Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 134 (2018) 75–82

Procedia Computer Science

www.elsevier.com/locate/procedia

The 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018)

Model-Based Runtime Monitoring of Smart City Systems

Koray Incki[a,*], Ismail Ari[a]

[a] *Ozyegin University, Cekmekoy Kampusu Nisantepe Mah. Orman Sk. 34794 Cekmekoy, Istanbul 34794, Turkey*

Abstract

The pace of proliferation for smart systems in city wide applications is unmatched. The introduction of Internet of Things (IoT), an enabler of smart city phenomenon, has incubated a productive environment for such innovations. Smart things equipped with IoT capabilities, allow for developing smart city applications at such large scale that each application can be represented as a system of systems (SoS). Nevertheless, the complexity of engineering such SoS has been a major challenge in developing and maintaining smart city applications. One of the engineering challenges that industry face today is the verification of a SoS smart city application at runtime. We introduce utilization of a model-based runtime monitoring approach for providing reliable service. We propose to use message sequence charts for representing a smart city application, later allow the practitioners to express expected behavior of an application in terms of complex-event processing patterns. We demonstrate the fidelity of our approach on a sample smart parking system. Our approach is one of its kind in enabling a non-intrusive monitoring of IoT behavior at runtime (online).

Peer-review under responsibility of the scientific committee of the 13th International Conference on Future Networks and Communications, FNC-2018 and the 15th International Conference on Mobile Systems and Pervasive Computing, MobiSPC 2018.

Keywords: runtime monitoring; component-based iot; model-based testing; internet of things; complex-event processing; intelligent transportation; smart city

1. Introduction

Thanks to improvements achieved in fitting such great computing power and memory in a small form factor that smartness flavor has permeated every domain of industry and research venues. Technological breakthroughs such as IoT have already been identified as enablers of smart objects in various applications [1]. Software and software systems have become so widespread that we cannot think of any aspect in everyday life that does not involve a product with such a system in it. The scale of computing capabilities has penetrated a diverse spectrum of application domains encompassing end user artifacts such as key-chains, all the way up to an industrial factory automation. A technology innovation driven by introduction of IoT phenomenon has increased the momentum towards implementing systems of systems with more smartness features [1]. The driver behind such an impetus of IoT is that it allows for engineering such integrated systems that merely require human interaction; thereby, enabling invention of new functionalities

* Corresponding author. Tel.: +90-505-408-0085.
E-mail address: koray.incki@ozu.edu.tr

Peer-review under responsibility of the scientific committee of the 13th International Conference on Future Networks and Communications, FNC-2018 and the 15th International Conference on Mobile Systems and Pervasive Computing, MobiSPC 2018.
10.1016/j.procs.2018.07.146

that is composed of autonomous endpoints operating in an accord with respect to a predefined service composition definition.

Every new capability built on groundbreaking concepts brings along a challenge in engineering the systems that we used to develop with older conventions. IoT systems have already shifted the engineering paradigm for building systems of systems (SoS). The SoS that are architected with IoT principles exemplifies a domain of large-scale systems with so intricate detailed design elements that it necessitates innovative engineering solutions to tackle with. Because, the size of an IoT system might scale from a system of tens of endpoints up to thousands (Fig.1). According to a recent research by Gartner[1] the estimated number of IoT devices installed in year 2020 will be 20,415 millions of units. The total number of IoT devices installed in year 2017 has already passed the total number of human beings on this planet.

