



UNIVERSITÀ
DEGLI STUDI
FIRENZE

FLORE

Repository istituzionale dell'Università degli Studi di Firenze

Smart-troubleshooting connected devices: Concept, challenges and opportunities

Questa è la Versione finale referata (Post print/Accepted manuscript) della seguente pubblicazione:

Original Citation:

Smart-troubleshooting connected devices: Concept, challenges and opportunities / Caporuscio M., Flammini F., Khakpour N., Singh P., Thornadtsson J.. - In: FUTURE GENERATION COMPUTER SYSTEMS. - ISSN 0167-739X. - ELETTRONICO. - 111:(2020), pp. 681-697. [10.1016/j.future.2019.09.004]

Availability:

The webpage <https://hdl.handle.net/2158/1398837> of the repository was last updated on 2025-01-31T08:51:53Z

Published version:

DOI: 10.1016/j.future.2019.09.004

Terms of use:

Open Access

La pubblicazione è resa disponibile sotto le norme e i termini della licenza di deposito, secondo quanto stabilito dalla Policy per l'accesso aperto dell'Università degli Studi di Firenze (<https://www.sba.unifi.it/upload/policy-oa-2016-1.pdf>)

Publisher copyright claim:

Conformità alle politiche dell'editore / Compliance to publisher's policies

Questa versione della pubblicazione è conforme a quanto richiesto dalle politiche dell'editore in materia di copyright.

This version of the publication conforms to the publisher's copyright policies.

La data sopra indicata si riferisce all'ultimo aggiornamento della scheda del Repository FloRe - The above-mentioned date refers to the last update of the record in the Institutional Repository FloRe

(Article begins on next page)

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/335837494>

Smart-troubleshooting connected devices: Concept, challenges and opportunities

Article in *Future Generation Computer Systems* · September 2019

DOI: 10.1016/j.future.2019.09.004

CITATIONS

36

READS

1,411

5 authors, including:



Mauro Caporuscio

Linnaeus University

87 PUBLICATIONS 1,308 CITATIONS

SEE PROFILE



Francesco Flammini

Mälardalen University

308 PUBLICATIONS 3,167 CITATIONS

SEE PROFILE



Narges Khakpour

Newcastle University

36 PUBLICATIONS 398 CITATIONS

SEE PROFILE



Prasannjeet Singh

Linnaeus University

6 PUBLICATIONS 58 CITATIONS

SEE PROFILE

Smart-Troubleshooting Connected Devices: Concept, Challenges and Opportunities

Mauro Caporuscio^a, Francesco Flammini^a, Narges Khakpour^a, Prasannjeet Singh^a, Johan Thornadtsson^b

^aLinnaeus University, Sweden

^bSigma Technology, Sweden

Abstract

Today's digital world and evolving technology has improved the quality of our lives but it has also come with a number of new threats. In the society of smart-cities and Industry 4.0, where many cyber-physical devices connect and exchange data through the Internet of Things, the need for addressing information security and solve system failures becomes inevitable. System failures can occur because of hardware failures, software bugs or interoperability issues. In this paper we introduce the industry-originated concept of "smart-troubleshooting" that is the set of activities and tools needed to gather failure information generated by heterogeneous connected devices, analyze them, and match them with troubleshooting instructions and software fixes. As a consequence of implementing smart-troubleshooting, the system would be able to self-heal and thus become more resilient. This paper aims to survey frameworks, methodologies and tools related to this new concept, and especially the ones needed to model, analyze and recover from failures in a (semi)automatic way. Smart-troubleshooting has a relation with event analysis to perform diagnostics and prognostics on devices manufactured by different suppliers in a distributed system. It also addresses management of appropriate product information specified in possibly unstructured formats to guide the troubleshooting workflow in identifying fault-causes and solutions. Relevant research is briefly surveyed in the paper in order to highlight current state-of-the-art, open issues, challenges to be tackled and future opportunities in this emerging industry paradigm.

Keywords: resilience, dependability, fault-tolerance, self-healing, self-repair, diagnostics, prognostics, event correlation, log analytics, embedded systems, cyber-physical systems, Internet of Things

1. Introduction

Modern computer systems will be increasingly based on interconnected devices manufactured by diverse vendors. Therefore, it will not be possible to troubleshoot all system failures (e.g. due to interoperability issues) based on single product information. Rather, practical troubleshooting will require appropriate failure detection, identification and information mining from heterogeneous sources. If performed manually, such a process can be highly time consuming and error-prone. By cooperating with industry addressing related technology chal-

lenges¹, we have identified the following core research question:

"How can we promptly recognize anomalies in heterogeneous connected devices (including Embedded Systems, Cyber-Physical Systems and the Internet of Things), and (semi)automatically apply appropriate troubleshooting solutions based on available information, in order to restore correct system operation, reduce the time-to-repair and the probability of maintenance mistakes?"

Such a research question is extremely relevant nowadays due to the growing complexity and pervasiveness of computer systems in many critical domains, including Smart-Cities, Industry 4.0 and Intelligent Transportation Systems. Therefore, it

Email addresses: mauro.caporuscio@lnu.se (Mauro Caporuscio), francesco.flammini@lnu.se (Francesco Flammini), narges.khakpour@lnu.se (Narges Khakpour), ps222vt@student.lnu.se (Prasannjeet Singh), johan.thornadtsson@sigmatechnology.se (Johan Thornadtsson)

¹ Sigma's Notes: Disruptive Innovation in the World Around Us (2015). <https://sigmatechnology.se/news/>

is expected that the complexity of failure modes of connected devices will make system-level failure detection and handling an increasingly challenging task. Due to the heterogeneity of connected devices, manufactured by diverse vendors, no single product information can be easily used to support troubleshooting. Instead, a large amount of information sources should be explored to build knowledge about failure modes, causes (intentional and unintentional), consequences, as well as how to diagnose them and finally repair the system.

Furthermore, restoring normal system operation could require a set of both manual and automatic actions to be coordinated in order to speed-up the repair interventions and avoid imperfect maintenance. Basic repair actions to tackle failures include restarting devices or specific software processes, or upgrading software modules on one or more devices, e.g., to comply with latest communication protocol specifications and prevent/resolve incompatibility issues. More complex actions allow prognostics, prevention of known/similar failure by holistic fault-avoidance, virtual patching, etc. In order to include all those actions, the framework needed to tackle the research challenge we address in this paper could be structured on several levels, including edge, fog and cloud computing. For instance, at edge level, self-diagnostics and self-healing mechanisms should be adopted on the single devices to allow automatic repair and also propagate diagnostic information to other devices and upper monitoring levels to enable troubleshooting.

In such a context, we define “smart-troubleshooting” as the (semi)automatic process required for the identification of system failures, the matching with relevant troubleshooting information, and the application of appropriate recovery and reconfiguration procedures. Please note that such a concept includes the ones of resilience and self-healing, but reinterprets them in a holistic system-of-systems view, also envisioning pragmatic aspects of human-based information processing and security management.

It is evident that the problem of smart-troubleshooting – although relatively easy to specify – is highly challenging due to its multifaceted and cross-discipline nature, entailing aspects of artificial intelligence, formal modeling and data analytics, in order to support online identification of failure patterns, mine troubleshooting information based on a combination of natural language and machine readable sources, match appropriate

solutions to recognized failures, apply those solutions through structured workflow management models and procedures. It also includes several “Self-*” aspects of autonomous and self-adaptive systems like self-monitoring, self-diagnostics, self-reconfiguration and self-healing/self-repair.

Finally, the problem has a very high industry relevance especially for system integrators and maintenance service providers who could have the possibility to implement new paradigms as *Troubleshooting as a Service (TaaS)*. In fact, this study has been performed in the context of an industry cooperation with the company Sigma Technology.

The paper is organized as follows. Section 2 provides background and preliminary definitions, whereas Section 3 discusses the study design of this paper. Findings are discussed in following sections: Section 4 addresses prevention techniques to avoid faults becoming errors and failures, Section 5 discusses error and failure detection, Section 6 discusses techniques for error handling and failure recovery, and finally Section 7 surveys techniques for adapting and evolving system resilience over time. Open-issues, challenges and opportunities are discussed in Section 8. Finally, Section 9 draws conclusions and provide hints about future developments.

2. Background and Basic Definitions

2.1. Reference Taxonomy

In this paper, we follow the taxonomy of dependable and secure computing defined by Avizienis et al. [11], where threats are classified as faults, errors and failures. A state of the system or a part of system state that may entail its eventual failure is called an *error*. Further, the speculated cause of the error is termed as a *fault*; finally, a *failure* occurs when the functionality of the system diverges from its correct behavior. A fault causes an error after a time known as the fault-latency; when the error also activates and propagates, it may cause a failure after the so called error-latency.

The manifestation of a fault is necessary but not sufficient condition for the occurrence of a failure [42]. Faults can be originated by random events (e.g. natural faults) or be intentional (e.g. human made malicious). Therefore, in the reference taxonomy, information security attacks are considered as faults of a different origin and that allows us to simplify and unify our discussion about threat management.

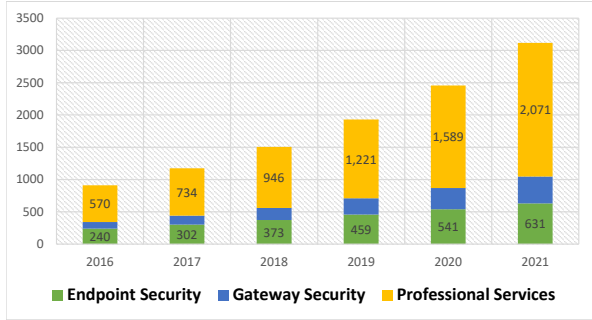


Figure 1: Projected Global IoT Security Spending, in Million USD.

2.2. Motivation

Most modern connected systems, like the ones used in smart houses, smart cities and smart factories, include concepts inherited from the Internet of Things (IoT) and Cyber-Physical Systems (CPS) domains [124].

All those systems include a collection of tangible components connected to the Internet. Such components may include a range of devices using various types of local area connections, such as RFID, NFC, Zigbee, Bluetooth, Wi-Fi, plus wide area connectivity, such as GSM, GPRS, 3G, LTE and LoRa. They can record human or other activities which can then be relayed to other components, such as the processing system, and this information can be used to operate the system effectively, to gather data, or to perform any other relevant control and supervision tasks.

Devices are normally manufactured by different vendors. Although failures *within* an individual component may be taken care of by their respective manufacturers, failures *between* different components, such as connectivity problems, will likely not be supported by specific manufacturers and not covered under warranty. A popular report by Forrester states that in 2010 the number of devices connected to the Internet surpassed the earth's human population and predicts that there will be a huge growth in the IoT industry in the next years². Additionally, Symantec³ predicts that there will be up to 21 billion connected devices by 2020, and not just the consumers, but cities and companies will also start adopting *smart* technologies in their operations.

² Forrester Report (2017). <https://www.forrester.com>

³ Symantec Report (2019). <https://us.norton.com/>

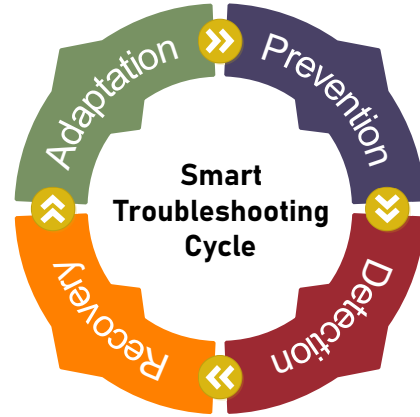


Figure 2: Smart Troubleshooting Cycle.

