Patterns for Things that Fail

ANTONIO RAMADAS, Faculty of Engineering, University of Porto, Portugal
GIL DOMINGUES, Faculty of Engineering, University of Porto, Portugal
JOAO PEDRO DIAS, INESC TEC, Porto, Portugal
ADEMAR AGUIAR, Departament of Informatics Engineering, Faculty of Engineering, University of Porto, Portugal
HUGO SERENO FERREIRA, INESC TEC, Porto, Portugal and Departament of Informatics Engineering, Faculty of Engineering, University of Porto, Portugal

Internet of Things is a paradigm that empowers the Internet-connected heterogeneous devices alongside with their capabilities to sense the physical world and act on it. Internet of Things has a wide range of application in a variety of areas and contexts, such as *smart spaces* like *smart homes* and *smart cities*. These systems have to deal with device-related problems, such as heterogeneity, fault-tolerance, privacy and security. This paper addresses some patterns about some recurring problems when designing and implementing Internet of Things systems. More concretely, the patterns address how to deal with highly-changeable, error-prone and failing devices and their networks. DEVICE REGISTRY demonstrates how to deal with devices that can change over time. DEVICE RAW DATA COLLECTOR explains how to check the health of heterogeneous devices by a constant collection of raw device data. DEVICE ERROR DATA SUPERVISOR shows how to leverage the continuous data-flow coming from devices to enable error detection and, consequently, processing and handling those errors. Lastly, PREDICTIVE DEVICE MONITOR show how to pro-actively predict devices operation behavior enabling maintenance actions.

Categories and Subject Descriptors: [**Computer systems organization**]: —*Embedded and cyber-physical systems*; [**Software and its engineering**]: —*Design Patterns*

Additional Key Words and Phrases: Internet of Things, Software Architecture, Software Engineering, Design Patterns

1. INTRODUCTION

Internet of Things (IoT) is a novel paradigm with a wide range of applications, pushing towards a worldwide network of interconnected heterogeneous entities. These entities, devices, are capable of contextual awareness, sensing capability, and autonomy [Hossain et al. 2015].

There exists a variety of domains where IoT has to take a fundamental role, including, but not limited to, smart spaces such as smart cities and smart homes [Tan 2010; Korzun et al. 2013]. However, as of today, there is a set of pending research challenges and problems in the scope of IoT, emerging from different components and layers [Mineraud et al. 2016]. IoT generally deals with heterogeneous devices [Miorandi2012], where we can focus on problems such as data throughputs and formats, programming languages, power/battery constraints, memory and computation limitations, network deficiencies, privacy and security [Diaz2016].

Therefore, the underlying technological aspects of IoT systems and the details of their implementations reveals recurring solutions for the inherent problems to the field. These time-tested solutions have been described as design patterns.

Efforts have already been made towards the definition and categorization of patterns [Chandra and Mahindra 2016], though there is still a large number of recurrent solutions implemented in IoT systems which are yet to be documented. Therefore, more in-depth research needs to be done to, on the one hand, reveal more design patterns laying in the IoT-based systems, and, on another hand, document the new and pre-existing patterns following the established pattern template documentation standards.

Initial contributions in the context of developing such design patterns were done by Reinfurt et al. [Reinfurt et al. 2016], establishing five patterns representing some key aspects of the IoT scenario, namely: DEVICE GATEWAY,

DEVICE SHADOW, RULES ENGINE, DEVICE WAKEUP TRIGGER and REMOTE LOCK AND WIPE. Futher work by the same authors focus on device energy supply types and operation modes [Reinfurt et al. 2017].

In Section 2 of this paper, the four patterns will be described in detail each with a figure presenting the sequence diagram. A summary of the patterns here described is shown in Table I.

Table I. Summary of the patterns discovered and described.

