Analyzing WSN-based IoT Systems using MDE Techniques and Petri-net Models*

Burak Karaduman[1], Moharram Challenger[2], Raheleh Eslampanah[2], Joachim Denil[2], and Hans Vangheluwe[2]

[1] International Computer Institute, Ege University, Turkey,
{bburakkaraduman}@gmail.com
[2] University of Antwerp and Flanders Make, Belgium
{moharram.challenger,raheleh.eslampanah,joachim.denil,
hans.vangheluwe}@uantwerpen.be

Abstract. There are various computation components, operating systems, and firmware used in the development of the Internet of Things (IoT). This variety increases the structural complexity and development cost and effort of the IoT systems. Besides, analyzing and troubleshooting these systems are time-consuming, costly, and cumbersome. To address these problems, this study aims to provide a higher level of abstraction for analyzing and developing IoT systems using Model-driven Engineering techniques and Petri-net models. To this end, a Domain-specific modeling Language (DSML), called DSML4Contiki, was presented in our previous study for the development of Wireless Sensor Systems (WSN) based IoT systems. The current study extends DSML4Contiki by providing an automated mechanism to analyze the IoT system at the early design phase, resulting in a reduction of the number of errors in the system and iterations in the development process. This is achieved using model transformation rules to transform the domain models at a high level to both the target platform artifacts as well as Petri-net models. By applying k-boundedness property checking on the Petri-net models, different analyses (such as power consumption, bottlenecks, and first crashing node) are realized for WSN based IoT systems. To evaluate the proposed approach, the engineering of a smart fire detection system is considered as a case study.

Keywords: Internet of Things, Wireless Sensor Networks, Model-driven Engineering, Petri-net, Smart Fire Detection System

1 Introduction

Internet of Things (IoT) is rapidly taking its place in different technologies and markets [17], such as home appliances, smart buildings, Industry 4.0 applications, and Digital Twin systems. IoT systems consist of different components such as sensors, actuators, and log managers for data management. These systems can benefit from Wireless Sensor Networks (WSN) to make their communication

topology more flexible (using the ad-hoc network provided by WSNs) and increase the coverage of the resulting IoT system in the physical environment, without a need for a direct Internet connection in all devices.

However, creating WSN based IoT systems require some more components such as source nodes, sink nodes, and gateways. The resulting system is complex with different components requiring to be programmed to work collaboratively. This complexity makes the design and analyses of these systems time-consuming, costly, and cumbersome. This complexity can be addressed with Model-Driven Engineering (MDE) techniques [10] to increase the level of abstraction and automatically synthesize the system artifacts. To this end, in [6] and [2], we have introduced a Domain-specific Modeling Language (DSML), called DSML4Contiki, for the design and development of WSN based IoT systems (using Contiki operating system). The domain models, i.e. the models designed in DSML4Coktiki, are used for automatically synthesizing the architectural code/configuration of WSN based IoT systems. More details on DSML4Contiki can be found in [2].

Using the model-centric development methodology, design models can be used for the early analyses and validation of the system. This can reduce the number of errors in the system under development, leading to increase the quality of the system. This early analyses of the system can also reduce the number of iterations in the development process of an IoT system, resulting in reduction of the development cost and effort. As these multi-component systems work based on some events to fulfill their tasks, their behavior [20] can be represented by the Petri-net models. Therefore, in this paper, we extend DSML4Contiki to synthesis the analysis models of the IoT system in Petri-net. In this way, the modeling language can generate the analysis models of the system from the design models, using model transformation techniques. By applying k-boundedness analysis on the automatically generated Petri-net models, various analyses (such as power consumption, bottleneck, and first crashing node) can be performed (semi-)automatically on the IoT models. To evaluate the proposed approach, a case study called fire detection system is used.

The paper is organized as follows: Section 2 discusses the related work. The proposed approach for analyzing the WSN-based IoT systems with Petri-net models are discussed in Section 3. Fire detection system is discussed as an evaluating case study in Section 4. A computational complexity analysis is provided as the evaluation for the proposed approach in Section 5. Finally, the paper is concluded and the future work is stated in Section 6.

2 Related Work

To analyze the design and deployment of WSN and IoT systems, several studies are proposed in the literature. In the study [18] the power analysis is realized using a tool called ArchWiSeN. Also, a modeling framework in [5] is proposed to provide multi-view architectural approach. The authors also offer a code generation for the simulation of the designed network. However, the simulation can

be made for limited devices. These two studies do not benefit from a formal modeling method, such as Petri-net.

