Microsystem Technologies
https://doi.org/10.1007/s00542-019-04631-9

TECHNICAL PAPER

Dynamic sampling rate algorithm (DSRA) implemented in self-adaptive software architecture: a way to reduce the energy consumption of wireless sensors through event-based sampling

Hatem Algabroun[1]

Received: 9 September 2019 / Accepted: 14 September 2019

Abstract
With the recent digitalization trends in the industry, wireless sensors are, in particular, gaining a growing interest. This is due to the possibility of being installed in inaccessible locations for wired sensors. Although great success has already been achieved in this area, energy limitation remains a major obstacle for further advances. As such, it is important to optimize the sampling with a sufficient rate to catch important information without excessive energy consumption, and one way to achieve sufficient sampling is using adaptive sampling for sensors. As software plays an important role in the techniques of adaptive sampling, a reference framework for software architecture is important in order to facilitate their design, modeling, and implementation. This study proposes a software architecture, named Rainbow, as the reference architecture, also, it develops an algorithm for adaptive sampling. The algorithm was implemented in the Rainbow architecture and tested using two datasets; the results show the proper operation of the architecture as well as the algorithm. In conclusion, the Rainbow software architecture has the potential to be used as a framework for adaptive sampling algorithms, and the developed algorithm allows adaptive sampling based on the changes in the signal.

1 Introduction

In the maintenance industry, condition monitoring can be defined as a technique used to monitor physical variables (e.g., temperature, vibration, and pressure) to draw conclusions about the condition of an asset being monitored. Condition monitoring techniques are used as a data source in condition-based maintenance to provide an early warning to plan and trigger cost-effective maintenance actions (Owen et al. 2009; Al-Najjar 2012). With the recent digitalization trends in industry, wireless sensors are, in particular, gaining a growing interest. This is due to the possibility of installing them in inaccessible locations for wired sensors, such as in embedded systems and rotating machines. This avoids the burden of unreliable electrical connections and wiring expenses (Owen et al. 2009).

Although great success has already been achieved in this area, energy limitation remains a major obstacle for further advances (Alippi and Anastasi 2010; Zhang et al. 2013). For this reason, energy harvesting for wireless sensors received considerable attention from several researchers (Owen et al. 2009; Bogue 2010, 2015). However, harvested energy sources are typically unstable and vary with time, weather, or the season. As such, managing energy consumption wisely is still desirable (Yan et al. 2012; Zhang et al. 2013).

Generally, it is assumed that the energy consumption for sensing is relatively far lower than it is for transmitting and receiving. However, in practice, this is not always the case. There are so-called "hungry sensors" (e.g., a gas sensor) whose energy consumption is relatively high compared to that of data transmitting and receiving (Alippi and Anastasi 2010; Shu et al. 2017). In such applications, efficient sampling management is a very important issue as reducing data sampling means, in many cases, reducing data acquisition, including computations and data transmission.

For wireless sensors, there is usually a trade-off between monitoring quality and energy consumption. This is because with the increase in collected data comes increased monitoring quality and increased energy consumption. However, if only a little data is collected, there is a lack of information, which eventually, will come with the cost of

✉ Hatem Algabroun
hatem.algabroun@lnu.se

[1] Mechanical Engineering Department, Faculty of Technology, Linnaeus University, Vaxjo, Sweden

poor monitoring quality and consequent bad decisions (Lu et al. 2017). Therefore, it is important to optimize the sampling with a sufficient rate to catch important information cost-effectively and without excessive energy consumption. One way to achieve sufficient sampling is to adapt the sampling rate dynamically to the events that occur in the physical phenomenon being monitored.

Several studies investigated adaptive sampling. For instance, in Shu et al. (2017), an algorithm for adaptive sampling was developed and tested for water quality. Two key parameters were used to evaluate the algorithm, dissolved oxygen (DO) and turbidity. Normalized mean error was used to evaluate the performance of those predetermined parameters. In comparison with fixed intervals, the study showed that battery life could be increased to 30.66% within 3 months of monitoring. In Alippi and Anastasi (2010), an adaptive sampling algorithm was proposed that estimates the optimal sampling rate online. The proposed algorithm determines the sampling rate by using the fast fourier transform algorithm and compares it with the maximum frequency. The study showed that the algorithm could reduce the number of samples up to 79% compared with the traditional fixed-rate approach. Yan et al. (2012) suggested an adaptive sampling algorithm that adapts the sampling mode to the present energy state. The test was made using a CO_2 sensor, and the results showed that the sensor sampling mode was changing based on the changes in the energy state. More detailed surveys in this domain can be found in Rault et al. (2014) as well as Khan et al. (2015).

