

On Internet of Things Programming Models

Dmitry Namiot¹(✉) and Manfred Sneps-Sneppé²

¹ Faculty of Computational Mathematics and Cybernetics,
Lomonosov Moscow State University, GSP-1, 1-52,
Leninskiye Gory, Moscow 119991, Russia
dnamiot@gmail.com

² Institute of Mathematics and Computer Science,
University of Latvia, Raina Bulvaris 29, Riga 1459, Latvia

Abstract. In this paper, we present the review of existing and proposed programming models for Internet of Things (IoT) applications. The requests by the economy and the development of computer technologies (e.g., cloud-based models) have led to an increase in large-scale projects in the IoT area. The large-scale IoT systems should be able to integrate diverse types of IoT devices and support big data analytics. And, of course, they should be developed and updated at a reasonable cost and within a reasonable time. Due to the complexity, scale, and diversity of IoT systems, programming for IoT applications is a great challenge. And this challenge requires programming models and development systems at all stages of development and for all aspects of IoT development. The first target for this review is a set of existing and future educational programs in information and communication technologies at universities, which, obviously, must somehow respond to the demands of the development of IoT systems.

Keywords: Internet of Things · Smart Cities · Streaming · Sensor fusion · Programming · Education

1 Introduction

The Internet of Things (IoT) world is becoming an important direction for technology development. In general, the IoT promotes a heightened level of awareness about our world. IoT plays a basic role in many other things. For example, IoT is a base for Smart Cities, etc.

IoT ecosystem is currently presented by multiple (sometimes - competing) technologies and platforms. IoT platforms (at least, nowadays) are varied across the vertical and horizontal segments of the markets. Of course, it complicates and delays the development and deployment, makes the support of IoT systems more expensive than it should be, etc. So, IoT standards are highly demanded [1].

In the same time, it is a very competitive area. We cannot expect that a general solution will be agreed upon by all players. Standards proposals in IoT (and M2M) come from formal standards development organizations (e.g., the European

Telecommunications Institute - ETSI) or non-formal groups (the Institute of Electrical and Electronics Engineers - IEEE). Standards can target the connectivity for a particular set of devices (e.g., Bluetooth Low Energy) or provide common application interfaces up to developers (e.g., oneM2M) [2]. In this paper, we would like to discuss the common elements of IoT programming models and perform this review from the perspective of educational programs.

The rest of the paper is organized as follows. In Sect. 2, we discuss programming systems for IoT. In Sect. 3, we discuss data models, data persistence, data processing and educational programs for IoT. In Sect. 4, we discuss cloud computing for IoT.

2 IoT Programming Models

The choice of programming languages for IoT platforms does not depend on a hardware platform. Also, new hardware platforms make programming embedded (nowadays - cyber-physical) systems easier. Even more, the diversity in hardware platforms enhances the interest to the platform independent on hardware.

Standards for the IoT could be classified as downward-facing standards that establish connectivity with devices and upward-facing standards that provide common application interfaces up to end users and application developers.

By our opinion, confirmed by the practical experience and academic papers, the key moment for software development in telecom and related areas (IoT is among them) is time to market indicator [3]. The main question to any software standard is the generalization. Shall the standard follow to the “all or nothing” model and covers all the areas of the life cycle? In software standards, the excessive generalization (unification) could be the biggest source of the problems. Actually, all the standards should make its implementation by the most convenient way for developers. Because only the developers are finally responsible for the putting new services in place.

It is especially true for such areas as IoT or Smart Cities. The services here are not finalized (and it is very probably that they could not be finalized at all). This means that we will constantly try (test) new services and to refuse from the old ones. Naturally, this process needs to be fast and inexpensive. As the next step, it means that the most of IoT application could be described as mashups [4]. Mashups use data from several data sources. On programming level, it should stimulate the interest in scripting languages and to the systems for fast prototyping. In the modern software architecture world, we can mention also micro-services approach [5]. Of course, these directions should have an appropriate reflection in educational programs.

Another direction, which is very close to mashups, actually, is so-called Data-as-a-Service (DaaS) approach [6]. In its technical aspects, DaaS is an information provision and distribution model in which data files (including text, images, sounds, and videos) are made available to customers over a network. The key moment is the separation for data and proceedings. It lets delivery data (e.g., in some open format, like JSON), rather than some API with the predefined model for data processing.