Pattern	Problem	Solution
Device Registry	When the number of devices in a network increase it is hard to manually keep a record of all the connected devices and their proprieties.	Build and maintain a registry with all devices in the system which scales and adapts as the devices connected to the system can change.
Device Raw Data Collector	IoT devices are typically producing large quantities of data. The complexity, heterogeneity and distributed architecture inherent to IoT-based systems leads to the creation of large amounts of data lacking a coherent structure, but it is essential to keep this raw data stored.	A collector should be part of the IoT-systems in order to collect all the raw data produced by the system from *device logs* to *network traffic*.
Device Error Data Supervisor	As the number of devices in an IoT system increases, dealing with errors that can appear at different architecture levels lead to a scalability and fault-tolerance problem.	A supervisor should be continuously processing data coming from the devices in the network, handling errors that may appear, triggering countermeasures or broadcasting alerts.
Predictive Device Monitor	IoT-based systems are deployed in a variety of environments and, depending on the criticality of such deployment scenarios, malfunctions on the system can have different levels of nefarious effects.	Using well-known pattern analysis techniques, predictive failure detection can be enabled as a way of detecting when and how a device will fail, improving the health of system through maintenance actions.

In order to identify the patterns here presented, existing products and technologies were analyzed and information about them collected and reviewed [Fehling et al. 2014].

2. INTERNET OF THINGS PATTERNS

In this section, the four patterns mentioned earlier shall be described, as well as the interactions between them. A pattern map of the discovered patterns is shown in Figure 1.

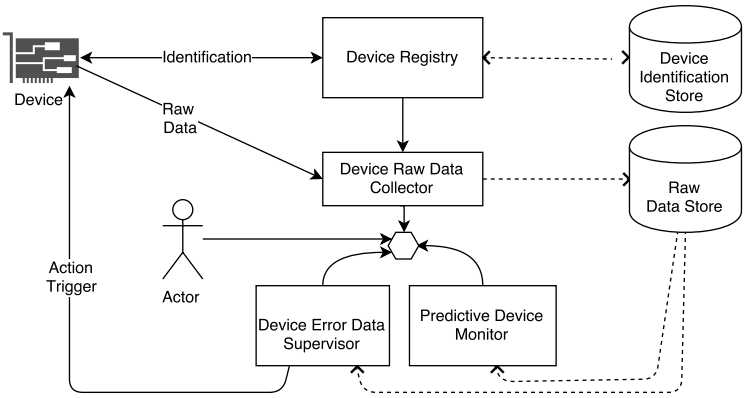

Fig. 1. Pattern mapping of the four different design patterns discovered, including their interactions and relationships.

2.1 Device Registry

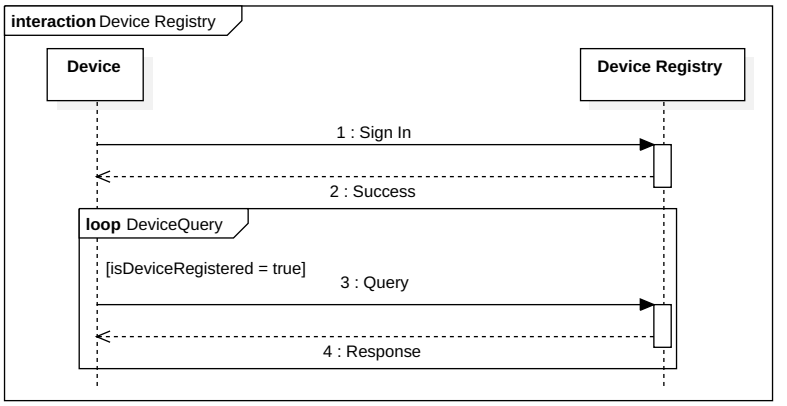

Fig. 2. Sequence Diagram

Aliases. Thing Registry, Device Hub

Context. An IoT implementation mostly consists in a group of devices which form a network. It should be possible to leverage the visibility of a set of devices to provide more complex features (e.g. the introduction devices whose focus is to process data). Nonetheless, as an admin of the network, one is aware that one of the main requirements of a network of IoT devices is its scalability. Therefore, the devices should be Plug and Play [Yang et al. 2015]. Regarding security and control over the network, it should be possible to disconnect the devices from the network if they get hacked, promote the mutual awareness or, in the opposite case, limit the exposure of the devices among each other.

Problem. Having a local registry in each device is not possible simply because the network is too large for an IoT device to keep track of. Moreover, it is also desirable to have an updated registry that every device may rely on.

Forces

Storage Constraint: The network has too many devices to allow an IoT device to have a local registry of all the connected devices, or at least the ones meaningful to it, taking up precious storage space. Having a larger storage per device is undesirable as it increases the hardware costs of the devices.