In general, an adaptive sampling technique is computational-dependent and, therefore, software plays a significant role in it. For this reason, a reference framework of software architecture is important to facilitate adaptive sampling design, implementation and modeling. Despite its importance, there is a lack of studies on developing a reference framework for adaptive sampling models. From the software engineering field, the architectural approach appears to be suitable for this task. It provides the right level of abstraction and generality to construct a knowledge representation scheme (Oreizy et al. 1998; Garlan et al. 2004; Kramer and Magee 2007; Weyns and Iftikhar 2016; Algabroun 2017). Garlan et al. (2004) proposed a self-adaptive based architecture, named Rainbow, that can be used as a reference for systems that are self-adaptable to uncertainties at the runtime. In this paper, we argue that this reference architecture is suitable for adaptive sampling applications.

The contribution of this paper is twofold: first, proposing a reference architecture, that is, Rainbow, for adaptive sampling algorithms, and second, developing a parametric algorithm, that is, dynamic sampling rate algorithm (DSRA), that dynamically adapts the sampling rate based on the events in the signal. Hence, the aims of this study are as follows: (1) to establish a reference framework for adaptive sampling algorithms, (2) to develop an adaptive sampling algorithm, and (3) to implement and test the proposed reference framework using the developed algorithm.

The remainder of the paper will be as follows: the next section will discuss the self-adaptive software architecture; Sect. 3 will explain the development of the algorithm, which will then be implemented in the proposed software architecture; in Sect. 4 the algorithm will be tested using two datasets; Sect. 5 will discuss the results; and in Sect. 6 conclusions will be drawn.

2 Self-adaptive software architecture

Self-adaptive software architecture is designed to endow a system with autonomous adaptation to dynamics at runtime (Kephart and Chess 2003; Gil De La Iglesia and Weyns 2015). Typically, this approach contains a managed system and a feedback loop, that is, a managing system. The managed system is concerned with the domain's functionality, while the managing system is concerned with the adaptation of the managed system (Oreizy et al. 1998; Kramer and Magee 2007).

This approach has several advantages (Algabroun 2017):

- It enables the design of a system that is autonomously adaptable at runtime.
- It is based on the principle of separation of concerns, where each component is assigned a distinct function. This facilitates repair, modification, and development.
- It allows the abstract design of a system that covers different domains.
- The abstraction provides a holistic view of a system exposing its system-level properties (Garlan et al. 2004).
- It is well supported by modeling languages and notations to express knowledge such as stitch (Cheng and Garlan 2012) and automata (Weyns et al. 2012).
- It allows treating a component as a black box, thereby increasing the possibility of its reusability (Oreizy et al. 1998).

Several frameworks have been proposed for self-adaptive software architectures (Oreizy et al. 1998; Kephart and Chess 2003; Garlan et al. 2004; Kramer and Magee 2007). Some studies, such as IBM's MAPE-K (monitor-analyze-plan-execute-knowledge) (Kephart and Chess 2003), considered developing a framework that guides the self-adaptation mechanism. Kramer and Magee (2007) considered three different hierarchical layers, which named component control, change management, and goal management. Others, such as Oreizy et al. (1998) and Garlan et al. (2004), provided a framework that enables assessment of the adaptation decision and allows runtime configuration. All of these frameworks aimed to provide leeway for the engineer to architect a software system. As such, it is nearly possible to reach rather a similar objective using any of them (Weyns et al. 2012). Despite this, the Rainbow framework appears to be the most suitable for the problem addressed in this paper, as it contains the necessary elements for conducting an adaptive sampling operation. These elements are collect data, detect changes in the condition, evaluate changes, decide the adaptation regime (i.e., the sampling rate), and execute the regime. Therefore, in this study we considered Rainbow architecture (Garlan et al. 2004). The next section will describe the Rainbow architecture in more detail.

