Modelling Contiki-Based IoT Systems*

Caglar Durmaz[1], Moharram Challenger[2], Orhan Dagdeviren[3], and Geylani Kardas[4]

1 **International Computer Institute, Ege University, İzmir, Turkey; and Integra Project Management Engineering Consultancy Ltd., İzmir, Turkey**
caglar.durmaz@gmail.com, caglar@integra-pm.com.tr
2 **International Computer Institute, Ege University, İzmir, Turkey**
moharram.challenger@ege.edu.tr
3 **International Computer Institute, Ege University, İzmir, Turkey**
orhan.dagdeviren@ege.edu.tr
4 **International Computer Institute, Ege University, İzmir, Turkey**
geylani.kardas@ege.edu.tr

Abstract

In this paper, we investigate how model-driven engineering (MDE) of Internet of Things (IoT) systems and Wireless-Sensor Networks (WSN) can be supported and introduce a domain-specific metamodel for modeling such systems based on the well-known Contiki operating system. The unique lightweight thread structure of Contiki makes it more preferable in the implementation of new IoT systems instead of many other existing platforms. Although some MDE approaches exist for IoT systems and WSNs, currently there is no study which addresses the modelling according to the specifications of Contiki platform. The work presented in this paper aims at filling this gap and covers the development of both a modeling language syntax and a graphical modeling environment for the MDE of IoTs according to event-driven mechanism and protothread architecture of Contiki. Use of the proposed modeling language is demonstrated with including the development of an IoT system for forest fire detection.

1998 ACM Subject Classification D.1.7 Visual Programming, D.2.6 [Programming Environments] Graphical Environments, D.4.7 [Organization and Design] Real-Time Systems and Embedded Systems, Embedded Software

Keywords and phrases Domain-specific Modelling, Metamodel, Model-driven Engineering, Internet of Things, Wireless Sensor Networks, Embedded Software, Contiki Operating System

Digital Object Identifier 10.4230/OASIcs.SLATE.2017.5

1 Introduction

Internet as a global information and communication infrastructure, is evolving to form of a platform for letting machines and smart objects communicate, dialogue, compute and coordinate [17]. The term "Internet-of-Things" (IoT) is broadly used for these interconnected objects which build the smart environments, e.g. smart homes [12]. IoT systems do not only present diverse context of smart implementations, but also present diverse computing and communication capabilities. This heterogeneity in devices brings management challenges in architectural and protocol issues [17] which requires network, embedded and distributed

* Authors would like to thank the Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) for covering SLATE conference attendance and paper presentation expenses under the project grant 115E449.

programming knowledge [2, 18]. Although advances in the development of low-cost and low-power micro-controllers play an important role during the construction of IoT systems, the scarcity of application developers who possess the required knowledge and experience for such systems, limits the power of using the micro-controller technologies in IoT. Proper software platforms, which facilitate the design and implementation of IoT applications, may bring more developers into this domain. For the sake of increasing need for application development on IoT, several operating systems for Wireless Sensor Network (WSN) and IoT motes are realized. Contiki [9] and TinyOS [14] are two of these best known sensor node operating systems (OS) [20]. TinyOS provides interfaces and components for common abstractions to implement algorithms designed for eneergy-efficient devices [1, 5]. Contiki gained popularity recently because of built in TCP/IP stack and lightweight preemptive scheduling over event-driven kernel [9] which is a very motivating feature for IoT.

Even if operating systems play the role of an abstraction layer for low-level hardware heterogeneity, distributed programming and network related concerns dominate the workload in software development. The separation of hardware-related and application related concerns will improve the software engineering processes of IoT systems [15] which may pave the way to deal with the system's structural complexity coming from the heterogeneity.

One possible approach to cope with this complexity is to increase the abstraction level using system models [16], for WSN and IoT in a Model Driven Engineering (MDE) approach. MDE moves software development from code to models and may increase productivity and reduce development costs [23]. Fruitfulness of this approach is demonstrated in several other domain studies, e.g. [4, 26].