In the study of [12], a discrete event simulation is made for IoT systems using high-tech and costly multi-core CPUs and GPUs. The study [13] offers stochastic model checking. However, they are bounded to hardware part that is operating with Contiki operating system. In our study, we consider an approach which covers all system components from WSN, IoT to Log Manager as well as providing a cost efficient development.

Furthermore, there are several studies in the literature which are using manually created Petri-net models for IoT systems to do analysis [16] [22] [21]. However, these approaches do not benefit from MDE techniques for the automatic development of Petri-net models and they need lots of effort for creating the models manually, especially for the industrial cases of IoT systems.

In a complex and multi-component system, such as an IoT system, a low-cost and effort development is very important. To this end, in our study, the Petri-net models for WSN based IoT systems are generated automatically (using MDE techniques) to do a set of early design phase analysis such as power consumption, bottlenecks, and first crashing node.

3 Analyzing IoT systems with Petri-net models

This section discusses the proposed analysis approach for WSN-based IoT systems using Petri-net models. These models are automatically generated from domain models (designed in DSML4Contiki). Figure 1 illustrates the mapping table, between DSML4Contiki elements to Petri-net model elements, to generate the Perti-net models both in PIPE and LoLA frameworks. Since LoLA has not a graphical interface (it runs in terminal), we decided to use PIPE's graphical interface for the representation of the system. On the other hand, PIPE has not k-boundedness feature, so, the k-boundedness analysis is made in LoLA.

For both of the Petri-net analysis tools, the code generation rules are written in Acceleo model to text transformation language. As it is shown in Figure 1, node elements such as the WSN sensor nodes, ESP8266 Wi-Fi module [11], RaspberryPI, and Log Manager elements are mapped to places in the Petri-net model. On the other hand, the messages such as Node messages and ESP messages are mapped to tokens in the Petri-net model. Finally, the relations in the domain-specific models are mapped to transitions and arcs, based on their situation.

DSML Element	Petri-Net Element	DSML Element	Petri-Net Element
Tag (Sensor Nodes)	Place	Node messages	Token
ESP8266	Place	ESP Messages	Token
IoT Log Manager	Place	Element Relations	Transition
RaspberryPI	Place	Element Relations	Arcs

Fig. 1. Mapping table between DSML4Contiki elements to Petri-net model elements

One of the important criteria to analyze the IoT systems is the data sampling period of its end-devices [4]. To reduce power consumption and increase battery life, designers can increase the sampling period. However, this may violate system requirements in some cases such as sampling temperature data for a fire detection system where the fire situation must be reported in a limited amount of time. However, setting the sampling time very high can bring the risk of bottleneck and also resulting some of the sensors to discharge their batteries fast, resulting in losing the coverage of part of the system. Therefore, analyzing the trade-off between these two ends and providing an appropriate sampling period is very essential. Specifically, realizing these analyses in the early design phase is important, as it decreases the cost of system development and avoids unforeseen critical errors in the system after deployment. Another parameter that needs to be analyzed is the distances between the nodes. In this section, these analyses, as well as some other analyses (such as cost analyses and propagation delay analysis), are elaborated using Petri-net models generated from the design models and the data provided by the user in the design models. These analyses are done using the k-boundedness feature of Petri-net models.

K-boundedness feature checks a place in a Petri-net model and counts the number of tokens that are passed in that place, then it compares that number with the k value, which is determined by system. In this study, the k value represents the bottleneck of the sensor network. If the number of tokens gets closer to k value for a node, the node (place) can have bottleneck problem and also the power consumption will increases in that node and the node may goes down soon. If the power is depleted in a node, a part of the network may be disconnected in the WSN. Therefore an optimal k value must be found by the designer and the topology design must be made considering this value. Moreover, the number of the tokens pass through a place must be below this k value. One way to reduce the k value is by adding extra nodes to decrease message traffic in this specific node, but this approach results in an extra cost to the system. Another way is tolerating the higher k value by having higher battery capacity in the node to ease the bottleneck problem, but this also results in additional cost. We need another trade-off analysis for this situation as well.