2.1 Rainbow framework

A framework refers to a structure representing a knowledge scheme, which serves as a template or guideline to construct a specific model (Guillén et al. 2016; Algabroun 2017). The Rainbow framework (Garlan et al. 2004) is a reusable software architecture for engineering systems that are self-adaptive at runtime. It aims to minimize the design efforts by providing a customizable framework that can be employed in different domains (Cámara et al. 2016).

The adaptation process of the Rainbow framework is as follows (see Fig. 1): the data of the managed system (in the system layer) is collected by the Probes component, aggregated by the Gauges component, and then used to update a model of the managed system that is maintained by the Model manager component. This is done through the Translation infrastructure layer that enables the communication by mapping the managed system in the system layer to the managing system in the architecture layer. Then the Constraints evaluator component evaluates if specific constraints were violated in the model if so, it triggers the Adaptation engine component. The latter component then selects an adaptation regime, and the Adaptation executor component executes it through Effectors at running time.

The Rainbow framework employs the notation architectural style to distinguish system-specific knowledge from commonalities between systems. This facilitates customization by determining four customization points (Cámara et al. 2016):

- The model of the managed system in the system layer, which is handled by Model manager.
- The constraints, which are handled by Constraint evaluator.
- The adaptation regime, which is handled by Adaptation engine.
- The configuration of Effector, which is handled by Adaptation executor.

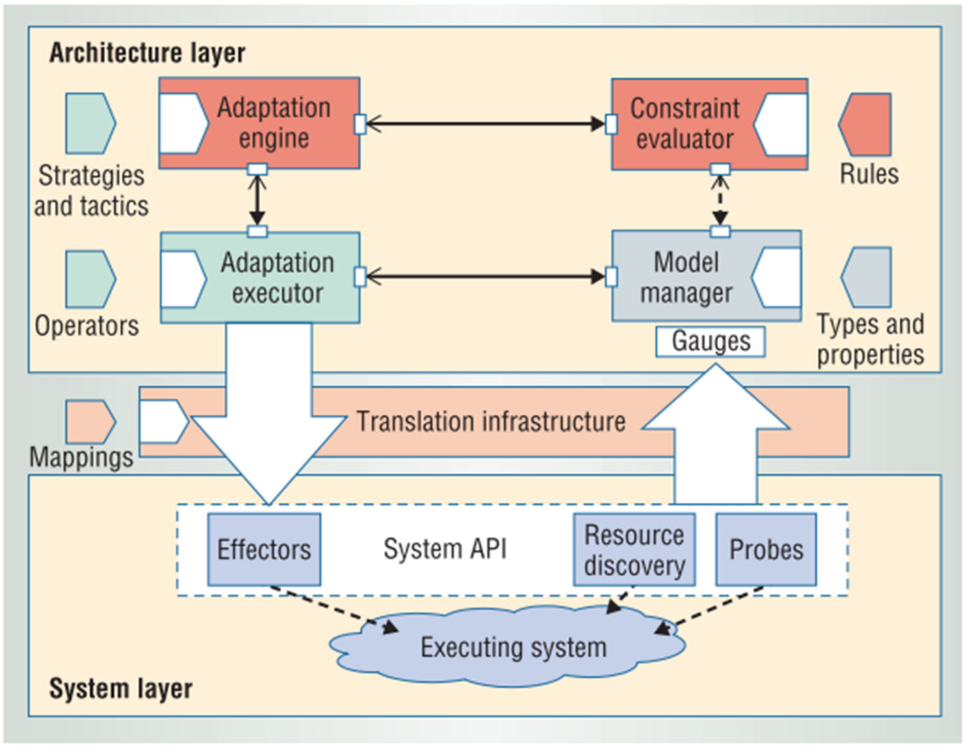

Fig. 1 The Rainbow framework (Garlan et al. 2004)

3 Algorithm development and the implementation in Rainbow

Sufficient sampling is very important, as once the obtained signal is distorted from insufficient sampling, it will be impossible to retrieve the original signal anymore. In a sampled signal, the pattern between the sampling points is unknown. From a practical point of view, the signal is usually assumed to have a smooth transition between these points (Cohen 2019).

The algorithm developed in this section is based on a realistic assumption to capture the signal changes; that is, we need more data when a change in the monitored phenomenon occurs. So, the sampling rate should be dependent on the change rate in the monitored signal. This motivates the need to consider the slope between two consecutive points. The more slope there is, the more change that occurred and the more following-up there should be; consequently, the higher sampling rate should be implemented. Based on this assumption, this algorithm is dealing with two main parts, that is, identifying the slope and determining the sampling rate.

