Study of IoT Architecture and Application Invariant Functionalities

1st Nahit Pawar
Institut Polytechnique de Paris
Télécom SudParis
nahit.pawar@telecom-sudparis.eu

2nd Thomas Bourgeau
KB DIGITAL AG, Switzerland
tbourgeau@kb-digital.ch

3rd Hakima Chaouchi
Institut Polytechnique de Paris
Télécom SudParis
hakima.chaouchi@telecom-sudparis.eu

***Abstract*—IoT is a conjunction of many technologies and it has spanned across diverse and multidisciplinary application domains. Each domain has its own set of application requirements thereby inhibiting the use of conventional programming models. As a result there is a strong need for intermediate software abstraction layer to hide various technological details underlying IoT. In this paper, we propose two high level concepts of IoT, first is the IoT architecture for systematic study of common characteristics and second is the IoT application invariant functionalities and programming patterns. We think that our work exposes a step forward to review on the overall IoT architecture and provides a lightweight approach when designing software abstraction layer with minimalist programming functionalities that are IoT application invariant.**

***Index Terms*—IoT system architecture, device heterogeneity, invariant functionalities, programming pattern**

I. Introduction

One of the widely known attribute of IoT is that it encompasses large network of *heterogeneous devices* [1] ranging from resource constrained devices used in battery operated low power wireless sensor actuator node (WSAN) to more powerful devices used in gateways and other compute intensive applications like autonomous vehicles, connected robots, drones, etc. An important challenge which is still being explored in IoT, is rapid prototyping or Proof-of-Concept (PoC), of IoT applications on heterogeneous devices [2]. Moreover, the presence of IoT in various application domains - *smart energy, smart health, smart buildings, smart transport, smart industry and smart city* - makes it a diverse and multidisciplinary field. Each application domain has its own diverse characteristics and requirements inhibiting the use of conventional programming models of distributed systems [3].

As mentioned in [4], [5] for widespread adoption of IoT-based system and services an intermediate *Software Abstraction Layer* (SAL) - platform, framework and operating systems - is needed to hide the details of various technologies underlying IoT. We believe that in order to design and evaluate this SAL a systematic study of IoT application characteristic and its invariant attribute (if any) is needed to capture the essence of IoT which is otherwise difficult to realize.

The main goal of this paper is to introduce two high level IoT abstractions (section III), the first is in the form of IoT reference architecture for systematic understanding of common characteristics of IoT - right from the network of low power devices to gateways then to cloud infrastructure and all the way up to device orchestration. A second abstraction is a set of programming concept that captures the most common IoT application invariant functionalities and programming pattern that we identified and gathered in a limited list of commands. Based on this work we are in a process to enhance our previous work [2] on minimalist IoT programming language (PrIoT-Lang) and API (PrIoT-API).

The concept of abstraction in IoT is not new, the abstraction defined in [6], [7] are rather coarse grained in contrast to ours but more interestingly they might allow us to reuse the various semantics defined in their research to better understand and define the scope of our programming patterns (PPs). Whereas [8] provides design patterns that are focused for edge devices, in contrast to their work we proposed programming patterns that model the operating modes of constraint devices as well for edge and cloud.

II. Related Work

In the context of IoT application development several solutions exist in both academic research and industrial products in the form of - platform, operating systems and frameworks. All these solutions rely on the underlying high level system architecture and target one or more problems associated with application development, deployment and maintenance. This section briefly describes some of the published work in IoT system architecture and also some of the famous frameworks.

a) IoT Architecture: In the literature number of architecture has been proposed by many industrial alliance OneM2M [9], *Open Connectivity Foundation* (OCF) [10], Thread [11], to name a few and industrial standardization bodies such as ITU, ETSI, IETF in the form of reference framework. Research community is also actively participating in defining and comparing the requirements for the future IoT architecture. The architecture of most publicly available IoT experimental testbeds follow either a two-tier or three-tier structure [12], where the structure refers to the organization of testbed hardware components. Although these architectural structures are suitable for experimentation and prototyping but do not cover IoT system management features. A new five layer IoT architecture has been proposed in [13] by merging the architecture of Internet and telecommunication management