One technology worth mentioning is the Software Defined Networking, or SDN [113], which will make a connected systems more secure[85]. Referring to Figure 1, a report from Gartner⁴ says that the worldwide spending in IoT Security was 1.1 Billion USD and it is projected to skyrocket at more than 3.1 Billion USD in 2021.

From the above discussion, it should be clear that diagnosing errors and troubleshooting failures in connected devices will be a huge issue in the near future. In this paper, we survey existing literature in fields related to the concept of smart-troubleshooting as defined above, including system- and device- level diagnostics/prognostics, threat (i.e. fault, error, failure) management (prevention, detection, tolerance), and recovery/reconfiguration. Furthermore, we will take a step ahead and explore promising possibilities of self-diagnostics and self-healing in order to detect and fix system-level problems at run time.

2.3. Reference phases

Smart-troubleshooting can be categorized into four primary phases which are sequential and recurrent (see Figure 2), namely *Prevention*, *Detection*, *Recovery*, and *Adaptation*.

Prevention is the ability of a system to anticipate and avoid threats. *Detection* addresses the capacity of the system to recognize and diagnose anomalies so that an appropriate fix can be applied. The application of the fix is defined as *Recovery*. finally, *Adaptation* is in charge of adapting and evolving

⁴ Gartner Report (2018). <https://www.gartner.com>

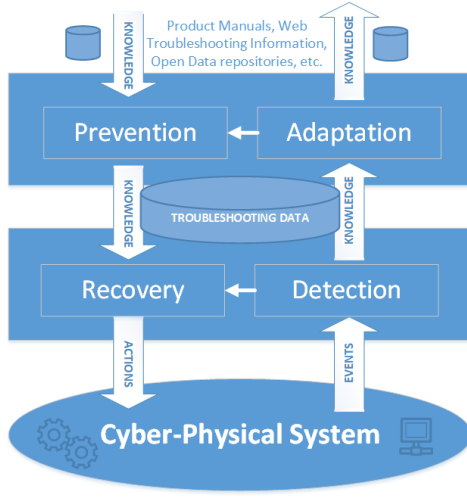


Figure 3: Smart-Troubleshooting Conceptual Diagram.

the system so that the same fault can be treated in a more effective and efficient way in the future.

Figure 3 shows another conceptual view where we assume to monitor a connected CPS and those phases are positioned at different levels. At lower levels, detection and recovery are used to quickly manage known anomalies in a short feedback loop. At higher levels, adaptation is used to improve threat management (e.g. by system reconfiguration) and prevent or mitigate any new occurrences of the same threat wherever possible. Hence prevention is initially fed by a-priori knowledge, but as the system executes it evolves with new knowledge extracted from the history of previous anomalies and their management. The troubleshooting knowledge base is therefore continuously updated with new information from both external sources and past experience. That also enables prognostics that is early detection of threats before they lead to severe consequences.

2.4. Related research

Related research fields have been investigated over the last years under various lexical analogues, one of them being Self-Healing [9], Anomaly Detection [1][78][111][99][71], Smart Maintenance [12][10][89][2], Intrusion Detection [117], Self-Protection and Self-Explanation [79], Predictive Safety Analytics [91], Failure Prediction [16], Proactive Fault Management [115], Resilient Systems [4][56].

Although none of those research achievements cover all the aspects that are required to implement

smart-troubleshooting, according to the conceptual model introduced in this paper, each of them inspects some relevant aspects that can be re-used or used as a starting point for related research. In particular, several surveys exist on Self-Healing [108][35][72][64]; however, they mostly address self-diagnostics and autonomous repairs at device or subsystem level, and not at a holistic system-of-systems level in the context of heterogeneous connected devices.

The research field of connected devices dependability and self-healing can be extremely broad. In this paper, we concentrate on aspects that are directly related to the smart-troubleshooting concept, and hence restrict our analysis to relevant research and related challenges.

3. Study Design

The replication package for this study is available publicly to investigators who would like to learn more or expand this manuscript⁵. It contains a list of all the keywords that were used to extract publications. Furthermore, it also contains a catalogue of publications that were extracted using those keywords. We start this section by describing the goal, followed by the research questions and the process used to extract relevant research publications. Finally, we list down keywords and delineate the search strings used.

3.1. Goal and Research Questions

According to the discussion and motivation above, we formulate our research questions as:

RQ1: *What is the need for troubleshooting connected devices?*

By answering this question, we wish to establish the importance for a system made up of connected devices to partially or fully recover from errors or failures.

RQ2: *What is the current state-of-the-art in methodologies, tools, and technologies in the field of prevention, detection, recovery and adaptation in the context of connected devices?*

By answering this question, we show the relevant research that has been done in the past to

⁵ P. Singh, M. Caporuscio, F. Flammini, N. Khakpour, Smart Troubleshooting Survey Resources (2019). <https://github.com/prasannjeet/Smart-TroubleshootingSurvey-Resources>

fix a system automatically in case of errors and failures. Moreover, we substantiate the current state of the art in prevention, detection, recovery and adaptation.

RQ3: *What are the open issues, challenges and opportunities in the field of smart-troubleshooting?*

As smart-troubleshooting is an emerging field, it is important to answer the above research question as a step towards building a roadmap for smart-troubleshooting connected devices.

3.2. Search and Selection

Figure 4 shows the search technique applied to obtain the initial list of publications used for this survey. Elsevier Scopus has been the primary search engine for title, abstract and keywords, as it indexes the largest number of quality sources including the ones by the most reputable publishers like IEEE, Wiley, ACM, Springer, etc. Then papers were checked and selected by accessing the related digital libraries (e.g. IEEE Digital Library).

The initial aggregation resulted in approximately 305 publications, most of which are included in the replication package. Shortlisting these publications finally resulted in a total of 78 publications.

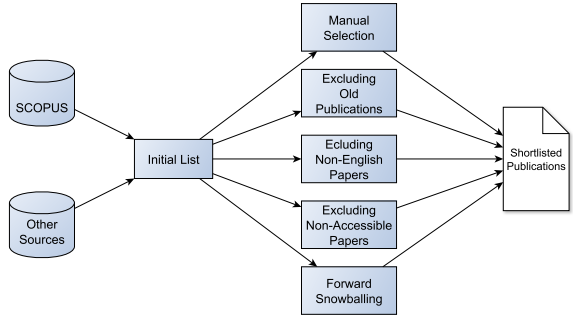


Figure 4: The technique exercised in shortlisting the primary publications for this survey.

The initial list was manually investigated to select the most recent and relevant publications. Publications not related to connected devices were discarded, together with papers in languages different than English or not accessible. Methods such as forward snowballing [135] were also used. Some secondary or tertiary studies such as survey papers, or systematic literature reviews were also considered if they appeared to be relevant.

Methodology	Domain
Troubleshooting	IoT
Resilien*	Internet of Things
Automat* * Recovery	Cyber Physical Systems
Fault*Tolerance	CPS
Error*Tolerance	Autonomous
Proactive Fa* Management	Embedded Systems
Self*Heal*	Real*Time Systems
Smart*Maintena*	Reactive Systems
Slef*Repai*	Dependable Systems
	Connected Devices
	Intelligent *
	Smart *
	Critical Systems
	SCADA
	VANET
	Wireless Sensor Networks
	WSN
	Industr* Critical Systems
	ICS
	Critical * Infrastructures
	System*Of*Systems

Table 1: List of keywords.

3.3. Keywords Used

In order to search for the relevant sources, a list of keywords were composed which is shown in Table 1. These keywords have been grouped into two categories, *methodology* and *domain*. Methodology contains all the keywords related to smart-troubleshooting, while domain lists the keywords related to connected devices. It should be noted that that keywords related to smart-grid were intentionally ruled out from the search, as most of the publications related to smart-grid addressed resilience from the viewpoint of electrical engineering. Furthermore, in the main research, the subject area was limited to *computer science* and only the most recent publications were included, since those were the ones dealing with heterogeneous connected devices in the context of modern paradigms like CPS and IoT. Older papers were added manually whenever deemed relevant for specific aspects of this study. Additionally, in order to obtain more precise results, the search string was only used to look into the title and keywords of the paper.

Although several strings were used to search for relevant publications, the general rule can be described as follows:

Let $m_1, m_2, \dots, m_i$ be a list of *methodology* keywords, and $d_1, d_2, \dots, d_j$ be a list of *domain* keywords, and $n_1, n_2, \dots, n_k$ be the list of keywords to

be excluded. The search string used for automatic search was:

TITLE-KEY((m_1 OR m_2 OR ... OR m_i)
AND (d_1 OR d_2 OR ... OR d_j) AND NOT
(n_1 OR n_2 OR ... OR n_k))

3.4. Classification of results

We used a two-dimensional classification to frame existing research as shown in Table 2. The four rows represent the four phases of the process, while the columns indicate prominent questions about threat identification and management, which we simply refer to as the *action*, i.e. When, Who, Where, How and What:

- **When:** At which phase of the system life-cycle the action happens, i.e. *design-time* (including engineering/specification, development/implementation, testing/verification) or *run-time* (i.e. operational stage). Due to the focus of this paper, most of the related research addresses threat management at run-time.
- **Who:** Who is the actor responsible for the action. We found out that in all surveyed papers the actors could be classified into the following categories: *humans*, through maintenance and manual repair interventions, the *operating system* or specific *application*, a specifically designed threat management *middleware*, an *external control agent* like a separate monitor/troubleshooter, or a fully *decentralized system* solution exploiting autonomy at device level.
- **Where:** Which is the highest level/layer among modern computing paradigms (i.e. *edge*, *fog* or *cloud* computing) involved in the action? Such classification is important to distinguish between device-level implementations, local area network or remote supervisors.
- **How:** Which techniques are employed to perform the action? We categorized those techniques using subsection titles of the following sections representing each stage of smart-troubleshooting (e.g. *Verification and Validation* is one technique used for prevention, *Log Analysis* is used in detection, etc.).
- **What:** What kind of threats are addressed by the action? We distinguished here among the two most important categories that are

intentional and *unintentional* threats. Intentional threats include all security attacks, while unintentional threats include errors and failures originated from random/natural faults like hardware malfunctions or software bugs.

Note that although comprehensive care was taken to fit each surveyed paper in all the columns of their corresponding row (Prevention, Detection, Recovery, or Adaptation), a small number of papers failed to cover all the columns, primarily due to insufficient information.

The publications investigated in this survey span through various domains, as shown in Table 1. Some existing research results are general and provide approaches used or usable in the context of Safety Critical Systems, Distributed Systems, Industrial IoT, Cyber Physical Systems, Computer Networks, Large Scale Applications, Autonomous Systems, etc.; while other publications cover specific domains such as Aerospace Systems, SCADA⁶, Smart Home, etc.

⁶ What is SCADA?, B-Scada. <http://scada.com/software/what-is-scada>.