To ease understanding and demonstration, the Petri-net based analyses are divided into two parts. The first part is realized in PIPE framework (with graphical representation) to provide the distance and propagation delay with respect to the sink node, and to perform a cost analysis and arrange a bill of material (BOM) for the system. The distance values are provided by the user for each node. The second part of the analysis is realized using the low-level Petri-net capability of LoLA (using k-boundedness for the places) for bottleneck analysis, power consumption analysis, and node crash (first node die) analysis. For the cost analysis, the BOM is calculated based on a (assumptive) price list.

To perform the analyses, the Petri-net model is generated automatically from the design models. The model-to-model transformation rules are implemented in the Acceleo[3] language to generate the Petri-net models. These transformation

[3] Acceleo: https://www.eclipse.org/acceleo/

rules accept the WSN based IoT system design in 4 viewpoints (modeled in DSML4Contiki framework) as inputs and after applying the transformation rules they return the Petri-net models in PIPE and LoLA format. An excerpt of LoLA model generation rules (to create the places) is demonstrated in Listing 1.1. To generate LoLA model, the elements in the *System view* and *Log Manager* view are used. The rules in Listing 1.1 traverse the design model to collect all ESP elements (Lines 2-4) and Tags (Lines 5-7) connected to the Log Managers (Line 1) defined in the system and provide the information for the places of the Petri-net models.

Listing 1.1. Excerpt of LoLA model generation rules in Acceleo for creating Places

```
1 [for (l:LogMan | IoTSys.logman)]
2     [for (e:ESP | IoTSys.esp)]
3         [e.Name/],
4     [/for]
5     [for (t : Tag | tag)]
6         [t.Name/],
7     [/for]
8 [/for]
```

However, calculating and transforming some of the semantic properties of the design models, such as distances of source nodes and sink node, are not straightforward and cannot be done using solely a template engine (in Acceleo) and constraint checking language (in AQL). In fact, we need an imperative language (such as Java) to do this kind of calculations and later to be transformed to the target model using a declarative transformation language such as Acceleo.

As an example, in Listing 1.2 the distance property is transformed from the design model to the Petri-net model. This rule uses the Java wrapper code that implements the *calcDist* function. In Listing 1.2, Lines 5-6 are calculating the BOM and Lines 7-10 are calculating the total distances of each source node to the sink node.

Listing 1.2. Semantic property calculation Rule for node distance

```
1  [query public calcDist(arg0:OrderedSet(Tag)):
2     OrderedSet(String) = invoke ...
3  Analyses Report [let abc : OrderedSet(String) = aIoTSystem.logman.tag
4     .calcDist(aIoTSystem.logman.tag)->asOrderedSet()]
5  [for(t:String|calcDist)] TmoteSky x [aIoTSystem.logman.tag->size()/]=
6     [100*aIoTSystem.logman.tag->size()/]$ ...
7  calcDist(Collection tag){for(int i=0; i<aTag.size(); i++)
8    for(int j=0; j<aTag.get(i).getSelftag().size(); j++)
9      aTag.get(i).getSelftag().get(j).setDistance(aTag.get(i).
10        getSelftag().get(j).getDistance()+aTag.get(i).getDistance());
11    ...
```

Finally, using k-boundedness feature, the highest risky nodes (the first to die nodes) can be detected. These nodes have the highest connectivity in the network. Considering the k value applied in the Petri-net model of the network,

the nodes with their connectivity value equals or nearest to the k value, are candidate risky nodes and the user need to be aware of the risk of connection due to the failure of these nodes.

4 Case Study: Smart Fire Detection System

In this section, the proposed approach is evaluated using a case study called smart fire detection system [1]. The system modeling (design), generation of Petri-net models and analyzing the case study using DSML4Contiki and the proposed approach are discussed in this section.

4.1 System Modeling

The requirements of the fire detection system are elicited from the needs of a municipality library system in Izmir, Turkey [8]. It is designed and modeled in DSML4Contiki using a WSN multi-hop topology [7]. Figure 2 shows the topology (Long Manager) viewpoint of the system in DSML4Contiki including the tags for the sensors and a log manager called "firedection".

The system aims to use a WSN based IoT system to detect the fire and notify the users to evacuate the building as soon as possible. This system has several WSN source nodes to sense the temperature at certain spots in the library and send the values to the gateway which delivers them to a log manager. Also, there are ESP8266s modules that sense the room temperature and directly send the temperature data to the log manager. Based on these temperature values, the system can decide about the states of a possible fire, Blue (Safe), Yellow (Warning), and Red (Alarm).