The next significant visible trend is the growing interest in the dynamic languages. And the perfect example here is JavaScript. We can mention the following reasons for JavaScript in IoT applications [7]. At the first hand, there is a big army of web developers. So, the entry level for programming is low. Most of the Internet applications already use JavaScript. JavaScript nowadays covers both server-side and client-side programming. It could be useful to use the same language across the whole project. So, it makes sense to extend the same standard platform to the Internet of Things, communicating to a larger set of devices using the same language.

JavaScript has matured as a language and international standards cover its extensions. JavaScript has a range of already existing libraries, plugins, etc. And what is also important, there is a huge Open Source community behind JavaScript.

Technically, this language has got a great support for event-driven apps. The nature of IoT project is mostly associated with asynchronous communications. And event-driven models are the most suitable solution. In JavaScript, it is very easy to implement models, where an application can receive and respond to events, then wait for a callback from each event that notifies us once it is complete. It lets respond to events as they happen, performing many tasks simultaneously as they come in.

Also, the recent development shows more and more direct involvement JavaScript into data processing. Actually, the winning data format (JSON) has its origin in JavaScript.

As a recent example of JavaScript in IoT, we can mention the developments from Samsung. Samsung Electronics recently opened the development of IoT.js, a web-based Internet of Things (IoT) platform that connects lightweight devices. Examples of lightweight devices include micro-controllers or devices with only a few kilobytes of RAM available [8]. The idea is to make all devices interoperable in the IoT space by enabling more devices to be interoperable, from complex and sophisticated devices such as home appliances, mobile devices, and televisions, to lightweight and small devices such as lamps, thermometers, switches, and sensors. The IoT.js platform is comprised of a lightweight version of the JavaScript engine, and a lightweight version of node.js. We think that JavaScript for IoT world should be in educational programs. This is very important because until now, this language is often seen as simple web pages scripting. But it's not for a long time already.

By the same reason, any attempt to replace JavaScript with the similar idea of portability could be also interested in IoT programming. In this connection, we should mention Dart programming language from Google [9].

3 Data Persistence and Processing

It terms of the data processing in IoT applications, we should pay attention in educational programs to the following two moments: sensor fusion and streaming. There are different data mining and data science approaches which are applicable

to IoT. And of course, they should be a subject of the separate courses for statistics, machine learning, etc. For example, in many cases, IoT (Smart City) measurements are time series. Of course, it should be a subject of a separate course among other data-mining techniques [10].

But one moment is important, in our opinion, and should be discussed separately for IoT applications. It is sensor fusion. Sensor fusion is combining of sensory data or data derived from disparate sources such that the resulting information has less uncertainty than would be possible when these sources were used individually [11]. It is illustrated in Fig. 1.

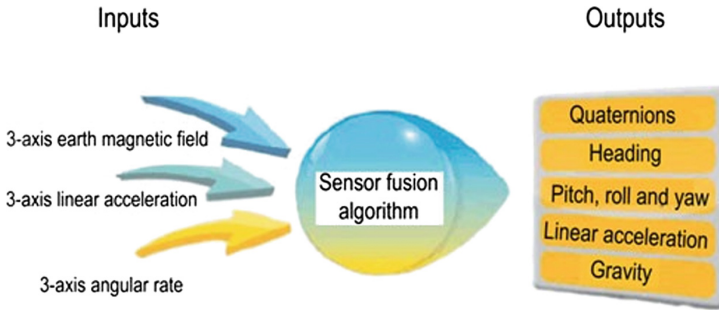


Fig. 1. Sensor fusion [12]

There are many ways of fusing sensors into one stream. Each sensor has its own strengths and weaknesses. The idea of sensor fusion is to take readings from each sensor and provide a more useful result which combines the strengths of each. Actually, such a fusion is the main idea for all IoT and Smart City projects, related to some measurements. The next big issue for IoT data processing is streaming. By our opinion, it is a key technology in data acquisition and proceeding for Smart Cities and IoT.

There are many tasks in IoT with the requirements for real-time (or near real-time) processing. In this case, the common architecture is associated with some messaging bus. And it is very important to present the software architectures associated with streaming. At the first hand, it is so-called Lambda Architecture [13]. Originally, the Lambda Architecture is an approach to building stream processing applications on top of MapReduce and Storm or similar systems (Fig. 2). Currently, we should link it to Spark and Spark streaming too [14].