	When	Who	Where	How	What
Prevention	Design Time[57, 56, 129] Runtime [100, 122, 65] [36] [77, 83, 116] [101, 15, 67, 25, 3] [94, 100, 95] [38]	Operating System/ Application [100, 101, 95, 3] Middleware[122, 12, 116] [14, 15, 25, 83, 94, 38] External Control Agent [10, 65, 77, 67] Humans[57, 56]	Edge [65, 57, 56] [14, 95, 129] Fog[16, 122, 116] [65, 94, 77, 10, 3] [15, 25, 36, 83, 38] Cloud[67, 12, 10] [114]	Verification and Validation [57, 100] Fault Tolerance [122, 65, 36] [94, 129, 114] Intrusion Prevention [118, 18, 77] [107, 47, 137, 83, 116] Virtual Patching [101] Firewalls [15, 67] [25, 3, 14, 13, 95, 38]	Intentional [3, 14, 15, 18, 25] [47, 67, 77, 83, 101, 107, 114] [116, 118, 137, 95, 38] Unintentional [94, 57, 56, 100, 122] [65, 36, 129]
	Runtime [71, 111, 2] [1, 10, 80, 42, 45] [49, 68, 69, 70, 87, 90] [92, 110, 112, 136, 138, 142] [16, 19, 43] [52, 55, 74, 86, 119] [140]	Operating System/ Application [71, 43, 138, 80, 112, 136] [90, 45, 87, 69, 142, 110] [68, 92, 49, 111, 117] [19, 42, 70, 74, 86, 119] Middleware [1, 12] [16, 55, 140] External Control Agent [78, 99, 10, 2] Decentralized System [52]	Edge [140] [43, 138, 80] [112, 136, 90] [45, 87, 69] [142, 110, 68] [92, 49, 19] [70] Fog [10, 89, 52, 16] [55, 74, 86, 119] Cloud [99, 12] [10, 2]	Log Analysis [42, 45, 48, 49] [68, 69, 70, 80, 87, 90] [92, 110, 112, 136, 138, 142] Model Based Analysis [16, 19, 21] [28, 34, 43, 52, 55, 74] [86, 103, 119, 127, 128, 140]	Intentional [48, 80] Unintentional [42, 45] [49, 68, 69, 70, 87, 90] [92, 110, 112, 136, 138, 142] [16, 19, 21, 28, 34, 43] [52, 55, 74, 86, 103, 119] [127, 128, 140]
Recovery	Runtime [120, 53] [141, 122, 39, 40, 121] [29, 125, 106, 97, 96, 98] [84, 27, 109, 126, 123, 41]	Operating System/ Application [5, 27, 29, 40, 84] [96, 97, 98, 121, 123] Middleware [53, 120] [122, 141] External Control Agent[78, 41, 125] Decentralized System [39, 106, 109, 126]	Edge[5, 125, 29] [84, 96, 97, 98, 121] Fog[120, 89, 40] [53, 122, 141] Cloud [41, 27, 106] [109, 123, 126]	Software Architecture [120, 41, 53] Hardware Redundancy [141, 76, 122] Modularity [39, 40, 121] [29] Software Rejuvenation [125] Machine Learning [106, 97, 96] [98, 84, 27, 109, 126, 123]	Intentional [120, 53] Unintentional [120, 41, 53] [141, 122, 39, 40, 121] [29, 125, 106, 97, 96, 98] [84, 27, 109, 126, 123]
	Runtime [55] [86, 74, 17, 20, 31] [33, 46, 58, 61] [66, 93, 32, 61, 62] [102, 131, 134, 26] [54, 51, 8, 23] [30, 81, 139, 120]	Operating System/ Application [23, 31, 81, 102, 139] Middleware [55, 120, 8, 26] [33, 46, 54, 58, 61] [62, 66, 134, 17] Decentralized System [30, 32, 51, 93]	Edge [81, 102, 17] [30] Fog[132, 55] [23, 120, 26, 31, 33] [46, 51, 54, 58, 61] [62, 66, 134] Cloud [8, 32, 93, 139]	Model-Based Approaches [17, 20, 31, 33, 46, 58] [61, 66, 93] Architecture-Basd Approaches [32, 61, 62, 82, 102, 120] [131, 134] Control Theory [26, 54] Machine Learning [51, 8, 23] [24, 30, 63, 81, 139]	Intentional [120] Unintentional [17, 20, 31] [33, 46, 58, 61, 66] [93, 32, 61, 62, 82, 102] [120, 131, 134, 26, 54, 51] [8, 23, 30, 63, 81] [139]

Table 2: Classifying research in the area of Smart Troubleshooting

4. Prevention

In their popular manifesto of computer dependability, Avizienis et al. define Fault Prevention as one of the four major means for a system to attain Dependability and Security [11]. Basically, the concept of prevention is associated to all the techniques needed to design a system that is less prone and vulnerable to faults. In smart-troubleshooting, part of this design can be postponed at run-time as an output of adaptation. When feasible and effective, fault-prevention is preferable to postponing fault management to later stages of system life-cycle, since the cost of fixing bugs is much higher when they are detected at later stages [50].

Since in this study we move around the central concept of “errors”, i.e. fault-originated alterations of system state that can be detected/diagnosed before they lead to failures, we include in this section the concept of “fault-tolerance” as a method to prevent errors. Also, since we defined smart-troubleshooting as a continuous loop, we could start from any phase of that loop. Prevention is naturally suitable to be the first phase according to a system life-cycle view; however, considering a system at run-time, we have highlighted in the previous section how the phase of prevention can actually follow the phases of detection and adaptation, since adaptation can enable system-level reconfiguration and preventive maintenance.

4.1. Verification and validation

The concept of prevention is traditionally associated to verification and validation (V&V), hence to testing, which is a simulation-based approach consisting in executing (part of) a system with given inputs and check whether the outputs match the expected ones. Since the complexity of connected systems is continuously increasing and the number of faults/bugs in a system is proportional to its complexity (see e.g., the basic metric based on lines of code [88]), it can be assumed that it is impossible to test the system against all possible faults.

Methodologies based on abstract testing can be used to increase verification effectiveness and efficiency in case of large and evolving system configurations, by automatically adapting generic test-suites to specific configurations of distributed embedded systems [57]. However, those methodologies lack the automatic discovery of new components, which need to be configured manually. Simulation-based approaches are the most common V&V tech-

niques but not the only ones. Static analyses, code walk through, and formal methods like model-checking and theorem proving are also essential in the V&V of critical systems. However, in the context of smart-troubleshooting we concentrate on techniques which can be used at run-time, during the operational stage, after the system has been designed, developed and installed. Therefore, it is not in the scope of this survey to address static and pre-deployment V&V techniques.

Furthermore, although works have been done previously in developing safe programming languages [100] that can condense the effects of system faults, those approaches was never used in mainstream industries. Therefore, introduction of faults in a system is inevitable, and more emphasis is given in developing *fault-tolerant* systems, rather than building *fault-less* systems. This is especially true when integrating components over which we do not have direct control, like COTS (Commercial Off the Shelf).

4.2. Fault tolerance

Fault-tolerance is perhaps the most popular technique to build dependable systems, mainly acting at device and subsystem levels. A system could be termed as *fault-tolerant* if it is able to prevent *faults* from turning into *failures*. Fault-tolerance can be implemented at both hardware and software level. Furthermore, we can distinguish between spatial and temporal fault-tolerance: spatial fault-tolerance is based on redundancy, fault detection, isolation and system reconfiguration; temporal fault-tolerance involves redundant (i.e. repeated) computations instead of redundant modules and can protect from transient faults.

Considering the literature relevant to smart-troubleshooting connected devices, the work in [122] proposes a fault tolerant mechanism which is implemented on a WuKong⁷ middleware. The mechanism provides failover for application components to comply with the requirements of fault tolerance. To prevent bottleneck throttling, it capitalizes the decentralized nature of IoT devices by adopting a decentralized algorithm. The work in [65] proposes a fault-tolerant architecture for healthcare IoT systems consisting of Wireless Sensor Networks (WSN). A 6LoWPAN (IPv6 over

⁷ K.-J. Lin, D. Shih, Y.-C. Wang, J. Hsu, N. Reijers, G. Chou, B. Tsai, WuKong - Intelligent Middleware for Internet-of-Things (2016). <https://newsabntu.github.io/wukong4iox/>

Low-Power Wireless Personal Area Networks) architecture is presented for healthcare environments that constructs an improved gateway solving the bottleneck problem in edge routers due to the limitations in 6LoWPAN, thus improving the network fault tolerance. Other faults such as the malfunction of the sink node hardware are also covered [36]. Misra et al. [94] propose a method for fault-tolerant routing in IoT. A fault-tolerant routing protocol is suggested for IoT systems that is based on mixed cross-layer and Learning Automata (LA). A successful packet delivery between nodes is assured even with the existence of faults. Kouwe [129] proposes *fault-injection*, where artificial faults are injected into a system in order to learn its behavior against activated faults and hence try to prevent system failures.

Cyber Physical Systems such as SCADA (Supervisory Control And Data Acquisition) are good examples to be considered when designing fault tolerant connected devices [114]. These systems can exploit cloud computing services and integrate with IoT to build Smart Industrial Systems to be employed in smart grids, smart transportation, and even eHealthcare.

4.3. Intrusion prevention

Intentional attacks or intrusions can also result in failures. These attacks can be considered as faults (malicious, human-made [11]). Thus, it is important to employ intrusion prevention mechanisms in connected devices to prevent those faults. Sicari et al. [118] discuss various methods to establish privacy and trust in IoT. Furthermore, various methods of intrusion detection are discussed and a taxonomy is proposed by Zarpeloa et al. [7]. In addition, connected devices with high processing power are prone to being compromised by external attacks, and eventually being used as a source for distributed denial of service (DDoS) through Botnets. Bertino and Islam [18] propose guidelines to prevent an IoT system from being compromised.

Sometimes, the IoT system itself can be a victim of denial of service attacks. Kasinathan et al. [77] propose a DoS Detection Architecture for 6LoWPAN IoT systems using *Suricata*, an open-source intrusion detection system that detects and eliminates the attack using appropriate countermeasures before network operations are disrupted.

Another efficient way of intrusion prevention is the installation of *honeypots* in a system. Honeypots can be in the form of a physical or virtual

computer that emulates genuine components along with activity monitoring and logging, which can help attackers waste resources and time while letting defenders examine attacks and build countermeasures. Likewise, *honeynets* are aggregations of honeypots intended to imitate servers and network services [107]. Honeypots are essentially a technique of deception, where the defender purposely hoodwinks the attacker into acting in ones favor [47]. While Yu et al. [137] reject the idea of Honeypots in an IoT setup due to non-scalability and dependency issues, La et al. [83] consider the possibility and use game theory to analyze the situation where both attacker and defender try to deceive each other.

The attack and defense activities between an attack and IDS are simulated, analyzed and modelled as a Bayesian signaling game of incomplete information. The results conclude that honeypots must be heavily deployed while there is a high concentration of active attackers. Scott and Carbone [116] have designed and implemented a Honeypot for SCADA networks 8, 9, 10.

4.4. Virtual patching

Another problem in an IoT system arises when a known vulnerability in a component of the system is identified but there is no simple way to update the component software. Smart-meters or domestic modems are fitting examples. While the devices are part of the IoT system, they generally do not possess the capability of Over The Air (OTA) updates. Updating these devices generally require users to manually download the firmware followed by establishing a connection with the device.

Moreover, after a vulnerability is disclosed, devices are prone to attacks within as little as a day, whereas it can take as much as a month for manufacturers to release a bug-free patch for that vulnerability¹¹ In this case, instead of patching the device itself, the patch can be applied in a *wrapper layer*.