MDE for IoT systems and WSNs is being researched in several studies especially for the purpose of separation of concerns [3, 11, 21, 25]. Although, some of these studies deal with the structural complexity of IoT, most of them do not provide a complete and/or a systematic approach for their MDE solutions. Moreover, most of these studies only consider the MDE of IoT only on TinyOS as reported in [11]. In order to contribute these studies and fill the gap for providing an MDE approach for developing IoT systems based on Contiki OS, in this paper, we introduce a domain-specific metamodel, which can be used for modeling Contiki applications. Based on the metamodel, a visual concrete syntax is derived and a graphical modeling environment is developed for modeling IoT systems. Use of the proposed modeling language is demonstrated with including the development of an IoT system for forest fire detection.

Rest of the paper is organized as follows. In Section 2, Contiki OS is briefly discussed. Section 3 introduces the metamodel proposed for Contiki. Section 4 covers the derived concrete syntax, its notations and the related graphical modeling environment. A use case study is given in Section 5 for exemplifying the use of the proposed modeling environment. Section 6 includes the related work. Finally, Section 7 concludes the paper and states the future work.

2 Contiki Operating System

Contiki[1] is a lightweight open source OS written in C for IoT. Contiki connects tiny low-cost, low-power micro-controllers to the Internet. As indicated in [24], Contiki and its micro IP (uIP) stack are used worldwide by hundreds of projects and companies. The uIP implements only the minimal set of features needed for a full TCP/IP stack.

[1] Contiki: `http://www.contiki-os.org/index.html`.

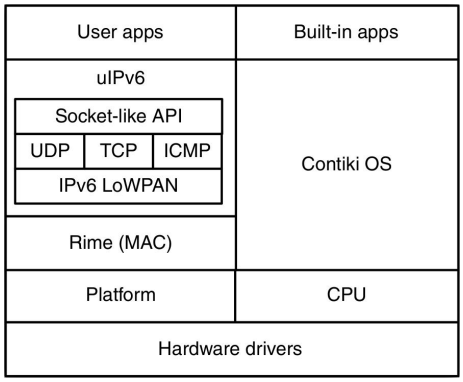

Figure 1 The architecture and main components of Contiki [24].

Operating systems, like TinyOS [14] and SOS [13] are based on event-driven model which is often used on memory constrained devices. Contiki is also event-driven and provides a lightweight thread model called protothreads [10] over event-driven kernel which is not available in the other peer operating systems. User and built-in applications can be implemented over protothreads which are basic OS threads. Protothreads simplify event-driven programming by reducing the need for explicit state machines by providing abstraction of conditional blocking wait operation [10]. PT_WAIT_UNTIL() statement in Contiki [10] blocks conditional execution of a process. On the other hand, PT_YIELD() statement blocks execution of process unconditionally. Protothread waits until the next time the protothread is invoked and continues executing the code following the PT_YIELD() statement.

The architecture of Contiki is shown in Figure 1. Hardware Drivers, Platform and CPU abstractions form the abstraction for the real low-level hardware. Platform and CPU layers are implemented independently due to portability concern. The Rime system provides medium access control and a set of lightweight communication primitives for network protocols. The uIPv6 stack makes use of Rime, and provides TCP, UDP and ICMP and also a socket-like API, protosockets [2]. The protosocket implementation makes use of Contiki protothreads.

The Contiki also provides timer support and dynamic linking capabilities as system services. Both updating the system which contains hundreds or even thousands of nodes with new functionality and correcting software bugs are often needed in deployed IoT systems [8]. Developing full system image replacement is a solution for this situations but it is not feasible to physically collect and reprogram all sensor devices [9]. By the help of dynamic linking, new code modules can be added at runtime to the application running on a node.

In this study, we address the modelling of the following sections of Contiki architecture given in Figure 1: ContikiOS, User apps, and uIPv6 (including Socket-like API, UDP, TCP). In this way, by using the provided metamodel of ContikiOS and uIPv6, we can model a user application for an IoT system.

3 The Metamodel for Contiki-based IoT Systems

In this section, a metamodel is presented for MDE of Contiki-based systems to provide the concepts and their relations pertaining to IoT domain. This metamodel can be used as an abstract syntax and pave the way for developing a modelling language for Contiki.

[2] ContikiOS 2.6 Documentation: `http://contiki.sourceforge.net/docs/2.6/index.html`.