The main idea behind this schema is the fact that an immutable sequence of source data is captured and fed into a batch system and a stream processing system in parallel. Of course, the negative impact of this decision is the need to implement business logic twice, once in the batch system and once in the stream processing system.

The Lambda Architecture targets applications built around complex asynchronous transformations that need to run with low latency. One proposed

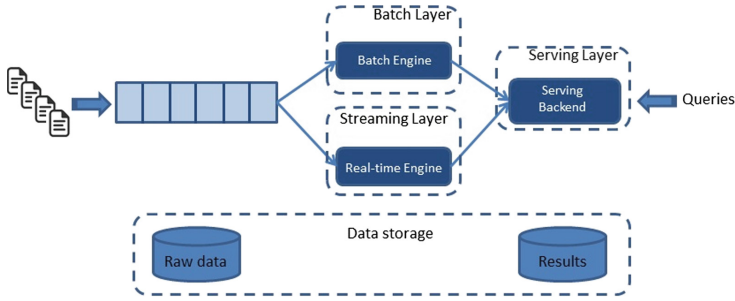


Fig. 2. Lambda architecture [15]

approach to fixing this is to have a language or framework that abstracts over both the real-time and batch framework [16].

Another solution here is so-called Kappa architecture [17]. The Kappa architecture simplifies the Lambda architecture by removing the batch layer and replacing it with a streaming layer (Fig. 3).

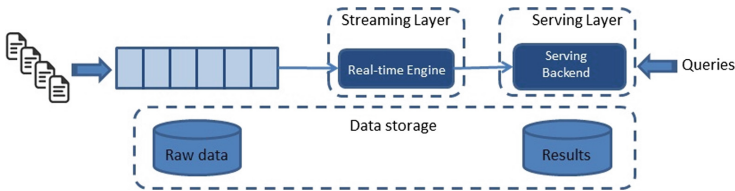


Fig. 3. Kappa architecture [15]

With Kappa, everything in the system is a stream. All batch operations become a subset of streaming operations. Data source (raw data) is persisted and views are derived. Of course, a state can always be recomputed where the initial record is never changed. This feature lets us support replay functionality. Computations and results can evolve by replaying the historical data from a stream. With Kappa, only a single analytics engine is required. It means that code is considerably reduced. Also, maintenance and upgrades are cheaper.

The hearth for such implementations is a scalable, distributed messaging system with events ordering and at-least-once delivery guarantees. At this moment, it is almost always Kafka system [18].

Apache Kafka is a distributed publish-subscribe messaging system. It is designed to provide high throughput persistent scalable messaging. Kafka allows parallel data loads into Hadoop. Its features include the use of compression to optimize performance and mirroring to improve availability, scalability. Kafka is optimized for multiple-cluster scenarios. In general, publish-subscribe architecture is the most suitable approach for mostly asynchronous measurements in IoT.

Technically, there are at least three possible message delivery guarantees in publish-subscribe systems:

- (1) At most once. It means that messages may be lost but are never redelivered.
- (2) At least once. It means messages are never lost but may be redelivered.
- (3) Exactly once. It means each message is delivered once and only once.

As per Kafka's semantics, when publishing a message, developers have a notion of the message being "committed" to the log. Once a published message is committed, it will not be lost. Kafka is distributed system, so messages are replicated to partitions. For message commit, at least one replicating broker should be alive.

Kafka guarantees at-least-once delivery by default. It also allows the user to implement at most once delivery by disabling retries on the producer and committing its offset prior to processing a batch of messages. Exactly-once delivery requires co-operation with the destination storage system (it is some sort of two-phase commit).

In connection with Kafka, we should highlight Apache Spark [19]. Apache Spark is an open-source cluster computing framework for big data processing. It has emerged as the next generation big data processing engine, overtaking Hadoop MapReduce.

Apache Spark provides a comprehensive, unified framework to manage big data processing requirements with a variety of diverse data sets (text data, graph data, etc.) and data sources (batch data and real-time streaming data). Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk. Spark lets developers write applications in Java, Scala, or Python using a built-in set of high-level operators. In addition to MapReduce operations, Apache Spark supports SQL queries, streaming data, graph data processing, and machine learning. Developers can use these capabilities stand-alone or combine them to run in a single data pipeline use case.