978-3-903176-32-4

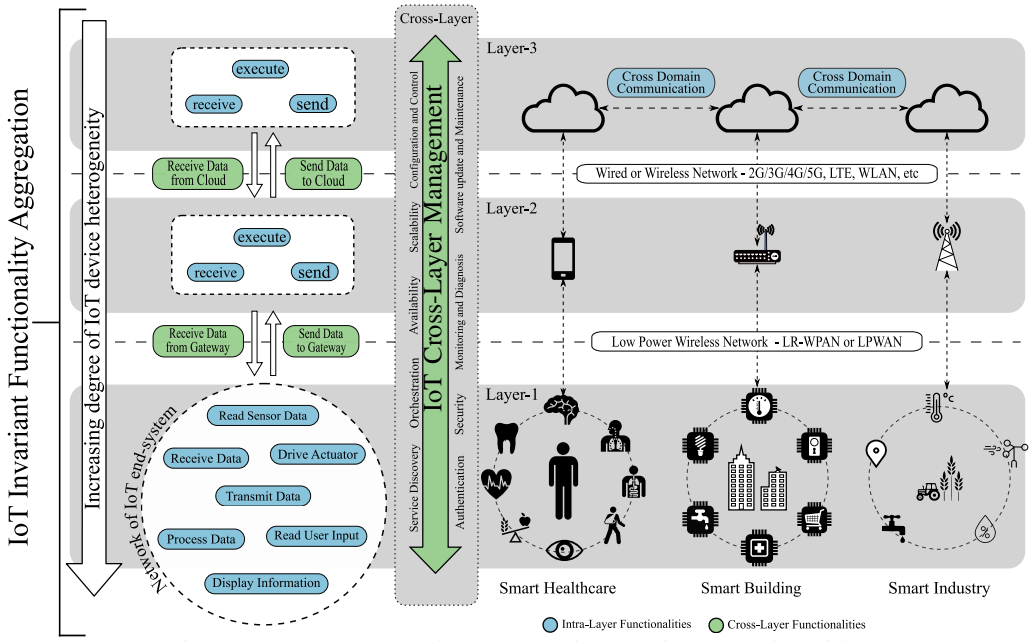

Fig. 1: IoT System Architecture and its Invariant Functionalities

network, thereby introducing the management functionalities and extending the three layer architecture. IoT-A [14] is another EU FP7-ICT project worth mentioning, although IoT-A is not an architecture but provides the architectural reference model in the form of building blocks, design guidelines for protocols, interface, algorithm and models of interoperability for designing future IoT architecture.

b) Frameworks: There are many industrial alliances and international standardization bodies that define the IoT framework. *OneM2M* is one such global organization that provides a framework defining the requirements, architecture, API specification, security and interoperability for M2M and IoT technologies. The OCF is another industrial group whose stated mission is to develop specification standards, interoperability guidelines and provide certification programs for software frameworks. For example, *IoTivity* [15] is a well known open source implementation of the OCF specification. While each industrial alliance promotes their own protocol, there are no real successful attempts to cover across all domains for the entire IoT echosystem. Meanwhile, international standardization bodies such as ITU, ETSI, IETF are all providing their specifications for standardized connected IoT architecture and protocols. The research community is still working in a broader range of open standards for the IoT and focuses mainly around IP compliant approaches for interoperability [16].

III. IoT Abstraction

In this section, we will explain two types of abstraction, first is the reference *"IoT Architecture"* (III-A) that is used in our work to understand, develop, deploy and maintain systematic IoT applications and second is the concept of *"IoT Application Invariant Functionality and Programming Pattern"* (III-B) - we found that although IoT applications are diverse but they still share many invariant functionalities and programming patterns which does not change from one application to another. Figure 1 shows our proposed architecture and also the glimpse of some common functionalities across three diverse IoT application domains and are explained in details hereafter.

A. IoT Architecture

An ideal IoT architecture which has the ability to capture all the characteristics of IoT scenarios is far from reality, mostly due to the multi- and inter-disciplinary nature of IoT. Nevertheless there are number of IoT architecture discussed in the literature [12]–[14], [17]. For our work we decided to adopt a more simplified 4-layer architecture as shown in Figure 1, that allows us to capture most IoT application characteristics as described hereafter.

The architecture in Figure 1 consist of 4 layers, namely *Device-Layer*, *Edge-Layer*, *Cloud-Layer* and *Cross-Layer*. We briefly describe each layer in the following:

a) ***Layer-1 - Device-Layer****:* This layer is responsible to *measure physical quantity*, *detect an event* or *control* in the environment of interest. One of the characteristics of this layer is that it consists of a (large) network of *heterogeneous* IoT end-devices like sensor, actuator, transceiver and processing unit - that together perform the desired application task. These devices are available with varying degrees of capabilities and are provided by various OEMs (Original Equipment Manufacturers). Selecting a particular device/hardware technology for a given application is not trivial, because of heterogeneous application, hardware and software requirements [18]. Another important characteristic of this layer is the availability of large number of communication and networking protocols (LR-WPANs and LPWAN) [19]. The reason for such a large selection is that these protocols are specifically designed to

satisfy application requirement and there is no one protocol that can satisfy the requirements of various IoT domains.
There are number of possible ways a network of IoT end-systems in layer-1 can communicate with the rest of the IoT sub system as shown in Table I along with example application and technologies. In conclusion this layer has the highest degree of heterogeneity in terms of hardware devices, software support, communication and network protocols.