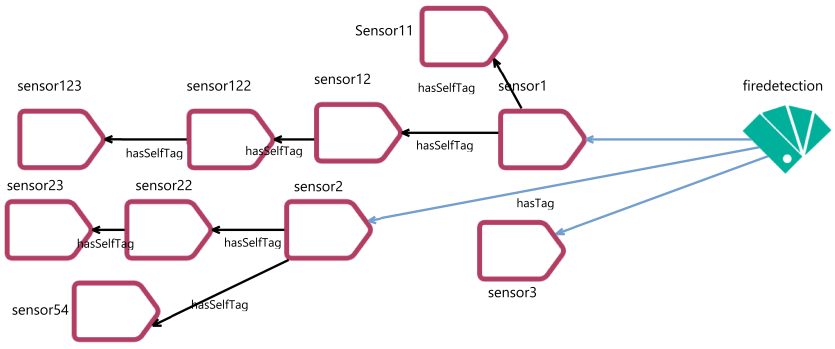

Fig. 2. Topology viewpoint to design WSN topology

4.2 Generating Petri-net Model

Using the transformation approach discussed in the previous section, a Petri-net model is generated from the domain model (mainly topology viewpoint in

Figure 2) of the fire detection system. The generated Petri-net model is shown in PIPE framework, see Figure 3. This figure shows the places, transitions, and arcs related to a multi-hop WSN for the fire detection system. The initialization of tokens is made according to the sensing activity of sink nodes. If a node (place) has a sensor data and has to transmit this data, then that node (place) must have a token.

Moreover, the distance between the sensor nodes and the sink node, and propagation delay results are calculated via the transformations and shown top-right side of Figure 3. They are calculated based on the information provided via the designer in the design models.

Finally, the generated BOM is generated and shown in the bottom-right side of Figure 3 with the number of materials which are used in the design.

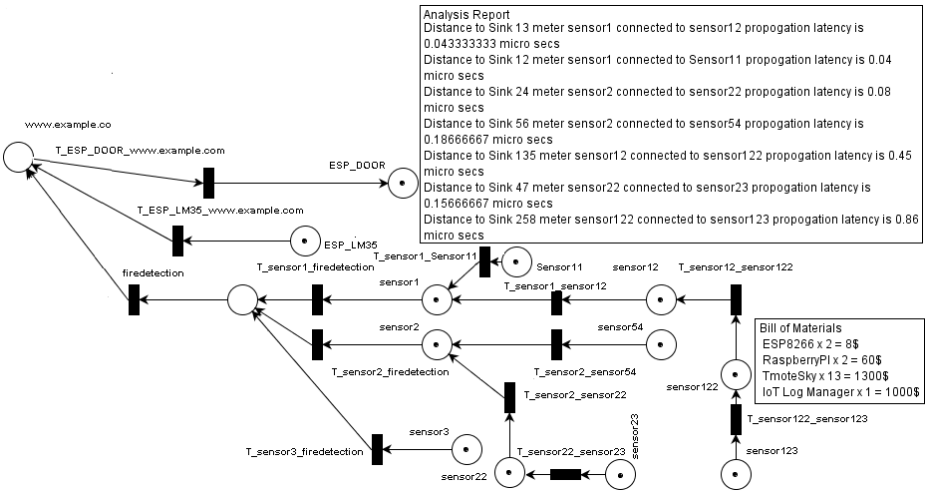

Fig. 3. Generated Petri-net model for smart fire detection system

4.3 Analyzing the Fire Detection System with Petri-net

Bottleneck problem and its relation with power consumption for IoT systems is discussed in the study of [3]. In our proposed IoT analysis approach, the bottlenecks can be detected using k-boundedness, if the designed system model can be transformed to a model in Petri-net. In this way, the power analysis can be done using the Petri-net model to find out the impact of bottleneck in the system. Generally, if there is a bottleneck problem in a node, the network lifetime, which has already planned, may not be valid anymore.

In our case study, the k-boundedness feature of the Petri-net models is used so that the nodes are analyzed for the power consumption as they may go out of battery and cause a part of the network to go down. Therefore, the user can go back to the design and can devise alternatives, such as adding new connections,

new nodes, or increasing battery capacity. To keep the planned/designed lifetime of the network, the bottleneck problem must be analyzed for the IoT system before developing and deploying the system in the physical environment.

