underway*. Advances in hardware development and the general availability of powerful but very inexpensive integrated chips will make it possible to embed connectivity and full-fledged virtual machines and dynamic language runtimes virtually everywhere – or literally everywhere. As a consequence, everyday things in our surroundings – light bulbs, door knobs, air conditioning systems, lawn sprinklers, vacuum cleaners, toothbrushes and the kitchen sink – will become connected and programmable dynamically. The future potential of this disruption will be every bit as significant as the mobile application revolution that was sparked when similar technological advances made it possible to open up mobile phones for third-party application developers in the early 2000's.

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

Since this is a forward-looking paper, the roadmap presented in this paper is to some extent subjective. The viewpoints presented in this paper stem from our own projects and collaboration efforts in the IoT area [2, 3, 4, 5] as well as from our past experiences in predicting and partaking in the evolution of mobile and web computing over the past twenty years. For instance, the emergence of virtual machines in mobile phones in the late 1990's – while technically not a dramatic achievement – opened up

mobile phones for the vast masses of developers, creating today's multi-billion mobile application industry. Although history rarely repeats itself, it often rhymes; in this case parallels with the past seem especially obvious as we are just reaching a point in which it will become possible to embed dynamic programming capabilities virtually everywhere.

2 The Emerging Common E2E IoT Architecture

The term *Internet of Things* is not new. Over twenty 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 started emerging around enabling technologies such as RFID and wireless sensor networks [7]. However, in recent years the usage of the concepts has spread rapidly into various new 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 Postscapes.com study listed 115 IoT cloud platforms (`http://postscapes.com/internet-of-things-platforms/`). We are currently witnessing a classic "crossing the chasm" period in which the ultimate winners have not been determined yet [8]. Most likely, by 2025 there will be only a handful of dominant IoT platforms and platform vendors left.

What is interesting about these IoT offerings is how similar they are at the conceptual level. In spite of the apparent diversity and the huge number of vendors targeting the IoT market, there is a common end-to-end (E2E) architecture emerging for IoT solutions, with a number of elements that are pretty much the same in all the systems [1, 9, 10]. The basic concepts are depicted in Fig. 1 and summarized below.

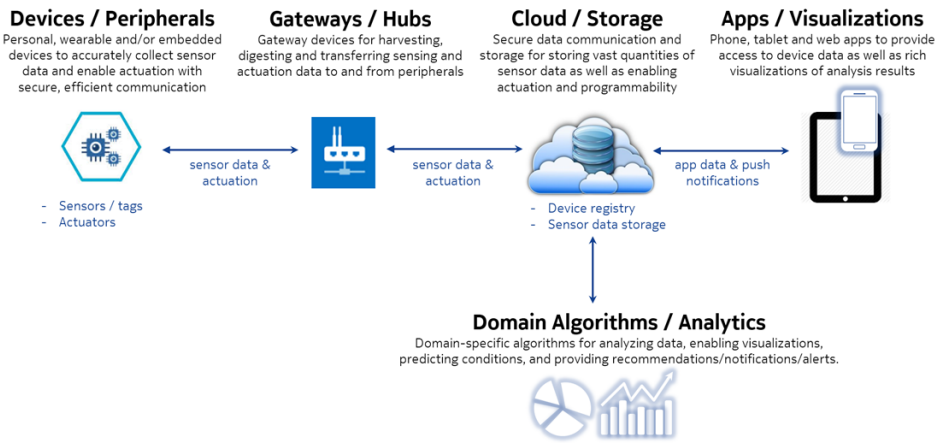

Figure 1: The Emerging Common E2E IoT Architecture.

Devices or **peripherals** are the physical hardware elements that collect sensor data and/or perform actuation, with built-in communication capabilities to submit the collected data to the broader IoT ecosystem. From a functional point of view, devices can be divided broadly into two categories:

1. *Sensors* provide information about the physical entity they monitor. Collected information may range from the identity of the physical entity to measurable qualities such as temperature, humidity, pressure, luminosity, sound level, fluid flow, vibration, abrasion, and so on. Sensors whose sole purpose is to facilitate an identification process are commonly known as *tags*.

