Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 130 (2018) 1057–1062

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Symposium on Internet of Ubiquitous and Pervasive Things (IUPT 2018)

Understanding IoT Systems: A Life Cycle Approach

Leila Fatmasari Rahman[a,*], Tanir Ozcelebi[a], Johan Lukkien[a]

[a]*Eindhoven University of Technology, P.O.Box 513, Eindhoven 5600 MB, The Netherlands*

Abstract

Internet of Things (IoT) systems and the corresponding network architectures are complex due to distributed services on many IoT devices collaboratively fulfilling common goals of IoT applications. System requirements for different types of IoT application domains are still not well-established. The life cycle view is one of the views used for system architecting, showing different stakeholders' concerns at every stage of the life cycle to derive system requirements. We employ the life cycle view to understand IoT systems in different IoT application domains. Our contribution is the definition of a generic life cycle model for IoT, which is specified by observations on life cycles of existing IoT solutions and generalizations taking into account important IoT functionalities and quality attributes. Such generic life cycle model for IoT was non-existent.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Generic Life Cycle Model; Internet of Things; IoT Systems; System Requirements; System Architecture; Conceptual Model; Application Life Cycle; Service Life Cycle; Device Life Cycle

1. Introduction

An Internet of Things (IoT) system consists of services distributed in distinct devices, collaborating with each other to fulfill common goals of IoT applications. To build a robust and effective IoT system, one has to initially derive the right architectural requirements of the system. As IoT is a young and complex technology domain, insights into its system requirements are not widely available and therefore not well understood.

As with other system architecting methods[1,2,3], a life cycle view helps turn ill-defined problems into a well-articulated and structured understanding of the problems and their potential solutions which embody an architecture description[4]. The life cycle view is particularly important in architecting IoT systems due to the many collaborating elements involved, each with its own unique life cycle. The life cycle view helps to identify the non-obvious problems entailed in a system's life cycle. The life cycle of a product or system is the series of stages it goes through from inception to decline. As an example, a typical life cycle for a software system consists of six stages: analysis, design, implementation, testing, deployment and maintenance. Through the life cycle view of a system, we gain insights into the activities involved in each stage of the life cycle, the stakeholders responsible for and affected by these activities and the problems that come with them. We can use these insights to identify the life cycle needs[2] of a system which will turn into system requirements realizable by certain system and software architectures.

* Corresponding author. Tel.: +31-40-2477382
E-mail address: l.f.rahman@tue.nl

1877-0509
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2018.04.148

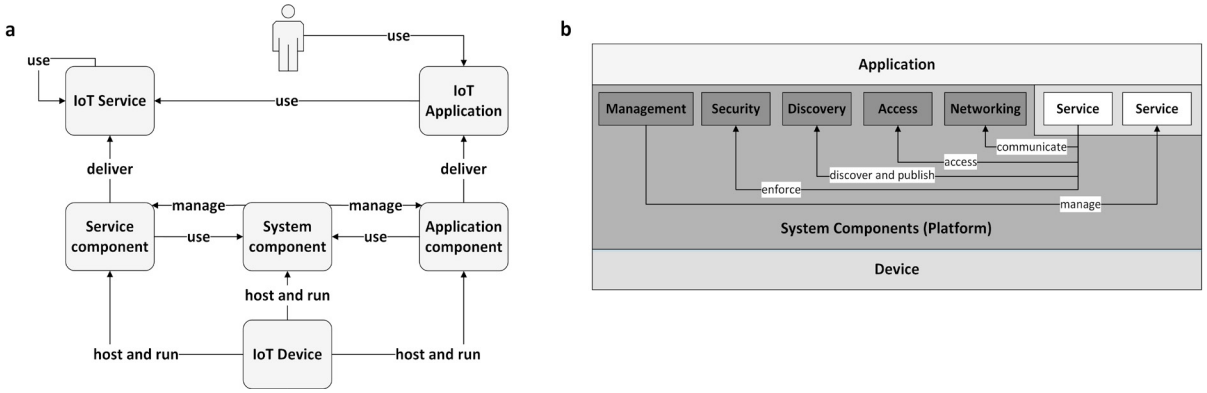

Fig. 1. (a) A user interfaces with the system through an IoT application. An IoT application is established by collaborating IoT services. IoT services and applications are delivered by software components, i.e. service and application components that reside in IoT devices. An IoT device hosts and runs system components that are used by both service and application components to perform system-related tasks. A system component also manages, i.e. installs, starts, stops, uninstalls service and application components [15]. (b) System components in an IoT device include management, security, discovery, access and networking components. The management component manages services and applications, and monitors the system. Other system components are used by services and applications to enforce security, discover, publish and access services and to communicate with other nodes in the network [15].

