Towards Model-driven IoT Maintenance

Muzaffar Artikov[1], Dilshodbek Kuryazov[2] and Andreas Winter[2]
[1]Urgench branch of Tashkent University of Information Technologies, Uzbekistan
[2]Carl von Ossietzky Universität, Oldenburg, Germany
muzaffar.artikov@ubtuit.uz, {kuryazov,winter}@se.uol.de

1 Motivation

Internet of Things (IoT) is a network of interconnected "things" that provides certain services in order to reach common goals. They are implemented by the combination of heterogeneous hardware, software, and network technologies. The "thing" can be any real-world object embedding sensors/actuators that are uniquely identifiable. They can collect data through their sensing capability and and their states can be changed through acting. In IoT, the most devices (i.e., things) are equipped with software support to autonomously provide "smarter" services. IoT systems have commonly been termed "Smart X" including Smart Home, Smart City, Smart Grid, Smart Car, Smart Traffic Control, etc [2].

IoT systems consist of various hardware devices, complicated processes and highly connected systems making up heterogeneous systems with huge load of data. IoT encompasses heterogeneous hardware components such as sensors, actuators, networks, gateways and software components, i.e., IoT operating systems, drivers, libraries, 3rd party services, mock-ups. Moreover, the network aspect of IoT is heterogeneous too using the variety of network protocols and different topology. Thus, developing and maintaining IoT systems is quite complex and developers often have to resort to ad-hoc solutions. These ad-hoc solutions might be unstable and their maintenance might require much effort.

IoT systems undergo various maintenance activities such as bug fixes, optimizations or extensions. Lientz and Swanson [3] defined four types of maintenance: *adaptive*, *perfective*, *corrective* and *preventive*. Since IoT is considered as software-intensive system, these four types of maintenance can also be considered in IoT maintenance. *Adaptive* IoT maintenance implies applying changes to IoT systems. For instance, if developers change certain network protocol within a IoT system to another. *Perfective* maintenance implies adding new functionality to IoT systems. For example, stakeholders may require to add new components (e.g., things) to the system. If some errors occur due to the break of some sensors or actuators, fixing them is considered as *corrective* maintenance. *Preventive* maintenance tends to avoid possible problems in the future.

The main reason that causes majority of challenges in IoT development and maintenance is heterogeneity of it's components. Heterogeneous nature of IoT components pertains to the hardware, software and network aspects of IoT, and complicates the development and maintenance of IoT systems. Diversity of development technologies and languages raises a need to shift IoT systems to more abstraction levels.

Like any software system, IoT systems should be maintained to keep them stable and operational. But, IoT systems are typically deployed in distributed environments where they operate making them difficult to maintain, test and debug remotely. Maintaining IoT systems in their operating environments (i.e., in production) is cumbersome. If they are maintained in production environment, it might lead to interruptions, failures or delays of production and serviceability.

In order to minimize risks in production and to continually improve the quality of IoT solutions, the developed IoT systems have to be maintained without halting their serviceability. Thus, they have to be shifted to some level of abstraction to handle their maintenance and to keep their components operate together.

2 IoT Abstraction by MDE

Model-Driven Engineering (MDE) is the modern style of abstracting planned or existing software-intensive systems. It supports well-suited abstraction concepts to develop and maintain software projects. A higher level of efficiency in IoT development, maintenance and integration can be achieved by the MDE-based abstraction of different soft- and hardware components and units.

Model-driven IoT [1] provides abstraction of physical components and their software support. It eases maintenance of IoT systems by isolating the problem area and predicting future errors in the system. This abstraction can demonstrate the impact of design changes, usage scenarios, environmental conditions, etc.