3.1 Algorithm development

To realize this data driven algorithm, when the monitored signal is relatively static and no considerable variation is assumed, it is reasonable to reduce the sampling rate. This is to reduce energy consumption since much of the data could be redundant.

The value of the sampling rate in the case of redundancy should be predetermined first. This value of the sampling rate serves as a means to explore if a change is present in the signal. This value is then subtracted from the slope of the signal trend to obtain the time between measurements (TBM). In the case where the signal has a high variation, for example, vibration, a complementary technique has to be used, such as averaging and cumulative sum (CUSUM) (Al-Najjar 2016). TBM could be mathematically defined as follows:

$$TBM = E - \left(S * \left| \frac{(y_i - y_{i-1})}{(M_i - M_{i-1})} \right| \right), \ (i \in 1, 2, 3, \ldots n) \qquad (1)$$

where E is the exploratory value, that is, the sampling rate in the normal case, which is meant to explore the presence of changes and S is a tuning constant that determines the sensitivity of the changes. It is desirable to utilize knowledge-expert and/or historical data to predetermine a proper value for those parameters, that is, E and S; y_i and y_{i-1} denote the current and previous sensor values, respectively and M_i and M_{i-1} denote the current and previous time of measurements, respectively.

The next measurement (M_{i+1}) could be then defined as follows:

$$M_{i+1} = M_i + TBM, \ (i \in 1, 2, 3, \ldots n) \qquad (2)$$

3.2 Algorithm implementation in the Rainbow framework

The adaptive process of the Rainbow framework described in Sect. 2.1 can be considered as a generic pattern of the adaptive sampling process. As such, this process can be mapped to the generic components of the Rainbow framework; each particular component will be assigned a related function as follows:

- Gauges: to collect, preprocess, and update the data, for example, implementing aggregation and averaging.
- Model manager: to handle and run the model being employed.
- Constraints evaluator: to evaluate if there is a violation of specific constraints to detect changes.
- Adaptation engine: to decide the proper sampling regime.
- Adaptation executor: triggering Probes to collect the data based on the adaptive plan decided by Adaptation engine.

A pseudo-code for implementing the developed algorithm in the Rainbow framework is represented below in Algorithm 1.

Algorithm 1. DSRA

```
1:  Initialization
2:    set the sampling rate at the normal case $E$;
3:    set the tuning factor $S$;
4:    set $C$ (Constraint);
5:    set time;
6:    Flag = False
7:  end initialization;
8:  while (True) do:
9:      Gauges
10:         get data from Probes;
11:         preprocess the collected data;
12:         update repository;
13:     Model Manager
14:         find Change Significance (i.e. $CS = S * \left|\frac{(y_i - y_{i-1})}{(M_i - M_{i-1})}\right|$ );
15:     Constraints Evaluator
16:         If ($CS \geq C$);
17:             Flag = True
18:         else;
19:             $TBM = E$
20:             $M_{i+1} = M_i + TBM$
21:      Adaptation Engine:
22:          If (Flag = True);
23:             find $TBM$ (i.e. $TBM = E - CS$);
24:             decide $M_{i+1}$ (i.e. $M_{i+1} = M_i + TBM$ );
25:             Flag = False
26:     Adaptation Executor
27:         while (time < $\boldsymbol{M}_{i+1}$)
28:             wait;
29:         end while;
```

4 Testing DSRA-based Rainbow architecture

Two datasets were created and used as examples to test the DSRA algorithm. The first example is intended to visualize the behavior of the algorithm; the second example is intended to show the approximation that could be achieved in a more realistic dataset.

4.1 Example 1

As a proof of concept, a dataset was created with different slope values to visualize the behavior of the algorithm implemented in the Rainbow architecture. These values are meant to represent different states of the physical phenomenon being monitored; the values of the slopes are 0, 1, and 2.

DSRA was implemented in the Rainbow architecture and programmed using Python 3.7.3 and open-source libraries Numpy and Matplotlib. After certain iterations using different values for E and S, the values of 5 and 2 for E and S respectively were selected as it allows intuitive visualization of the algorithm's behavior (see Fig. 2).

