Towards Self-Healing in the Internet of Things by Log Analytics and Process Mining

Prasannjeet Singh[a], Mauro Caporuscio[a], Francesco Flammini[a], Narges Khakpour[a]

[a]*Linnaeus University, Sweden*

Abstract

The Internet of Things (IoT) will be used in increasingly complex and critical applications where heterogeneous devices will work together in connected systems. In this paper we address methods for log-analytics and process mining in order to support automatic problem detection and diagnosis in IoT. Those methods are essential to provide a foundation for the future generation IoT systems that will be capable of self-healing.

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

1.1. Connected Cyber-Physical Systems

Connected systems is not a new concept; however, it is used more than ever in today's day and age, even on a personal basis. There are hardly any personal electronic devices which are used independently without being connected to a system, or the internet. This phenomenon is not just limited to personal systems and is also prevalent in other industries like heavy machinery, automobiles, etc., where every individual system is connected to one or more systems, which monitors or controls them according to the requirement. For instance, Vivint.com [1] competently describes smart home, as a home that is equipped with technology to remotely control and automate household systems like lighting, doors, thermostats, entertainment systems, security alarms, surveillance cameras and other connected appliances. The Internet of Things (IoT) can be described as a system that includes the collection of tangible components that are connected, either wirelessly or otherwise, and the system is connected to internet. The components may include a range of sensors which can use various types of local area connections such as RFID, NFC, Wi-Fi, Bluetooth and Wide Area Connectivity, such as GSM, GPRS, 3G and LTE, and can record human or other activities which can then be relayed to another component, such as the processing system, and this information can be used to operate the system effectively, to gather data, to troubleshoot the system, or to perform any other relevant tasks[2]. The deployment diagram of a typical IoT system is shown in Figure 1. Multiple connectivity between entities can be observed. Some devices are connected wirelessly where as some have a wired connection. Moreover, all the sub-systems belong to different manufacturers. However, all of them work together to form an orchestration.

Like the Figure 1, more often than not, most of the components in an IoT system are manufactured by different vendors. Though many failures *within* an individual component may be taken care of by their respective manufacturers; however, it is likely that any other failure *between* different components in an IoT system, such as a simple connectivity problem will not be supported by the manufacturers, as it is generally not covered under the warranty. A popular report from Forrester[3] predicts that there will be a huge growth in the IoT industry in the next years where

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

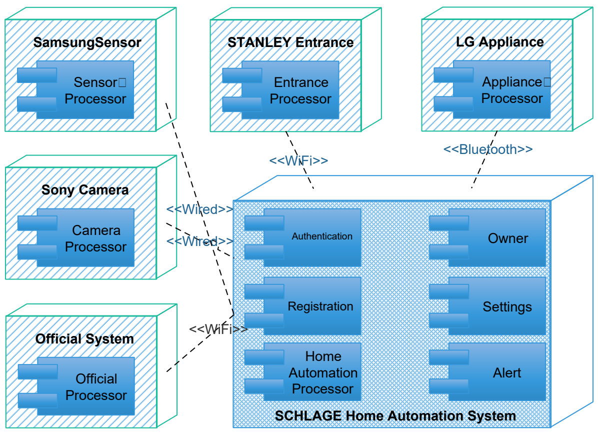


Figure 1: A Typical Smart Home Deployment Diagram.

enterprises will increase their efforts to introduce voice-based services to consumers and new European guidelines will allow commercializing the IoT data. In 2010, the number of objects connected to the Internet surpassed the earth's human population[4]. Additionally, Symantec[5] 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. A report from Gartner[6] 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, which is a direct result of the increase in vulnerabilities in the systems.

The above discussion leads us to believe that diagnosing and troubleshooting failures in an IoT system will be a huge problem in the near future which may not be covered by any manufacturer, unless all the entities within the system are manufactured by the same company. In this paper, we will address the issue of IoT self-healing that is automatically diagnosing and troubleshooting IoT systems. Other related concepts such as error prevention will also be looked at. This field has been prolifically researched over the years under various other lexical analogues, one of them being Self-Healing [7, 8, 9]. Several surveys have been done previously on Self-Healing[7, 10, 11, 12], however, they do not specifically discuss the phenomenon for IoT.

There are many ways to detect if an error has occurred in a system [13]. One of them is System-Level Monitoring that is used by several commercial enterprise applications. Another notable method is to detect errors/failures at the application layer. However, the most common way to examine a system for detecting any error, fault or failure is by analyzing the event logs generated by the system. Therefore, in this paper, we will focus on the research works that emphasize on analyzing event logs for detecting and diagnosing errors.

The organization of this paper is as follows: IoT error prevention detection and diagnosis will be briefly surveyed in Section II. Specific methods for log analysis and process mining will be addressed in Section III. Finally, Section IV will draw conclusions and hints about future developments.

2. Prevention, Detection and Diagnosis in IoT

For the sake of consistency in this paper, we will follow the taxonomy for dependable and secure computing as proposed by Avizienis et al.[14]. A state of the system or a portion of state of the system that may entail its eventual failure is labeled as an *error*. Further, the speculated cause of the error is termed as a *fault*; and finally, when the functionality of the system diverges from its correct behavior, it is termed as a *failure*. A fault, when activated causes an error, and the propagation of this error causes a failure[15].