2. *Actuators* are components that modify the state of a physical entity by taking energy, usually transported by air, electric current or liquid, and converting that energy into a state change, thus affecting one or more physical real-world entities.

Gateways or **hubs** are devices for collecting, preprocessing and transferring data from peripheral IoT devices and their sensors, utilizing different (usually wireless) communication protocols such as Wi-Fi or Bluetooth Smart. Gateways provide secure protocol translation between peripheral IoT devices and cloud, and may support additional tasks such as intermediate sensor data storage and preprocessing, service discovery, geo-localization, verification and billing. In addition, gateways also deliver the actuation requests from the cloud to the devices.

In some systems, the IoT devices themselves are capable of uploading sensing data directly to the cloud, as well as receiving actuation requests directly from the cloud, e.g., via Wi-Fi, 3G/4G or soon NB-IoT and 5G networks. In such solutions, no dedicated gateway devices are required at all. Furthermore, it is not uncommon to have IoT solutions in which IoT devices themselves are capable of acting as gateways to other devices, possibly forming peer-to-peer (P2P) / mesh topologies via local connectivity.

Cloud computing and **cloud-based storage and analytics** solutions play a central role in most IoT platforms today. The main roles of the cloud in the generic end-to-end IoT architecture can be summarized as follows.

1. *Data acquisition, storage and access.* A fundamental functionality area 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 in-premise databases to massive replicated, fault-tolerant, scalable storage clusters. Query and notification APIs for accessing collected data are provided as well.

2. *Data analytics.* Data analytics refers to the process of examining, cross-connecting and transforming acquired sensor data in order to

discover and present useful information, e.g., 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*, in contrast, refers to batch-style operations that are performed after large datasets have already been accumulated. Machine learning and data mining technologies and algorithms play an important role in this area.

3. *Actuation support.* Data flows in IoT systems are not just unidirectional. In addition to sensor data collection from the devices to the cloud, secure device actuation from the cloud to the devices is also important. Actuation capabilities are a fundamental enabler for remote device programmability.

In addition to the core IoT cloud features discussed above, a cloud solution supporting IoT solutions typically includes a number of administrative functions such as device management, user account management, usage logging, server status monitoring, reporting capabilities, and so on.

3 Roadmap to the Programmable World

Looking above and beyond data acquisition, analytics, data visualization and other currently prominent IoT topics, we believe that the future of the Internet of Things lies in the *ability to orchestrate and program large, complex topologies of IoT devices remotely.* As argued by Wasik [11], once devices are connected to public or private clouds, with sensor data flowing in and actuation capabilities being widely available, focus will eventually shift from sensor data collection and analytics features to application programming capabilities that can be used for manipulating complex real-world systems. Actuation capabilities offered by IoT devices form the foundation for all this. These capabilities will literally put the world at our fingertips, making it possible 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.

The transition towards the Programmable World will be driven by advances in hardware development. IoT hardware capabilities and price points are rapidly reaching an inflection point in which it will be possible to run full-fledged operating systems such as Linux or virtualized software stacks or dynamic language runtimes on almost any type of device. The low-cost computing chips that have become available in the mid-2010's already match or exceed the memory and processing power capabilities that mobile phones had in the late 1990's, just before the emergence of the Java 2 Platform, Micro Edition (J2ME) [12] launched the industry-wide mobile application revolution leading to today's multi-billion mobile application industry. We

take it for granted that similar cross-manufacturer programming capabilities will soon find their way to everyday objects and things around us.

Another major trend driving the industry towards the Programmable World is the emergence of edge computing. Classic cloud computing systems are highly centralized. While centralized computing has significant benefits, it can also be very costly in terms of communication and power consumption. For instance, if an IoT environment consists of a large collection of devices that are in close proximity of each other, it may be inefficient to transmit all the data from those devices to a faraway data center for processing, and then transmit actuation requests back from the remote data center to the individual devices. In an IoT deployment with tight latency requirements, latency overhead alone can make such solutions impractical.