4.2 Example 2

A dataset of a signal with variations in its amplitude was created, and the algorithm was implemented on this dataset. After several iterations with different E and S parameters, values of 12 and 15 were selected. These tuning parameters were selected as in this case, the reconstructed signal from the algorithm appeared to have a lower number of samples and a higher approximation to the original signal (see Fig. 3).

The mean absolute percentage error (MAPE) was calculated for the two signals (i.e., the reconstructed signal from DSRA and the test signal) as a mean for pattern similarity and it was 0.326% for 300 points. MAPE was employed due to its popularity and intuitive interpretability as an absolute percentage error (Kim and Kim 2016).

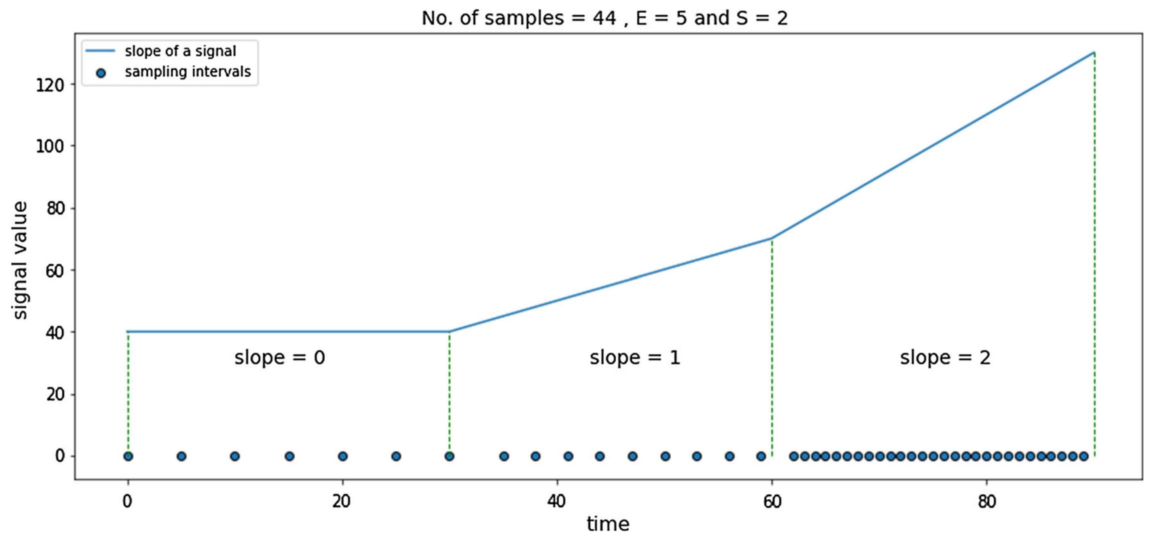

Fig. 2 DSRA sampling rate over different slopes

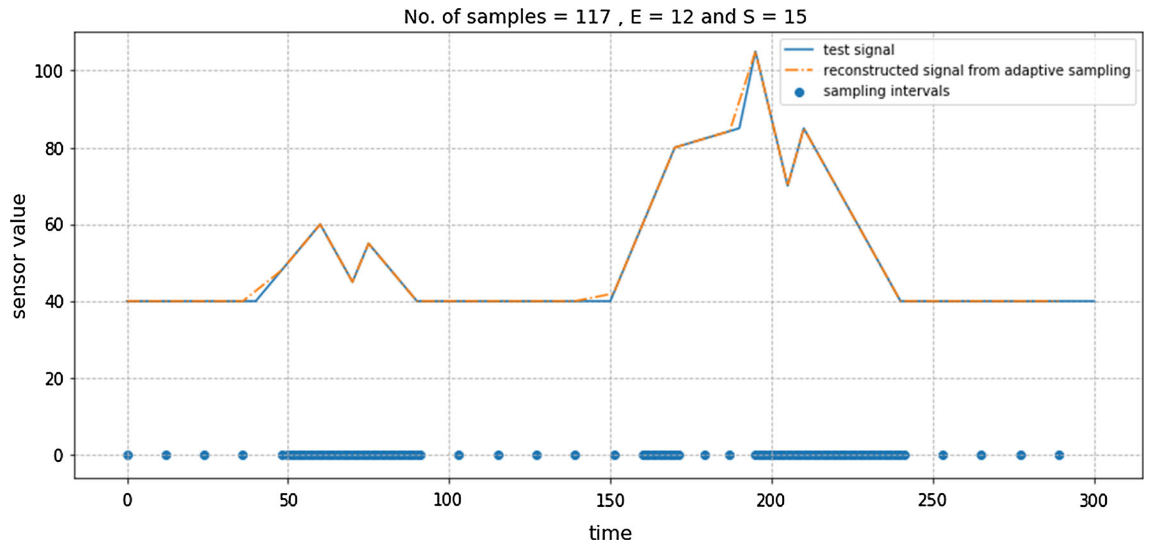

Fig. 3 Implementing DSRA on a dataset

5 Discussion

Based on the algorithm implementation and the results of the examples, the Rainbow software architecture provided a knowledge scheme for adaptive sampling to follow and use. It follows systematic steps to perform adaptive sampling that are: data collection, detection of a change in the signal being monitored, evaluation of the change significance, selection of the adaptation strategy, and eventually, execution of the selected strategy. There was no conflict or illogical process in the flow of the Rainbow software architecture to be reported.

Figure 2 shows that the algorithm sampling increases when the slope increases. When the slope value is 0, the total number of samples is $\sim$ 7. When the slope value is 1, the total number of samples is 9, and when the slope is 2, the total number of samples is 28. This is when E and S are 5 and 2, respectively.

In Fig. 3, when there is a redundancy in the signal (i.e., the amplitude value of the signal is stable, and there is no variation in its value), fewer samples were collected. When there were variations in the test signal, higher sampling was exhibited to capture the changes in the signal. The reconstructed signal from DSRA appears to have a pattern close to that of the test signal with MAPE = 0.326% for 300 points. This is when E and S are 12 and 15, respectively. It is crucial to assign a proper value for E and S to achieve efficient sampling with a good approximation to the signal being monitored. The optimization strategies and techniques for tuning E and S are out of the scope of this study and will be the subject of future work.