Our contribution in this paper is the proposal of a generic life cycle model for IoT which answers the question: how are life cycles of IoT systems structured? The generic model can be instantiated into concrete life cycle views of specific IoT systems in any specific domain. Such generic life cycle model for IoT was non-existent. The goal of our life cycle model for IoT differs from that of Product Lifecycle Management (PLM) [5]. PLM seeks to improve business models through shared data between people and processes in a product life cycle, whereas our life cycle model is used for understanding the requirements of an IoT system.

We define the generic life cycle model for IoT by first observing IoT life cycles of existing IoT solutions including Amazon Web Services (AWS) IoT [6], Works with Nest [7 8], Apple HomeKit [9], Philips Hue [10] and Open Architecture for Intelligent Solid State Lighting Systems (OpenAIS) [11 12]. We observed the problems that need to be solved in their life cycles and the various stakeholders involved. Secondly, we generalize by taking into account the realization of important functionalities and quality attributes in IoT [8 13]. For example, an *update* stage exists in the IoT device life cycle (Fig. 2) so that the system can adapt to future requirement changes; a *bootstrapping* [14] activity exists in the *commissioning* stage of the generic IoT device life cycle (Fig. 2) to realize secure communication within the network. Finally, we verify the universality of the generalized life cycles by evaluating how existing IoT solutions instantiate these life cycles. This step also helps us to complete or adjust the stages and their relevant activities in the life cycles.

2. Generic Life Cycle Model for IoT

As IoT systems adopt the Service Oriented Architecture (SOA) [8], we identify three main elements of an IoT system: IoT device, IoT service and IoT application. We model their relationships in Fig. 1. Our generic life cycle model for IoT represents the stages that any IoT device, IoT service and IoT application go through from construction to destruction. Inside each stage are relevant activities necessary for fulfilling the objective of the stage. The objective can be expressed as quality attributes that the stage help to satisfy, such as modifiability [16], interoperability [17], security [13], functional appropriateness [18], availability [13], usability [13], performance [13], deployability [13] and adaptability [18]. Based on these objectives and the activities in the life cycle stages, we can draw the life cycle needs of an IoT system.

2.1. Generic Life Cycle for IoT Device

As shown in Fig. 1, an IoT device hosts software components that are categorized into system, service and application components. A generic life cycle for IoT device describes the stages that an IoT device goes through from (re)construction to decommissioning (Fig. 2). The stakeholders responsible for or affected by these stages can vary from one system to another. The (re)construction stage pertains to the construction of a device hardware and its initial