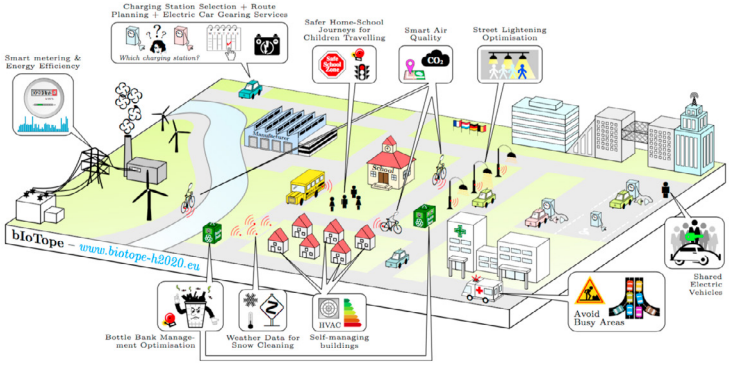


Fig. 1: A Large-Scale Smart City Application[2]

IoT systems are enabled by several underlying technologies (e.g., communication protocols, wireless sensor networks, service-oriented architecture, complex-event processing, etc.). One of the those technologies that makes those system easy to design, develop and test are the application layer communication protocols. Event though those systems might be built around hundreds or thousands of devices that are manufactured by different companies, the service-oriented architecture (SOA) principles adopted by some of the protocols facilitates the engineering process. Constrained-Application Protocol (CoAP), an OSI Layer-7 application layer protocol [3], is designed with RESTful API guidelines ([4]) as the building block of the protocol. The IETF Constrained RESTful Environments (CoRE) working group ([5]) has compiled the best practices as CoAP standard for supporting a SOA-based application development for resource constrained devices in IoT. RESTful services perspective has a tremendous advantage over other architectural approaches when it comes to discovery, composition, integration and execution of various services procured by different subsystems. Thus, engineering an IoT as a SoS is extremely streamlined with the introduction of such application protocols as CoAP.

A smart city infrastructure (Fig.1) hosts diverse applications of IoT systems [2]. The elements of such a system might contain smart parking, smart traffic lights, smart metering and similar smart vertical domain applications. That's why, such a system might be composed of thousands of interconnected IoT enabled endpoints. This paper extends our research reported in [6] and [7] onto smart city domain, and address the verification challenge of engineering a smart parking system with IoT devices. We particularly focus on runtime verification (RV) of such a system. We first explore how model-based testing can be utilized for representing the expected behavior of such a system. Then, we transform the messaging interactions amongst the IoT endpoints in the network into executable runtime monitors in the form of complex-event processing (CEP) statements by using a seamless model-to-text (M2T) transformation in a model-based testing (MBT) environment. We believe that researchers and practitioners of smart city applications will benefit from our solution approach.

The remaining of the paper is organized as follows: Section-2 explains the domain of runtime verification, where we borrow the runtime monitor definition. In Section-3 we outline the case strudy by defining the faults under scrutiny

[1] https://www.gartner.com/newsroom/id/3598917

in smart parking domain. Then, Section-4 explains our contributions in MBT approach. Later, we cover a short list of related work in Section-6, then we conclude the paper with future directions in Section-7.

2. Background

Recent developments in *software-as-a-service* business has made it possible for software systems to become much more prevalent than ever. Those systems operate more autonomously thanks to innovations introduced as agent-based smart objects [8]. Such improvements in engineering software systems necessitate innovative approaches to verification of those in order to enable reliable system operation.

RV has usually been considered as an intimidating verification method by the practitioners ([10]). That's why, it is poorly utilized in the industry. However, large-scale systems such as IoT call for feasible solutions for RV, because of the fact that those systems are deployed with such devices that are line-replaceable-units (LRUs); which means that an IoT device can be easily swapped with another one providing similar or new services. This capability has been introduced owing to the SOA based CoAP protocol.

With a broad perspective, verification methods can be listed as (i) *theorem proving*, (ii) *model checking*, (iii) and *testing* [9] (see Fig.2). Theorem proving explores the correctness of a software by using certain proof techniques. Model checking, a technique that generally requires formal analysis of the system under test (SUT), is more related with verifying a software automatically at design time (i.e., without executing the SUT on production environment), which is usually represented as finite-state machines. On the other hand, testing is practically examining the SUT to demonstrate non-existence of faults in it [9].