Inter-Layer Communication	Example Application and Technology
L-1 ↔ L-1	M2M, Mesh network, Infrastructure network, Autonomous deployment
L-1 ↔ L-3	GSM/GPRS/3G/4G
L-1 ↔ L-2	M2M, Edge Computing, Intranet, Access Network
L-1 → L-2 → L-3	Monitoring
L-1 ← L-2 ← L-3	Actuator Control, configuration command, etc.
L-3 ↔ L-3	Cross-domain
L-4 ↔ (L-1/L-2/L-3)	Configuration & Control

TABLE I: Inter/Intra-Layer Communication

*b) **Layer-2 - Edge-Layer**:* At the very least this layer act as bridge to exchange data between device-layer and cloud-layer without any data manipulation but in other cases it might implement some complex cloud services which also referred to as "Fog Computing" or "Edge Computing". The facility to port complex cloud services closer to layer-1 allows reduced bandwidth utilization and latency for critical applications. The hardware system used in this layer is generally made up of processing unit with abundant resources to support state of the art operating systems. The presence of operating system at this layer hides the hardware heterogeneity, thereby relieving application developers from knowing the underlying hardware details. Furthermore, data exchange at this layer rely on well know standard protocols from the application layer of the Internet protocol suite - like HTTP, MQTT, CoAP, etc - over wireless/wired technologies from 3GPP (3G, LTE, 5G, etc.), IEEE (802.11, 802.2, etc.), etc. In conclusion, the use of resource abundant hardware devices at this layer facilitate the use of high end operating systems (Linux, Windows, etc.) that allow development of hardware independent compute intensive functionalities or services.

*c) **Layer-3 - Cloud-Layer**:* Bulk data received from the previous layer are stored in the cloud for further processing. This layer expose the required tools to help in the development of various cloud applications and services like visualization, analytics, cross-domain data exchange, cognitive computing, big data, machine learning algorithm and artificial intelligence (AI) to name a few. This layer expose close to no hardware heterogeneity as IoT application developers take advantage of available cloud services like SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) from various service providers like Amazon, Google, Microsoft, etc. to name a few. The services at this layer can benefit from uniform APIs (Mesos [20], Terraform [21], etc.) and standard secure access protocols (MQTT, AMQP, HTTP, etc.) that simplifies the portability of application at this layer.

*d) **IoT Cross-Layer Management**:* IoT system management plays a very important role in the sustainability of IoT system and can be considered as a backbone of the IoT architecture. Provisioning for system management helps in the *deployment* and *maintenance* of IoT end-system and is also considered an important requirement from business point of view [22]. For interested readers, a complete survey on IoT management can be found in [23] which expose certain features such as service orchestration, software maintenance, deployment diagnosis and configuration to name a few.

As shown in Table I, there can be many possibilities for inter- and intra-layer communication depending on the producer and consumer of data. Finally, in contrast to existing work, the architecture discussed in this section highlights the common characteristics observed in various application domains which is useful for understanding and developing various functionalities of IoT and also it introduces an important system management cross-layer for deployment and maintenance of IoT systems.

Function Name	Description
configure	Configure IoT end-devices
read	Read data from sensor or HMI
write	Write data to actuator or HMI
send	Send data using transceiver
receive	Receive data using transceiver
execute	Execute user defined function
wait	Delay, Wait and Sleep

TABLE II: Layer-1 - Invariant Functionalities

B. IoT Application Invariant Functionality (IF) and Programming Pattern (PP)

In this section we examine that within each layer of IoT architecture, an IoT application perform various IF and follow certain PP which does not change from one application to another.