-
- 8 L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, J. Smith, Conpot ICS/SC/SCADA Honeypot (2013). <http://conpot.org/>
 - 9 D. G. Peterson, Honeynet Project tries SCADA (2007). <http://www.digitalbond.com/>
 - 10 V. Pothamsetty and M. Franz, Scada honeynet project: Building honeypots for industrial networks, Critical Infrastructure Assurance Group(CIAG). Cisco Systems, Inc., 2005 <http://scadahoneynet.sourceforge.net/>
 - 11 TrendMicro, Virtual Patching: Patch Those Vulnerabilities before They Can Be Exploited - Security News (2018). <https://www.trendmicro.com/>

An ideal point where the patch can be applied is the gateway. This type of patching is termed as *Virtual Patching*. As per Open Web Application Security Project (OWASP), Virtual Patching is defined as a “*security policy enforcement layer which prevents the exploitation of a known vulnerability*”¹². It is worth noticing that in virtual patching, while the vulnerability in the component still exists, any incoming connection to the vulnerability is blocked by the virtual patch applied in the wrapper layer. Nelson[101] has proposed a virtual patching based solution to combat brute-force attacks in a SDN environment.

4.5. Firewalls

While virtual patching is often mentioned as a Web Application Firewall (WAF), an IoT system also needs a full-fledged firewall as some of the embedded systems that are part of the IoT system can have little to no security. Legacy systems are even more vulnerable to attacks as they were built with a notion that the components will work in a closed “safe” environment, thus ignoring security.

An efficiently built firewall in IoT can prevent an attack before it begins. Firewalls combined with encryption and authentication can provide maximum multi-layered security in IoT. A *Filtering Engine* based firewall can be implemented that scans all the incoming packets and blocks unwanted packets based on a set of rules[104]. A recent improvement on the traditional firewalls for IoT is the so-called *Smart Firewall*, which is a hardware-based device unlike traditional software-based firewall. It is capable of managing several devices at once more efficiently than traditional firewalls[15]. Gupta et al.[67] have proposed a IoT firewall implementation using Raspberry Pi as a gateway.

Other legacy research works in the field of firewall and security of IoT include the following references: [25][3][14][13].

4.6. Software over the air

With rapid technological advancements and increased vulnerability discoveries, it is imperative for an IoT system to frequently receive software updates. We foresee regular software updates and event-driven patching for embedded systems will

become the most basic and important technique to support prevention in the future.

Now that agile methodology is also used for developing IoT systems[60], product development has now become part of a dynamic process where software is continuously updated, improved and adapted to customer requirements that are prone to change occasionally. Therefore, it is crucial to always keep the software up to date. The most efficient way to achieve this is via over the air updates, i.e. SOTA (Software Over The Air). While SOTA is a more general term, FOTA (Firmware Over The Air) refers specifically to updating *firmwares* over the air. A prominent use case is the modern automobile industry, as automobiles are no more just a mechanical hardware but an IoT system consisting of multiple sensors connected to each other.

With a wide range of software used in multiple components, it is important to keep them up to date. Although SOTA would be the preferred option, often updates in older systems have to be performed manually in workshops via wired connections¹³.

5. Detection

The process of detection addresses several concerns such as, when error detection should be done, what information should be used for detection, how the information used for the analysis should be extracted or represented, which techniques can be used to detect/diagnose an error, etc.

Error detection can be done at design-time or at run-time. Design time error detection is performed in the context of verification and validation, including static code analysis, testing and model-checking, as already discussed when addressing prevention. Run-time detection can be performed *on-line* by monitoring a live system (e.g. by checking assertions at run-time) or it can be carried out by logging the relevant system events/data and analyzing them after the system executes. In offline detection, there is no interference between the system behavior and the detection procedure, and hence less overhead at run-time. However, the offline approach suffers from *late detection*, i.e. it can detect an error when it has become too expensive to mitigate its effects. In contrast, online methods have

¹² OWASP, Virtual Patching Best Practices (2017). <https://www.owasp.org/>

¹³ O. Weinmann, Software updates in the IoT: an introduction to SOTA (2018). <https://blog.bosch-si.com/>

the advantage of early detection but their performance overhead can be high. Offline methods have the advantage of more precise analysis as they usually use a complete model of the system compared to the online methods that rely on partial traces or partial logged information. In this section, we focus on run-time error detection methods.

Furthermore, the error detection process can be done in a *monolithic* way where a single component is responsible to analyze the gathered events/data to detect an error [140][43], or it can be done in a *distributed* way where several components in the system analyze the events and perform the detection in a collaborative way, e.g. [52]. Decentralized error detection can be coordinated by a single component or it can be carried out in a collaborative way by several error detection modules.

An important task in detection is collecting the data of interest from the running system. Data collection can be event-driven or data-driven: the event-driven approaches observe the events (activities) that occur in the systems to extract an execution trace that indicates the sequence of actions happened, while in the data-driven methods a snapshot of the system is taken during predefined periods. An error can be detected and diagnosed at different levels, such as class/method level, module level, architecture level, system level or system-of-systems level. In this paper, we mainly focus on macro-level error detection methods, i.e. at the architecture, system or system-of-systems levels. Hence, data should be collected from different components (i.e. modules, sub-systems or systems) and communication channels in a distributed way. The collected data can be processed by the error detection module in a synchronous or an asynchronous way. In case of synchronous detection, the collected data might not be stored. However, the collected data should be stored in the majority of approaches.

We categorize the methods in two classes of model-based detection methods, such as runtime verification, and model-free detection methods such as log analysis techniques using statistical and machine learning.

5.1. Log Analysis

In asynchronous and offline detection methods, the information used for diagnosis is extracted from logs produced by the system. Logs usually contain valuable information that can be helpful to understand the system behavior. The usage of logs pro-

duced by a system can be different, e.g. for troubleshooting an error, performance analysis, security analysis, prediction etc.

A system of connected devices can generate up to hundreds of thousands event log instances each hour. As a consequence, logs produced by a system usually contain a huge amount of data. The content and format of log information could vary from one system to the other, or even from one component to another component in the same system. Log heterogeneity is even more serious in IoT systems, as such systems are usually heterogeneous and consist of various devices provided by different vendors. Logs produced by different devices in a system can be inconsistent, irrelevant, unstructured or incomplete, e.g. some logged information is not helpful to detect failures while it can be helpful for other purposes and it can decrease the readability of error logs by humans and increase the processing cost. Logs can be incomplete, as they may not provide sufficient information for error detection. One solution to this problem is to develop more effective logging mechanisms while designing the system, e.g. the authors in [42] propose an approach for rule-based logging with the aim of detecting threats more effectively. Logs produced by different devices in a systems should be filtered, normalized and represented in a standard format. They should also be reduced and refined to be able to provide consistent, relevant, complete (if possible), and homogeneously structured information.

The process of log file reduction and normalization usually go through a cumbersome work of parsing and analysis, and sometimes do not lead to reasonable results. In addition, the accuracy of the detection phase depends on the quality of provided log data that cannot be always ensured, due to the lack of sufficient log information. An alternative is to design new logging mechanisms and/or standards that can provide consistent and complete information. BSD (Berkeley Software Distribution) syslog¹⁴ and IETF syslog¹⁵ are two standards that have been developed for event messaging and data logging. IDMEF (Intrusion Detection Message Exchange Format) is another data logging standard [48] which defines the data formats and exchange protocols for sharing information of interest to intrusion detection and response systems.

A log analyzer usually has a component to iden-

¹⁴ <https://tools.ietf.org/html/rfc3164>

¹⁵ <https://tools.ietf.org/html/rfc5424>

tify the patterns of log messages and another component to match them against undesired/desired patterns in order to identify anomalies. Based on the violated pattern, a possible fix can be determined and applied, either automatically or by guiding maintenance staff to perform the right repair intervention. There has been a stream of log analysis methods based on machine learning and data mining algorithms for fault identification and diagnosis, or mining the root cause of abnormalities. Some methods use supervised learning approaches such as SVM (Support Vector Machines) [138][80] and deep learning [112] to categorize faults and problems. However, they usually require a large number of labeled data that often need to be done manually. Unsupervised learning [136][90], including clustering [45][87][69], is another common approach for problem identification.

UiLog [142] is a system that analyzes different logs gathered from several components, categorize them into different failure types, locate failures and determine their root causes. The authors in [110] present an approach for distributed failure-monitoring and the propagation of faulty information in the system, with the goal of identifying failures, and troubleshooting them in large-scale systems. The logs are categorized into a sequence of failure events that are used to identify the root cause of the failure. Hamoon et al [68] proposed LogMine that is a framework to extract patterns from massive heterogeneous logs produced from different systems, in particular IoT systems. Mariani et al [92] proposed a heuristic-based log analysis approach for failure identification based on a sequence of events extracted from logs. They construct a model of the system behavior in terms of finite state automaton specifying the expected behavior of the system. The event traces extracted during failed executions are compared with the system model to identify the undesired events that can help the user determine the location or the cause of a failure.

Anomaly detection aims at detecting abnormal system behavior in a timely manner to help the admin pinpoint the issues and resolve them promptly, to avoid system downtime. Log analysis has been widely used for anomaly detection by researchers and in industry. [70] provides a review and evaluation of log-based anomaly detection approaches, including a few supervised methods and unsupervised methods. LogLens [49] is a log-based anomaly detection system to detect operational problems from any software system logs. LogLens learns the

system behavior and constructs models to specify the normal system behavior. It employs the learnt models to analyze logs in order to detect anomalies.

5.2. Model-Based Analysis

Detection methods, such as run-time verification or model-based learning algorithms, which work based on the analysis of a system model, need techniques for reverse engineering and extracting such models from execution traces and log files. *Process mining* is a well-known technique that can be used for model extraction by identifying the process elements and their temporal relation. Such techniques automatically elaborate event logs to create a model, called process model, that represents the system behavior. There are various existing algorithms that extract a process model from event logs. One of the most prominent and rather naive algorithm is the α -algorithm [128]. Event logs often include the exact sequence of steps the system underwent to complete any task. Additionally, using a surfeit of event logs for modelling will result in the inclusion of all the events that might be less likely to happen, as discussed above.

A process model is capable of providing different views in different abstraction levels for the same system and can be represented in various models/languages such as Hidden Markov Models [16], Petri Nets [127][128], automaton [21], etc. The extracted process models can sometimes be composed with each other, in a sequential or concurrent way, to obtain more complex processes.

Another common approach to build a system model is *automata learning*. Finite state machines, such as state charts and state transitions systems, are often used to express and specify system behavior. These models consist of a set of states and the transitions between them. Automata learning, i.e. reversing the design process, gradually constructs a state machine from log files and observed executions that represent system behavior. There are several algorithms to induce the structure or the probabilities of an automaton, e.g. the state merging method [34], algorithms based on guaranteeing PAC-learnability [103] and L*-based algorithm [21]. Learning state machine has received a lot of attention from software engineering and formal methods communities, as learning a system model can provide an insight on complex systems and their properties, e.g. [19][119].

Run-time verification is an attractive complement to design-time verification and a useful tech-

nique for verification of complex systems that change at run time, such as self-adaptive systems. In run-time verification, an observer monitors executions of a software system at run time and verifies whether or not the behavior of the software system adheres to a set of formal specifications. The specification formalizes the behavior of the system typically in logics (such as variants of Linear Temporal Logic, LTL) or finite-state machines. Then monitors are synthesized based on the formal specification to run alongside the system. Since only one execution path is examined at a time, run-time verification is more efficient compared to design-time verification.