Another model is the recently introduced Kafka Streams. Kafka models a stream as a log, that is, a never-ending sequence of key/value pairs. Kafka Streams is a library for building streaming applications, specifically applications that transform input Kafka topics into output Kafka topics (or calls to external services, or updates to databases, or whatever). It lets you do this with concise code in a way that is distributed and fault-tolerant [20].

The above-mentioned models describe the modern view of the building IoT systems from the position of data architecture. A review for some IoT and/or Smart Cities related program (of course, we target technology-related education only) is presented in [21], for example.

Time-series databases historically play the important role for IoT applications. Technically, most of the applications (especially, in M2M area) collect and proceed some measurements. And time-series are the natural way of saving measurements.

One important element of IoT programming is associated with meta-data. In the most cases, the public APIs IoT systems are dealing with are based on

REST model. It is true for data persistence interfaces too. REST architecture proposes the uniform interface. In REST model, all resources present the same interface to clients. And it is one of the reasons for REST popularity. Alternatively, the Service Oriented Architecture (SOA) approach may offer personalized interfaces for the different resources. The whole SOA model is based on the idea that different services have different interfaces. In SOA, we need to provide the definition for used interfaces. The definition of the services (Web Service Definition Language - WSDL) is a key part of SOA. Any WSDL definition of a Web Service defines operations in terms of their underlying input and output messages. Unlike this, REST is based on the self-described messages. WSDL defines the form of the data that accompany the messages in SOA. REST does not provide this information. In other words, SOA has got a rich set of metadata, where REST model does not have meta-data at all.

Metadata support lets discover information about interfaces programmatically. It is a key moment. With the program-based discovery, we can automate the programming. And automation is a very important issue for Internet of Things due to high diversity in hardware (e.g., sensors, actuators, etc.). So, in our opinion, adding some standard form of metadata for REST API is very important for Internet of Things programming.

The educational program should include the following parts (elements):

- sensing,
- network connectivity,
- IoT security
- data integration, data processing, and applications.

In the first part, we present some overview for the modern sensors. Network connectivity section should discuss IoT networks, such as Bluetooth, Bluetooth Low Energy, ZigBee, Wi-Fi, WiMAX, LTE. There are several key dimensions for IoT protocols: their communication range, application duty cycle, data rate, and battery consumption. Also, we should talk here about data protocols, such as CoAP, MQTT, HTTP (HTTP/2).

Data integration elements should include IoT middleware, data storage options, principles of processing for unstructured data. This topic should cover data architectures for IoT systems too.

Data processing part includes real-time processing engines and algorithms, as well as stream analytics.

Applications-related section should include gateways, user applications as well as top-level architectures, such as edge processing and fog computing. Also, we will discuss here such things as localization: localization algorithms, indoor and mobile localization. In this section, we place also context-aware applications and utilizing sensors to gain greater visibility and real-time situational awareness.

In general, we followed the following schema (Fig. 4) in our educational program.

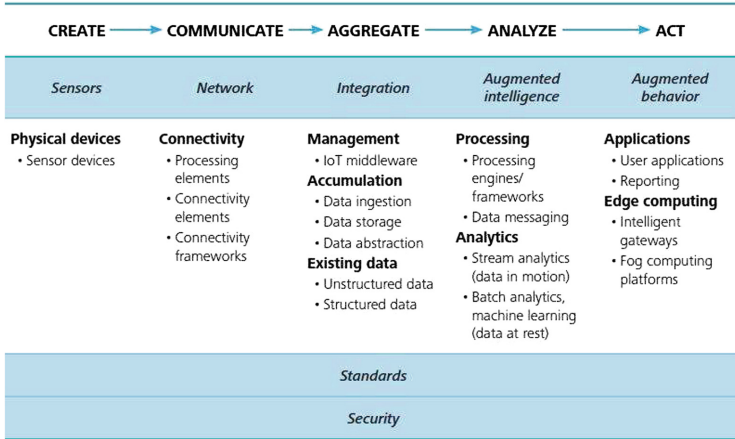


Fig. 4. Sensor fusion [22]

4 Cloud Computing and Related Areas for IoT

In this section, we would like to discuss the cloud computing models for IoT. Cloud architecture provides new opportunities in aggregating IoT data (e.g. data from sensors) and exploiting the aggregates for larger coverage and relevancy. In the same time, cloud models affect privacy and security.