Since the expected number of bugs in a system is proportional to its lines of code [16], it can be safely assumed that it is impossible to exhaustively list down all possible faults in a system. Therefore, introduction of faults in a system is inevitable, and more emphasis is given in developing a *fault-tolerant* system, rather than building a *fault-less* system. A system could be termed as *fault-tolerant*, if it is able to prevent the *fault* from turning into a *failure*. [17] proposes a fault tolerant mechanism which is implemented on a WuKong[18] middleware. The mechanism furnishes 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. [19] proposes a fault-tolerant architecture for healthcare IoT systems consisting Wireless Sensor Networks (WSN). A 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) architecture is presented for healthcare environments that constructs an improved gateway that solves 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[20] hardware are also covered. Mishra et al. in their research[21] 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, in their thesis[22] propose *fault-injection*, where artificial faults are injected into a system in order to learn the behavior of a system with activated faults, and eventually prevent system failure.

Cyber Physical Systems such as SCADA (Supervisory Control And Data Acquisition) is an appropriate example for the current state-of-the-art in fault tolerant Internet of Things system[23]. These systems are competent in offering flexibility, stability and fault tolerance. These systems exploiting cloud computing services, integrated with Internet of Things can be judged as Smart Industrial Systems which are predominantly employed in smart grids, smart transportation, eHealthcare and smart medical systems.

Intentional attacks are common threats in IoT. Thus, it is important to employ an IoT intrusion prevention mechanism in to prevent those threats. Various methods of intrusion detection are discussed and a taxonomy for the same is proposed by Zarpelãoa et al.[24]. Bertino and Islam[25] propose various guidelines that can prevent an IoT system from being compromised. Kasinathan et al.[26] 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 the network operations are disrupted. Another efficient way of intrusion prevent is the installation of Honeypot in a system. Likewise, Honeynets are an aggregation of Honeypots that are intended to imitate usual servers and network services[27]. In general, honeypots are essentially a technique of deception, where the defender purposely hoodwinks the attacker into acting in one's favor[28]. While Yu et al.[29] rejects the idea of Honeypots in an IoT setup due to non-scalability and dependency issues, La et al.[30] considers the possibility and uses game theory to analyze the situation where both attacker and defender try to deceive each other. While virtual patching is a type of firewall often mentioned as a Web Application Firewall (WAF), an IoT system also needs a full-fledged firewall as most of the embedded systems that are part of the IoT system have little to no security. A recent improvement on the traditional firewalls for IoT is *Smart Firewall*, which, unlike traditional software-based firewall, is a hardware-based device[31]. Gupta et al.[32] have proposed an implementation of firewall for Internet of Things using Raspberry Pi as a gateway. Other legacy research works in the field of firewall and security of IoT include [33, 34, 35, 36].

3. Log Analysis and Process Mining

Traditional logging techniques in major applications aren't structured and many the errors logged by them don't lead to faults, while these errors might be useful in some cases, most of the times they only decrease the readability of error logs by humans. Furthermore, there are some instances where errors, which were directly responsible for the failure of a system, were failed to be captured by these logging techniques. Considering that we are only focused on errors that lead to a failure, these techniques lead to a lot of false positives and false negatives. Here we focus only on the effective errors, or the errors which leads to the activation of faults, which in turn leads to a failure.

A lot of work has been done previously to overcome the problems that occur by these traditional logging techniques, few of them being parsing the error logs to filter out the relevant content. While it is a great way to refine the error logs, it must go through a cumbersome work of parsing and at the same time, it does not address the core issue with error logs, which is designing a new logging mechanism.

Most of the logging patterns try to detect errors and log it by placing a line of code at the end of a block of instructions, these techniques may not be able to detect errors such as infinite loops, etc. Therefore, instead of working on the resultant error logs, this paper aims to develop a new logging mechanism which takes care of these issues, and others such as false positives and false negatives by monitoring changes in the control flow of the program, and they do this by placing the logging instructions strategically in the code.

Moreover, it aims to detect only those errors which cause failure.

Case ID	Event ID	Time	Activity
1	85477	15-01-2019:14.05	connection request received
1	85478	15-01-2019:14.06	SYN
1	85479	15-01-2019:14.06	SYN-ACK
1	85480	15-01-2019:14.08	pin verification
1	85481	15-01-2019:14.08	verification successful
1	85482	15-01-2019:14.08	ACK
1	85483	15-01-2019:14.09	connection successful
2	92199	11-02-2019:09:11	connection request received
2	92200	11-02-2019:09:12	SYN
2	92200	11-02-2019:09:12	SYN-ACK
2	92201	11-02-2019:09:13	pin verification
2	92201	11-02-2019:09:15	verification failed
2	92202	11-02-2019:09:16	pin verification
2	92202	11-02-2019:09:17	verification failed
2	92200	11-02-2019:09:18	pin verification
2	92201	11-02-2019:09:18	verification successful
2	92201	11-02-2019:09:20	ACK
2	92202	11-02-2019:09:21	connection successful

Table 1: A Sample Event Log

3.1. The Rule-Based Logging-Mechanism

In[15], a set of entities has been identified in a system that interacts with each other, and are prone to cause an error. This is done by referring to high-level or low-level models available at the design time, for example the architectural model, system conceptual model, or the Unified Modeling Langua. However, if these models are not available, or the logging mechanism is supposed to be implemented in an already developed system, other reverse engineering techniques can be used to isolate the entities starting from source code.

Further, a set of error-modes are established, based on the widely accepted taxonomy in the dependability area[14]. Such as Service Error (SER: Prevents an invoked service from reaching the exit point.) Other error modes are also discussed in[15]. Different types of code injections in the form of events are then done in the source code of system which helps in the logging. These are called Login Rules (LR). Let us discuss two such Login Rules:

LR-1: Service Start (SST) and LR-2: Service End (SEN) are logged as the very first and very last instructions of a service, respectively. These login rules are made for the first error-mode, which is Service Error (SER). Note that just like LR-1 and LR-2, there are other Logging Rules (till LR-8) discussed in the paper which addresses all four errors described above. A notable Logging Rule worth mentioning is the LR-7, Heartbeat (HTB), which logs a heartbeat event periodically.