Probabilistic verification techniques have been employed for run-time analysis of adaptive software systems, with the purpose of checking if the system satisfies its quantitative requirements and adapting it, if required, e.g. [55][28]. Runtime verification of cyber-physical systems has also received some attentions recently [140][43][86]. Cotard et al [43] developed an error detection service for in-vehicle multicore and real-time embedded systems. This service generates a monitor directly from formal specification that observes the data flows between communicating tasks and detects violations of the expected communication patterns encoded in the monitor. Leucker et al [86] proposed an approach based on service-oriented architectures in which interconnected medical devices publish their connectivity interfaces via web services that provide data and can be controlled. A formal specification of the desired data streams of systems using temporal assertions is used to generate a monitor.

Incki et al [74] proposed an approach for runtime verification of IoT systems which monitors component interactions, uses Event Calculus to specify system constraints and employs an event processing algebra to detect failures based on the specified constraints. Finally, El-Hokayem et al. [52] use a decentralized runtime verification approach where each monitor has its own specification and communicates with other monitors to detect undesired behaviors.

6. Recovery

Connected devices can be large-scale distributed systems composed of resource-constrained, wireless and battery powered devices that are prone to crash failures, omission failures, timing failures, response

failures and arbitrary failures [44]. Although devices could implement some aspects of self-healing at the edge level, their limited capabilities suggest to orchestrate intelligent monitoring and recovery functionality from more powerful and reliable entities located at fog or cloud levels [22]. Operating at higher hierarchical levels also allows engineers to account for failures that are not detectable at device level since they arise from the coordinated interaction among more devices. This section briefly discusses methods and techniques for achieving *error handling* and *failure recovery* in such a context. Although some techniques described in this section also address aspects of fault-tolerance that we have anticipated while discussing prevention, we concentrate here on specific aspects that can be used for handling situations in which faults have already turned into errors and – possibly – system-level failures.

6.1. Software Architectures

Sousa et al. [120] have proposed FOREVER, a framework that makes a replicated system able to handle errors originated from malicious attacks. FOREVER is deployed beneath a hybrid architecture as a secure real-time subsystem that is segregated from the targeted application to be protected, to make sure that the framework itself does not become a prey to malicious strikes. It uses *event-triggered reactive recoveries* and *time-triggered periodic recoveries* when malicious behavior is detected. Xu et al. [5] proposed one of the initial frameworks for enhancing dependability in a concurrent object-oriented system via coordinated error recovery. The framework also involves strategies to handle errors and provide coordinated recovery within a group of collaborating objects.

A possible approach to respond to failures by dynamically computing SPD (Security, Privacy and Dependability) metrics, react and reconfigure the system accordingly has been defined in the context of the SHIELD research project[53]. The SHIELD framework used semantic technologies to model failures and recovery actions to perform at system level through a dedicated middleware.

A smart home architecture has also been introduced so that a neural network can be used to predict missing sensor data and hence provide redundancy to recover from failures [41].

6.2. Hardware Redundancy

Hardware redundancy has been proposed through sensors that are not directly compatible with failed sensors, where compatibility is identified via regression analysis and data combination [141]. Another way to provide redundancy along with the reduction of deployment cost was proposed by [76] where the authors introduce the NewIoTGateway-Select algorithm for smart cities that determines the minimum number of required gateways. By guaranteeing a minimum of k routes between gateways, the algorithm studies gateway and link failures. The simulations indicate that using the algorithm results in lesser number of gateways. Another decentralized solution is proposed in [122], in which services are assigned to devices and the devices are arranged in a ring topology. Here the responsibility is shifted to a redundant device in order to recover from errors.

6.3. Modularity

Cherrier et al. in their research have proposed a framework called Distributed Logic for IoT Services (D-LITe) facilitating the creation of IoT applications [39] where different objects that are part of the whole application interact with each other using REST web services. The complete application is categorized as a choreography that is deployed in real time on numerous objects. However, in contrast to orchestrations, the application logic is spread to various objects in choreography, which increases the chances of system failures. Therefore, the authors have proposed a methodology for error recovery to reestablish coherence in applications while keeping the advantages of choreography. The methodology consists in transforming objects into a coherent stage in case of a fault, as ascertaining which part was responsible for the induction of error seems to be out of reach [40]. Another framework called FLORA proposed by Sozer et al. [121] suggests breaking down the software architecture into modular units so that individual recoverable-units (RUs) can be recovered in isolation, also known as *local recovery*, which seems applicable to IoT architectures. This is done to maintain the quality concerns of the system, such as reuse, adaptability and performance. The suitability of local recovery is also argued, since the erroneous unit can locally be recovered, while keeping the rest of the system running, as opposed to shutting the whole system down while also decreasing the mean time to recovery [29]. The set of reusable abstractions included

in the framework defines the RUs and introduces communication, detection and diagnosis control between them. The framework currently supports the detection of deadlock and fatal errors.

6.4. Software Rejuvenation

One basic yet promising approach which can be adopted to recover from software failures due to *software aging* is known as software rejuvenation [125]. It is a matter of fact that most software failures belong to the so called Heisenberg category that is to say their origin and dynamics can not be easily diagnosed. Software rejuvenation is aimed at handling software failures regardless of their origin by restarting specific processes, applications or whole devices seen as black-boxes, according to predefined and fine-tuned strategies. Software rejuvenation can be used as a preventive maintenance technique and as such it should be classified as an approach for prevention. However, we foresee an increasing application of this technique for structured and systematic failure recovery due to its practicality and easy implementation even in heterogeneous systems.

6.5. Machine Learning

A recent research [106] proposed a microservice-based fault tolerant framework by means of two complementary microservices using complex event processing to recover from failures by online machine learning.

In another research, Nakamura et al [97] proposed a methodology for error recovery in a maintenance robot or a manipulation system using task stratification and error classification after the error is diagnosed in real time. The authors have defined a *task layer*, and based on the nature of the error, whether small or large, the methodology uses backward or forward correction, where the process either returns to the lowest or high-ranking layer of the task hierarchy and it is re-executed from the preceding step [96][98].

Moreover, the authors have proposed the usage of Artificial Intelligence such as deep learning for correction of parameters such planning, modeling and sensing when error recovery is performed in case of a small error because of the availability of a large quantity of past data. Similar methods of reverse execution have been used in [84] for automatic error recovery in robot assembly operations.

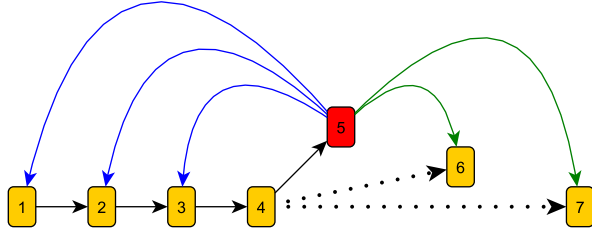


Figure 5: Figure describing forward and backward error recovery.

The method of forward error correction for error recovery is also used in other domains like vehicle ad-hoc networks (VANETs) [130][27][109][126]. The methods of forward and backward error-recovery is widely used for system level fault tolerance as briefly explained in Figure 5 [123]. The yellow nodes represent stable states of the system and solid black arrows represent a transition. The red node (node-5) represents an error state and the dotted arrows represent an expected stable transition. While backward error recovery rolls back the system to a previous stable state (blue solid arrows), forward error recovery transforms the system to a stable, or correct state (green solid arrows).

7. Adaptation

A key objective for smart-troubleshooting is to support the evolutionary and adaptive behaviors of connected devices to guide prevention and better respond to threats. *Evolution* aims at changing (in the long-term) the system functionality in response to the new or changing user requirements or to the need for maintenance. Evolution usually requires a major update of the system, as it might involve both its organization and implementation: e.g., adding, removing, and replacing components, and integrating device updates.

On the other hand, *adaptation* aims at preserving (in the short-term) the system behavior in response to the dynamics of both the system itself and the execution environment. Hence, adaptation requires monitoring the connected devices and their environment to update them accordingly. Adaptations can range from changing the value of a single parameter up to modifying the whole architectural configuration.

Traditionally, changes are handled off-line, during the maintenance phase. When dealing with connected devices, changes must also be handled at run time, while the system is operating. To

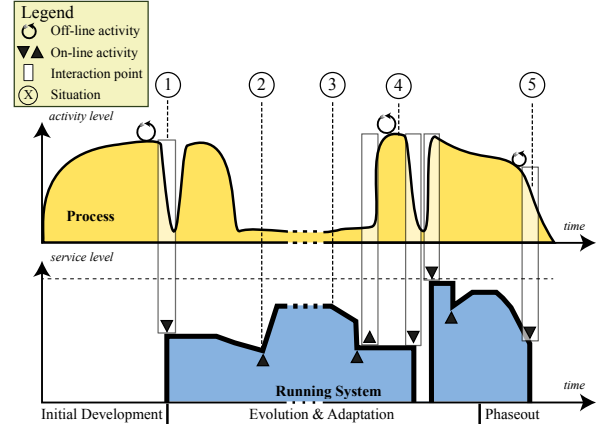


Figure 6: Evolution and Adaptation and (timeline view)

achieve this vision, systems must be able to reason about themselves and their state as they operate, through adequate run-time mechanisms. They must be able to monitor the environment, compare the data they gather against a model representing their expectations, and detect possible behavioral changes. Whenever a deviation is found, an adaptation step must be performed, which modifies system structure (e.g., components and/or devices re-binding) and/or behaviour (e.g., algorithm replacement).

In this setting, the clear and clean traditional separation between development time and run time becomes blurring. Referring to Figure 6, the interaction points indicate that off-line activities (i.e., process) impact on-line activities (i.e., adaptation logic) and vice versa [6]. It is worth noticing that adaptation and evolution may be initiated and controlled by both off-line and on-line activities.

When the *Initial Development* phase is terminated, and the system is operational (cf. ①), off-line evolution and adaptation activities are enacted to the connected devices in order to react to the changes detected in the environment. *Adaptations*, where the connected devices themselves detect and adapt, is usually simple and fast (cf ②). On the other hand, when the system detects a problem but it is not able to find a suitable run-time “troubleshooting” solution to address it, a complex and time-consuming off-line *evolution* is required (cf ③). This can be very costly and time-consuming. Moreover, when the system is re-engineered off-line, it can happen that an on-line update of all connected devices cannot take place due to its complexity. Applying such change requires for off-line evolution

activities followed by a new deployment (cf ④), which in turn affects device availability. Finally, the shutdown and decommission activities are usually planned and terminate both the system and its process life-cycle (cf ⑤).

Since connected systems are built as a composition of autonomous and heterogeneous devices, *self-adaptation* is considered as an effective approach to deal with the run-time uncertainty inherent to such a dynamic environment [37]. Self-adaptive systems are typically decomposed into a *managed subsystem* that implements the system functionality and a *managing subsystem* that implements the adaptation logic.

A general reference architecture for Self-Adaptive Systems (SAS) is given by the MAPE-K model [73], with Monitor, Analyze, Plan and Execute activities, plus a Knowledge that maintains relevant information for the other components (e.g., system state, adaptation rules). Specifically, *Monitor* aims at collecting information about both the managed system and the execution environment, whereas *Analyze* and *Plan* aim at selecting the set configurations that best serve to resolve existing non functional requirements (e.g., performance). Finally, *Execute* actually enacts the reconfiguration, so leading to the maintenance of the required requirements. When systems are large and heterogeneous, as in the case of connected devices, multiple MAPE-K components might coordinate with each another to decentralize the control [133].