If k increases, the bottleneck problem may occur and as a result power consumption can be increased. A k value must be found to provide the desired lifetime with planned battery capacity in the system. Some of these analyses are discussed below for the fire detection system.

Load bottleneck (First Crashing Node Analysis):

The load in a node is the number of messages to be received and/or transmitted to the next node(s). Naturally, the message load of a node increases when the node has more sub-nodes (neighbor nodes). This can create a bottleneck and results in more power consumption, which can result in a node crash earlier than the planned lifetime. Moreover, a node can have message overflow due to the limited buffer (as they have limited memory). To analyze these two problems, the number of operations in each node (place) should be calculated using the number of its neighbors (k).

To find the number of operations for a place/node, the number of messages which are received and transmitted in a node should be considered, and it can be calculated using the equation 1. The value of k (neighbors) is multiplied by two since a node that receives a message should transmit it to the next node(s); a send and receive actions are considered two different operations. The node also sends its own sampling data to the sink node, which adds 1 operation. It worth to note that in fire detection system, each sensor node periodically samples the temperature and send it to the sink node via intermediate neighbor nodes.

$$NumberOfOperations = (2 * k) + 1 \tag{1}$$

Using these calculations throughout the Petri-net model with a specific initial configuration, the user can monitor the behavior of the system in terms of the node's load (and probable bottlenecks) in the network. The nodes with highest number of operations (or highest k value) are the candidate nodes to crash the first (high risk nodes). The user can modify the design to resolve the problems (by adding new nodes or battery capacity) and continue to the development. Alternatively, the designer may resolve the problem by reducing the number of neighbors for the node. This needs a trade-off analysis.

Power consumption:

Power consumption depends on the number of operations for each node. Therefore, to bound the network with a value of k, the user first should decide on the nodes' lifetime, the battery capacity and the message sending period/sampling rate.

The lifetime of a node can be calculated using equation 2. If the user increases battery capacity, then the lifetime of nodes increase. The power consumption of a send and receive operation (using internal antenna) is averaged with ($RxTxAvg_{mA}$). The average of receiving and transmitting operations is averaged to ease the calculation. T represents the number of using antenna in an hour (as the unit of battery capacity is also in mAh). The value of T is bounded

to the sampling period of the nodes and can be calculated using equation 3. Since our Petri-net analyses are based on tokens instead of time, if the system's lifetime is determined then k value can be found to analyze the Petri-net model in LoLa.

$$LifeTime_{hour} = \frac{BatteryCapacity_{mAh}}{(NumberOfOperations) * (RxTxAvg_{mA}) * T} \tag{2}$$

$$T = \frac{3600}{SamplingPeriod_{seconds}} \tag{3}$$

In this study, TmoteSky motes are used to create a multi-hop WSN. To do the above-mentioned calculations, the power consumption of transmission and reception values are found from the data-sheet of TmoteSky. In the case study, the nodes send temperature values periodically, in each 120-seconds. According to equation 3, nodes use their antenna 30 times in an hour. In experiment, a 5-volt (10000mAh) power-bank was used. It can provide 33000mAh battery capacity since TmoteSky runs at a voltage level of 3.3v. To ease the calculations the average of transmitted and received current consumption is calculated (based on the data-sheet), which is approximately 20mA. In equation 2, if k is 5, a node with 5 sub-nodes (see equation 1) can run for 5 hours. We should consider that k = 5 is a very highly connected network with very high reliability.

These manual calculations show how to perform the trade-off analyses between k and lifetime of the network (via power consumption). In the system design phase, network size can be huge where it is not easy to track the paths and do the analyses manually, so to test the k value and to bound the network, the proposed automatic analyses in LoLA can be used. For example, the sensor topology shown in Figure 2, is transformed to LoLA code and the network is tested for a node named *sensor1* using a k value and the output result is shown in Listing 1.3. If a node cannot be bounded with given k value, then alternatives must be devised by the user.

Listing 1.3. Application of k-boundedness to calculate power consumption using LoLA

```
1  lola --formula=AG sensor1 < 5 IoTSystem.lola : lola:result:no
```

Response time for alarm:

The first minutes of indoor fire events are critical considering the evacuation possibility of the building. Therefore, the response time of a fire detection system must be as low as possible. If the fire starts exactly after sampling of temperature (the worst case), then the fire spreads until the time for the next sampling time. To keep the sampling interval more often, either the battery capacity must be increased or several sink nodes must be added.