In Rule-Based logging, these Login Rules don't write the log-file directly but are processed dynamically by a framework called as LogBus. For example, if Service Start (SST) event is logged in LogBus, but the Service End (SEN) event fails to log within a stipulated time, it can be concluded that a Service Error (SER) has occurred. This is when LogBus updates the log file with an entry. The stipulated time, as discussed above is calculated using historical time when the system had a clean run. This is explained methodically in the paper. Methods to identify the right areas to insert these Logging Rules code is difficult, and may require usage of regular expression to parse, and later insert codes, however, the paper does not discuss any tool which can be used to insert these codes into the system automatically. Furthermore, other Login Rules like the LR-6 and LR-8 may have to be introduced into the code manually, which is a drawback. Finally, authors discuss two case studies and show that the Rule-Based logging mechanism performs drastically better than the inbuilt logging techniques in the system. Moreover, the time overhead because of introducing extra lines of logging codes are negligible.

The above logging mechanism can be extended to work within an IoT system by introducing a new entity within the system which can be in the form of a framework. Let us name it as the *LogMonitor*. Contrary to traditional logging mechanisms where codes written within the system update the error logs, the LogMonitor will monitor all the events logged by the aforementioned logging rules, and eventually update the error log. This way a consistent error log can be generated for the entire system. However, this mechanism assumes that we have access to modify the internal codes of each entity within the IoT system. Other tedious methods, such as parsing of event logs[37, 38] from each entity may have to be done, in case the internal access is unavailable.

In the previous section, an approach was discussed to make an event log consistent in an IoT system. However, there still is a need to define the event logs and categorize them in such a way that appropriate tools can be developed that works on each category efficiently. Some event logs contain time-stamps; some contain user-readable texts while others do not. Some contain just a few sets of information in the events, while some may contain hundreds of information in a single line of event log. The replication package[39] contains three samples of event logs generated by various programs in the Widows Operation System. As it can be observed, event logs can range from being very complex to extremely simple. Thus, for a broad understanding, let us consider a general event log sample[40], as shown in Table 1, that is readily comprehensible. The given event log can represent all possible types of event log generated by a system, as it includes some identification, a time stamp and in this case, four different attributes, viz. Activity, Award Discount, Resource and Cost. The number of attributes for an event log has no restriction. Note that just like all the entries in an event log may or may not have values for every attribute, the same has been reciprocated in the event log shown in Table 1. These event logs are bound by some underlying assumptions[41]. One important assumption among the aforesaid is that each event refers to some activity. For example, in Table 1, every event consists of an activity, such as register request, check ticket, etc. A system such as the LogMonitor, as discussed in the section IIB above, could be programmed in such a way that it generates the event logs in the above format.

As can be observed, the event logs in Table 1 are initially divided into Cases. Each case is further divided into Event IDs. It is worth noting that events listed under a case should only relate to precisely one case. Moreover, all the events in a case should be chronologically ordered, and at least one attribute is mandatory for an event log to exist. Issues may arise when identical events are executed in a single case. If these identical events have a corresponding termination event, it can be difficult to identify which initial event this termination event is associated with. Errors like these are categorized in *Primary and Secondary Correlation Problems*[41]. Also, every entry in the event log should relate to an activity. For example, in Table 1, entries refer to activities like connection request received, pin verification, verification successful, etc. In other words, the first attribute, which is *activity* should definitely have a legit value for an entry in the event log to be present. The author also defines a term *trace*, which is a finite sequence of unique events within a case. Furthermore, each event has been named based on its primary attribute, i.e. *activity*. Therefore, the presence of this attribute has been made obligatory. In the same vein, a *Simple Trace* is defined as a sequence of *activities* and analogously, a *Simple Event Log* is defined as a multi-set of traces. In other words, a Simple Trace has only one attribute, which is 'activities' in this case. That being said, any event log can be easily transformed into a simple event log by choosing one particular attribute from the set of attributes possessed by that event log. This process is particularly beneficial, if a simple approach has to be applied to a rather complex event log and can be analogized with the process of Dimensionality Reduction in machine learning.

3.2. Process Mining

An IoT system can be expected to generate anywhere from a thousand to hundreds of thousands of events logs instances in an hour, and as a result it is imperative to extract meaningful information from them that can help us troubleshoot any errors. Amongst many other approaches, one of them is to find possible patterns in the event logs, and then expect the IoT system to follow the pattern which was modelled. If the system deviates from the determined pattern, it can be assumed that an error has occurred and based on which part of the pattern was violated, a possible fix can be determined and automatically applied, if possible. Additionally, if the fix is determined but automatic application is not possible, for example replacement of certain parts, a repair-person can procure the replacement part in their first visit, just by accessing the event logs, thus reducing the service cost and time.

This methodology of capturing patterns in event logs can be achieved through Process Mining. Process Mining is an excellent tool to capture meaningful information from event logs. It is mainly used to offer new medium to discover, monitor and improve processes in a quality of application domains. However, it can also be used to analyze and rectify errors in a system. It is a comparatively new research discipline that tries to extract knowledge from event logs of real processes (i.e. not assumed processes) which are readily available in today's information systems. Additionally, with the technology advancements, it has become easier to capture detailed event logs from any system, and thus, the quantity of event logs generated by any device in a period of time is increasing with every newly manufactured device.