However, how the adaptation activities are actually architected and implemented must take into account the specific characteristics of the managed system and its operating environment. To this end, the next subsections present and discuss a set of SAS best-practices that can be employed to engineer connected devices that are able to self-adapt at run-time.

7.1. Model-based Approaches

Model-Driven Engineering (MDE) is a software development paradigm that aims to use models as the main development artifact instead of code [59]. However, dealing with run-time changing conditions requires a paradigm shift from using models at design-time to using models at run-time. Exploiting models at run time is not a new concept since different approaches have been proposed to extend MDE applicability to run time [20]. In fact, MDE capabilities (e.g., model transformation and code generation) are considered viable means to

enable on-line system monitoring, model analysis and adaptation [66]. For instance, in [31] model-based analysis is exploited to manage QoS level of software systems at run-time, whereas in [61] architectural models are used to achieve both run-time adaptation and evolution. Further, in [46] MDE techniques have been exploited for specifically modeling and simulating self-adaptive decentralized cyber-physical systems.

Nevertheless, models are only a supporting tool for reasoning and need to be accompanied by additional techniques for adapting the managed systems, such as aspect-oriented programming [58], component-based programming [17], and service-oriented approaches [33][93].

7.2. Architecture-based Approaches

In order to guarantee the provision of functionality in an unknown and ever-changing environment, connected devices should be arranged in fluid architectures [32] and be able to self-adapt their structure depending on the changing situation. To this end, many different architecture-based approaches have been proposed to develop and reason about SAS.

The well-known architectural approach proposed by Kramer and Magee [82] offers a three-layer architecture model for self-management: (i) the *Component Control* layer controls the managed system, (ii) the *Change Management* layer monitors the managed system, and implements adaptation plans, and (iii) the *Goal Management* layer works out adaptation plans based on the specific goals the SAS should fulfill.

The Rainbow framework [61] implements a rule-based adaptation approach and makes use of a run-time architectural model. The architecture has different elements, which define the adaptation plans. A translation infrastructure translates and controls the deployment of these plans in the managed system layer.

The FOREVER framework [120] alters the vulnerabilities such as passwords and open ports when the recovery of a certain replicated system is processed. It also supports online upgrades such as patches, version updates, etc., to evolve with time while having the least impact on the availability of the framework.

Finally, other approaches use agent-related techniques for implementing architectural change [134][131] or dedicated architectural components for controlling the adaptation [62].

All architecture-based approaches implement the MAPE-K functionality (mostly implicitly, i.e., without dedicated components) and represent the architecture in the form of models for reasoning about the system's structure. The architectural models are used in combination with metrics such as constraints [102][62] and layers [61][82].

7.3. Control Theory

Knowledge from control theory can play an important role for implementing adaptation logic [26][54] in the context of connected devices.

Control loop systems can be divided into feedback control systems and feedforward control systems: feedback loop reacts on changes of the output variable, whereas the feedforward loop adjusts process variables by anticipating the effects on the output variable. Feedback control loops can be adaptive [26].

Control engineering for SASs focuses on feedback loops which result in reactive adaptation [105]. These approaches usually address the whole MAPE functionality.

7.4. Machine Learning

Machine learning has been widely recognised as a viable technique for achieving run-time self-adaptation. In fact, it enables SAS to continually learn about its structure, behavior, and execution environment, in order to become more efficient.

Several approaches use reinforcement learning for structural optimization. Dowling et al. present a collaborative reinforcement learning approach for decentralized coordinated self-adaptive components [51]. Kim and Park rely on Q-learning to improve the mapping between environmental change and software configuration in a SAS at run time [81]. Other approaches apply reinforcement learning for decentralized self-adaptive service assembly [30], and for generation and evolution of adaptation rules [139].

Furthermore, Arcaini et al. [8] have used MAPE-K to provide decentralized adaptation control for autonomic and self-adaptive systems. Here, the knowledge element can be dynamically updated to register past adaptations so that a re-occurrence of an error can quickly be mitigated by the managing system. However, since an IoT system is prone to be constantly updated with new services and devices that were not anticipated at the time of deployment, there are big odds that older adaptation

knowledge becomes obsolete. This phenomenon is also known as *Concept Drift* [63]. This issue is addressed by Bouchachia [23], where Incremental Learning is used to incrementally and continually accommodate new knowledge whenever it becomes available. These properties are known as stability and plasticity, respectively. The author also gives an overview of various incremental learning algorithms for adaptive systems such as fuzzy ARTMAP, generalized fuzzy min-max neural network, incremental learning based on function decomposition, etc. [24].

8. Challenges and Opportunities

The short survey provided by this paper has shown that while there are numerous approaches to tackle specific issues related to smart-troubleshooting, often for specific domains, there is no concrete cross-domain solution that can handle all these aspects using a cohesive, holistic and adaptive framework shaped as a feedback loop. Moreover, as different connected components generate their own event logs, comprehending logs in diverse formats represents a challenge. Even in case of log format standardization, the events and states recorded could differ in each device. In order to generate logs in a specific format with the required information, it is possible to design a centralized logging unit capturing events from connected components. Another option is to parse and convert error logs into a consistent format. Such an approach should also support introduction of new components at runtime. Transmitting event logs to the cloud can be a promising option to support the analysis of a huge number of logs from diverse systems using machine learning. Several research effort in error correction propose machine learning algorithms to accelerate the recovery process which requires large amount of past error-data. However, it should be noted that the accessibility of a large amount of error-data deduces that an outbreak of error has occurred, which contradicts with the ideal performance of any methodology. Therefore, in this case a trade-off must be established. Additionally, artificial intelligence techniques could be applied to adaptive systems, as suggested by Weyns [132], which might improve the systems ability to deal with unanticipated changes, thus evolving in a more effective way. It must be noted that most of the research discussed above proposes approaches requiring active local area and/or Internet connections;

as a result, if the connectivity itself is hampered, there might not be a solution available to automatically repair the system unless a local agent is kept updated and available in case of disconnections.

Digital twins is another important concept that can play a pivotal role in troubleshooting IoT systems in the future. It is essentially a virtual representation, or a *virtual doppelganger*, as termed by IBM¹⁶, of a cyber-physical asset. It enables the interaction of the application with devices in a consistent manner by providing an appropriate abstraction layer. Digital twins are conceptualized to follow the life-cycle of a device and its associated data¹⁷. It can also be used to simulate system operation in a specific scenario without actually running the system, in order to anticipate any faults-errors-failures that might occur over time. The simulations can also be “fast-forwarded” to foresee the effects of updates, repairs, preventive maintenance, and even future threats. Although physical objects have been digitally represented for decades, IoT digital twins is still a fledgling concept. It can provide a holistic view of most of the abilities an IoT system has so as to orchestrate various facets of the system¹⁸. A report from Gartner in 2017 described Digital Twins as a “disruptive trend that will have increasingly broad and deep impact over the next five years and beyond¹⁸”. Moreover, it was labeled as one of the top ten IoT technologies to be investigated starting from year 2018¹⁹. The same report remarks that billions of dollars worth savings is plausible in the field of maintenance repair and operation (MRO) and optimized IoT asset performance with the use of digital twins. Apart from avoiding a failure in an IoT system, using digital twins can also help predict the future, increase the accuracy and reduce costs²⁰. Various cloud vendors have started providing solutions for digital twins in IoT, such as Microsoft Azure Device-Twin²¹, Amazon Device-Shadow²², IBM Device-Twin²³. Addi-

tionally, Jung et al. [75] have discussed numerous approaches and challenges for the simulation of IoT systems. They have also proposed a concept that addresses the challenges by individually modelling each IoT component in separate modelling tools in order to better adapt to heterogeneity. Despite of those initiatives, digital twins for IoT can still be considered as a “concept”, and future efforts will be needed to make IoT digital twins a reality. Therefore, we believe IoT digital twins can be a promising research area which is destined to generate key technologies which can effectively support smart-troubleshooting.

It may not always be possible to automatically recover from failures occurring in a system. Device hardware failures that cannot be masked require procurement and replacement of faulty devices that can be impossible to automate. Human involvement will still be essential to recover from those failures, but semi-automatic troubleshooting can help to guide and orchestrate manual interventions through precise step-by-step instructions provided to maintainers. Another aspect that will be possibly semi-automated is the analysis of technical documentation via human-centered text mining, information visualization and multivariate analysis²⁴.

9. Conclusion

In this paper, we have introduced the concept of smart-troubleshooting in the context of interconnected devices, which are widespread in modern cyber-physical systems (CPS), including the emerging paradigm of the Internet of Things (IoT). We have described the current research challenges to tackle in order to address and implement the most relevant aspects of automated and intelligent troubleshooting, and to ultimately achieve a higher resilience in embedded systems, CPS and the IoT. The study has shown that while many approaches exist that could be partly re-used, including recent advances in fault-tolerance, log file analytics, process mining, etc., many tools and technologies still need to be researched and developed to evolve towards “self-X” approaches (self-monitor, self-diagnose, self-repair, etc.), especially at a holistic, system-of-systems level.

16 Wired Brand Lab, Digital Twin: Bridging the physical-digital divide (2017). <https://www.ibm.com/>

17 I. Skerrett, The Reality of Digital Twins for IoT - DZone IoT (2018). <https://dzone.com/>

18 G. Glocker, A primer on digital twins in the IoT (2018). <https://blog.bosch-si.com/>

19 K. Panetta, Gartner Top 10 Strategic Technology Trends for 2018 (2017). <https://www.gartner.com/>

20 McCarthy Dave, 5 reasons digital twins matter to your IoT deployment (2018). <https://www.networkworld.com/>

21 D. Betts et al., Understand Azure IoT Hub device twins (2018). <https://docs.microsoft.com/>

22 Device Shadow Service for AWS IoT. <https://docs.aws.amazon.com/iot/>

23 IBM, Introduction to data management (2018). <https://>

console.bluemix.net/

24 N. Kellingley, Information Visualization: An Introduction to Multivariate Analysis (2016). <https://www.interaction-design.org/>

In our study, we have categorized related works into four main phases named Prevention, Detection, Recovery and Adaptation. This is only one view of the possible dimensions of the problem, which can be observed and tackled from several perspectives.

In order to better structure our study, we have posed and answered three research questions. The first one (RQ1) was related to the motivation for introducing the concept of smart-troubleshooting. We have explained why this concept is novel and extremely relevant throughout the whole study, starting from the introduction. The second research question (RQ2) addressed the core of this paper that is the study of the state-of-the-art of relevant research, which we have mainly surveyed in the sections entitled Prevention, Detection, Recovery and Adaptation. We have used a combination of systematic literature review (SLR) to address the most relevant and recent research related to the main concept of smart-troubleshooting as a whole, plus other sources more focused on specific aspects, with the aim of evaluating the transfer to this area of results achieved in related fields. The third research question (RQ3) addressed challenges and opportunities. We have highlighted some open issues associated to the specific phases in each related section of this paper, and then discussed the main challenges plus some long-term and visionary perspectives (including Digital Twins) in a dedicated section.

The problem of smart-troubleshooting is so broad and cross-discipline that it is nearly impossible to address all its facets in a single paper. Some challenges we did not extensively address in this survey include distributed event processing, information fusion and event correlation for failure pattern recognition and matching, and natural language processing for product/troubleshooting information text mining. In particular, the latter is essential to collect and elaborate troubleshooting information from unstructured repositories which are not machine-readable. That would be a challenging goal to achieve until product and troubleshooting information will evolve to machine-readable representations.