In calculation of the response time (from sensing to triggering alarm), the message transmission times should be also considered. Therefore, if the distance to the sink node is too far for a node in a multi-hop network, the system's total propagation delay for a message can be a problem. If the distance increases, the

propagation delay increases. Although there is a processing delay in the nodes, it is trivial amount of time comparing to the message transmission time and it can be ignored in the calculations. Consequently, the time to trigger the fire alarm in the worst case needs to be calculated with respect to the node which has the furthest distance to the sink node. These kinds of trade-off analyses can be done by the user using the automatically calculated propagation delays (see Figure 3) to reach a proper response time.

5 Evaluation

In this section, the computational complexity of the proposed approach is evaluated. The main process for system design using the proposed approach is model transformations. So, the complexity of these transformations are focused for the evaluation.

First, we present the assumptions of evaluating transformations.

1. M represents the set of Log Managers in the system
2. N_m represents the set of the Tags for Log Manager m
3. $\forall m \in M, N = \cup N_m$
4. $\forall n_i, n_j \in N_m \rightarrow n_i \neq n_j$
5. $\forall n_i \in N_p \wedge n_j \in N_q \rightarrow n_i \neq n_j$

Items 1-3 gives the definitions for log managers, their tags, and total number of tags in the system. Assumption number 4 means that the Tags of each Log manager are unique and assumption 5 means that the Log Managers have no common Tags. This is because the log managers store the data for the devices and the system should not have redundancy in storing the data.

The transformation from design models to Petri-net models is given in Listing 1.2. This transformation routine contains two nested loops, the outer one traverses all log managers of the system and the inner one transverses all of the tags for the log manager indicated by the outer loop. Although, the total complexity seems to be $O(|M| * |N_m|)$, however, this estimation is too high as the sets of Tags for log managers are disjoint (assumption 5). So, the total number of iterations for these two nested loops is the number of log managers M (for the outer loop) plus the total number of Tags N in the system (for the inner loop), resulting to $O(|M| + |N|)$.

If the total number of Log managers are much less than total number of Tags (i.e. $M << N$), then the complexity of the algorithm becomes $O(|N|)$. Therefore, the computational complexity depends on the number of the nodes (Tags). If M=N, only one Tag is used in each Log manager to connected a device, which is a poor design. In other words there are lots of Log managers and each Log manager just have one node (single Tag). Then the complexity is $O(|M| + |N|)$ $= O(|N| + |N|) = O(2|N|)$. This complexity is equal to $O(|N|)$. As a result, in any case the computational complexity of the transformation is dependent on the total number of Tags in the system (or the devices which are connected to the IoT system). This complexity is linear and has a reasonable computational cost.

6 Conclusion and Future Work

The current study extends DSML4Contiki by automatic generation of the Petri-net models from the design models and applying different analyses on these models using k-boundedness property of Petri-net models. Using this development approach, the developers can apply analyses such as network lifetime (first crashing nodes), bottleneck, and power consumption analyses in the early design phase. This prevents the extra iterations of re-design and re-deployment for the system. The proposed approach not only can generate a high ratio of the codes from the system but also can apply analyses based on the same domain-specific models. The study is evaluated using a smart fire detection system, and various analyses are applied in this case study and the results are discussed.

In the future, it is planned to raise the level of abstraction on the Platform-Independent modeling level. It will extend this study to support the modeling of IoT systems, independent of their target platforms, such as RIOT [9] and TinyOS [14]. This will be achieved using the model to model transformations. Moreover, we aim to use Multi-agent Systems [15] [19] in the modeling, analysis, and implementation of IoT systems.