This behavior is expected, as a larger quantity of event logs implies a bigger scope to analyze the error. Thus, a robust methodology is needed to diagnose and fix the errors in an IoT system that works with event logs generated by older as well as newer systems. A process model generated by process mining using the event logs can tackle this issue. Furthermore, many commercial and academic systems have implemented process mining algorithms. The manifesto[42] also claims that an active group of researchers are working on process mining and it has become one of the "hot topics" in Business Process Management (BPM) research. Additionally, process mining functionality is added by more and more software vendors to their tools. Notable examples are: Discovery Analyst (StereoLOGIC), ARIS Process Performance Manager (Software AG), and Comprehend (Open Connect). Aforementioned reasons led us to believe that Process Mining can be an effective tool for solving the problem of Self-Healing in IoT systems. Besides this, as of July, 2015, there were 18 PhD students being funded by Philips who were working in analyzing the Philips X-Ray machines and electronic shaving devices among other items to see how these machines are used in the field, when do they fail and why do they fail, etc.[43]. Moreover, most of the X-Ray machines manufactured by Philips use Process Mining for fault diagnosis[44, 42].

Since process mining primarily deals with event logs, it is assumed that the event logs used are in accordance with the guidelines mentioned in the section *Defining Event-Logs* (vide supra). The event-log is the starting point and can be used to conduct three types of process mining. We will briefly describe these three parts below.

Process Discovery: The most important part in Process Mining, it creates a model (called as Process Model) using event logs of real processes, but not using any theoretical or deduced information.

Conformance Checking: Here, a process model which was created through Process Discovery is compared with an unseen event log of the same process. This is done to check if in reality, an event log conforms to the model and vice-versa.

Enhancement: As the word says, here the objective is to improve the existing process model using additional information recorded in the event log.

Figure 2 describes the three types of process mining in terms of the input and output. Process Mining also covers different perspective, based on the goal that we need to achieve, these are the control-flow perspective, organizational perspective, case perspective and the time perspective.

As can be seen in various machine learning algorithms, a common approach to solve a problem is by assuming an appropriate model initially (by observing the dataset) and then finding the best set of hyper-parameters; however, in Process Mining, we start from a bunch of behavior (event logs, in our case) and automatically construct the model based on the logs. This is because more often than not, there are no appropriate existing models, or the models are flawed or incomplete. Reference[45] describes process model which is *roughly an anticipation of what the process will look like*.

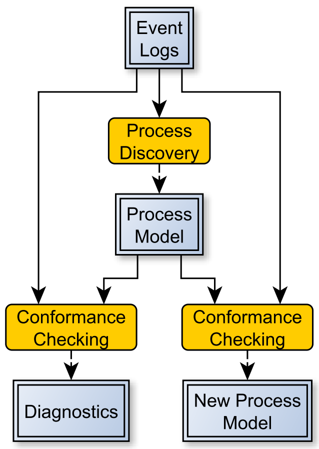


Figure 2: Different types of Process Mining in terms of Input and Output.

A process mining model can be represented using various notations, and the fact that there are these many notations available demonstrates the significance of process modeling. A simple event log as shown in the Figure 3a may be represented using an intuitive model. However, in case of an IoT system, or any electronic system whatsoever, the event logs are far more complex than the one illustrated above. As a result, a legitimate, well-defined notation should be used to represent a process model. Moreover, process models for complicated systems are prone to many problems, some of which are outlined underneath.

- Oversimplification of the model: Most models have a propensity to focus only on the desirable behavior, and in a way overlook the events that are less likely to happen. There are chances that these “overlooked” events cause most of the errors and thus, the model may not be helpful.

- Incompetence in satisfactorily capturing human behavior: Simple and mundane process involving human engagement are prone to modifications, as human behavior is unpredictable. These changes, although minor, should nevertheless be incorporated in a model, as these nonconformities by and large result in an eventual fault.

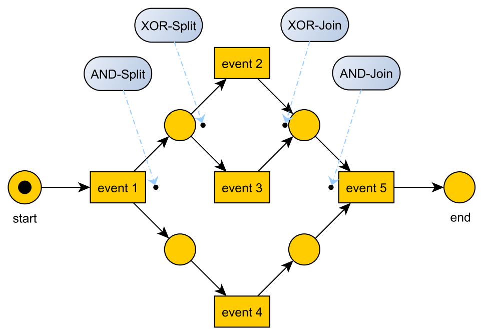
- Restricted or redundant abstraction level: It is important that an appropriate abstraction level is chosen based on the objective and the input data. There is a plausibility that the model is too abstract to answer a detailed question, or conversely, a model is too detailed, and thus redundant, to answer a simple question.

A manually composed process model are susceptible to the aforementioned (and similar) problems. Moreover, a poorly designed model may engender wrong conclusions. To eradicate these problems, process mining takes the help of event logs to create a process model. An event log notifies the exact steps that 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. Moreover, a process model is capable of providing different view in different abstraction levels for the same system.