There are several important moments. Firstly, we would like to highlight the important role of Amazon S3 for all tasks related to media data persistence. Almost all existing projects use Amazon Simple Storage Service (S3) for media data. The classical model is when Amazon S3 stores media objects and a separate relational database (it could be some NoSQL database, e.g., key-value store) keeps keys for objects.

The OpenStack project [23] produces the open standard cloud computing platform for both public and private clouds. OpenStack has a modular architecture with various code names for its components. OpenStack Object Storage (Swift) is a scalable redundant storage system [24]. With Swift, objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. It lets scale storage clusters scale horizontally simply by adding new servers. Swift is responsible for replication its content.

Another important moment for IoT programming and cloud is so-called MBaaS (Mobile Backend As A Service). It is a model for providing the web and mobile app developers with a way to link their applications to backend cloud storage [25]. The key moments are simplicity for the developers and time to market factor. Actually, the additional services are the key idea behind MBaaS. MBaaS provides public application program interfaces (APIs) and custom software development kits (SDKs) for mobile and web developers. Also, MBaaS provides such features as user management, push notifications, and integration

with social networking services. The key moment here is the simplicity for the developers.

Usually, MBaaS API (SDK) hides data persistence details from the developers. So, in the most cases, for the developers, it looks like some unrestricted key-value data store.

Also, in this connection, we can mention such IoT frameworks as oneM2M or FIWARE. They are pretending to be IoT standards. But standards in IoT (M2M) do not provide dedicated data persistence solutions. They rely on the existing cloud solutions.

Our vision for the future of cloud computing in IoT is based on the conception of fog computing. Fog computing is an extension of classic cloud computing to the edge of the network. It has been designed to support IoT applications, characterized by latency constraints and a requirement for mobility and geo-distribution [26]. It is illustrated in Fig. 5.

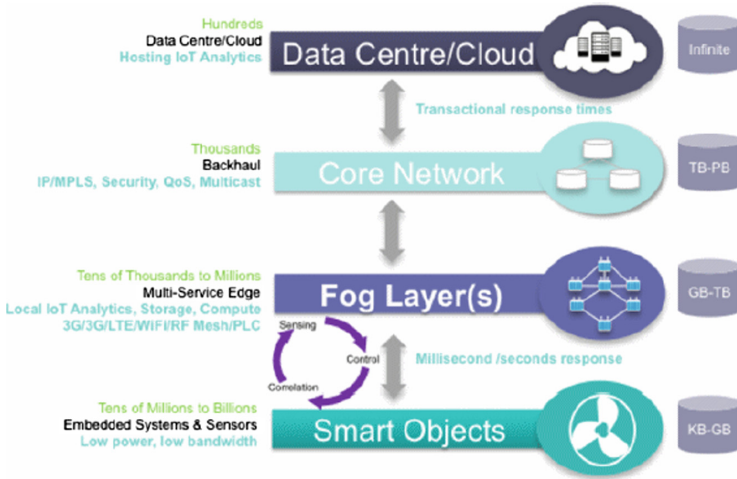


Fig. 5. Fog computing [27]

Fog computing architecture is not just about aggregation or concatenation of various physical data. It is about real-time distributed intelligence.

The common model is a bit more complex. It will include also extreme edge computing and software defined networks. It is illustrated in Fig. 6.

As per Cisco, extreme edge part includes data collecting elements: vehicles, ships, railways, roadways, factory floors. And of course, nowadays data could be processed on the collecting devices too. In general, the most time-sensitive should be analyzed on the node closest to the data collector (collectors). It is the main idea behind the fog computing. In this paradigm, we could analyze data even on the data collector itself. Especially, if our collector has the same computing power as fog's node.