As shown in Figure 2, the proposed metamodel provides the meta-entities and their relations required for the structural architecture of Contiki programs. Entities of the metamodel are given in italics throughout the paper.

Since Contiki is primarily an OS, main entities of Contiki are process and thread inside each *Node*. A process in Contiki consists of a single protothread. *Process_Thread* is used to define this single protothread of the process [10]. A process thread can also call several stand alone protothreads represented as *PT_Thread* in the metamodel by the use of *PT_Thread_Call*. Boolean value of *Autostart* attribute of *Process_Thread* designates whether the thread will be started at the beginning of the mote execution, or not. Also, *Name* attribute is the declaration of the thread while *Description* attribute represents a short description for debugging concerns. Being an event-driven OS, Contiki kernel sends events to *Process_Thread* with event and data arguments. The attributes, *EventArgument* and *DataArgument* in the metamodel, define variable names of event and data arguments.

PT_Thread_Call holds the destination *PT_Thread* and a concrete PT_Thread_Struct argument of *PT_Thread*. PT and Psock are concrete structures that can be sent as arguments when calling *PT_Threads*. Psock ,which can only be used in *PT_Threads*, offers TCP and UDP socket implementations in Contiki.

Messaging among processes and threads is another core function of operating systems. Due to being again event-driven, messages received from other processes are handled by a specialized event, *Process_Event*, and the messages are sent by *Process_Post*. *Process_Post* has exactly one Data attribute that stores message payload and exactly one *Process_Event* that is going to be triggered. Sync attribute of *Process_Post* defines the execution of caller process thread which is going to be synchronous or asynchronous.

Besides this messaging between processes in the same mote, messaging among processes residing in different motes is another requirement of an IoT OS. Contiki is differentiated itself from other WSN operating systems by implementing network IP stack and built-in TCP/IP and UDP support. Related feature is modelled in the proposed metamodel as follows: One mote can start a connection and send first message via *Client_Connection* with an outgoing *Data* with *RemoteIP* and *RemotePortNumber* attributes. This connection can also be used to retrieve incoming messages from the relevant remote host. Contiki fires *TCPIP_Event* when a UDP or TCP packet is arrived. There is no special event type for UDP IP events. By the help of connection and *TCPIP_event*, messages from other host can be detected and processed. On the other side, *Server_Connection* entity can receive messages from hosts by defining *ListeningPort*. Same *TCPIP_Event* and connection entity types are also used to process messages on the server side. Connections can be used for several messages with the remote host, several incoming and outgoing data may be related to a connection in the model. The Last event type is *Time_Event* and it occurs when *Etimer* gets to zero. *Etimer* has *Name* attribute which holds a variable name in *Process_Thread*, and *Period* attribute which defines durations between the current moment and the time of next firing of *Time_Event*.

4 The Concrete Syntax

The metamodel introduced in the previous section represents an abstract syntax for a modeling language for Contiki-based IoT systems. While the abstract syntax includes the concepts and their relations between those concepts, a concrete syntax provides a mapping between these concepts (meta-elements) and their textual and/or graphical representation. In this study, we also introduce a graphical concrete syntax for modelling Contiki applications. Table 1 lists the graphical notations for some of the important entities discussed in Section 3.