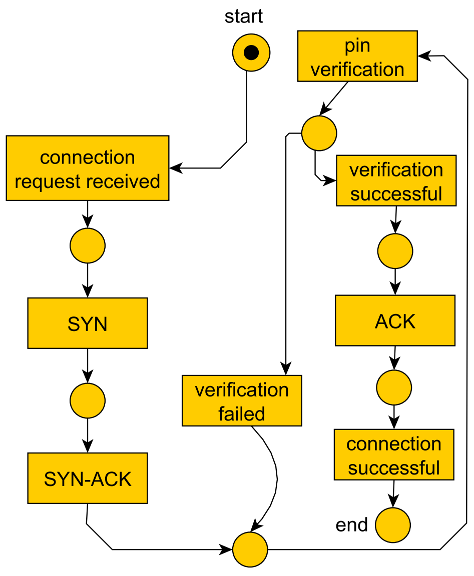
Algorithms such as the -algorithm can automatically generate process models based on event logs, as will be discussed in future sections. Notations such as EPC’s, BPMN, UML activity diagrams, etc. can be used to describe process model. It is also possible to easily convert one model to another. The primary objective of a process model is to resolve which activities need to be performed in what order. Note that activities can be executed sequentially or concurrently. They can also be optional. Furthermore, same activities can also be repeated. Let us quickly go through few of the notations that can be used to describe a process model[46]:

Transition Systems: A transition system can be considered to be the most basic process modeling notation. It consists of states and transitions. Diagrammatically, the notation resembles a non-deterministic finite automata (NFA), which can have more than one initial state and final states. The final states can also refer to as the “accept” states. The transitions, also called as “edges” in NFA denote the set of activities of the system, as described in the event log. For a path to be successfully terminated, it has to end in the final state. Moreover, if it ends at a non-final state without any ongoing transitions, it can be considered as a *deadlock*. Additionally, there is also a prospect of a *livelock* in case it is becomes impossible for a transition to reach the final state. A transition system can easily be translated to higher level languages such as those discussed above. However, the reverse may not be possible in some cases. These modes are simple, but they face huge issues in describing concurrent process. For example, for a process with ten parallel activities, a transition system will have to delineate more than three million execution sequences (10!), whereas a *Petri-net* needs only ten transitions. Therefore, in view of the concurrent nature of process that can be observed in many IoT systems, it is imperative to use more communicative models, such as Petri-Nets for satisfactorily portraying process mining results.

Petri Nets: As previously discussed, Petri nets are capable of describing concurrent processes without much effort like the transition systems. This is one of the oldest process modeling language. Due to its broad usage, it has also been extensively investigated; and as a result, there exists many tools to analyze them[43, 44, 42, 47]. A Petri net, as suitably described by authors in[48] is “a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles).” Petri nets have a static network structure. *Tokens* flow through the network governed by certain firing rules. The distribution of tokens over places (also referred as *marking*) determines the state of the Petri net. [49] provides a broad understanding of Petri nets. Furthermore, a *labeled Petri net* is an extension of the basic Petri nets, including a set of activity labels. Figure 3a depicts a labelled Petri-Net diagram which describes the purport of each junction. Furthermore, Figure 3b is a petri-net



(a) A Labeled Petri-Net



(b) A petri-net process model for the sample event log displayed in Table 1

Figure 3: Examples for Petri-Net Diagram

model for the sample event log shown in Table 2.

Workflow Nets (WF-net): A special type of Petri net, which is suitable for expressing workflows is used to create a process model. It is called as workflow nets[50]. A WF-net has a dedicated source place and sink place where the process starts and ends respectively. Another property, called soundness is defined for a WF-net. A WF-net is sound if and only if the places cannot hold multiple tokens at the same time, a proper completion is possible, there always is an option to complete and there is an absence of dead parts[46].

YAWL: The acronym stands for Yet Another Workflow Language. As the name suggests, it is a workflow modeling language. It is also an open-source workflow system. The Workflow Patterns Initiative heavily influenced its development[51, 52]. Amongst all the open-source workflow systems, YAWL is currently the most widely used. It is similar to WF-nets that every process has a start and end condition, however activities here are called tasks. Nevertheless, just like there are places in Petri-Nets, YAWL has conditions, though tasks in this case can also be connected to each other directly, without any condition in between. It is noteworthy that YAWL also supports *cancellation regions*. There may exist a cancelation region comprising of tasks, conditions and arcs. All the tokens are removed from this region after the completion of the task.

BPMN: For modeling business processes, BPMN, or the Business Process Mining Notation has recently emerged as the most expansively used language supported by many vendors and standardized by the OMG[53].

Apart from these, other distinguished notations such as *Event-Driven Process Chains (EPCs)*[54] and Casual Nets are also be utilized. While EPC covers a lot of functionality provided by BPMN and YAWL, it still is only a subset of these notations as many important functionalities aren't available in in EPC. Moreover, the semantics of EPC wasn't clearly defined by the creators, which led to the decrease in EPC's usage with time[55]. Note that these notations will not be elaborated upon in further sections as they are never used in the course of this survey. However, if one is more comfortable with a specific notation, note that most of the notations can be convert to the other by simple mathematical rules.

There are various existing algorithms that convert event logs to a process model as above. One of the most prominent and rather naïve algorithm is the -algorithm[56]. Additionally,