The term *edge computing* was coined around 2002, and it was originally associated with the deployment of applications over Content Delivery Networks (CDNs) – the main objective was to benefit from the proximity of CDN edge servers to achieve better scalability and lower latency. Edge computing IoT solutions harness the edge of the network (usually gateway devices such as routers or base stations) for computation, e.g., to preprocess sensor data and trigger alerts and actuation requests locally based on predefined criteria. Such systems may leverage *local connectivity* technologies (e.g., Bluetooth LE or Wi-Fi Direct) to enable direct, more efficient and decentralized communication between sensor devices. *Virtualization* technologies also play a central role in enabling the migration of computation between different entities.

Table 1 presents an anticipated roadmap predicting how IoT systems will evolve over the next ten years. The table has been divided into two columns: one looking at the evolution from the data viewpoint, and another looking at the evolution from the programming angle. In this paper, the primary focus is on programming aspects, so in the remainder of the paper we will dive deeper only into challenges related to programmability.

The table is based on a number of sources:

- observed trends within the industry and in the academia, reflecting recent theses, academic papers and books (e.g., [1, 5, 7, 9, 10, 13, 14]);
- expert views, building on practical product and prototype development experiences in both industry and academia [2, 3];
- personal experiences and past observations from the mobile application revolution as mobile phones evolved from closed, voice-centric devices to application-rich smartphones [12]; and
- personal experiences and past observations from the evolution of the Web from a simple document distribution environment to a platform that supports rich application development and instant worldwide deployment [15].

Table 1: Anticipated Evolution of IoT Systems

Year	Data Viewpoint	Programmability Viewpoint
2015 —	• Data acquisition already possible on a massive scale • Monitor, track, route, command and control, mining for trends and behaviors • Cloud-centric data analytics, including both real-time and offline analytics support • A lot of focus on data visualization and simple IFTTT (If-This-Then-That) alerts • Open source technologies available and widely used for implementing data acquisition and analytics features	• The majority of serious computation in IoT systems performed in the cloud • Devices support 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 (e.g., NodeRED) emerging for implementing device control applications in a more portable fashion • Standards emerging (e.g., OMA LWM2M, 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, utilizing local connectivity • Adapt, enhance, extend; automatic determination and selection of computing and analytics resources (cloud vs. edge)	"Edge IoT Era" • 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, e.g., 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 computations 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	"Universal IoT Era" • Universal, containerized application deployment and execution model supported across multiple manufacturers and industries • Industry-wide, cross-manufacturer programming APIs allowing device discovery, data acquisition, remote device programming and device management in a generalized fashion • "Universal remote control" applications and "universal device consoles" possible • Dynamic remote (re)programming of devices widely supported, enabled by the widespread use of virtual machines, containerization and common developer APIs

Basically, we expect the evolution of the IoT programming capabilities to progress from simple actuation features, vendor-specific apps and device APIs, and cloud-centric data acquisition and processing to systems that leverage edge computing, virtualization and containers extensively. Such future systems will support portable, cross-manufacturer and cross-industry application development, and allow – whenever required – flexible migration of computation and data between the cloud and heterogeneous edge devices.

While it is debatable whether there will ever be a single set API to cover IoT devices from entirely different domains and verticals, it is safe to predict that in five to ten years IoT devices and their APIs will have converged significantly. It is simply impractical for people to use a large number of vendor-specific apps to control all the devices in their surroundings. It is also likely that the necessary infrastructure will grow around the already existing IP networking and web infrastructure, enhanced with local connectivity technologies in order to support edge computing.

4 How is IoT Development Different from Mobile and Web Application Development?

Below we summarize the basic differences between IoT development and mainstream mobile and client-side web application development today. These differences are a key reason for the implications and technical challenges presented later.

- **IoT devices are part of a system**. IoT devices are almost always *part of a larger system of devices*, e.g., an installation of interoperating devices in a home, office, factory, shopping mall or in an airplane. Furthermore, IoT devices are usually just a small part of an end-to-end architecture presented in Section 2. Granted, PCs and smartphones today have major dependencies with cloud-based services as well, but from the developer's perspective they are still primarily seen as standalone devices, with the application developer targeting a single computer or smartphone when writing software.

- **Rebootables vs. systems that never sleep**. Personal computers, 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 are "*systems that never sleep*" that in most cases should not or cannot be shut down in their entirety. Although individual devices may be shut down, the system in its entirety must be designed to be resilient to device and network outages.