This approach might not be suitable for highly fluctuating signals (e.g., high noise signals, vibration signals), and in this case, a further process for this signal is required (e.g., averaging, CUSUM). However, further research is still required to investigate its performance on such signals.

6 Conclusion and future work

In order to overcome the energy limitation in wireless sensor, it is important to optimize sensor sampling to catch only important information without excessive samplings that cause unnecessary energy consumption. A way to achieve this is by adapting the sampling rate to the events that occur in the signal being monitored.

As software plays an important role in the techniques of adaptive sampling, a reference framework for software architecture is important in order to facilitate their design, modeling, and implementation.

The software reference framework for adaptive sampling should allow one to address the following three key questions: is there a change in the signal, is it considerable change, and what adaptation strategy should be conducted? The Rainbow software architecture appears to have a mechanism that answers those questions, that is, after collecting data, Model manager component detects the change in the signal, Constraint evaluator component determines the relevance of the change, and Adaptation engine component selects the suitable adaptation strategies that are then executed by Adaptation executor component. It is believed that the proposed architecture has the general steps of adaptive sampling and, as such, can be generalized and used as a reference software architecture for a self-adaptive sampling algorithm.

An algorithm for adaptive sampling, namely DSRA, was developed in this study and implemented in Rainbow architecture. As proof of concept, this algorithm was implemented in two examples, and the results prove that the algorithm has adaptive sampling based on the changes in the signal. However, this algorithm might not be suitable for signals with high fluctuation, in such a case, further signal processing, such as averaging and CUSUM, might be necessary. The applicability of DSRA on such a signal still needs to be investigated. Future work could include investigating optimization strategies for E and S parameters and techniques to include multiple sensor signals in the algorithm, as well as examining the Rainbow software architecture with other sampling algorithms.

Acknowledgements Open access funding provided by Linnaeus University. This research was supported by PreCoM-H2020, FoF 09, 2017-202. PreCoM has received funding from European Union's Horizon 2020 research and innovation program under grant agreement No. 768575.