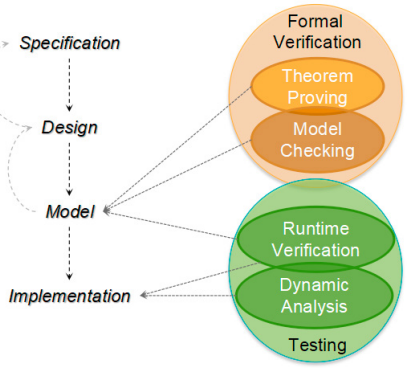


Fig. 2: Verification Techniques

Fig.2 summarizes the relation between different verification techniques. Theorem proving and model checking are types of formal verification, which involve mathematical models and proofs of system correctness through formal algebra. Those two types of verification constitute static analysis methods. However, RV, having its roots in formal specification of a SUT and being applied to SUT at runtime, is a complementary technique between functional testing and formal verification. It allows for reacting to unexpected behaviors at runtime [9].

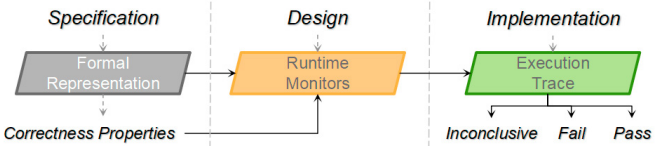


Fig. 3: Runtime Verification Recapped

Fig.3 recaps the principles of RV technique. *Runtime verification* is defined as examining a systems execution trace against its specification expressed in correctness properties. A *correctness property* formally captures the expected

behavior of a particular system specification, which enables checking whether a runtime instance of the system conforms to that specification or not via runtime monitors. A *runtime monitor* is a tool that observes execution trace of a SUT and returns verdicts on the verification of a particular specification against its correctness property. *Execution trace* represents a finite run of a program, usually expressed in SUT's states.

3. Smart Parking Problem Re-Defined

Smart parking systems (SPS) are prevalently deployed in many of the smart cities around the world[2]. Those systems are designed with usability considerations in order to maximize the effective utilization of limited parking resources around a city. The smart parking capability is enabled by installing smart sensors (Fig.4) with IoT features that seamlessly integrate to a back-end system.

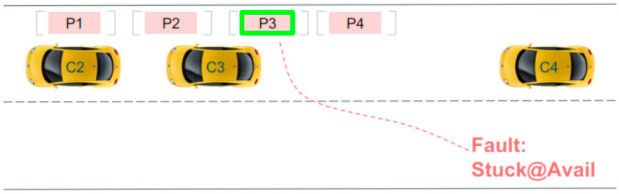
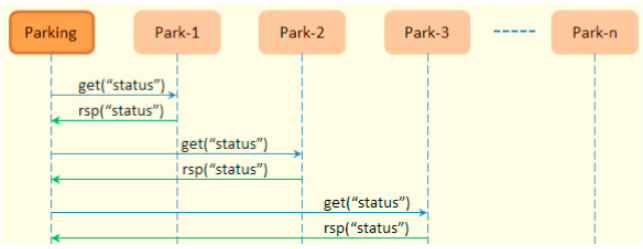


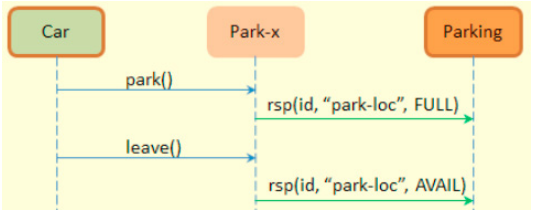
Fig. 4: Smart Parking Problem

The usability of a SPS is achieved through a component-based design of overall architecture, which provides its resources as services. The service-oriented architecture is enabled by using CoAP ([3]) on IoT devices. CoAP allows designers to describe system behavior by using RESTful-like APIs; thereby, supporting utilization of message-sequence charts [3] (MSC) as design method. In a previous work [6] we proposed to use MSC as the representation tool for expected behavior of an IoT system; because, it enables us to express the behavior in terms of event calculus (EC) ([6]). A *Parking* back-end system periodically collects the status of each parking space, then each sensor at a particular slot (i.e., *Park-1, Park-2,..*) responds with a current value of its state, via CoAP message exchanges (Fig.5a).