- **Pets vs. cattle**. The number of *computing units (devices/CPUs) in IoT systems is often dramatically larger* than in traditional computing

environments, consisting potentially of hundreds, thousands or even millions of units. Unlike personal computers and smartphones that are seen by the users almost like "pets", IoT devices in a large system are more like "cattle" that must be managed *en masse* instead giving them personal attention and care.

- **IoT devices are embedded and often invisible**. IoT devices are often embedded in our surroundings in a manner that makes them *physically invisible and unreachable*. Devices may be permanently buried deep in the ground or physically embedded in various materials (e.g., vibration sensors placed in mining equipment). It may be impossible to attach physical cables to those devices, or replace hardware or embedded software when problems arise.

- **IoT systems are highly heterogeneous**. Computing units in an E2E IoT system 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, while many devices present no visible user interface at all.

- **IoT systems have weak connectivity**. IoT devices tend to be *weakly connected, with intermittent and often unreliable network connections*. From the software developer's viewpoint, such an environment places special requirements on constantly preparing for failure. Applications written with insufficient error handling are likely to stall during a device or network outage, infinitely waiting for an answer packet, excessively consuming memory, battery or other resources.

- **IoT systems have dynamic topologies**. In IoT systems, *device topologies can be highly dynamic and ephemeral*. For instance, in factory environments there may be countless of constantly moving pieces of equipment with sensing capabilities, or products (or parts thereof) that stay within the factory perimeter only transiently. When combined with the much larger number of devices overall, this will require programming and deployment technologies that can cope with dynamically changing "swarms" of devices.

In addition to the basic differences listed above, there are many additional issues that arise from the *relative immaturity of the IoT area*. Such differences include:

- **Today's IoT systems have incompatible, immature development APIs**. At this stage, common industry-wide IoT developer APIs are still missing. Unlike in mobile and web application development where significant convergence has already taken place, IoT

APIs still tend to be vendor- and hardware-specific. This severely hinders the creation of software that would work across IoT devices and systems from different manufacturers and vendors. A number of standardization efforts are underway, e.g., by the *Industrial Internet Consortium*, *IPSO Alliance*, *Open Connectivity Foundation* (formerly *Open Interconnect Consortium*), and *Open Mobile Alliance (OMA)*. However, it will still take several years until standardization efforts reach consensus and maturity.

- **Today's IoT systems are very cloud-centric**. Today's IoT systems are highly *cloud-centric* in the sense that nearly all the data is collected in the cloud; most of the computation/actions on the collected data are typically performed on the cloud side as well. However, as IoT devices and gateways become more capable, it becomes possible to perform computation in various places – in devices/peripherals, in gateways/hubs or in the cloud. Optimal behavior of IoT systems relies on the ability to migrate computation and data flexibly to those devices where computations make most sense at each time.

5 Implications and Challenges for Software Development

To summarize the differences presented above, an IoT developer needs to consider several dimensions that are unfamiliar to most mobile and client-side web developers, including: (1) multi-device programming, (2) reactive and always-on nature of the system, (3) heterogeneity and diversity, (4) distributed, highly dynamic and potentially migratory nature of software, and (5) the general need to write software in a fault-tolerant and defensive manner. A typical IoT application is *continuous* and *reactive* in nature. Based on observed sensor readings, computations get (re)triggered and eventually result in various actionable events. The programs are essentially *asynchronous*, *parallel* and *distributed*.

Strictly speaking, these are by no means new characteristics in software development. Any developer who has built software for distributed systems or mission-critical systems is at least to some degree familiar with the challenges arising from such qualities. The developers of today's cloud backend server cluster software are commonly faced with these aspects as well.

Tackling the fallacies of distributed computing. Based on our experiences the average mobile or client-side web application developer today is not well-equipped to cope with challenges presented by IoT systems development. As L. Peter Deutsch aptly summarized in his *fallacies of distributed computing* back in 1994, there is a set of (false) assumptions that programmers will invariable make when writing software for distributed sys-

tems and applications for the first time. Unfortunately, the original Sun Microsystems blog site summarizing the fallacies (`http://blogs.sun.com/jag/resource/Fallacies.html`) is no longer online, but material on the topic is readily available on the Web [16].