Uptime: The device's uptime may be intermittent, increasing the odds of missing the join event of a new device in the network. Additionally, it is undesirable to ask the network for the log of events that occurred, as its size may be large. Therefore, the broadcast channels should be free for meaningful communication.

Scalability: The network size is aimed to grow and the initial configuration is subjectable to changes. However, the costs regarding storage and communication channels must be reduced to their minimum.

Mutability: The network is not locked to its initial configuration (new features may be added and old ones removed). Moreover, a device is mutable and its capabilities may support different features during its lifespan. These changes must be propagated accross the network, but without flooding the communication channels.

Consistency: Expanding the network must never leave it in a inconsistent state. Every device must be recognized and its capabilities known so it can be used. However, the devices may be hacked and disrupt the network.

Security: The IoT devices are exposed and constitute the bridge between the attacker and the network. As such, given that the devices have weak capabilities, they are a frequent target for attacks aiming to disrupt the network (e.g. flooding the communication channels or sending erroneous messages directly to other devices). Lastly, breaching one device allows one to snap the structure of all the devices present (e.g. their location and features).

Solution. Have a single module known and reachable by the whole network of components. This module may be queried for the address of a given device or to list a group of devices having a specific set of features. Also, a device joining the network will have to register itself in this module. In order to leave the network, the procedure is the same. The devices may subscribe to certain events (e.g. registration of devices with a specific capability) so they make the best use of the network. The module may request authentication for a device to register itself into the network. Afterwards, all communications may be accomplished using a token to enhance security. The communication between devices is not handled by this module. Also regarding security, a hacked device should be excluded of the network by not acknowledging it when asked and its exclusion should be broadcasted so no device communicates with it. If the contents of the network are sensitive, then each device should only be visible to devices that need to communicate with it. This way, each device has a limited view of the network. Only this module has access to the registry in order not to compromise it. Additionally, this module must be operational and visible at all times so it doesn't limit the network operation. Lastly, the device registry, if replicated through multiple devices, may be placed in a subnetwork which only contains these devices, thus enabling communication through multicast between them. The information the device registry should contain of each device is its identifying name, its type (sensor/actuator), its variant (e.g. light sensor, temperature sensor, motor), its IP and the devices it is allowed to communicate with.

Resulting Context. Scalability: the registration is made on a single module with some metadata regarding the device capabilities. Mutability: The subscrition of certain events empowers the network to change itself at runtime. Furthermore, adding or removing features is as easy as posting the information to the module. Consistency and Atomicity: the registry is located on a single module. Security: a device may only be aware of the rest of the network if registered. Not only that, but the devices may only be aware of the devices allowed by the registry module. Additionally, the module has the power to cut off a hacked device from the network. Responsabilities: there's a clearer separation of concerns and more awareness of the network capabilities. Single point of failure: a single module holding the registry has its advantages, as stated previously, but one unavoidable downside is creating a component which the system heavily relies on to properly function. Attacks: grouping all the information of the devices in the network in a single module increases the desirability of attacks to it in order to retrieve informations of the network.

Related Patterns. SERVICE REGISTRY[Sarma and Bhagavatula 2008]

Examples. Take an IoT system which contains an actuator that controls the opening and closing of window blinds given the average temperature of the environment the system is in. Consider now that a new temperature sensor is added in a room which did not have one already. Using device registry, the actuator mentioned before will automatically be aware of the recently added temperature sensor and request temperature readings from it as well, without the need to be manually reprogrammed to include the knowledge of the new device's location. This pattern is already in use as described by some of the biggest companies such as Amazon Web Services IoT (AWS IoT) [Services 2017] with the *Thing Registry* and in Microsoft Azure IoT [Corporation 2017] with the IoT Hub identity registry.

2.2 Device Raw Data Collector

Aliases. Device State Store

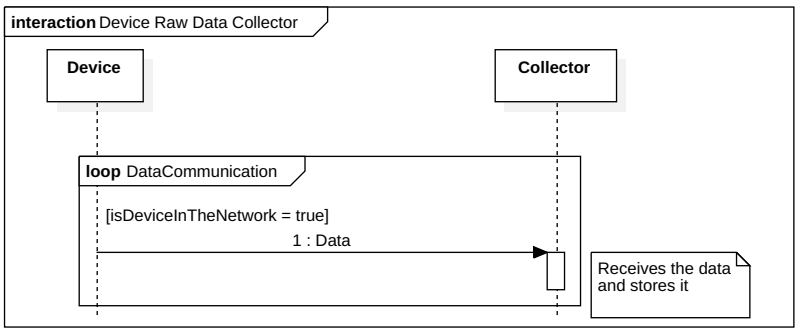

Fig. 3. Sequence Diagram

Context. Both the raw data collected by the devices and the logs they produce can be the source of a wealth of information about the status of the system and the functioning of the devices. This pattern facilitates the compilation of data and logs from the devices in a singular, known location, from where it can later be extracted and further processed.