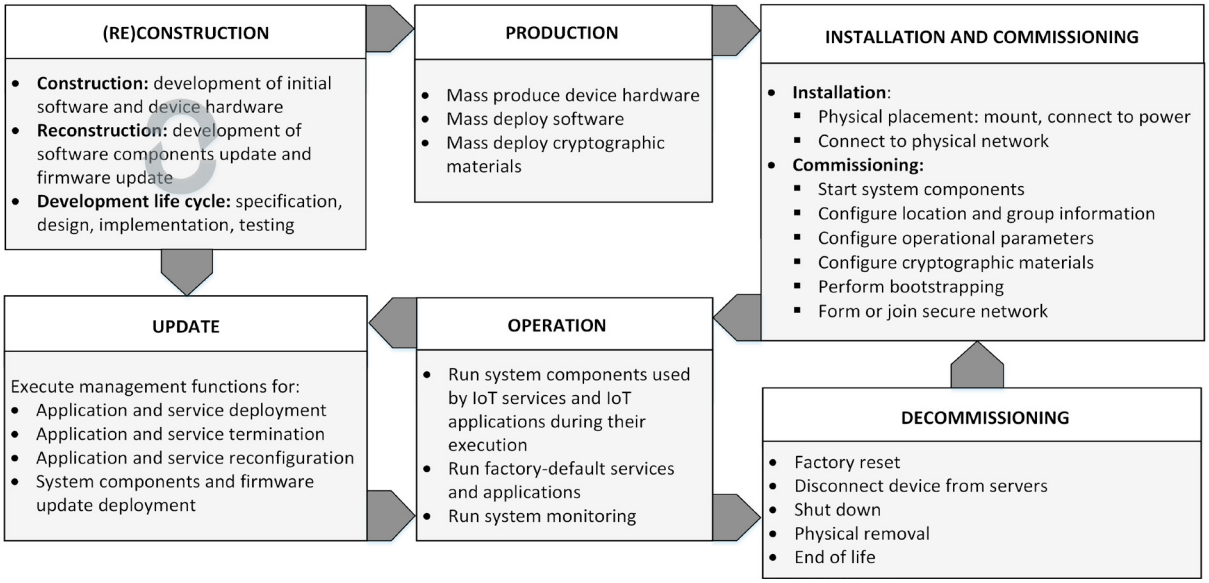

Fig. 2. Generic life cycle for IoT devices.

software (factory default software components), and to the reconstruction of device firmware and the initial software. Both of these activities should follow the common product development life cycle of specification, design, implementation and testing to ensure functional appropriateness[18]. Consideration on standard specifications, plug-fest testing and certification during software and hardware development help to achieve the desired levels of interoperability[17]. Once construction is done, an IoT device goes to the production stage, while the reconstruction activity starts its iteration independently, parallel to the rest of the device life cycle. One iteration of reconstruction results in a software component or firmware update, which can then interrupt the device life cycle to go from operation to the update stage.

The production stage pertains to the large scale production of the device. This includes manufacturing the hardware and mass deploying the initial software onto the hardware. During this stage, mass deployment of cryptographic materials can also be performed.

The installation and commissioning stage prepares the device for operation and secure communication within the network. Fig. 2 shows the activities involved in installation and commissioning. Configuring the location and group information for the device during commissioning allows the realization of functionalities that require this information. Many IoT use cases depend on such information, e.g. controlling the temperature in a certain area or controlling the lighting in a certain room[12 11]. Configuring operational parameters during commissioning enables the device to operate as desired. Configuring keys and certificates for bootstrapping can be done during commissioning if they are not yet deployed during production. Bootstrapping in IoT relates to the process of acquiring configurations and secret keys for authentication and secure communication with other devices in the network[14]. To summarize, the activities during the installation and commissioning stage play a role in achieving functional appropriateness[18] and security[13]. The stakeholders should also be able to perform these activities in a simple manner in order to increase usability[13].

The operation stage supports the execution of IoT services and applications by running relevant system components. The performance of the system depends on how well these system components perform their functions. For example, the use of a light-weight application protocol[19] for accessing the services would decrease message latency and consequently increase the performance of application execution. The choice of overall system architecture and network technology would also affect the performance of these system components. The management component monitors the system components' run-time during this stage in order to achieve high availability[13].

The device goes into the update stage when it needs to run management functions for updating the device. These updates come in different formulations as shown in Fig. 2. All of these updates should be done via the network (as opposed to an expert manually deploying the update on each node) as IoT systems typically consist of a high number of devices, distributed in remote places. This is important for modifiability[16 13] of IoT systems, as IoT services and applications evolve continuously over a long period of time.