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology does not change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Failure to take into account these false assumptions will result in various kinds of errors, e.g., open sockets listening to devices that are no longer present, assuming immediate responses or infinitely waiting for reply packets, and consuming memory and battery unnecessarily. As Leslie Lamport once famously wrote, in a distributed system a "failure of a computer you did not even know existed can render your own computer unusable".

That said, there is also a danger that the developer will have to write *too much code* to cope with any potential error and exception, burying the actual program logic under thousands of lines of error handling "boilerplate" code and thus making programs much more difficult to understand and maintain. Therefore, 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 does not unnecessarily 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 may amount up to 75% of the total software development costs [17].

Inadequate languages and tools for programming and orchestrating IoT systems. The programming languages and development tools that people use for IoT development today are largely the same that are used for mainstream mobile and client-side web application development. For instance, the software development toolkits available for today's popular IoT development boards – for instance, Arduino, Espruino, Intel's Edison and Galileo, and Tessel – provide a choice between C, C#, Java, JavaScript or Python development.

Against all odds, JavaScript and Node.js (the server-side version of JavaScript) – because of their popularity in web development – are becoming central tools for IoT development. This is rather unfortunate, since JavaScript was not originally designed for writing asynchronous, distributed applications, or for programming-in-the-large more broadly. In recent years, the expression *"callback hell"* was popularized in the context of JavaScript to characterize situations in which application logic becomes impossible to follow because of asynchronous function calls and the separate event handler functions that are used for writing the success and error handler routines (see, e.g., `http://callbackhell.com/`). The *Promise* mechanism added in the ECMAScript 6 standard alleviates this problem, but nevertheless it is fair to say that JavaScript is hardly an optimal development language for any distributed software system.

The currently popular IoT development languages do not really address the programming-in-the-large aspects either, i.e., they do not provide any facilities for orchestrating systems that consists of thousands of devices, or mechanisms that would allow code to be flexibly migrated between the cloud, gateways and IoT devices.

The dynamic nature of IoT systems poses additional challenges. Software development has generally become much more agile and dynamic in the past 10-15 years. Most applications are connected to backend services nowadays. Dynamic programming languages such as JavaScript and Python have gained popularity. The emergence of the World Wide Web has enabled instant worldwide software deployment. Developers expect to be able to push updates to their applications on a dramatically faster pace than ten years ago – often even several times a day. The DevOps development and deployment model [18] has largely replaced earlier practices in automating the process of software delivery and infrastructure.

While these advances have been largely beneficial to the software industry, the extremely dynamic nature of IoT system poses additional challenges. For instance, *debugging and testing of IoT systems can be very challenging* because of the large number of devices, dynamic topologies, unreliable connectivity, and heterogeneous and sometimes invisible nature of the devices.

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

Again, for the developers of distributed or mission-critical software, or for the developers of process automation systems these are not necessarily new challenges. However, the vast majority of mobile and client-side web developers have not faced these kinds of challenges, and thus they are likely to underestimate the effort and potential problems associated with debugging and testing.

Looking ahead – while learning from the past. The observations and challenges above 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 full power of the Programmable World, new software engineering and development technologies, processes, methodologies, abstractions and tools will be required. Past experiences and technologies for the development of distributed systems and mission-critical software play an important role in avoiding duplicated work and reinventing the wheel. To a large degree the challenges need to addressed by *educating software developers to realize that IoT development truly is different from mobile and client-side web application development.*

Security. Last but not least, it is obligatory to note that a major challenge for the realization of the Programmable World vision is *security.* The remote management of complex installations of IoT devices in environments such as factories, power plants or oil rigs is obviously something that requires utmost attention on security. Remote actuation and programming capabilities can pose very 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.

6 Putting It All Together

Advances in hardware development and the general availability of powerful but very inexpensive integrated chips will make it possible to embed connectivity and full-fledged virtual machines and dynamic language runtimes everywhere, thus leading us to a *Programmable World* in which everyday things around us will become universally connected and programmable. We believe that the future potential of the Programmable World disruption will be every bit as significant as the mobile application revolution that was sparked when similar, seemingly subtle hardware advances and small-footprint virtual machines made it possible to open up mobile phones for third-party application developers in the early 2000's.