Problem. Edge devices in IoT systems are constantly reporting data about their functioning. It is therefore important to have an efficient way to store it consistently. The problem arises due to the fact that the devices are spread geographically and the information needs to be sent to a remote location due to the limitation of the devices' storage. That is, there needs to be a way to receive, collect and distribute raw data and logs produced by the devices. It may be desired to attach some labels to the data such as timestamps. Lastly, the delay between the moment the data is generated and its storage is crucial in order to allow real time monitoring by other entities. Nonetheless, ensuring data integrity is also a key point, but it is as important to have security and abstraction layers in order to prevent outsiders from having access to the stored data.

Forces

Channel bandwidth: The amount of information to be sent should be limited by the capabilities of the communication channels of the system.

Edge processment: The edge devices are able to pre-process data and/or control the flow of data emitted, so that it does not flood the network.

Solution. Have a centralized collector server responsible for capturing logs and raw data from every edge device. The collected data can later be used for a variety of purposes such as error detection (DEVICE ERROR DATA SUPERVISOR), system status analysis or extracting information from the data. This collector server may exist locally or be located in the cloud, but its location needs to be known to every edge device (DEVICE REGISTRY). Each sequence of raw data or log captured should be stored by the collector along with a reference of which device it was emitted by. Additional labels may be attached either requested be the device or by the server.

Resulting Context. Facilitates the detection of errors in edge devices. Raw data received from the devices can be used as a source of information for posterior analysis. By having the raw data stored, a more powerful device, or even a dedicated one, can process it. It is possible to identify the device from which sequence piece of data originates, thus making it possible to analyze individual devices, the whole system or parts of it.

The server described will need enough disk space to store a history of the device data and logs.

Related Patterns. RAW DATA [DeLano 1998], CENTRALIZED DATA [DeLano 1998], LOG AGGREGATOR [Brown and Woolf 2016], CENTRALIZED SYSTEM LOGGING [Bijvank et al. 2013]

Examples. Consider an IoT system in which about 100 edge devices are present. With the DEVICE RAW DATA COLLECTOR all the data an logs produced by the devices will be sent and stored in a repository, clearly identifying the device from which each piece of data originated from, and said data can then later be used for a variety of purposes, e.g. error detection.

Already in use by a multiplicity of entities such as syslog-ng[1], Amazon Simple Storage Service[2] and Microsoft Azure Storage[3].