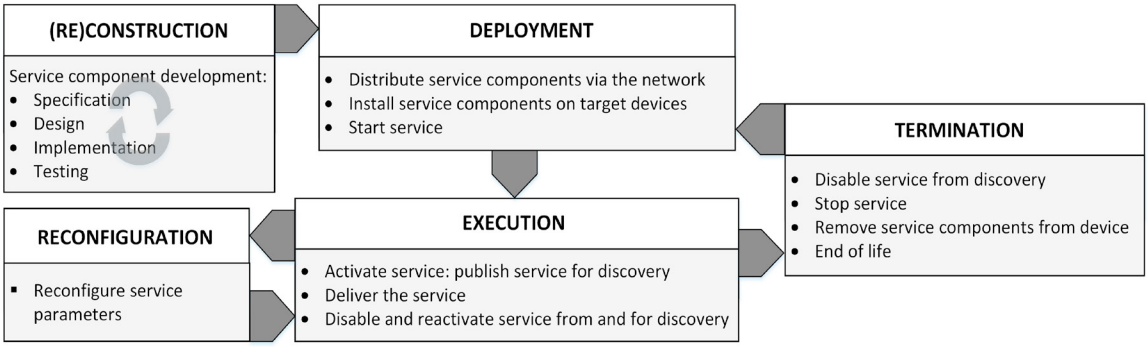

Fig. 3. Generic life cycle for IoT services.

The last stage in the device life cycle is decommissioning. In some cases, an IoT device needs to be decommissioned, for example because it will be commissioned in another network, or because it is due for replacement which is the case when the device reaches its end of life. If the device is to be commissioned in another network, resetting the device to its factory configuration is an important activity so that the device can be perceived as a new device which enables bootstrapping and joining the new secure network. This activity should be handled properly to improve usability. Another important activity in decommissioning is to disconnect the device from servers. This is to notify the system that the device is no longer available, therefore the system can adapt quickly to this change. Allowing the device to perform decommissioning properly will help usability and adaptability [13].

2.2. Generic Life Cycle for IoT Service

A service is a contractually specified overall functionality (semantics) of an entity. As shown in Fig. 1, an IoT service is used by IoT applications to achieve their goals. Any IoT service should follow the generic life cycle for IoT service as shown in Fig. 3. However, the realizations of the activities inside the life cycle stages and the corresponding stakeholders concerns vary from one system to another.

The (re)construction stage pertains to the construction and reconstruction of service components for commissioned devices. One iteration of reconstruction results in a new update for the service component and such iteration is in parallel with the rest of the service life cycle. When this update is available, the currently running service goes to the deployment stage, following the path of the service life cycle. Both construction and reconstruction follow the common software development life cycle of specification, design, implementation and testing to ensure functional appropriateness [18]. By taking into account certain standard specifications and plug-fest test during service development, we can ensure that the service is interoperable with applications and other services during run-time. The goal of the (re)construction stage is therefore to develop services and ensure their functional appropriateness and interoperability with applications and other services during run-time.

When a service is ready for install, it goes to the deployment stage. Fig. 3 shows the activities inside this stage. The goal of this stage is to realize service deployment via the network, as once IoT devices are installed and commissioned, physical deployment is no more feasible due to their high number and remote placement. If the deployment stage is properly implemented, service deployability [13] for the system will be achieved. A proper realization of the deployment stage also allows for cost-effective service update during system run-time which is important for modifiability.

The next stage is execution. During execution, the service is published for discovery and is ready to serve requests from applications or other services. The service discovery can be disabled from and reactivated for discovery. The goal of this stage is to enable the service to be used by a growing number of applications and other services while maintaining acceptable performance, or in other words, to achieve service scalability. Certain architectural decisions, such as the use of proxy and caching or the implementation of the publish and subscribe architectural style can help to achieve this goal.

The reconfiguration stage allows the service parameters to be adjusted via the network. This is supported by the device management functions during the update stage of the device life cycle (Fig. 2). A proper implementation of this stage helps modifiability as service parameters can be adjusted during run-time.