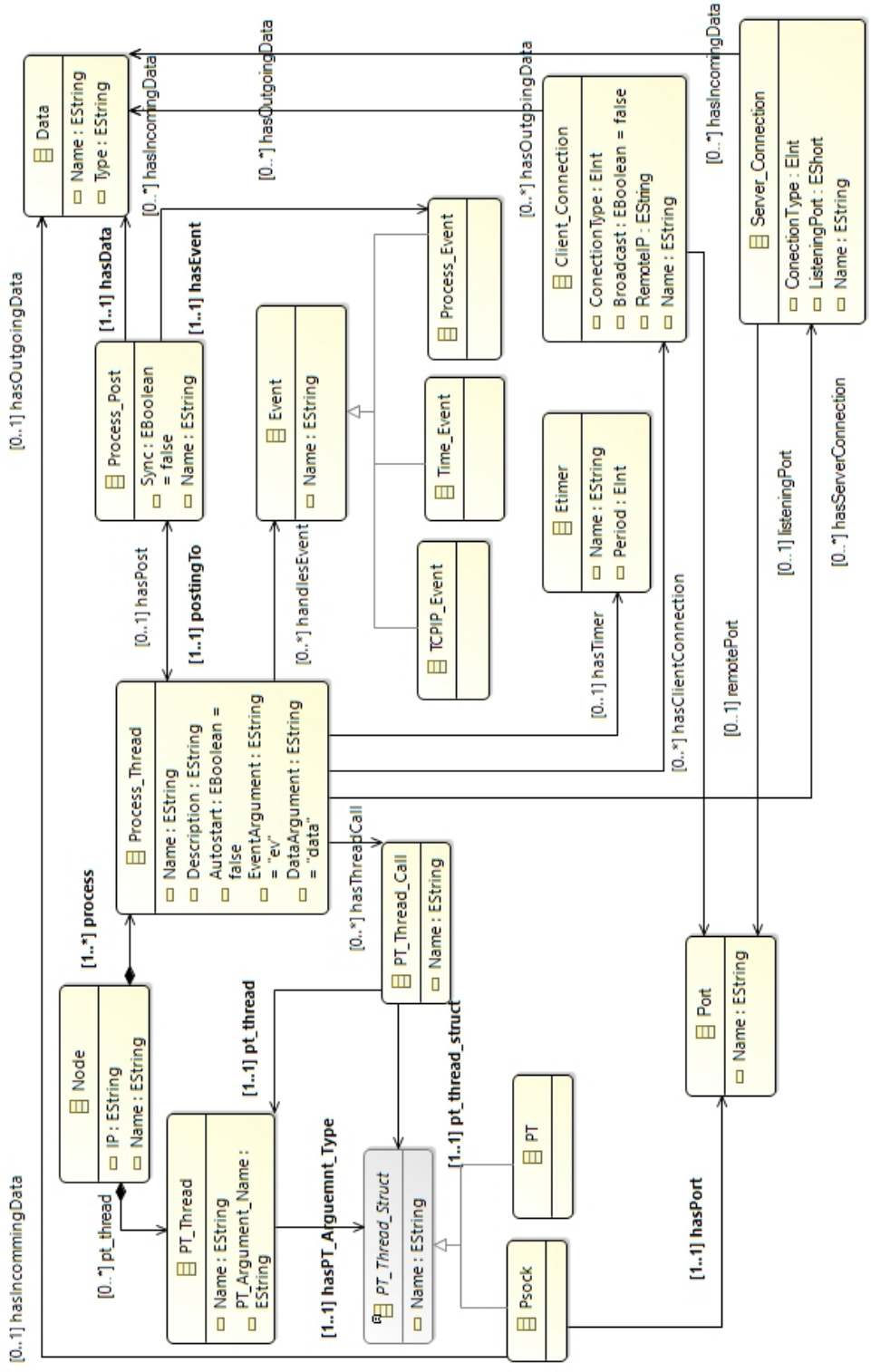

Figure 2 A domain-specific metamodel for Contiki OS.

Table 1 Some of the concepts and their notations for the Contiki modeling environment.

Concept	Notation	Concept	Notation
Node		Data	
Process_Thread		Process_Event	
PT_Thread		TCPIP_Event	
PT_Thread_Call		Port	
Process_Post		Server_Connection	
Etimer		Client_Connection	
Event		Psock	
Time_Event		PT	

Some of the graphics used for the concept representations are adopted and modified from Flaticon[3].

As it is shown in Table 1, the notations are selected in a way that the concepts with the same range of semantics have common symbols. For example, the elements with interior structure of processes and threads, such as *Process_Thread*, *PT_Thread*, *Process_Event*, and *PT*, have the same symbol of a gear as part of their notation. Similarly, the event related concepts, such as *Event*, *Time_Event*, *Process_Event*, and *TCPIP_Event* have the symbol of a screen as part of their notations. The notations are selected in a way that their presentation in both black-white or color will let the user to differentiate them from each other.