A use case scenario of SPS can be depicted by using MSC's as in Fig.5b. The reservation of a certain parking space starts by a *Car park*ing in an empty slot. The *Park-x* slots interrogates the *id* of the *Car* via a built-in system on it or through a software tool on the customer's smart phone. Then, *Park-x* updates its status to *Parking* back-end system. The *leaving* of *Car* from *Park-x* renders the slot available again.



(a) A sample sequence diagram for smart parking



(b) Expected operation of smart parking

Fig. 5: Smart Parking Represented in MSCs

As we explained in Section-1, IoT is made possible via several enabling technologies; which means that a fault can occur due to many reasons. However, we're interested in two particular failures of a SPS for demonstration of our contribution. Specific fault that we seek to detect is *Stuck-At*, which might arise due to a failure at the sensor node.

[2] http://www.smart-cities.eu/
[3] http://www.itu.int/rec/T-REC-Z.120/en

Stuck-at can occur when a parking sensor gets stuck at a particular status. When this fault arises, new *Car*'s are not notified correctly on the availability of the parking spaces, which hinders the usability of SPS. In the next section, we describe how to utilize MBT for expressing those faults by using EC and CEP methods.

This kind of failures cause a SPS to lose its grip on providing an effective, efficient, and green smart parking solution. Because, *Cars* are oriented towards, in fact *occupied* spaces, causing those to consume more time and fuel.

4. Model-Based Runtime Monitoring for Smart Parking

Providing an online and non-instrumented RV solution for such IoT systems as described above is the main motivation of this paper. We seek for an online solution because SoSs engineered with LRUs require instant verification of sustained reliability. The driving idea behind providing a non-instrumented approach in such environments as IoT where those devices from various manufacturers can be plugged in and out of the SoS at will, is that we can not inject any instrumentation code to proprietary commercial-off-the-shelf products. That's why we adopt the framework that we proposed in [7], and do not go into a detailed design in this paper.

We streamline the process of designing for runtime verification, which establishes a foundation on representing messaging model of CoAP in terms of events, then utilizing CEP techniques to yield success/failure verdicts on observed behaviors of a SoS. The MBT approach is achieved through deriving an extended UML profile on an open-source modeling tool [7]. Owing to the domain-specific modeling capability provided with UML profile extension, we can use sequence diagrams (SD) to describe both the expected behavior and faulty behavior of a system.