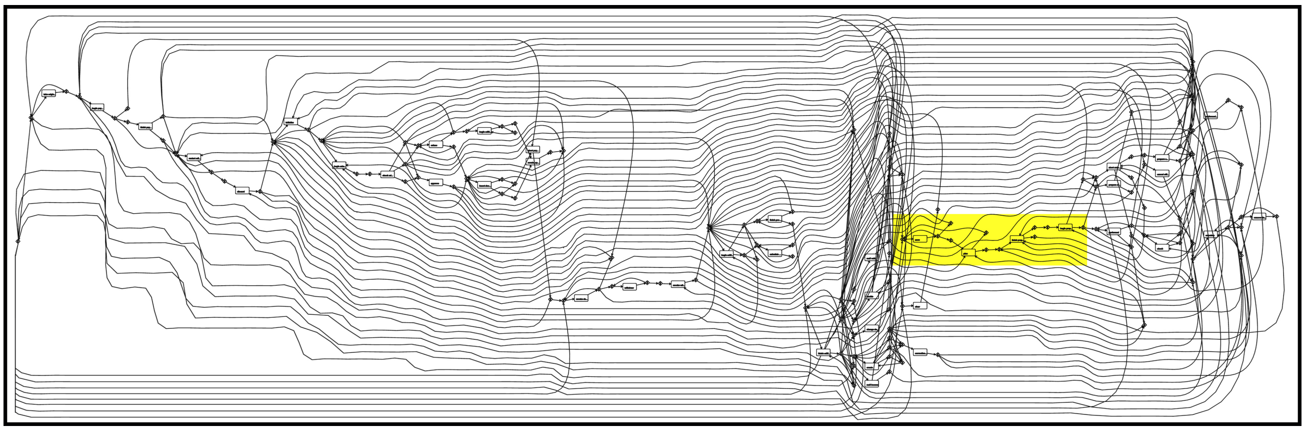


Figure 4: A complex process-model using petri-nets.

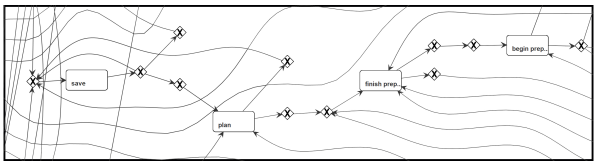


Figure 5: A zoomed-in version of the shaded region showing in the Figure 4.

Figure 4[1] is an example of a seemingly more complicated process model. It was created using ProM Tools. Figure 5 shows a zoomed-in version of the shaded region of the process model in Figure 4. Similarly, process models also contain detailed information, but relevant information may be available only in a particular section.

4. Conclusion

As we see the importance and growing presence of IoT and Connected Cyber-Physical systems around the world, we stressed the fact that there is need for methodologies, tools, technologies, procedures and frameworks supporting automatic error detection and – possibly – repair. However, bisecting process models requires various protocols to be followed[43], which are not covered in this survey. Once a robust process model is built, the following things can easily be found:

1. Something that should have happened but did not.
2. Something that shouldn't have happened, happened.
3. If the IoT system is being used as it was intended to be.
4. Models can also be used to analyze what is the most frequent path followed by the user, etc.

There are various tools available in the market to create a process model, e.g. ProM, Disco, etc.. However, to create a process model for a new system, a plugin may be needed so that the event logs can easily by comprehended by the tool. Note that apart from self-healing, process mining may also capable of solving the problem of error predictability, where an end-user may be warned beforehand, if their actions have a high probability of causing an error.

1 This sample process model was created using the Event-Logs of BPI Challenge 2018 (https://data.4tu.nl/repository/uuid:3301445f-95e8-4ff0-98a4-901f1f204972). The log-file contains 2,514,266 instances of events that resulted in the process model as shown in the figure.

References

[1] Vivint.com, Home Automation Systems and Smart Home Devices | Vivint.
URL https://www.vivint.com/packages/home-automation

[2] Lopez Research, An Introduction to the Internet of Things (IoT), Tech. rep. (2013).
URL https://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf

[3] M. Pelino, J. S. Hammond, C. Dai, P. Miller, J. Belissent, J. A. Ask, N. Fenwick, F. E. Gillett, T. Husson, M. Maxim, C. Voce, C. Garberg, D. Lynch, Predictions 2018: IoT Moves From Experimentation To Business Scale, Tech. rep., Forrester (2017).
URL https://www.forrester.com/report/Predictions+2018+IoT+Moves+From+Experimentation+To+Business+Scale/-/E-RES139752

[4] A. Al-fuqaha, S. Member, M. Guizani, M. Mohammadi, S. Member, Internet of Things : A Survey on Enabling 17 (4) (2015) 2347–2376 (2015).
URL http://ieeexplore.ieee.org.proxy.queensu.ca/document/7123563/

[5] Symantec, 5 Predictions on the future of the Internet of Things.
URL https://nr.tn/2O9VlxE

[6] W. Bamiduro, R. van der Meulen, Worldwide IoT Security Spending Will Reach $1.5 Billion in 2018, Tech. rep., Gartner (2018).
URL https://www.gartner.com/newsroom/id/3869181

[7] A. Asghar, H. Farooq, A. Imran, Self-Healing in Emerging Cellular Networks: Review, Challenges and Research Directions, IEEE Communications Surveys & Tutorials (c) (2018) 1–1 (2018). doi:10.1109/COMST.2018.2825786.
URL http://ieeexplore.ieee.org/document/8335292/

[8] C. Chandraratne, R. T. Naayagi, T. Logenthiran, A. Background, Smart Grid Protection through Self-Healing (2017) 1–6 (2017).

[9] K. Khalil, O. Eldash, M. Bayoumi, A Cost-Effective Self-Healing Approach for Reliable Hardware Systems (2018).

[10] H. Psaier, S. Dustdar, A survey on self-healing systems: Approaches and systems, Computing (Vienna/New York) 91 (1) (2011) 43–73 (2011). doi:10.1007/s00607-010-0107-y.

[11] A. A. Hudaib, H. N. Fakhouri, F. E. Al Adwan, S. N. Fakhouri, A Survey about Self-Healing Systems (Desktop and Web Application), Communications and Network 09 (01) (2017) 71–88 (2017). doi:10.4236/cn.2017.91004.
URL http://www.scirp.org/journal/doi.aspx?DOI=10.4236/cn.2017.91004

[12] D. Ghosh, R. Sharman, H. Raghav Rao, S. Upadhyaya, Self-healing systems - survey and synthesis, Decision Support Systems 42 (4) (2007) 2164–2185 (2007). doi:10.1016/j.dss.2006.06.011.