References

Algabroun H et al (2017) Maintenance 4.0 framework using self-adaptive software architecture. In: Proceedings of 2nd International Conference on Maintenance Engineering, IncoME-II 2017, University of Manchester, UK, pp 1–11

Alippi C, Anastasi G (2010) An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors. IEEE Trans Instrum Meas 59:335–344. https://doi.org/10.1109/TIM.2009.2023818

Al-Najjar B (2012) On establishing cost-effective condition-based maintenance. J Qual Maint Eng 18:401–416. https://doi.org/10.1108/13552511211281561

Al-Najjar B (2016) Determination of potential failure initiation time using cumulative sum chart. IFAC-PapersOnLine 49:43–48. https://doi.org/10.1016/J.IFACOL.2016.11.008

Bogue R (2010) Powering tomorrow's sensor: a review of technologies—part 2. Sens Rev 30:271–275. https://doi.org/10.1108/02602281011072125

Bogue R (2015) Energy harvesting: a review of recent developments. Sens Rev 35:1–5. https://doi.org/10.1108/SR-05-2014-652

Cámara J et al (2016) Incorporating architecture-based self-adaptation into an adaptive industrial software system. J Syst Softw 122:507–523. https://doi.org/10.1016/j.jss.2015.09.021

Cheng SW, Garlan D (2012) Stitch: a language for architecture-based self-adaptation. J Syst Softw 85:2860–2875. https://doi.org/10.1016/j.jss.2012.02.060

Cohen M (2019) Understand the Fourier transform and its applications. Udemy, Inc. https://www.udemy.com/fourier-transform-mxc/. Accessed 25 Feb 2019

Garlan D et al (2004) Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer 37(10):46–54

Gil De La Iglesia D, Weyns D (2015) MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. ACM Trans Auton Adapt Syst 10:1–31. https://doi.org/10.1145/2724719

Guillén AJ et al (2016) A framework for effective management of condition based maintenance programs in the context of industrial development of e-maintenance strategies. Comput Ind 82:170–185. https://doi.org/10.1016/j.compind.2016.07.003

Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36:41–50. https://doi.org/10.1109/MC.2003.1160055

Khan JA et al (2015) Energy management in wireless sensor networks: a survey. Comput Electr Eng 41:159–176. https://doi.org/10.1016/j.compeleceng.2014.06.009

Kim S, Kim H (2016) A new metric of absolute percentage error for intermittent demand forecasts. Int J Forecast 32:669–679. https://doi.org/10.1016/j.ijforecast.2015.12.003

Kramer J and Magee J (2007) Self-managed systems: an architectural challenge. In: 2007 Future of Software Engineering. IEEE Computer Society. pp 259–268. https://doi.org/10.1109/fose.2007.19

Lu T et al (2017) Distributed sampling rate allocation for data quality maximization in rechargeable sensor networks. J Netw Comput Appl 80:1–9. https://doi.org/10.1016/j.jnca.2016.12.021

Oreizy P, Medvidovic N, Taylor RNRN (1998) Architecture-based runtime software evolution. Proc ICSE 1:2. https://doi.org/10.1109/icse.1998.671114

Owen TH et al (2009) Self powered wireless sensors for condition monitoring applications. Sens Rev 29:38–43. https://doi.org/10.1108/02602280910926742

Rault T, Bouabdallah A, Challal Y (2014) Energy efficiency in wireless sensor networks: a top-down survey. Comput Netw 67:104–122. https://doi.org/10.1016/j.comnet.2014.03.027

Shu T et al (2017) An energy efficient adaptive sampling algorithm in a sensor network for automated water quality monitoring. Sensors 17:2551. https://doi.org/10.3390/s17112551

Weyns D and Iftikhar MU (2016) Model-based simulation at runtime for self-adaptive systems. In: 2016 IEEE International Conference on Autonomic Computing (ICAC). 14–18 March, Suita, Japan IEEE, pp 364–373. https://doi.org/10.1109/icac.2016.67

Weyns D, Malek S, Andersson J (2012) FORMS: unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans Auton Adapt Syst 7:8. https://doi.org/10.1145/2168260.2168268

Yan J et al (2012) Local adaptive sampling for wireless sensor network powered by energy harvesting. Optik 123:2195–2197. https://doi.org/10.1016/j.ijleo.2011.11.011

Zhang Y et al (2013) Distributed sampling rate control for rechargeable sensor nodes with limited battery capacity. IEEE Trans Wirel Commun 12:3096–3106. https://doi.org/10.1109/TCOMM.2013.050613.121698