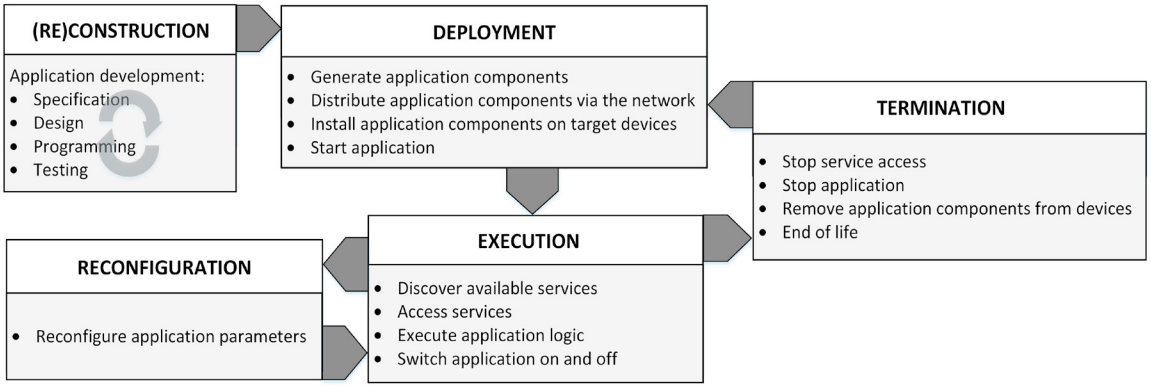

Fig. 4. Generic life cycle for IoT applications.

The last stage of the service life cycle is termination. Two cases are possible for service termination: first, service component update is available and ready to be deployed; second, the service is no longer used or developed, which is the case when the service reaches its end of life. Proper termination of the service is important for maintaining adaptability [18] of the system, as removal of the service from the system needs to be detected quickly.

2.3. Generic Life Cycle for IoT Application

IoT applications provide abstractions for users from the network and platform details of an otherwise very complex IoT system. To that end, it is important to understand the life cycle for IoT application as shown in Fig. 4. Construction and reconstruction of an IoT application are normally done by users who are not experts in software development. Application development in this stage should therefore be a simplified process of software development. It entails orchestrating available services to fulfill a desired goal. In the simplest case, this is restricted to inspecting collected data. In more advanced cases this entails the configuration of complex sensor/logic/actuator connections in the lower layers of the system. Possible means for developing IoT applications include: a graphical user interface (GUI), a domain specific language (DSL), advanced APIs or voice commands. Therefore, a proper choice of process and means to develop IoT applications is required to achieve usability.

When users develop an IoT application, they do so without full knowledge of the target platform. Such knowledge is to be added during deployment. For that reason, the deployment stage entails generating application components, distributing the components via the network, installing the components to target devices and starting the components. An application source code, e.g. represented in a DSL, can be compiled into a specific programming language code, such as C, C++ or Java, or into scripts, such as Python or JavaScript. In the first case, the code is further compiled into either a machine code, a byte code or an executable. An IoT device that hosts the application may need an interpreter for executing scripts or byte codes; or a link-loader for generating executables from machine codes. The goal of the deployment stage is to enable application deployment via the network as users have no physical access to nor knowledge of the target devices. The realization of automatic application deployment via the network improves modifiability [16 13]. Other than that, a proper realization of this stage is important for achieving application deployability [13].

Once deployed, the IoT application goes into execution stage. At this point, the application is ready to perform its expected behavior. It does so by first discovering available services identified in its instructions. Once discovered, the application accesses these services and uses them to execute its specified application logic. During execution, the application can also be switched on and off. When it is off, the application still exists, but it stops executing temporarily until it is switched back on. During execution, users experience the behaviors and functionalities that they expect from the application. Therefore, a proper realization of this stage is important for achieving functional appropriateness [18].

The reconfiguration stage allows users to adjusts application parameters via the network. Users require an easy-to-use interface to reconfigure the application. A proper realization of this stage will improve usability and modifiability.

The last stage in the generic IoT application life cycle is termination. Two cases are possible for termination. First, an update of the application is available and ready to be deployed. In this case, the application goes to termination and

then goes to deployment stage. Second, the application is no longer used or developed. In this case, the application reaches its end of life. In the termination stage, the application should stop accessing services, stop executing and remove itself from its host devices. A proper realization of these termination activities helps adaptability as the system can recognize application removal in a timely manner and synchronizes its state accordingly.