The metamodel discussed in the previous section is encoded in Ecore format inside Eclipse Modelling Framework (EMF[4]). Using this Ecore file, the notations depicted in Table 1, are mapped in Eclipse Epsilon Framework[5] to develop a graphical editor, as shown in Figure 3. To this end, the Ecore model, as our abstract syntax, is converted to Epsilon format which is used by Epsilon Eugenia tool. Then the required configurations are applied to inject the concrete syntax related information as some annotations in the Epsilon file.

Since we already provide some constraints in the abstract syntax (the metamodel), such as multiplicities, the graphical editor can check some of the connection rules during modelling, e.g. checking the source and target and also the number of relations for an element. Furthermore, we have benefited from the features of Eclipse Graphical Modeling Framework (GMF[6]) for automatically checking some consistency constraints when the model is modified, e.g. removing all the input/output links for an element when the element is removed from the model.

In addition to the GMF-based constraint checks, we have provided some domain rules to be checked to restrict the user to provide a more accurate model. This will lead to have artifacts with less errors in the generation phase. To this end, some static semantic rules

[3] http://www.flaticon.com
[4] http://www.eclipse.org/modeling/emf/
[5] http://www.eclipse.org/epsilon/
[6] https://www.eclipse.org/modeling/gmp/

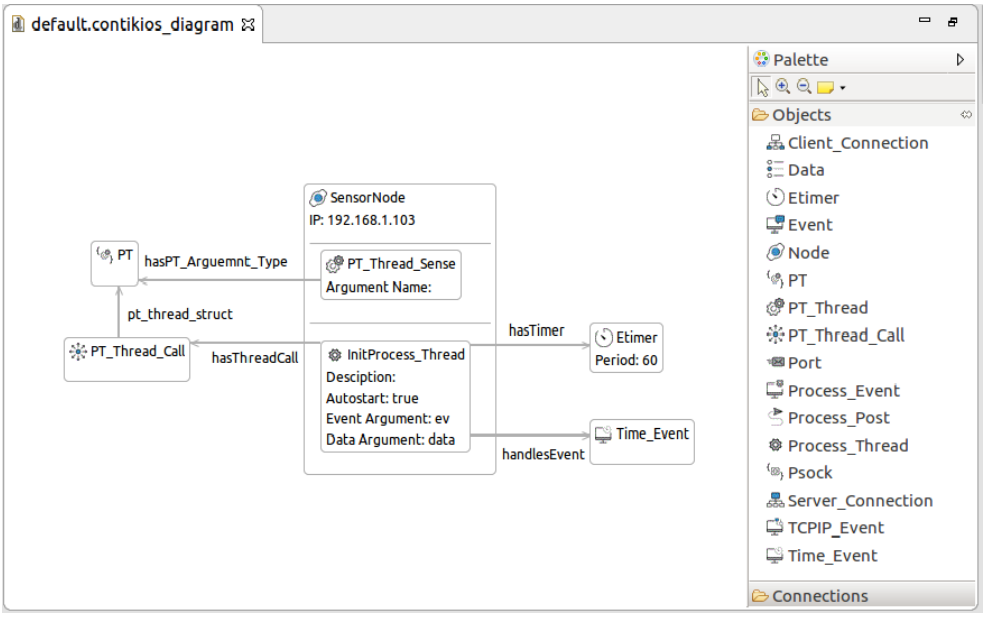

Figure 3 A screenshot from the developed graphical editor for modeling Contiki-based IoT systems.

are provided for the system. These rules are implemented in Eclipse Validation Language (EVL[7]) to be integrated with the provided graphical modeling environment. Some of these rules are given below:

- A *Port* element in an instance model must have exactly one incoming link from a *Server_Connection*, but it should also have at least one incoming link from some *Client_Connections*. In this way, no Port element will be used improperly (without client side or server side links).
- The message payloads which are held in *Data* element are forced to be same in the relevant client and server connections.
- All client and server Connections are forced to be linked to *Process_Threads*, as each *Process_Threads* may have 0-* *Client_Connection* and/or *Server_Connection*.
- All events must be handled by a *Process_Thread*

The graphical editor developed in this study provides a convenient modeling environment in which the developers can create instance IoT models conforming to our metamodel by using the graphical concrete syntax. As can be seen in Figure 3, entities are listed in the palette residing in the right side of the graphical modeling editor. A developer can drag and drop an entity from this palette to create an instance of this entity. While the instances are added into the model, the related associations between the entities are automatically established and controlled by the defined constraints.