4.1. Event Stream Processing for Verification

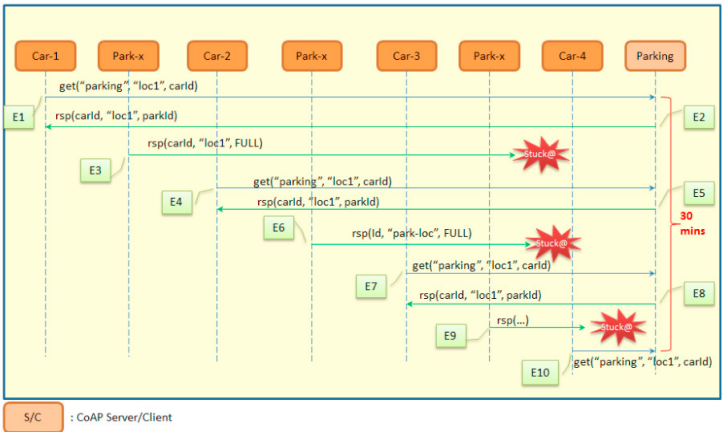


Fig. 6: A sequence diagram depicting "Stuck-At" fault

Let us now describe how we can represent those faults by using SD's. Note that, in [6] we showed how Esper[4] can be utilized in revealing the complex relations between simple events taking place in an IoT network. Fig.6 illustrates the behavior manifested by the SUT whenever a *Stuck-At* fault occurs. This scenario depicts that a parking sensor, *Park-x*, has a *Stuck-At* fault, which causes it to report its status as *available*, even though the space is *occupied*. The failure manifests itself when 4 *Cars* consecutively queries for available space in *Loc1 within 10 minutes*, and *Parking* system does *NOT* receive any status update from *Park-x* for its new status as *occupied*. This failure raises a problem in the SPS as directing *Cars* to *occupied* spaces supposing they are *available*.

[4] http://www.espertech.com/

E1: {GET, "car1", "parking", "loc1"} | **E2:** {RSP, "loc1", parkId} | ¬**E3:** {RSP-OBS, "car1", "loc1", FULL}
E4: {GET, "car2", "parking", "loc1"} | **E5:** {RSP, "loc1", parkId} | ¬**E6:** {RSP-OBS, "car2", "loc1", FULL}
E7: {GET, "car3", "parking", "loc1"} | **E8:** {RSP, "loc1", parkId} | ¬**E9:** {RSP-OBS, "car2", "loc1", FULL}
E10: {GET, "car4", "parking", "loc1"}

A car parking scenario consists of tuples of events:
$P=\langle E_i,E_j,E_k \rangle$ where $\{\exists (i,j,k) \mid E_i.msgId = E_j.msgId \ \& \ t_i < t_j < t_k \ \& \ E_i.carId = E_k.carId\}$

Stuck@Avail Fault occurs when $\langle E_1,E_2\rangle$, $\langle E_4,E_5\rangle$, $\langle E_7,E_8\rangle$, $\langle E_{10},E_{11}\rangle$ 4 pairs of events occur without any E_{park}(carId, locId, F) event occurring in between those pairs in 30 minutes.

Fig. 7: Events of a *Stuck-At* case

4.2. Capturing Complex Relations

The events of Fig.6 can be defined as in Fig.7. By applying the process described in [7], we can have a Esper statement as in Fig.8. Those EPL [6] statements constitute the *runtime monitors* as described in Section-2.

```
insert into E1.unique(carId, locId)
insert into E2.unique(carId, locId)
create TimedEventWin with parkId when select parkId, carId from E2.unique(carId, locId)
insert into TimedEventWin select * from E1.unique(carId, locId) as e1, E2.unique(carId, locId) as e2
        where e1.carId = e2.carId and e1.locId = e2.locId
select parkId from TimedEventWin.length_batch(4) where count(parkId) group by parkId
within.time(30min)
```

Fig. 8: CEP statements for *Stuck-At* expressed in Esper's EPL

5. Discussion

Smart parking systems have been deployed in various cities of the world since 1996 with the introduction of Stadinfo Köln project [12]. The survey conducted by Lin et al. [12] emphasizes the significance of smart parking solutions in driving the motivation for smart city projects. According to the survey, there are 50000 sensors installed in city of Moscow, which is the largest installation case for a smart parking solution. Considering the scale of those parking systems, it's obvious that engineering such large scale SoS's require an immense effort in maintaining those, as well.

Model-driven engineering not only enables the practitioners with an design-based development of systems, but also promises to perform a seamless maintenance and fault-tolerance experience on those systems. Runtime monitoring of smart parking systems is a phase of maintenance and verification processes. That's why the contributions of this paper equips the practitioners with the ability to deploy un-intrusive monitoring solution on those large-scale systems with such an ease.

The framework proposed in [7] employs a model-driven engineering solution for runtime monitoring. Our solution approach adopts an event-based monitoring of network communication messages between CoAP endpoints. The architecture uses a complex-event processing (CEP) engine for analyzing the patterns of interactions in order to detect a deviation from an expected behavior, which is to be specified via a sequence diagram. By using our solution, one can utilize UML diagrams not only for specifying design of interactions, but also implicitly describe runtime monitors for examining performance of the final application. The framework seamlessly analyzes the sequence diagrams, and derives event relations out of those, then automatically generates CEP expressions. Those CEP expressions are later deployed in a CEP engine for runtime monitoring. As we stated in [7], the complexity of CEP engine for monitoring a smart parking system is linearly proportional to the unique instances of interactions in the system.