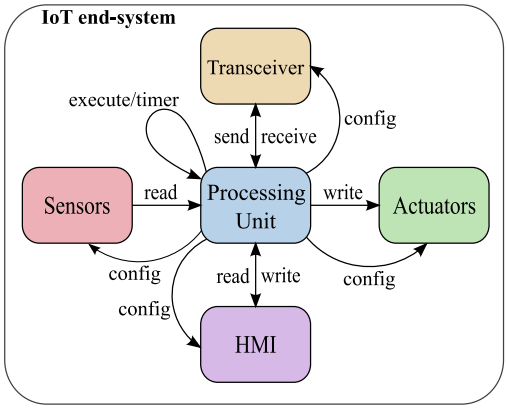

Fig. 2: IoT end-system

Layer-1 IF and PP - It is fair to generalize that *Layer-1* consist of network of IoT *end-systems* and each IoT *end-system* is further composed of IoT *end-devices* (sensor, actuator, transceiver, HMI, processing unit) as shown in figure 2. Irrespective of IoT domain and its requirements an IoT end-system performs some basic high-level invariant functions as explained hereafter and are summarized in Table II. These functions are programmed in the memory of processing unit (PU) and is responsible for controlling

the behavior of other IoT end-devices. One of the function issued from PU is to configure (**configure**) the end-device into desired functional state depending on the type of IoT end-device. Other functions perform operations that are based on the type of end-device they are associated with. For example, a **read** function associated sensor is used for reading sensor data, while the similar read function for HMI devices is used for reading user input. Similarly a **write** function is used controlling the action of actuators and also it can be used to display information on HMI devices like OLED displays. Also based on the underlying communication technology and network protocol a PU can issue **send** and **receive** function for transmitting and receiving data to and from layer above (layer-2 and layer-3) or directly to another IoT end-system in layer-1. Moreover, a PU performs some auxiliary functions for example, an **execute** command can call a library functions or a user defined custom functions (for example, an algorithm for local sensor data processing before transmitting). Finally the **wait** command can execute standard delay, interruption or sleep operation. We observed

Layer-1 Programming Pattern
read → send → sleep
read → execute → send or sleep
receive → write → sleep
receive → execute → read → send → sleep
receive → execute → write → sleep
Layer-2 Programming Pattern
receive → execute → send
Layer-3 Programming Pattern
receive → execute
execute → send

TABLE III: Layer-1,2,3 - Programming Pattern

that an application code for IoT end-system follows some basic programming patterns [18] listed in Table III. Although these patterns are limited but, they can be used to program highly practical IoT scenarios. These programming patterns can be easily visualized as finite-state machine, where each state is one of the basic operation performed by processing unit as shown in Figure 3 along with respective inter-layer communication.

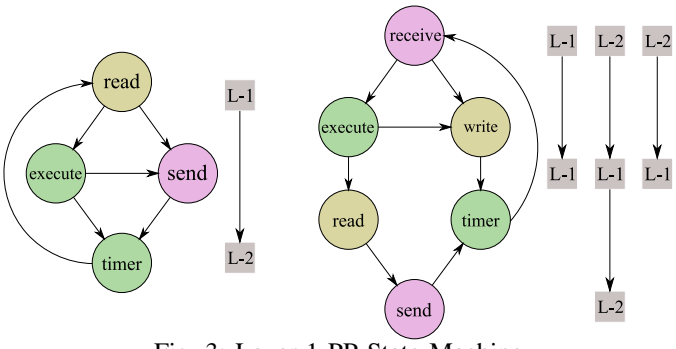

Fig. 3: Layer-1 PP State Machine

Layer-2 IF and PP - The network of gateways at this layer consists of two sets of **send** and **receive** functions, the first set of functions is used to exchange data from layer-1 using the available low power wireless communication (LR-WPAN or LPWAN) and the second set of functions is for the Internet backhaul using standard wired or wireless communication protocol like 3GPP/4G, LTE, WLAN, etc. One of the important feature of this layer is the capabilty to execute relevant cloud services closer to layer-1, therefore a gateway can expose a high level **execute** command that a user can bind to one or more services. We have observed that a gateway follows a very simple programming pattern to communicate with layer-1 and layer-2 as shown in Table III and Figure 4.

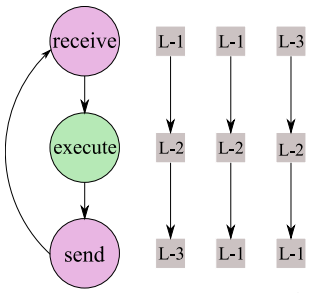

Fig. 4: Layer-2 PP State Machine

Layer-3 IF and PP - Similar to layer-1 and layer-2 this layer also consists of **send** and **receive** functions to exchange data with layer-2 or directly with layer-1 bypassing gateway. The most important feature is to **execute** various cloud services and **configure** the cloud instances on demand. Table III shows PP for layer-3.