2.3 Device Error Data Supervisor

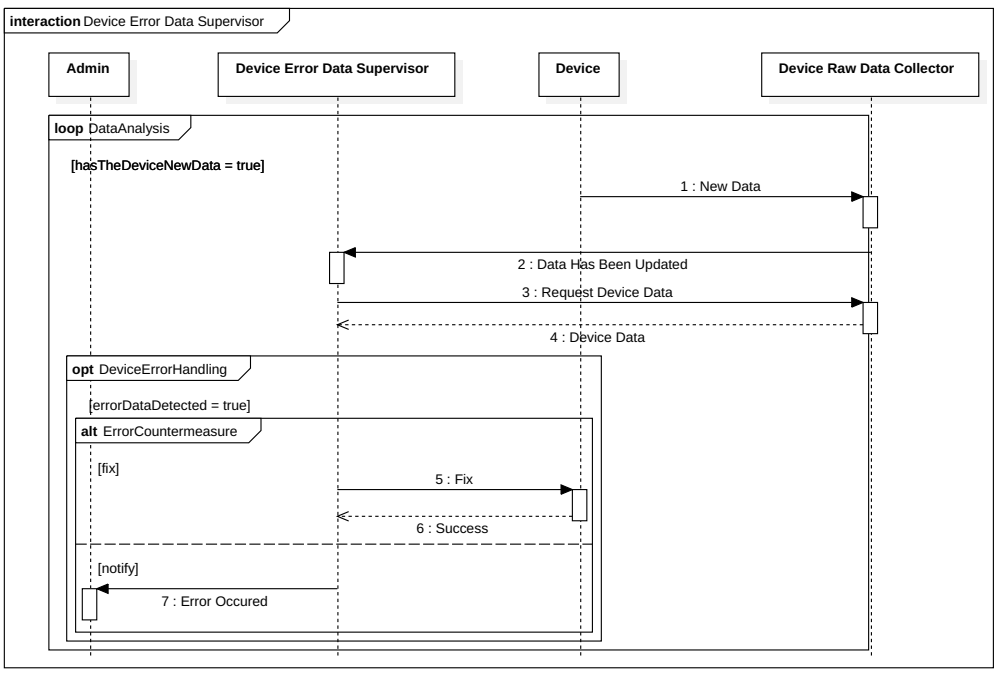

Fig. 4. Sequence Diagram

Aliases. Device Error Processing

Context. IoT-based systems are typically constituted by a large quantity of nodes, highly complex and distributed. This makes the occurrence of errors in edge devices more common due to the inherent heterogeneity as well as the wide-range of deployment scenarios where these nodes can coexist. By collecting information from the data streams associated to the edge systems, such as *logs*, it becomes possible to notice errors that occur in the system. The handling and processing of such errors is fundamental in order to mitigate failures and, consequently, reduce downtimes and other nefarious effects in the deployed IoT systems.

[1] https://syslog-ng.org
[2] https://aws.amazon.com/s3/
[3] https://azure.microsoft.com/en-us/services/storage

Problem. Edge devices in IoT systems are susceptible to errors which can affect the reliability of the data collected by sensors and influence the correct functioning of actuator systems. When the errors reported by the devices themselves or edge-device monitoring systems are not correctly handled and processed this can lead to the appearance of malfunctions in the systems, with consequences depending on the criticality of these systems.

Forces

Heterogeneity: Edge devices have a variety of origins. Thus complexity and diversity are inherent characteristics of IoT-based systems, and error-rates associated with these systems are increased by them.

Reliability: Edge devices are error-prone, since they are susceptible to security breaches, software and/or hardware failures or malfunctions. These errors reduce the reliability on the system due to the possibility of the system entering in malfunction or failure states, which can lead to undesired/unintended behaviors.

Recoverability: When a device enters in a error state there's the need of restoring its default behavior. This becomes even more relevant if the device is placed in remote or limited-access locations.

Impact: Depending on the deployment environment and its criticality, errors can produce different ranges of nefarious effects.

Solution. Since devices themselves, or even monitoring systems, are capable of reporting information about their current status and activities, and such data is collected, it is possible to continuously analyze and process it in order to detect explicit or implicit errors on the data. Whenever an error occurs some actions must be taken, as countermeasures, to mitigate or reduce its impact on the IoT system.

Resulting Context. To be able to offer DEVICE ERROR DATA SUPERVISOR functionalities, the edge devices must be continuously giving feedback about their current functioning status. Such information can be given by the devices themselves or by some third-party monitoring mechanism, and can be accomplished by implementing the DEVICE RAW DATA COLLECTOR. By analyzing the continuous flow of data we can detect and process errors, enabling its correct handling and activating warning mechanisms. Therefore, we can consider certain benefits on the error data handling, namely, by activating available countermeasures in order to mitigate or reduce the impact of device failures in IoT-based systems.

So, DEVICE ERROR DATA SUPERVISOR comes as a way of increasing reliability in the highly heterogeneous IoT systems, providing the possibility of enabling recoverability mechanisms in case of error detection thus enabling the reduction of the impact that malfunction can have on deployed systems. Moreover, the correct error handling in IoT systems can lead to the reduction of maintenance costs and efforts.

Notwithstanding, the error data handling can have some drawbacks, such as the increase of response time due to more extensive data processing. Such problem can appear specially when dealing within resource-constrained situations. Furthermore, some false-positive errors can appear in the data and countermeasures or warnings may be activated unnecessarily, which can have more or less of an impact depending on the situation.