Acknowledgments

The study presented in this paper has been co-funded by Linneaus University Centre for Data Intensive Sciences and Applications (DISA) through SEED funding. We would like to thank Diana

Unander, Welf Löwe, Morgan Ericsson and Diego Perez for supporting this initiative.

References

- [1] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.
- [2] A. M. Alexandru, M. Fiasché, C. Pinna, M. Taisch, L. Fasanotti, and P. Grasseni. Building a smart maintenance architecture using smart devices: A web 2.0 based approach. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016.
- [3] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, and M. Rossi. Low power link layer security for iot: Implementation and performance analysis. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 919–925, July 2013.
- [4] S. Amin, G. Schwartz, and S. S. Sastry. Resilient Cyberphysical Systems. In T. Samad and A. Annaswamy, editors, *The Impact of Control Technology*, pages 127–128. IEEE Control Systems Society, 2 edition, 2011.
- [5] and B. Randell, A. Romanovsky, C. M. F. Rubira, and R. J. S. and. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 499–508, June 1995.
- [6] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel. Software engineering processes for self-adaptive systems. In R. de Lemos, H. Giese, H. A. Müller, and M. Shaw, editors, *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, pages 51–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [7] B. B. Z. ao, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga. A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, 84:25–37, 2017.
- [8] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15*, pages 13–23, Piscataway, NJ, USA, 2015. IEEE Press.
- [9] A. Asghar, H. Farooq, and A. Imran. Self-healing in emerging cellular networks: Review, challenges, and research directions. *IEEE Communications Surveys Tutorials*, 20(3):1682–1709, thirdquarter 2018.
- [10] M. Ashjaei and M. Bengtsson. Enhancing smart maintenance management using fog computing technology. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1561–1565. IEEE, 2017.
- [11] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

- [12] S. Ayad, L. S. Terrissa, and N. Zerhouni. An iot approach for a smart maintenance. In *2018 International Conference on Advanced Systems and Electric Technologies (IC-ASET)*, pages 210–214. IEEE, 2018.
- [13] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad. Proposed security model and threat taxonomy for the internet of things (iot). In N. Meghanathan, S. Boumerdassi, N. Chaki, and D. Nagamalai, editors, *Recent Trends in Network Security and Applications*, pages 420–429, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [14] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad. Proposed embedded security framework for internet of things (iot). In *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*, pages 1–5, Feb. 2011.
- [15] R. Bala Krishnan and N. Sakthivel. Development of smart firewall load balancing framework for multiple firewalls with an efficient heuristic firewall rule set. *Journal of Theoretical and Applied Information Technology*, 35(2):201–207, 2012. cited By 2.
- [16] R. Baldoni, L. Montanari, and M. Rizzuto. On-line failure prediction in safety-critical systems. *Future Generation Computer Systems*, 45:123–132, 2015.
- [17] N. Bencomo and G. Blair. Using architecture models to support the generation and operation of component-based adaptive systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009.
- [18] E. Bertino and N. Islam. Botnets and internet of things security. *Computer*, 50(2):76–79, Feb. 2017.
- [19] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE '09*, pages 141–150, New York, NY, USA, 2009. ACM.
- [20] G. Blair, N. Bencomo, and R. B. France. Model-run.time. *Computer*, 2009.
- [21] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of nfa. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, pages 1004–1009, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [22] A. Botta, W. de Donato, V. Persico, and A. Pescapé. Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684–700, 2016.
- [23] A. Bouchachia. Incremental learning with multi-level adaptation. *Neurocomputing*, 74(11):1785 – 1799, 2011. Adaptive Incremental Learning in Neural Networks Learning Algorithm and Mathematic Modelling Selected papers from the International Conference on Neural Information Processing 2009 (ICONIP 2009).
- [24] A. Bouchachia, B. Gabrys, and Z. Sahel. Overview of some incremental learning algorithms. In *2007 IEEE International Fuzzy Systems Conference*, pages 1–6, July 2007.
- [25] M. Brachmann, S. L. Keoh, O. G. Morchon, and S. S. Kumar. End-to-end transport security in the ip-based internet of things. In *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–5, July 2012.
- [26] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009.
- [27] P. Bucciol, J. L. Zechinelli-Martini, and G. Vargas-Solar. Optimized transmission of loss tolerant information streams for real-time vehicle-to-vehicle communications. In *2009 Mexican International Conference on Computer Science*, pages 142–145, Sept. 2009.
- [28] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, Sept. 2012.
- [29] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot – A Technique for Cheap Recovery. In *IEEE Internet Computing*, volume 5 of *OSDI'04*, pages 46–55, Berkeley, CA, USA, 2004. USENIX Association.
- [30] M. Caporuscio, M. D'Angelo, V. Grassi, and R. Mirandola. Reinforcement learning techniques for decentralized self-adaptive service assembly. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.
- [31] M. Caporuscio, A. Di Marco, and P. Inverardi. Model-based system reconfiguration for dynamic performance management. *Journal of Systems and Software*, 2007.
- [32] M. Caporuscio and C. Ghezzi. Engineering Future Internet applications: The Prime approach. *Journal of Systems and Software*, 2015.
- [33] M. Caporuscio, V. Grassi, M. Marzolla, and R. Mirandola. Goprime: A fully decentralized middleware for utility-aware service assembly. *IEEE Transactions on Software Engineering*, 2016.
- [34] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications*, pages 139–152, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- [35] C. Chandraratne, R. T. Naayagi, and T. Logenthiran. Smart grid protection through self-healing. In *2017 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, pages 1–6, Dec. 2017.
- [36] F. Chen and R. Li. Sink node placement strategies for wireless sensor networks. *Wireless personal communications*, 68(2):303–319, 2013.
- [37] B. H. C. Cheng and et al. 08031 – software engineering for self-adaptive systems: A research road map. In *Software Engineering for Self-Adaptive Systems*, volume 8031 of *Dagstuhl Seminar Proceedings*. IBFI, 2008.
- [38] C. Cheng, C. Wang, W. Ku, T. Chen, and J. Wang. A scalable high-performance virus detection processor against a large pattern set for embedded network security. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(5):841–854, May 2012.
- [39] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel. D-lite: Distributed logic for internet of things services. In *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pages 16–

- 24, Oct. 2011.
- [40] S. Cherrier, Y. M. Ghamri-Doudane, S. Lohier, and G. Roussel. Fault-recovery and coherence in internet of things choreographies. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 532–537, Mar. 2014.
 - [41] P. K. Choubey, S. Pateria, A. Saxena, , and K. K. J. and. Power efficient, bandwidth optimized and fault tolerant sensor management for iot in smart home. In *2015 IEEE International Advance Computing Conference (IACC)*, pages 366–370, June 2015.
 - [42] M. Cinque, D. Cotroneo, and A. Pecchia. Event logs for the analysis of software failures: A rule-based approach. *IEEE Transactions on Software Engineering*, 39(6):806–821, June 2013.
 - [43] S. Cotard, S. Faucou, J.-L. Bechenec, A. Queudet, and Y. Trinquet. A data flow monitoring service based on runtime verification for autosar. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 1508–1515. IEEE, 2012.
 - [44] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–79, 1991.
 - [45] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering, ICSE '12*, pages 1084–1093, Piscataway, NJ, USA, 2012. IEEE Press.
 - [46] M. D'Angelo, A. Napolitano, and M. Caporuscio. Cyphef: A model-driven engineering framework for self-adaptive cyber-physical systems. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE '18*, pages 101–104, New York, NY, USA, 2018. ACM.
 - [47] D. C. Daniel and K. L. Herbig. *Strategic military deception: Pergamon policy studies on security affairs*. Elsevier, 2013.
 - [48] H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format (IDMEF). Technical report, The IETF Trust, 2007.
 - [49] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan. Loglens: A real-time log analysis system. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1052–1062, July 2018.
 - [50] B. Dhanalaxmi, G. A. Naidu, and K. Anuradha. A review on software fault detection and prevention mechanism in software development activities. *Journal of Computer Engineering*, 17(6):25–30, 2015.
 - [51] J. Dowling and V. Cahill. Self-managed decentralised systems using k-components and collaborative reinforcement learning. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems, WOSS '04*, pages 39–43, New York, NY, USA, 2004. ACM.
 - [52] A. El-Hokayem and Y. Falcone. Bringing runtime verification home. In *International Conference on Runtime Verification*, pages 222–240. Springer, 2018.
 - [53] A. Fiaschetti, V. Suraci, and F. D. Priscoli. The shield framework: How to control security, privacy and dependability in complex systems. In *2012 Complexity in Engineering (COMPENG). Proceedings*, pages 1–4, June 2012.
 - [54] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. Software Engineering Meets Control Theory. In *Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, 2015.
 - [55] A. Filieri, G. Tamburrelli, and C. Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, Jan. 2016.
 - [56] F. Flammini, S. Dambrot, D. de Kerchove, W. Kinser, L. MacDonald Glenn, and R. Saracco. Symbiotic Autonomous Systems - An FDC Initiative White Paper II. *IEEE Symbiotic Autonomous Systems*, 11 2018.
 - [57] F. Flammini, N. Mazzocca, and A. Orazzo. Automatic instantiation of abstract tests on specific configurations for large critical control systems. *Software Testing, Verification and Reliability*, 19(2):91–110, 2009.
 - [58] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J. M. Jezequel. Modeling and validating dynamic adaptation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009.
 - [59] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FoSE 2007: Future of Software Engineering*, 2007.
 - [60] J. R. D. Garibay, A. Almeida, S. A. Kados, A. G. Corcuera, and D. L. D. Ipiña. Codesign-oriented platform for agile internet of things prototype development. In *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 387–392, July 2014.
 - [61] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 2004.
 - [62] J. C. Georgas, A. van der Hoek, and R. N. Taylor. Using architectural models to manage and visualize runtime adaptation. *Computer*, 2009.
 - [63] A. Gepperth and B. Hammer. Incremental learning algorithms and applications. *European Symposium on Artificial Neural Networks (ESANN)*, 2016.
 - [64] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya. Self-healing systems—survey and synthesis. *Decision support systems*, 42(4):2164–2185, 2007.
 - [65] T. N. Gia, A.-M. Rahmani, T. Westerlund, P. Liljeborg, and H. Tenhunen. Fault tolerant and scalable iot-based architecture for health monitoring. In *2015 IEEE Sensors Applications Symposium (SAS)*, pages 1–6. IEEE, 2015.
 - [66] H. J. Goldsby and B. H. Cheng. Automatically generating behavioral models of adaptive systems to address uncertainty. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008.
 - [67] N. Gupta, V. Naik, and S. Sengupta. A firewall for internet of things. In *2017 9th International Conference on Communication Systems and Networks (COM-SNETS)*, pages 411–412, Jan. 2017.