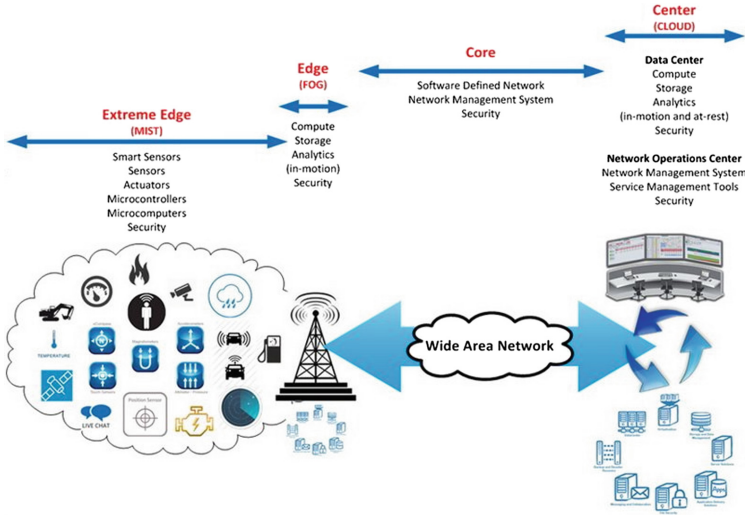


Fig. 6. The common architecture computing [28]

Technically, for or extreme edge is a new intelligent layer at or near the source of the data (data collector). And this layer can filter and normalize the data before passing them to the cloud or send commands directly to actuators.

With this architecture, data might not need to ever travel to the cloud layer. We can process data in real-time at the edge of our network.

With this architecture, we can store data on our devices (network fabric data store [29]) rather than in cloud-based data-center. Of course, we can follow to mixed model too and log, for example, some history on our cloud. But anyway, this model helps as to avoid always passing big data to our cloud and reduce the cost of transportation.

The edge-based processing is always stream-based processing. It is almost mandatory, because with the network fabric data store it is not so common to save big volumes of data. This emphasizes again the importance of stream processing, which we discussed in the section devoted to training. Stream processing is a key element for IoT programming.

Some of the authors describe this model as a shift in the way business is organized. The infrastructure based model is being replaced by the service-based model [30]. It corresponds to the common trend in which business is being virtualized and digitalized.

In this connection, we should mention the importance of another element in Fig. 5 - Software Defined Networks [31]. With Software Defined Networks (SDN) we can virtualize and digitalize network's hardware. Firstly, SDN separate (decouple) the data plane and control plane. SDN establish open interfaces between them. Secondly, SDN proposes a centralized control plane, thus having a global view of the network.

The ability to program the network, provided by SDN, can be compared with the mobile applications running on a mobile Operating System (Android or iOS) [31]. Similar to mobile applications, network-based applications can use resource and services (public interfaces) provided by SDN.

For IoT applications, SDN model brings yet another set of public interfaces. As per reviews, the specific SDN capabilities which will be useful for IoT application, are dynamic load management, service chaining, and bandwidth management.

Dynamic load management enables to monitor and orchestrate bandwidth changes automatically depends on the overall load of the network. It helps providers to support data peaks from IoT devices. Service chaining enables to sequence application-specific processing procedures to a given clients job [32]. SDN will ease the provisioning and service management processes for IoT devices. Service chaining allows to integrate unchanged network service software that is unaware of its operating environment. Many IoT devices (e.g., sensors) send data (measurements) periodically. Bandwidth management allow scheduling when and how much traffic an IoT application will need at a given time [33].

References

1. Chen, Y.-K.: Challenges and opportunities of internet of things. In: Design Automation Conference (ASP-DAC), pp. 383–388. IEEE Press, New York (2012)
2. Namiot, D., Sneps-Snepe, M.: On IoT programming. *Int. J. Open Inf. Technol.* **2**(10), 25–28 (2014)
3. Namiot, D., Sneps-Snepe, M.: On software standards for smart cities: API or DPI. In: ITU Kaleidoscope Academic Conference: Living in a Converged World-Impossible Without Standards? pp. 169–174. IEEE Press, New York (2014)
4. Im, J., Seonghoon, K., Daeyoung, K.: IoT mashup as a service: cloud-based mashup service for the internet of things. In: 2013 IEEE International Conference on Services Computing (SCC), pp. 462–469. IEEE Press, New York (2013)
5. Namiot, D., Sneps-Snepe, M.: On micro-services architecture. *Int. J. Open Inf. Technol.* **2**(9), 24–27 (2014)
6. Bahrami, M., Singhal, M.: The role of cloud computing architecture in big data. In: Pedrycz, W., Chen, S.-M. (eds.) *Information Granularity, Big Data, and Computational Intelligence*, pp. 275–295. Springer, Heidelberg (2015)
7. Raggett, D.: The internet of things: W3C plans for developing standards for open markets of services for the IoT. *Ubiquity* **10**, 3–6 (2015)
8. Samsung Iot.js. <https://news.samsung.com/global/samsung-electronics-opens-development-of-iot-js-an-iot-platform-that-expands-interoperability-to-lightweight-devices>. Accessed May 2016
9. Who uses Dart. <https://www.dartlang.org/community/who-uses-dart.html>. Accessed May 2016
10. Aggarwal, C.C.: *Managing and Mining Sensor Data*. Springer Science & Business Media, New York (2013)
11. Wang, M., et al.: City data fusion: sensor data fusion in the internet of things. arXiv preprint [arXiv:1506.09118](https://arxiv.org/abs/1506.09118) (2015)
12. Introduction to sensor fusion. <http://projects.mblab.com/introduction-to-sensor-fusion/>. Accessed May 2016