[13] L. M. Silva, Comparing Error Detection Techniques for Web Applications: An Experimental Study, 2008 Seventh IEEE International Symposium on Network Computing and Applications (2008) 144–151 (2008). doi:10.1109/NCA.2008.57.
URL http://ieeexplore.ieee.org/document/4579650/

[14] A. Avižienis, J. C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 1 (1) (2004) 11–33 (2004). doi:10.1109/TDSC.2004.2.

[15] M. Cinque, D. Cotroneo, A. Pecchia, Event logs for the analysis of software failures: A rule-based approach, IEEE Transactions on Software Engineering 39 (6) (2013) 806–821 (2013). doi:10.1109/TSE.2012.67.

[16] M. Lipow, Number of Faults per Line of Code, IEEE Transactions on Software Engineering SE-8 (4) (1982) 437–439 (1982). doi:10.1109/TSE.1982.235579.

[17] P. H. Su, C. S. Shih, J. Y. J. Hsu, K. J. Lin, Y. C. Wang, Decentralized fault tolerance mechanism for intelligent IoT/M2M middleware, 2014 IEEE World Forum on Internet of Things, WF-IoT 2014 (2014) 45–50 (2014). doi:10.1109/WF-IoT.2014.6803115.

[18] K.-J. Lin, D. Shih, Y.-C. Wang, J. Hsu, N. Reijers, G. Chou, B. Tsai, WuKong - Intelligent Middleware for Internet-of-Things (2016).
URL https://newslabntu.github.io/wukong4iox/

[19] T. N. Gia, A.-M. Rahmani, T. Westerlund, P. Liljeberg, H. Tenhunen, Fault tolerant and scalable IoT-based architecture for health monitoring, in: 2015 IEEE Sensors Applications Symposium (SAS), IEEE, 2015, pp. 1–6 (apr 2015). doi:10.1109/SAS.2015.7133626.
URL http://ieeexplore.ieee.org/document/7133626/

[20] F. Chen, R. Li, Sink node placement strategies for wireless sensor networks, Wireless Personal Communications 68 (2) (2013) 303–319 (jan 2013). doi:10.1007/s11277-011-0453-x.
URL http://link.springer.com/10.1007/s11277-011-0453-x

[21] S. Misra, A. Gupta, P. V. Krishna, H. Agarwal, M. S. Obaidat, An adaptive learning approach for fault-tolerant routing in Internet of Things, in: 2012 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2012, pp. 815–819 (apr 2012). doi:10.1109/WCNC.2012.6214484.
URL http://ieeexplore.ieee.org/document/6214484/

[22] E. van der Kouwe, Improving software fault injection, Ph.D. thesis, Vrije Universiteit Amsterdam (2016).

[23] A. Sajid, H. Abbas, K. Saleem, Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges, IEEE Access 4 (2016) 1375–1384 (2016). doi:10.1109/ACCESS.2016.2549047.
URL http://ieeexplore.ieee.org/document/7445139/

[24] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, S. C. de Alvarenga, A survey of intrusion detection in Internet of Things, Journal of Network and Computer Applications 84 (2017) 25–37 (apr 2017). doi:10.1016/J.JNCA.2017.02.009.
URL https://www.sciencedirect.com/science/article/pii/S1084804517300802

[25] E. Bertino, N. Islam, Botnets and Internet of Things Security, Computer 50 (2) (2017) 76–79 (feb 2017). doi:10.1109/MC.2017.62.
URL http://ieeexplore.ieee.org/document/7842850/

[26] P. Kasinathan, C. Pastrone, M. A. Spirito, 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), IEEE, 2013, pp. 600–607 (oct 2013). doi:10.1109/WiMOB.2013.6673419.
URL http://ieeexplore.ieee.org/document/6673419/

[27] N. Provos, T. Holz, Virtual honeypots : from botnet tracking to intrusion detection, Addison-Wesley, 2008 (2008).

[28] D. C. D. C. Daniel, K. L. K. L. Herbig, Strategic military deception: Pergamon policy studies on security affairs, Pergamon Press, 1982 (1982).

[29] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, C. Xu, Handling a trillion (unfixable) flaws on a billion devices, in: Proceedings of the 14th ACM Workshop on Hot Topics in Networks - HotNets-XIV, ACM Press, New York, New York, USA, 2015, pp. 1–7 (2015). doi:10.1145/2834050.2834095.
URL http://dl.acm.org/citation.cfm?doid=2834050.2834095

[30] Q. D. La, T. Q. S. Quek, J. Lee, S. Jin, H. Zhu,

Deceptive Attack and Defense Game in Honeypot-Enabled Networks for the Internet of Things, IEEE Internet of Things Journal 3 (6) (2016) 1025–1035 (dec 2016). doi:10.1109/JIOT.2016.2547994.
URL http://ieeexplore.ieee.org/document/7442780/

[31] K. Colburn, Smart firewalls protect the Internet of Things — which are the best? | WTOP (2017).
URL https://wtop.com/tech/2017/10/smart-firewalls-protect-internet-things-best/

[32] N. Gupta, V. Naik, S. Sengupta, A firewall for Internet of Things, in: 2017 9th International Conference on Communication Systems and Networks (COMSNETS), IEEE, 2017, pp. 411–412 (jan 2017). doi:10.1109/COMSNETS.2017.7945418.
URL http://ieeexplore.ieee.org/document/7945418/

[33] M. Brachmann, S. L. Keoh, O. G. Morchon, 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), IEEE, 2012, pp. 1–5 (jul 2012). doi:10.1109/ICCCN.2012.6289292.
URL http://ieeexplore.ieee.org/document/6289292/