Related Patterns. DEVICE RAW DATA COLLECTOR, PREDICTIVE DEVICE MONITOR, RULES ENGINE [Reinfurt et al. 2016], SERVICE MONITOR [Sarma and Bhagavatula 2008]

Examples. IoT dashboards such as Grafana[4] includes functionalities of device monitoring and error data handling, called *Alerting Engine*, which work by continuously analyzing the time-series data flow, verifying defined threshold values or incoherent readings, thus emitting warning messages or triggering countermeasure actions when configured to. Other approaches like the AWS CloudWatch Alarms let one define alarms that watch a single metric over a timespan and performs one or more actions based on the read value relatively to a given threshold.

[4] https://grafana.com

Furthermore, IBM allows the processing and handling of device errors by integrating devices with the IBM Watson IoT Platform[5] and Node-RED[6], followed by alert broadcasting using the IBM Alert Notification Service.

2.4 Predictive Device Monitor

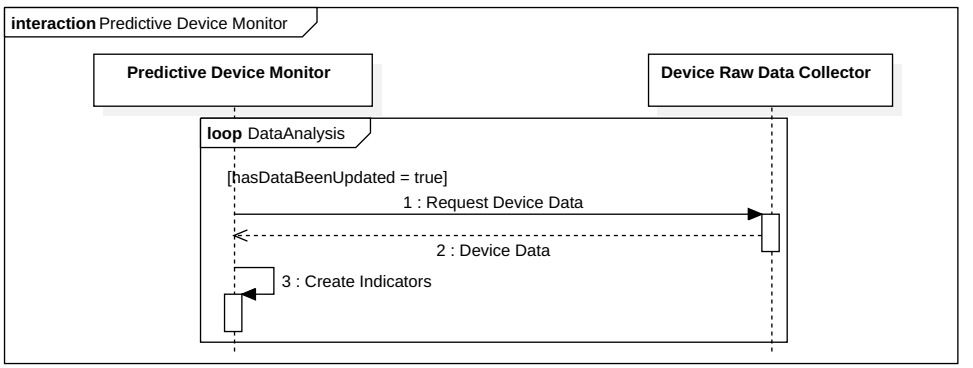

Fig. 5. Sequence Diagram

Aliases. Predictive Device Watcher

Context. Device maintenance in IoT systems can be considerably time costly when malfunctioning. The possible geographical dispersion and difficult access to the edge components can impact negatively on the time required to repair said components. Therefore it would be advantageous to efficiently and preemptively estimate the *Remaining Useful Life* of an edge component.

Problem. Maintenance on edge devices is an important operation for resource reliability endurance over time. Whether it be security vulnerabilities, software malfunction or hardware malfunction, such problems are usually not detected until after they occur, which results in a longer unexpected downtime. In that sense, it would be extremely advantageous for the management of an IoT system to become aware of a possible issue as soon as possible so that actions can be taken to mitigate it.

Forces

Maintenance Scheduling: Repairs should be performed at the appropriate time, as to prevent malfunctions of devices instead of simply correcting them after they occur, as well as to prevent downtime in a perfectly functional device, avoiding the misuse of a time and resource costly action.

Downtime: Device malfunctions or failures can lead to increasing costs and disruptions on normal system function.

Reliability: Edge devices are error-prone and this errors reduce the reliability on the system due to the possibility of the system entering in malfunction or failure states, which can be mitigate with *a priori* failure detection.

[5] https://www.ibm.com/us-en/marketplace/internet-of-things-cloud
[6] https://nodered.org

Solution. Based on the RAW DEVICE DATA COLLECTOR, through the use of monitorization operations like the ones provided by the algorithms of machine learning and data mining, analyze the data in such a way that allows the detection of abnormal entries in the data records associated with the devices. These kind of indicators should be known states or data samples that have a high likelihood of preceding a failure, so the fault prediction will learn the state that precedes an error with every error that occurs in the system. Upon detection of these indicators, the system must be able to warn the user of the likelihood of a malfunction, as well as have a schedule of predictable device failure.

Resulting Context. It becomes possible to avoid errors instead of simply repairing them after their occurrence. Unexpected downtime due to hardware or software faults is decreased.

The location where the data used for the prediction is stored as well as the component generating that prediction are centralized. The use of machine learning algorithms may produce false results initially, either positive or negative, before the patterns leading to those errors are learned by the model.

Variants. PREVENTIVE DEVICE MAINTENANCE: Shares the same problematic and forces behind the described pattern, but instead of trying to predict when a device will require maintenance actions, it consists on scheduling preventive maintenances in order to reduce the probability of device failure. The major drawback of such approach is that maintenance operations can occur without being needed, increasing costs.