At the moment, the programming features in IoT systems are provided mainly in the form of *custom apps* that are emerging from different manufacturers to control equipment that is specific to their ecosystem. For instance,

lightbulbs produced by Philips can be controlled with a Philips Hue app, while GE or 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 does not scale well and is bound to be only a temporary solution until industry-wide standards become available.

We foresee the nature of IoT development changing over the coming years. At the moment, there are no universal, interoperable software development environments that would allow a developer to effortlessly write a single IoT application that would be capable of running 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 current immaturity and fragmentation of the IoT market, as well as the technical limitations of the devices, e.g., 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. Traditional binary software and hardware-specific IoT development kits are at a significant disadvantage if the same code needs to be executable (or adaptable to execution) on a broad variety of devices.

In this paper we have presented a roadmap for IoT system evolution from today's cloud-centric, data-driven IoT systems to systems in which the edge of the network becomes universally programmable. Furthermore, we highlighted important software development challenges and implications that necessitate developer education and deserve deeper investigation above and beyond those topics that receive the most attention in the IoT area today. We are excited by the opportunities and challenges presented by the Programmable World and we hope that this paper – for its part – encourages people to work hard in this area.

References

[1] R. Stackowiak, A. Licht, V. Mantha, and L. Nagode, *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," in *Proceedings of the 42nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'2016, Limassol, Cyprus, August 31 - September 2, 2016).*

[3] F. Ahmadighohandizi and K. Systä, "Application Development and Deployment for IoT Devices," in *4th Workshop on CLoud for IoT (CLIoT 2016), to appear*.

[4] A. Gallidabino, C. Pautasso, V. Ilvonen, T. Mikkonen, K. Systa, J.-P. Voutilainen, and A. Taivalsaari, "On the Architecture of Liquid Software: Technology Alternatives and Design Space," in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)*, Venice, Italy, April 2016.

[5] J. Miranda, N. Mäkitalo, J. Garcia-Alonso, J. Berrocal, T. Mikkonen, C. Canal, and J. M. Murillo, "From the Internet of Things to the Internet of People," *IEEE Internet Computing*, vol. 19, no. 2, pp. 40–47, 2015.

[6] Postscapes, "A Brief History of the Internet of Things," 2014. [Online]. Available: http://postscapes.com/internet-of-things-history

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[8] G. Moore, *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers, 3rd Edition.* HarperBusiness, January 2014.

[9] O. Said and M. Masud, "Towards Internet of Things: Survey and Future Vision," *International Journal of Computer Networks*, vol. 5, no. 1, pp. 1–17, 2013.

[10] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[11] B. Wasik, "In the Programmable World, All Our Objects Will Act as One," *Wired*, 2013. [Online]. Available: http://www.wired.com/gadgetlab/2013/05/internet-of-things/all

[12] R. Riggs, A. Taivalsaari, and M. VandenBrink, *Programming Wireless Devices with the Java 2 Platform, Micro Edition.* Pearson Education, Java Series, 2001.

[13] S. Greengard, *The Internet of Things.* MIT Press, 2015.

[14] N. Balani, *Enterprise IoT: A Definitive Handbook.* CreateSpace Independent Publishing, 2015.

[15] D. Ingalls, K. Palacz, S. Uhler, A. Taivalsaari, and T. Mikkonen, "The Lively Kernel: A Self-Supporting System on a Web Page," in *Self-Sustaining Systems (S3'2008, Potsdam, Germany)*. Springer LNCS5146, 2008, pp. 31–50.

[16] A. Rotem-Gal-Oz, "Fallacies of Distributed Computing Explained," http://www.rgoarchitects.com/Files/fallacies.pdf, [Online; accessed 20-April-2016].

[17] J.-C. Laprie, "Dependable Computing: Concepts, Limits, Challenges," in *Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing*. IEEE, 1995, pp. 42–54.

[18] P. Debois, "Devops: A Software Revolution in the Making," *Journal of Information Technology Management*, vol. 24, no. 8, pp. 3–39, 2011.