In order to achieve the higher abstraction levels of IoT systems, the following main model-driven abstraction languages (i.e., domain-specific languages) can be defined to describe their different aspects:
Thing Description. As "things" in IoT can be hardware components, these components and their behavior should be described in some ways how they behave, where they are located, how they are utilized, e.g., how their APIs look like. These hardware components can be abstracted by using a device description language.
Hardware-Software Mapping Description. In IoT, all hardware components are associated with a piece of software, e.g., drivers, network protocols or other soft-

ware services. Thus, mappings between hard- and software components have to be described using a language in order to wire up right software components with right hardware components.
Rule System/Logic Description. "Things" in IoT operate based on control logic consisting of states, state-transitions, events and actions. This control logic follows certain rule systems which define the chains of states, transitions and actions. For developing the control systems in IoT, their rule systems have to be defined using a rule description language.
Communication Description. As IoT systems consist of various individual embedded units and components, they have to communicate with each other to fulfill certain tasks. Thereby, they perform specific tasks by the orchestrations of various soft- and hardware components based on control and data flow. Communication among various IoT components can be abstracted by a communication description language.
Integration Description. After having a set of description languages, they have to be integrated to achieve common goals. This can be accomplished by defining an integration description language [4].

These are the minimal set of description languages for abstracting IoT systems in order to achieve a higher lever of efficiency in IoT development and maintenance.

3 Maintenance

Section 2 has given some ideas for abstracting IoT systems by MDE concepts. These abstractions help to resolve IoT maintenance challenges explained in Section 1 focusing on adaptation, perfection, correction and prevention [3]:
Adaptation. IoT systems provide various smart services, yet, their underlying hardware and software support is similar. This common underlying technology base has to be adapted and configured differently to provide various services. These adaptations and configurations can be easily done using the MDE-based abstractions, e.g., the *Thing Description* language instead of having knowledge on several implementation technologies. For instance, if a hardware component is replaced by another, it can be described by the *Thing Description* language and mapped to right software using the *Hardware-Software Mapping* language.
Perfection. Usually, there is a need for optimizing and extending IoT components which yields perfection. As the MDE-based abstraction languages are completely independent from any implementation technologies and languages, improvements and perfections of IoT systems are easy using these languages (all languages in Section 2) rather than focusing on each individual technology.
Correction. Like any system, IoT systems are prone to errors and bugs. If errors or bugs occur due to the break of sensors or actuators, they have to be debugged and fixed. Bug fixes are usually done by testing. The MDE-based IoT systems can be tested based on the recorded data and predefined success states without interrupting real systems. As the integral part of the fast prototyping, the success states, test cases and scenarios can be predefined before actually further maintaining IoT systems enabling test-driven development.
Prevention. It is quite essential to prevent possible future failures in IoT systems as their services are directly concerned with the daily lives of people and industries. Thus, possible failures have to be predicted and prevented in advance without interrupting the whole system's activity. In case of the industrial IoT systems, they and their components have to be maintained without interrupting production. It can initially be assured before actually deploying in production if they are abstracted and separated from their actual production. For instance, this can be done by describing success states using the *Rule Description* language which allows for easy maintenance and fast prototyping of IoT solutions with high quality.

4 Conclusion

IoT systems are heterogeneous and realized using various implementation technologies and languages making them difficult to develop and maintain. This research proposes abstraction of IoT systems from their underlying implementation technologies and their servicing environments. There, MDE concepts provide a means to reverse engineer and design IoT systems enabling separation of concerns which eases reusability, applicability and adaptability in development and maintenance of IoT systems. As the future work, this research intends to develop MDE-based abstraction languages defined in Section 2 for covering the challenges in Section 1 and meant to be enabler for easy development and maintenance of IoT systems.

References

[1] F. Ciccozzi and R. Spalazzese. MDE4IoT: supporting the internet of things with model-driven engineering. In *International Symposium on Intelligent and Distributed Computing*, pages 67–76. Springer, 2016.

[2] D. Kuryazov, A. Winter, and C. Schönberg. Towards Collaborative Smart City Modeling. In *International Scientific-Practical and Spiritual-educational Conference to explore the importance of information and communication technologies in the innovative development of real sectors of the economy, TUIT*, volume 5, Tashkent, 04 2018.

[3] B.P. Lientz and E.B. Swanson. Software maintenance management. Addison-Wesley, 1980.

[4] J. Meier and A. Winter. Model Consistency ensured by Metamodel Integration. In R. Hebig and T. Berger, editors, *6th International Workshop on The Globalization of Modeling Languages (GEMOC), co-located with ACM/IEEE MODELS 2018*, pages 408–415, Copenhagen, 10 2018. CEUR Proceedings of MODELS'18 Workshops.