3. Conclusion

Our proposed generic life cycles for IoT devices, IoT services and IoT applications provide better understanding on the problems that need to be solved by IoT systems throughout their life cycles. As an example, through the generic life cycle for IoT applications, we understand that IoT applications are not restricted to orchestrating collected data from some database services, as presumed in many IoT discussions. Instead, they can also be configurations in the lower layers of the system, i.e combining sensing and actuating services with application logic on the lower layer devices. Our proposed generic life cycles shed light on the problems entailed for realizing these types of IoT applications. We conclude that our proposed generic life cycles for IoT can help system architects specify and design robust and effective IoT systems. Without taking into account the life cycle needs of an IoT system, desired functionalities and quality attributes for the system are in danger of becoming unrealizable.

Acknowledgment

This research was performed within the framework of the strategic joint research program on Intelligent Lighting between TU/e and Philips Lighting B.V.

References

1. Muller, G.. *CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity*. PhD dissertation; Technische Universiteit Delft; 2004.
2. Muller, G.. System Architecting. Tech. Rep.; 2016. URL: http://www.gaudisite.nl/SystemArchitectureBook.pdf.
3. Muller, G.. CAFCR+ Introduction. 2018. URL: http://architectingsystemperformance.eitdigitalx.eu/rich_media_page/cafcr-introduction-3; accessed: 2018-02-01.
4. ISO/IEC/IEEE 42010 A Conceptual Model of Architecture Description. 2018. URL: http://www.iso-architecture.org/42010/cm/.
5. Yoo, M.J., Grozel, C., Kiritsis, D.. Closed-loop lifecycle management of service and product in the internet of things: Semantic framework for knowledge integration. *Sensors* 2016;**16**(7).
6. Massingham, I.. The Lifecycle of an AWS IoT Thing from Manufacture to Retirement. 2016. URL: https://www.youtube.com/watch?v=D-5X_Ti1T4U; accessed: 2018-02-01.
7. Works with Nest. 2018. URL: https://nest.com/works-with-nest/; accessed: 2018-02-01.
8. Rahman, L.F., Ozcelebi, T., Lukkien, J.J.. Choosing Your IoT Programming Framework: Architectural Aspects. In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2016, p. 293–300. doi:10.1109/FiCloud.2016.49.
9. Apple HomeKit Framework. 2018. URL: https://developer.apple.com/homekit/; accessed: 2018-02-01.
10. Bui, T.V., Lukkien, J.J., Frimout, E., Broeksteeg, G.. Bridging light applications to the IP domain. In: *2011 IEEE International Conference on Consumer Electronics (ICCE)*. 2011, p. 235–236. doi:10.1109/ICCE.2011.5722558.
11. Mathews, E., Guclu, S.S., Liu, Q., Ozcelebi, T., Lukkien, J.J.. The internet of lights: An open reference architecture and implementation for intelligent solid state lighting systems. *Energies* 2017;**10**(8).
12. Final reference architecture of OpenAIS system. Tech. Rep. d2.7, v1.0; OpenAIS: Open Architectures for Intelligent Solid State Lighting Systems; 2017. URL: http://www.openais.eu/user/file/openais_final_reference_architecture_(d2.7)_v1.0-pub.pdf.
13. Bass, L., Clements, P., Kazman, R.. *Software Architecture in Practice*. Addison-Wesley Professional; 3rd ed.; 2012. ISBN 0321815734, 9780321815736.
14. Garcia-Carrillo, D., Marin-Lopez, R.. Lightweight CoAP-Based Bootstrapping Service for the Internet of Things. *Sensors* 2016;**16**(3).
15. Stolikj, M.. *Building Blocks for the Internet of Things*. PhD dissertation; Technische Universiteit Eindhoven; 2015.
16. Gielen, F.. Software Architecture for the Internet of Things: QA - Modifiability. Coursera Lecture; 2018. URL: https://www.coursera.org/learn/iot-software-architecture/lecture/JTIBq/qa-modifiability; accessed: 2018-02-01.
17. Gielen, F.. Software Architecture for the Internet of Things: QA - Interoperability. Coursera Lecture; 2018. URL: https://www.coursera.org/learn/iot-software-architecture/lecture/KXaEr/qa-interoperability; accessed: 2018-02-01.
18. ISO/IEC 25010 software product quality. 2018. URL: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010.
19. Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.. Performance evaluation of MQTT and CoAP via a common middleware. In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. 2014, p. 1–6.