Related Patterns. DEVICE RAW DATA MONITORING, ANYCORRECTIVEACTION STABLE [Singh and Fayad 2010]

Examples. Consider an IoT system which contains a device a temperature sensor known to be prone to issues from time to time which required substitution of the sensor. With this pattern, using a stream of data from the device it would be possible to predict when the device would be requiring its next repair. This pattern can be found in IBM IoT Asset Management Solution [IBM 2016].

Acknowledgments

This work was supported by Project “NanoSTIMA: Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring and Analytics/NORTE-01-0145-FEDER-000016" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

REFERENCES

Roland Bijvank, Wiebe Wiersema, and Christian Köppe. 2013. Software Architecture Patterns for System Administration Support. In *Proceedings of the 20th Conference on Pattern Languages of Programs (PLoP '13)*. The Hillside Group, USA, Article 1, 14 pages. http://dl.acm.org/citation.cfm?id=2725669.2725671

Kyle Brown and Bobby Woolf. 2016. Implementation Patterns for Microservices Architectures. (2016).

Gullena Satish Chandra and Tech Mahindra. 2016. Pattern language for IoT applications. (2016).

Microsoft Corporation. 2017. Understand identity registry in your IoT hub | Microsoft Docs. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry. (2017). Accessed: 2017-05-28.

David E DeLano. 1998. Telephony Data Handling Pattern Language. In *Pattern Languages of Program Design '98*, Vol. 53.

Christoph Fehling, Johanna Barzen, Uwe Breitenbücher, and Frank Leymann. 2014. A Process for Pattern Identification, Authoring, and Application. , Article 4 (2014), 9 pages. DOI:http://dx.doi.org/10.1145/2721956.2721976

Md. Mahmud Hossain, Maziar Fotouhi, and Ragib Hasan. 2015. Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things. *2015 IEEE World Congress on Services* (2015), 21–28. DOI:http://dx.doi.org/10.1109/SERVICES.2015.12

IBM. 2016. IBM IoT Services Asset Manager. https://www.ibm.com/blogs/internet-of-things/wp-content/uploads/2016/05/IG_preventiveMaintenance15_ss.pdf. (2016). Accessed: 2017-05-28.

Dmitry G Korzun, Sergey I Balandin, and Andrei V Gurtov. 2013. Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges. 48–59. DOI:http://dx.doi.org/10.1007/978-3-642-40316-3_5

Julien Mineraud, Oleksiy Mazhelis, Xiang Su, and Sasu Tarkoma. 2016. A gap analysis of Internet-of-Things platforms. *Computer Communications* 89-90 (2016), 5–16. DOI:http://dx.doi.org/10.1016/j.comcom.2016.03.015

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2016. Internet of things patterns. *Proceedings of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16* (2016), 1–21. DOI:http://dx.doi.org/10.1145/3011784.3011789

Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2017. Internet of Things Patterns for Devices. (2017), 117–126.

N. Vinod Sarma and Srinivas Rao Bhagavatula. 2008. Freeway Patterns for SOA Systems. In *Proceedings of the 15th Conference on Pattern Languages of Programs (PLoP '08)*. ACM, New York, NY, USA, Article 6, 10 pages. DOI:http://dx.doi.org/10.1145/1753196.1753204

Amazon Web Services. 2017. Managing Things with the Thing Registry - AWS IoT. http://docs.aws.amazon.com/iot/latest/developerguide/thing-registry.html. (2017). Accessed: 2017-05-28.

Shivanshu K Singh and Mohamed E Fayad. 2010. The AnyCorrectiveAction stable design pattern. In *Proceedings of the 17th Conference on Pattern Languages of Programs*. ACM, 24.

Lu Tan. 2010. Future internet: The Internet of Things. *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)* (2010), V5–376–V5–380. DOI:http://dx.doi.org/10.1109/ICACTE.2010.5579543

Fan Yang, Nelson Matthys, Rafael Bachiller, Sam Michiels, Wouter Joosen, and Danny Hughes. 2015. uPnP: Plug and Play Peripherals for the Internet of Things. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. ACM, New York, NY, USA, Article 25, 14 pages. DOI:http://dx.doi.org/10.1145/2741948.2741980