References

1. S. Arslan, M. Challenger, and O. Dagdeviren. Wireless sensor network based fire detection system for libraries. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 271–276. IEEE, 2017.
2. T. Z. Asici, B. Karaduman, R. Eslampanah, M. Challenger, J. Denil, and H. Vangheluwe. Applying model driven engineering techniques to the development of contiki-based iot systems. In *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*, pages 25–32. IEEE, 2019.
3. S. Bosmans, S. Mercelis, J. Denil, and P. Hellinckx. Challenges of modeling and simulating internet of things systems. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 457–466. Springer, 2018.
4. J. Cheng, H. Jiang, X. Ma, L. Liu, L. Qian, C. Tian, and W. Liu. Efficient data collection with sampling in wsns: making use of matrix completion techniques. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5. IEEE, 2010.
5. K. Doddapaneni, E. Ever, O. Gemikonakli, I. Malavolta, L. Mostarda, and H. Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Proceedings of the Third International Workshop on Software Engineering for Sensor Network Applications*, pages 1–7, Piscataway, USA, 2012.
6. C. Durmaz, M. Challenger, O. Dagdeviren, and G. Kardas. Modelling Contiki-Based IoT Systems. In *6th Symposium on Languages, Applications and Technologies (SLATE 2017)*, volume 56 of *OpenAccess Series in Informatics (OASIcs)*, pages 5:1–5:13, Dagstuhl, Germany, 2017.
7. B. Karaduman, T. Aşıcı, M. Challenger, and R. Eslampanah. A cloud and Contiki based fire detection system using multi-hop wireless sensor networks. In *Proceedings of the Fourth Intl. Conf. on Engineering & MIS 2018*, page 66. ACM, 2018.

8. B. Karaduman, M. Challenger, and R. Eslampanah. ContikiOS based library fire detection system. In *2018 5th Intl. Conf. on Electrical and Electronic Engineering (ICEEE)*, pages 247–251. IEEE, 2018.
9. B. Karaduman, M. Challenger, R. Eslampanah, J. Denil, and H. Vangheluwe. Platform-specific Modeling for RIOT based IoT Systems. In *2nd International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT 2020)*, pages 1 – 8 (Accepted), 2020.
10. G. Kardas, Z. Demirezen, and M. Challenger. Towards a dsml for semantic web enabled multi-agent systems. In *Proceedings of the International Workshop on Formalization of Modeling Languages*, page 5. ACM, 2010.
11. N. Karimpour, B. Karaduman, A. Ural, M. Challengerl, and O. Dagdeviren. Iot based hand hygiene compliance monitoring. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2019.
12. S. Kim, J. Cho, and D. Park. Accelerated devs simulation using collaborative computation on multi-cores and gpus for fire-spreading iot sensing applications. *Applied Sciences*, 8(9):1466, 2018.
13. A. Lekidis, E. Stachtiari, P. Katsaros, M. Bozga, and C. K. Georgiadis. Model-based design of iot systems with the bip component framework. *Software: Practice and Experience*, 48(6):1167–1194, 2018.
14. H. M. Marah, R. Eslampanah, and M. Challenger. Dsml4tinyos: Code generation for wireless devices. In *2nd International Workshop on Model-Driven Engineering for the Internet-of-Things (MDE4IoT), 21st International Conference on Model Driven Engineering Languages and Systems (MODELS2018)*, Copenhagen, Denmark, 2018.
15. V. Mascardi, D. Weyns, A. Ricci, C. B. Earle, A. Casals, M. Challenger, A. Chopra, A. Ciortea, L. A. Dennis, Á. F. Díaz, et al. Engineering multi-agent systems: State of affairs and the road ahead. *ACM SIGSOFT Software Engineering Notes*, 44(1):18–28, 2019.
16. K. Nakahori and S. Yamaguchi. A support tool to design iot services with nusmv. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pages 80–83, Jan 2017.
17. L. Özgür, V. K. Akram, M. Challenger, and O. Dağdeviren. An iot based smart thermostat. In *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, pages 252–256. IEEE, 2018.
18. T. Rodrigues, T. Batista, F. C. Delicato, P. F. Pires, and A. Y. Zomaya. Model-driven approach for building efficient wireless sensor and actuator network applications. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 43–48. IEEE, 2013.
19. B. T. Tezel, M. Challenger, and G. Kardas. A metamodel for jason bdi agents. In *5th Symposium on Languages, Applications and Technologies (SLATE'16)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
20. H. M. Wanniarachchi, K. S. Perera, and M. Goonatillake. Behavioral modeling of wireless sensor nodes using meta-data. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–5. IEEE, 2015.
21. R. Yang, B. Li, and C. Cheng. A petri net-based approach to service composition and monitoring in the iot. In *2014 Asia-Pacific Services Computing Conference*, pages 16–22. IEEE, 2014.
22. Y. Zhang, W. Wang, N. Wu, and C. Qian. Iot-enabled real-time production performance analysis and exception diagnosis model. *IEEE Transactions on Automation Science and Engineering*, 13(3):1318–1332, 2015.