6. Related Work

Monitoring of activities and data has also been studied in provenance domain [13–18]. These studies mainly focused on W3C metadata standards such as OPM and PROV-O used for defining the lineage of the data. In [19], they contribute a solution for tackling the challenges introduced with verification of an IoT system running CoAP against CoAP standard. The proposed framework operates in an offline fashion, which incurs logging of all CoAP traffic, then exerting verification test scenarios on those raw packages. The test cases that examine observance of certification are pre-coded and fixed cases. This solution can not be easily ported to an online testing scenario.

Medhat et al. [22] propose to implement such a runtime monitoring approach that buffers any events that take place in between any two consecutive calls to monitor, and later examined by the monitor whenever it's called to do so. In their approach, the monitored entity is the embedded device itself, so they don't provide a SoS perspective on interconnection of those devices. Moreover, their solution requires to interfere with the original execution of a device by instrumenting at the source code level. Therefore, this solution entails more resources in terms of memory and processing power when compared to our solution.

A predictive runtime monitoring approach has been proposed in [23] for cyber-physical systems (CPS). Their motivation is to provide a solution that will prevent any failure before happening by using predictive techniques. In order to achieve this goal, the SUT must be instrumented at the source-code level, which will generate events at runtime. Again, this solution does not deal with system aspects of IoT devices.

A runtime monitoring architecture has been contributed to the literature by Kane in [24]. In his research, Kane observes the runtime behavior of a safety-critical vehicular systems by adopting a black-box testing approach. The solution architecture is built on CAN network by using a passive bus-monitor. The bus-monitor attaches itself to the network of SUT, and records the runtime behavior as images. Afterwards, the monitor analyzes those recorded images for revealing any faulty behavior. Although, this research builds on similar principles as ours, we differentiate from their research in terms of formal specification method, the model-driven engineering approach for generating runtime monitors, using CEP as runtime monitoring medium, and finally the domain of discourse, IoT. Moreover, our solution operates on live network traffic, whereas they propose to use recorded images of execution traces of an SUT.

7. Conclusion

RV is a complementary verification technique along-side model-checking and theorem proving. Contrary to the value it adds to reliability of software systems, it has been poorly utilized in the industry. This is due to the fact that the formal methods employed in RV has been intimidating the practitioners. In this paper, we explained how we promote to proliferation of RV by contributing a MBT based runtime monitoring approach. The case study on smart parking demonstrates that representing CoAP behaviors on SD's help express them in event domain, whereby, we utilize CEP statements as runtime monitors. The future holds promising research in extending the work to cover other fault types and system configurations. The contributions in this research encourages investigation of new approaches to RV challenges of large-scale systems, especially those that are engineered with SOA principles (e.g., CoAP). That's why, we intend to explore portability of the contributions other domains with those principles.

References

[1] G. Fortino and P. Trunfio, "Internet of Things Based on Smart Objects, Technology, Middleware and Application", *Springer*, 2014, doi: 10.1007/978-3-319-00491-4

[2] S. Kubler, "Building an IoT OPen innovation Ecosystem for connected smart objects, Horizon 2020", 2015. [Online]. Available: https://cordis.europa.eu/project/rcn/200391_en.html.

[3] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," IETF RFC-7252. 2014.

[4] R. T. Fielding and R. N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115150, May 2002.

[5] Core, "IETF Constrained RESTful Environments (core) Working Group," 2018. [Online]. Available: https://datatracker.ietf.org/wg/core/about/.