[34] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, M. Rossi, Low power link layer security for IoT: Implementation and performance analysis, in: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE, 2013, pp. 919–925 (jul 2013). doi:10.1109/IWCMC.2013.6583680.
URL http://ieeexplore.ieee.org/document/6583680/

[35] S. Babar, A. Stango, N. Prasad, J. Sen, 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), IEEE, 2011, pp. 1–5 (feb 2011). doi:10.1109/WIRELESSVITAE.2011.5940923.
URL http://ieeexplore.ieee.org/document/5940923/

[36] S. Babar, P. Mahalle, A. Stango, N. Prasad, R. Prasad, Proposed security model and threat taxonomy for the Internet of Things (IoT), in: Communications in Computer and Information Science, Vol. 89 CCIS, Springer, Berlin, Heidelberg, 2010, pp. 420–429 (2010). doi:10.1007/978-3-642-14478-3_42.
URL http://link.springer.com/10.1007/978-3-642-14478-3_42

[37] P. He, J. Zhu, S. He, J. Li, M. R. Lyu, Towards automated log parsing for large-scale log data analysis, IEEE Transactions on Dependable and Secure Computing 15 (6) (2017) 931–944 (2017).

[38] M. Du, F. Li, Spell: Streaming parsing of system event logs, Proceedings - IEEE International Conference on Data Mining, ICDM (2017) 859–864 (2017). doi:10.1109/ICDM.2016.160.

[39] P. Singh, M. Caporuscio, F. Flammini, N. Khakpour, Smart Troubleshooting Survey Resources (2019).
URL https://github.com/prasannjeet/Smart-Troubleshooting-Survey-Resources

[40] E. U. o. T. Process Mining Group, Math&CS department, Event logs and models.
URL http://www.processmining.org/event_logs_and_models_used_in_book

[41] W. M. P. van der Aalst, Getting the Data, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 95–123 (2011). doi:10.1007/978-3-642-19345-3_4.
URL https://doi.org/10.1007/978-3-642-19345-3_4

[42] G. Vossen, The process mining manifesto - An interview with Wil van der Aalst, Information Systems 37 (3) (2012) 288–290 (2012). doi:10.1016/j.is.2011.10.006.
URL http://dx.doi.org/10.1016/j.is.2011.10.006

[43] W. van der Aalst, Process Mining Data Science in Action (2015).
URL https://www.youtube.com/watch?v=kIeLaNzw9hI&t=704s

[44] W. Van Der Aalst, Process mining, Communications of the ACM 55 (8) (2012) 76 (2012). arXiv:arXiv:1011.1669v3, doi:10.1145/2240236.2240257.
URL http://dl.acm.org/citation.cfm?doid=2240236.2240257

[45] C. Rolland, A Comprehensive View of Process Engineering, Lecture Notes in Computer Science 1413 (c) (1998) 1–24 (1998). doi:10.1007/BFb0054216>.
URL http://www.springerlink.com/index/XVER7FV40KKXMXJG.pdf

[46] W. M. P. van der Aalst, Process Modeling and Analysis, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 29–57 (2011). doi:10.1007/978-3-642-19345-3_2.
URL https://doi.org/10.1007/978-3-642-19345-3_2

[47] W. van der Aalst, C. Stahl, Modeling business processes : a petri net-oriented approach.
URL https://mitpress.mit.edu/books/modeling-business-processes

[48] G. Manoj, J. Samson Immmanuel, P. S. Divya, A. P. Haran, Modelling of System Configuration and Reconfiguration for IMS, Springer, Berlin, Heidelberg, 2012, pp. 285–292 (dec 2012). doi:10.1007/978-3-642-35594-3_40.
URL http://link.springer.com/10.1007/978-3-642-35594-3_40

[49] C. Petri, W. Reisig, Petri net, Scholarpedia 3 (4) (2008) 6477 (2008). doi:10.4249/scholarpedia.6477.
URL http://www.scholarpedia.org/article/Petri_net

[50] W. M. P. van der Aalst, Making Work Flow: On the Application of Petri Nets to Business Process Management, Springer, Berlin, Heidelberg, 2002, pp. 1–22 (2002). doi:10.1007/3-540-48068-4_1.
URL http://link.springer.com/10.1007/3-540-48068-4_1

[51] A. ter Hofstede, W. van der Aalst, Workflow Patterns.
URL http://www.workflowpatterns.com./

[52] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow Patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51 (2003). doi:10.1023/A:1022883727209.
URL http://link.springer.com/10.1023/A:1022883727209

[53] V. epa, Cooperation of Business Processes – A Central Point of the Content, Technical, and Human Aspects of Organization Management, Springer, Berlin, Heidelberg, 2013, pp. 78–90 (2013). doi:10.1007/978-3-642-40823-6_7.
URL http://link.springer.com/10.1007/978-3-642-40823-6_7

[54] A.-W. Scheer, Business Process Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 1994 (1994). doi:10.1007/978-3-642-79142-0.

URL http://link.springer.com/10.1007/978-3-642-79142-0

[55] J. Mendling, M. Nüttgens, EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC), Information Systems and e-Business Management 4 (3) (2006) 245–263 (jul 2006). doi:10.1007/s10257-005-0026-1.
URL http://link.springer.com/10.1007/s10257-005-0026-1

[56] W. M. P. van der Aalst, Process Discovery: An Introduction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 125–156 (2011). doi:10.1007/978-3-642-19345-3_5.
URL https://doi.org/10.1007/978-3-642-19345-3_5