- [68] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1573–1582, New York, NY, USA, 2016. ACM.
- [69] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 60–70. ACM, 2018.
- [70] S. He, J. Zhu, P. He, and M. R. Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, Oct. 2016.
- [71] A. J. Höglund, K. Hätönen, and A. S. Sorvari. A computer host-based user anomaly detection system using the self-organizing map. In *ijcnn*, page 5411. IEEE, 2000.
- [72] A. A. Hudaib, H. N. Fakhouri, F. E. Al Adwan, and S. N. Fakhouri. A survey about self-healing systems (desktop and web application). *Communications and Network*, 9(1):71–88, 2017.
- [73] IBM. An architectural blueprint for autonomic computing. Technical report, IBM, June 2005.
- [74] K. İnçki and I. Ari. A novel runtime verification solution for iot systems. *IEEE Access*, 6:13501–13512, 2018.
- [75] T. Jung, N. Jazdi, and M. Weyrich. A survey on dynamic simulation of automation systems and components in the internet of things. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, Sept. 2017.
- [76] S. A. Karthikeya, J. K. Vijeth, and C. S. R. Murthy. Leveraging solution-specific gateways for cost-effective and fault-tolerant iot networking. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–6, Apr. 2016.
- [77] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits. Denial-of-service detection in 6lowpan based internet of things. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 600–607, Oct. 2013.
- [78] C. M. Kennedy. *Distributed reflective architectures for anomaly detection and autonomous recovery*. PhD thesis, Citeseer, 2003.
- [79] C. M. Kennedy. Distributed meta-management for self-protection and self-explanation. *School of Computer Science Research Reports-University of Birmingham CSR*, 3, 2008.
- [80] L. Khan, M. Awad, and B. Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB journal*, 16(4):507–521, 2007.
- [81] D. Kim and S. Park. Reinforcement Learning-Based Dynamic Adaptation Planning Method for. *Seams*, 2009.
- [82] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. In *FoSE 2007: Future of Software Engineering*, 2007.
- [83] Q. D. La, T. Q. S. Quek, J. Lee, S. Jin, and H. Zhu. Deceptive attack and defense game in honeypot-enabled networks for the internet of things. *IEEE Internet of Things Journal*, 3(6):1025–1035, Dec. 2016.
- [84] J. S. Laursen, U. P. Schultz, and L. Ellekilde. Automatic error recovery in robot assembly operations using reverse execution. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1785–1792, Sept. 2015.
- [85] S. Lee, M. Bae, and H. Kim. Future of IoT Networks: A Survey. *Applied Sciences*, 7(10):1072, 2017.
- [86] M. Leucker, M. Schmitz, and D. à Tellinghusen. Runtime verification for interconnected medical devices. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, pages 380–387, Cham, 2016. Springer International Publishing.
- [87] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 102–111, New York, NY, USA, 2016. ACM.
- [88] M. Lipow. Number of faults per line of code. *IEEE Transactions on Software Engineering*, SE-8(4):437–439, July 1982.
- [89] Y. Liu, Y. Wu, and Z. Kalbarczyk. Smart maintenance via dynamic fault tree analysis: A case study on singapore mrt system. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 511–518. IEEE, 2017.
- [90] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li. Mining invariants from console logs for system problem detection. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 24–24, Berkeley, CA, USA, 2010. USENIX Association.
- [91] J. T. Luxhøj. Predictive safety analytics for complex aerospace systems. *Procedia computer science*, 20:331–336, 2013.
- [92] L. Mariani and F. Pastore. Automated identification of failure causes in system logs. In *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, pages 117–126, Nov. 2008.
- [93] D. A. Menascé, H. Gomaa, S. Malek, and J. P. Sousa. Sassy: A framework for self-architecting service-oriented systems. *IEEE Software*, 2011.
- [94] S. Misra, A. Gupta, P. V. Krishna, H. Agarwal, and M. S. Obaidat. An adaptive learning approach for fault-tolerant routing in internet of things. In *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 815–819, Apr. 2012.
- [95] A. Moosa and E. M. Alsaffar. Proposing a hybrid-intelligent framework to secure e-government web applications. In *Proceedings of the 2Nd International Conference on Theory and Practice of Electronic Governance*, ICEGOV '08, pages 52–59, New York, NY, USA, 2008. ACM.
- [96] A. Nakamura and T. Kotoku. Systematization of error recovery in skill-based manipulation. *Artificial Life and Robotics*, 14(2):203, Dec. 2009.
- [97] A. Nakamura, K. Nagata, K. Harada, and N. Yamano. Technique of recovery process and application of ai in error recovery using task stratification and error classification. *Journal of Robotics, Networking and Artificial Life*, 5:56–62, 2018.

- [98] A. Nakamura, K. Nagata, K. Harada, N. Yamanobe, T. Tsuji, T. Foissotte, and Y. Kawai. Error recovery using task stratification and error classification for manipulation robots in various fields. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3535–3542, Nov. 2013.
- [99] K. Nandhini, M. Pavithra, K. Revathi, and A. Rajiv. Anomaly detection for safety monitoring. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6, Mar. 2017.
- [100] G. C. Necula, S. McPeak, and W. Weimer. Cured: Type-safe retrofitting of legacy code. In *ACM SIGPLAN Notices*, volume 37, pages 128–139. ACM, 2002.
- [101] B. Nelson. *Virtual Patching: Fighting Brute Force Attacks in a Software Defined Network*. PhD thesis, Texas A&M University, 2018.
- [102] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, pages 177–186, Apr. 1998.
- [103] N. Palmer and P. W. Goldberg. Pac-learnability of probabilistic deterministic finite state automata in terms of variation distance. *Theor. Comput. Sci.*, 387(1):18–31, Nov. 2007.
- [104] H. Park and J. Kim. Design and implementation of a high-speed rfid data filtering engine. In X. Zhou, O. Sokolsky, L. Yan, E.-S. Jung, Z. Shao, Y. Mu, D. C. Lee, D. Y. Kim, Y.-S. Jeong, and C.-Z. Xu, editors, *Emerging Directions in Embedded and Ubiquitous Computing*, pages 423–434, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [105] T. Patikirikorala, A. Colman, J. Han, and L. Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2012.
- [106] A. Power and G. Kotonya. A microservices architecture for reactive and proactive fault tolerance in iot systems. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 588–599, June 2018.
- [107] N. Provos and T. Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.
- [108] H. Psaiar and S. Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, 2011.
- [109] N. Qadri, M. Altaf, M. Fleury, M. Ghanbari, and H. Sammak. Robust video streaming over an urban vanet. In *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 429–434, Oct. 2009.
- [110] R. Rajachandrasekar, X. Besseron, and D. K. Panda. Monitoring and predicting hardware failures in hpc clusters with ftb-ipmi. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 1136–1143, May 2012.
- [111] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection in online data sources. In *proceedings of the 24th International Conference on Software Engineering*, pages 302–312. ACM, 2002.
- [112] R. Ren, J. Cheng, Y. Yin, J. Zhan, L. Wang, J. Li, and C. Luo. Deep convolutional neural networks for log event classification on distributed cluster systems. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1639–1646, Dec. 2018.
- [113] C. E. Rothenberg, M. Ieee, S. Azodolmolky, S. M. Ieee, S. Uhlig, and M. Ieee. Software-Defined Networking : A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14 – 76, 2015.
- [114] A. Sajid, H. Abbas, and K. Saleem. Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges. *IEEE Access*, 4:1375–1384, 2016.
- [115] F. Salfner and M. Malek. Architecting dependable systems with proactive fault management. In *Architecting dependable systems VII*, pages 171–200. Springer, 2010.
- [116] C. Scott and R. Carbone. Designing and implementing a honeypot for a scada network. *SANS Institute Reading Room*, 2014.
- [117] B. Shanmugam and N. B. Idris. Improved intrusion detection system using fuzzy logic for detecting anomaly and misuse type of attacks. In *2009 International Conference of Soft Computing and Pattern Recognition*, pages 212–217. IEEE, 2009.
- [118] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.
- [119] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana. Hvlearn: Automated black-box analysis of hostname verification in ssl/tls implementations. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 521–538, May 2017.
- [120] P. Sousa, A. N. Bessani, and R. R. Obelheiro. The forever service for fault/intrusion removal. In *Proceedings of the 2Nd Workshop on Recent Advances on Intrusion-tolerant Systems, WRAITS '08*, pages 5:1–5:6, New York, NY, USA, 2008. ACM.
- [121] H. Sözer, B. Tekinerdoğan, and M. Akşit. Flora: a framework for decomposing software architecture to introduce local recovery. *Software: Practice and Experience*, 39(10):869–889, 2009.
- [122] P. H. Su, C. Shih, J. Y. Hsu, K. Lin, and Y. Wang. Decentralized fault tolerance mechanism for intelligent iot/m2m middleware. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 45–50, Mar. 2014.
- [123] F. Tartanoglu, V. Issarny, A. B. Romanovsky, and N. Lévy. Coordinated Forward Error Recovery for Composite Web Services. In *22nd Symposium on Reliable Distributed Systems : SRDS 2003*, pages 167–176, Florence, Italy, 2003.
- [124] D. Tokody, J. Papp, L. B. Iantovics, and F. Flammini. *Complex, Resilient and Smart Systems*, pages 3–24. Springer International Publishing, Cham, 2019.
- [125] K. S. Trivedi and K. Vaidyanathan. Software rejuvenation - modeling and analysis. In R. Reis, editor, *Information Technology*, pages 151–182, Boston, MA, 2004. Springer US.
- [126] M. Tsai, C. Shieh, T. Huang, and D. Deng. Forward-looking forward error correction mechanism for video streaming over wireless networks. *IEEE Systems Journal*, 5(4):460–473, Dec. 2011.
- [127] W. van der Aalst and C. Stahl. *Modeling business processes : a petri net-oriented approach*. Springer Berlin Heidelberg, 2011.
- [128] W. M. P. van der Aalst. *Process Mining*, volume 5. Springer Berlin Heidelberg, 2011.
- [129] E. van der Kouwe. *Improving software fault injection*.

- PhD thesis, Vrije Universiteit Amsterdam, 2016. Exacte Wetenschappen Naam instelling promotie: Vrije Universiteit Amsterdam Naam instelling onderzoek: Vrije Universiteit Amsterdam.
- [130] J. Vella and S. Zammit. A survey of multicasting over wireless access networks. *IEEE Communications Surveys Tutorials*, 15(2):718–753, Second 2013.
 - [131] D. Weyns. *Architecture-based design of multi-agent systems*. Springer, 2010.
 - [132] D. Weyns. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering*, 2017.
 - [133] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Miranda, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka. On patterns for decentralized control in self-adaptive systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013.
 - [134] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An architectural approach to autonomic computing. In *Proceedings - International Conference on Autonomic Computing*, 2004.
 - [135] C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. Citeseer, 2014.
 - [136] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSOP '09*, pages 117–132, New York, NY, USA, 2009. ACM.
 - [137] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks, HotNets-XIV*, pages 5:1–5:7, New York, NY, USA, 2015. ACM.
 - [138] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated known problem diagnosis with event traces. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 375–388. ACM, 2006.
 - [139] T. Zhao, W. Zhang, H. Zhao, and Z. Jin. A Reinforcement Learning-Based Framework for the Generation and Evolution of Adaptation Rules. In *Proceedings - 2017 IEEE International Conference on Autonomic Computing, ICAC 2017*, 2017.
 - [140] X. Zheng, C. Julien, R. Podorozhny, F. Cassez, and T. Rakotoarivelo. Efficient and scalable runtime monitoring for cyber-physical system. *IEEE Systems Journal*, 12(2):1667–1678, 2018.
 - [141] S. Zhou, K. Lin, J. Na, C. Chuang, and C. Shih. Supporting service adaptation in fault tolerant internet of things. In *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 65–72, Oct. 2015.
 - [142] D.-Q. Zou, H. Qin, and H. Jin. Uilog: Improving log-based fault diagnosis by log analysis. *Journal of Computer Science and Technology*, 31(5):1038–1052, Sept. 2016.