13. Marz, N., Warren, J.: *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., Greenwich (2015)
14. Ranjan, R.: Streaming big data processing in datacenter clouds. *IEEE Cloud Comput.* **1**, 78–83 (2014)
15. Applying the Kappa architecture in the telco industry. <https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry>. Accessed May 2016
16. Villari, M., et al.: AllJoyn Lambda: an architecture for the management of smart environments in IoT. In: *Smart Computing Workshops (SMARTCOMP Workshops)*, pp. 9–14. IEEE Press, New York (2014)
17. Erb, B., Kargl, F.: A conceptual model for event-sourced graph computing. In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pp. 352–355. ACM, New York (2015)
18. Garg, N.: *Apache Kafka*. Packt Publishing Ltd., Birmingham (2013)
19. Shanahan, J.G., Laing, D.: Large scale distributed data science using apache spark. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2323–2324. ACM, New York (2015)
20. Kafka Streams. <http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple>. Accessed May 2016
21. Namiot, D.: On internet of things and smart cities educational courses. *Int. J. Open Inf. Technol.* **4**(5), 26–38 (2016)
22. Inside the Internet of Things (IoT). <http://dupress.deloitte.com/dup-us-en/focus/internet-of-things/iot-primer-iot-technologies-applications.html/>. Accessed Aug 2016
23. OpenStack. <https://www.openstack.org/>. Accessed Aug 2016
24. Jackson, K., Bunch, C., Sigler, E.: *OpenStack Cloud Computing Cookbook*. Packt Publishing Ltd., Birmingham (2015)
25. Sneps-Sneppe, M., Namiot, D.: On mobile cloud for smart city applications. arXiv preprint [arXiv:1605.02886](https://arxiv.org/abs/1605.02886) (2016)
26. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: *Proceedings of the 1st edn. of the MCC workshop on Mobile Cloud Computing*, pp. 13–16. ACM, New York (2012)
27. Byers, C.C., Wetterwald, P.: Fog computing distributing data and intelligence for resiliency and scale necessary for IoT. *Ubiquity* **11**, 1–12 (2015)
28. Edge Computing - Where data comes alive! <https://vividcomm.com/2016/04/08/edge-computing-where-data-comes-alive/>. Accessed Sept 2016
29. Greenberg, A., et al.: VL2: a scalable and flexible data center network. *ACM SIGCOMM Comput. Commun. Rev.* **39**(4), 51–62 (2009)
30. di Costanzo, A., de Assuncao, M.D., Buyya, R.: Harnessing cloud technologies for a virtualized distributed computing infrastructure. *IEEE Internet Comput.* **13**(5), 24–33 (2009)
31. Caraguay, V., Leonardo, A., et al.: SDN: evolution and opportunities in the development IoT applications. *Int. J. Distrib. Sens. Netw.*, 1–10 (2014)
32. Blendin, J., et al.: Software-defined network service chaining. In: *2014 Third European Workshop on Software Defined Networks*, pp. 109–114. IEEE, New York (2014)
33. Kim, H., Feamster, N.: Improving network management with software defined networking. *IEEE Commun. Mag.* **51**(2), 114–119 (2013)

Distributed Computer and Communication Networks
19th International Conference, DCCN 2016, Moscow,
Russia, November 21-25, 2016, Revised Selected
Papers

Vishnevskiy, V.M.; Samouylov, K.E.; Kozyrev, D.V. (Eds.)

2016, XV, 670 p. 257 illus., Softcover

ISBN: 978-3-319-51916-6