Cross-Layer IF and PP - Most of the challenges related to cross-layer functionalities relates to orchestrating the deployment and behavior of devices application logic, maintaining the status of their firmware version and monitoring the data exchanged even in case of network failure. For these functionality it is important to keep a recurrent connection to the devices through the network with **send** and **receive** control commands. Furthermore, keeping the system updated to its latest version require to **configure** the devices with a correct version of the firmware while tracking the version of the application version deployed. For such cases, devices should embed a boot loader that has the ability to run firmware updates over the air such as Mender [24]. The configuration logic also spans the other layers when defining the configuration of the cloud architectures in L3 (number of virtual machines, the type of database, the selection of Machine Learning Algorithm, etc.) or when configuring the building blocks of an Edge container or the type of network access to forward data.

IV. Conclusion and Future Work

Software Abstraction Layer (SAL) is necessary for easy and rapid IoT application development. In this paper we have introduced our reference IoT architecture and presented some common characteristics of IoT applications and also various invariant functionalities and programming patterns observed in many IoT applications. This approach is implemented in our previous work [2], were we have shown that it is possible to construct lightweight SAL, language and API that can expose these invariant functionalities and programming pattern and still be able to cover most IoT application scenarios.

REFERENCES

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] Nahit Pawar, Thomas Bourgeau, and Hakima Chaouchi. PrIoT: Prototyping the Internet of Things. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 216–223. IEEE, 2018.

[3] Ryo Sugihara and Rajesh K Gupta. Programming Models for Sensor Networks: A Survey. *ACM Transactions on Sensor Networks (TOSN)*, 4(2):8, 2008.

[4] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(1):70–95, 2016.

[5] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of Middleware for Internet of Things: A Study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.

[6] Franco Zambonelli. Key Abstractions for IoT-Oriented Software Engineering. *IEEE Software*, 34(1):38–45, 2017.

[7] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. Internet of Things Patterns. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*, pages 1–21, 2016.

[8] Soheil Qanbari, Samim Pezeshki, Rozita Raisi, Samira Mahdizadeh, Rabee Rahimzadeh, Negar Behinaein, Fada Mahmoudi, Shiva Ayoubzadeh, Parham Fazlali, Keyvan Roshani, et al. IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 277–282. IEEE, 2016.

[9] oneM2M - Standards for M2M and the Internet of Things. https://www.onem2m.org/. Accessed on 01 January 2021.

[10] Open Connectivity Foundation (OCF). https://openconnectivity.org. Accessed on 01 January 2021.

[11] Thread Group. https://www.threadgroup.org/. Accessed on 01 January 2021.

[12] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. A Survey on Facilities for Experimental Internet of Things Research. *IEEE Communications Magazine*, 49(11), 2011.

[13] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the Architecture of Internet of Things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–484. IEEE, 2010.

[14] IoT-A - Internet of Things Architecture. https://cordis.europa.eu/project/rcn/95713_en.html. Accessed on 01 January 2021.

[15] IoTivity. https://iotivity.org/. Accessed on 01 January 2021.

[16] Zhengguo Sheng, Shusen Yang, Yifan Yu, Athanasios Vasilakos, Julie Mccann, and Kin Leung. A Survey on the IETF Protocol Suite for the Internet of Things: Standards, Challenges, and Opportunities. *IEEE Wireless Communications*, 20(6):91–98, 2013.

[17] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani. Architecting the Internet of Things: State of the Art. In *Robots and Sensor Clouds*, pages 55–75. Springer, 2016.

[18] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 8. ACM, 2016.

[19] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of Things (IoT) Communication Protocols. In *2017 8th International conference on information technology (ICIT)*, pages 685–690. IEEE, 2017.

[20] Apache mesos: A Distributed Systems Kernel. https://http://mesos.apache.org/. Accessed on 01 January 2021.

[21] Terraform - An open-source infrastructure as code software tool. https://www.terraform.io/. Accessed on 01 January 2021.

[22] In Lee and Kyoochun Lee. The Internet of Things (IoT): Applications, Investments, and Challenges for Enterprises. *Business Horizons*, 58(4):431–440, 2015.

[23] S. Sinche, D. Raposo, N. Armando, A. Rodrigues, F. Boavida, V. Pereira, and J. S. Silva. A Survey of IoT Management Protocols and Frameworks. *IEEE Communications Surveys Tutorials*, 22(2):1168–1190, 2020.

[24] Mender: Over-the-air software updates for connected Linux devices. https://mender.io. Accessed on 01 January 2021.