[6] K. Incki and I. Ari, "A Novel Runtime Verification Solution for IoT Systems," *IEEE Access*, vol. 6, pp. 13501-13512, 2018. doi: 10.1109/ACCESS.2018.2813887

[7] K. Incki and I. Ari, "Democratization of Runtime Verification for Internet of Things", *Elsevier Computers & Electrical Engineering*, vol. 68, 2018, pp. 570-580, doi:10/1016/j.compeleceng.2018.05.007.

[8] G. Fortino, W. Russo, C. Savaglio, W. Shen, and M. Zhou, "Agent-Oriented Cooperative Smart Objects: From IoT System Design to Implementation," *IEEE Trans. Syst. Man, Cybern. Syst.*, pp. 118, 2017.
[9] M. Leucker and C. Schallhart, "A brief account of runtime verification," *J. Log. Algebr. Program.*, vol. 78, no. 5, pp. 293303, 2009.
[10] J. P. Bowen and M. G. Hinchey, "Seven More Myths of Formal Methods," *IEEE Software*, vol. 12, no. 4, pp. 3441, Jul. 1995.
[11] C. Smythe, "Initial Investigations into Interoperability Testing of Web Services from their Specification Using the Unified Modelling Language," in *Proc. of International Workshop on Web Services Modeling and Testing (WS-MaTe 2006)*, 2006.
[12] T. Lin, H. Rivano, F. Le Moul. A Survey of Smart Parking Solutions. IEEE Trans. on Intelligent Transportation Systems, IEEE, 2017, 18 (12), pp. 3229-3253. doi: 10.1109/TITS.2017.2685143¿
[13] P. Chen, B. Plale, M.S. Aktas, Temporal representation for scientific data provenance, E-Science (e-Science), 2012 IEEE 8th International Conference on, 1-8, 2012
[14] S. Jensen, B. Plale, M.S. Aktas, Y. Luo, P. Chen, H. Conover, Provenance capture and use in a satellite data processing pipeline, IEEE Transactions on Geoscience and Remote Sensing 51 (11), 5090-5097, 2013
[15] M.S. Aktas, B. Plale, D. Leake, N.K. Mukhi, Unmanaged workflows: Their provenance and use, Data Provenance and Data Management in eScience, 59-81, 2012.
[16] M.S. Aktas, M. Astekin, Provenance aware runtime verification of things for selfhealing Internet of Things applications, doi.org/10.1002/cpe.4263, Concurrency and Computation: Practice and Experience, 2017
[17] M.J. Baeth, M.S. Aktas, A large scale synthetic social provenance database, Proceedings of the Ninth International Conference on Advances in Databases, Knowledge, and Data Applications, pp. 16-22, 2017.
[18] P. Chen, B. Plale, M.S. Aktas, Temporal representation for mining scientific data provenance, Future generation computer systems, vol 36, pp. 363-378, 2014.
[19] N. Chen, C. Viho, A. Baire, X. Huang, and J. Zha, "Ensuring Interoperability for the Internet of Things: Experience with CoAP Protocol Testing," *Automatika*, vol. 54, no. 4, 2013.
[20] A. Ahmad, F. Bouquet, E. Fourneret, F. Le Gall, and L. B., "Model-Based Testing as a Service for IoT Platforms," in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*. ISoLA 2016, vol. 9953, M. T. and S. B., Eds. Springer, 2016.
[21] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovskia, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *J. Syst. Softw*., vol. 136, pp. 1938, 2018.
[22] R. Medhat, B. Bonakdarpour, D. Kumar, and S. Fischmeister. "Runtime Monitoring of Cyber-Physical Systems Under Timing and Memory Constraints",*ACM Trns. on Embedded Computing Systems*, Vol. 14, No. 4, 2015.
[23] K. Yu, Z. Chen and W. Dong, "A Predictive Runtime Verification Framework for Cyber-Physical Systems," *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, 2014.
[24] A. Kane, "Runtime Monitoring for Safety-Critical Embedded Systems," *Dissertations CMU*, 532, 2015.