5 Case Study: Modeling an IoT System for Forest Fire Detection

There are many applications of WSNs [27] and IoT [19]. One of these applications is a Forest Fire Detection System [28]. In this application domain, IoT components are used to recognize

[7] `http://www.eclipse.org/epsilon/doc/evl/`

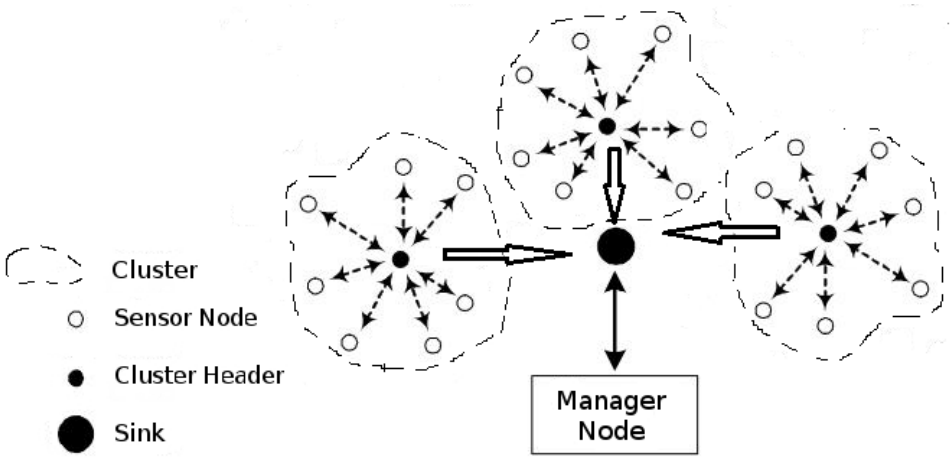

Figure 4 A WSN for real-time forest fire detection.

the symptoms of a fire in an area such as a jungle and take in-time reactions. By the help of periodic measurements of some critical parameters in a wide area, the system predicts the risk of forest fire and warns in order to minimize the loss of forests, wild animals, and people in the forest. In this section, to give some flavour of using both the proposed metamodel and our modeling environment, we discuss the development of such an IoT system.

Densely deployed large number of sensor nodes collect measured data and send them to their respective cluster nodes that collaboratively process the data. Battery limitation forces WSN researchers to minimize power consumption. Since, communication among nodes consume more energy over computing, the option of sending aggregate data from sensor nodes to cluster header and from cluster header to sink node is chosen as an architectural design. The total size of messages transmitted in the system is reduced by clustering nodes and aggregating measured data.

The system specifications require collecting regular measurements, getting immediate fire alarms from nodes and querying instant measurements of a particular cluster from management office for analyzing. Tasks of cluster headers are to transfer raw messages among sink and sensor nodes, to prepare aggregate reports and to transfer them in the name of cluster. This use case is analyzed to extract the main scenarios of the system (see Figure 4). Sensor, cluster header and sink nodes in Figure 4 are considered by focusing on the internal node structure and communication of node groups.

Following the analysis, the system to be built is designed by using Contiki modeling tool previously introduced in Section 4. It is worth indicating that the modeling discussed in here takes into account the main process of the system and does not cover the setup procedure including clustering.

The instance model conforming to our metamodel for this case study is given in Figure 5. A *SensorNode* can measure environment temperature, relative humidity and smoke in every minute. *RegularProcess* in *SensorNode* gathers data of some consecutive measurements, e.g. 10 times, and posts their average to *SendProcess* to be sent to the respective cluster head. The node *ClusterHeader*, see Figure 5, calculates the weather index from the messages, which is held in *RR_Data* payload, in *SensorListenerProcess*. Then it sends the weather index which is encapsulated in process report called *PR_Data*, to *SinkNode* by the use of *ClientCon_Report* connection.

Each *SensorNode* can generate three classes of data packets:

1. Regular Report (*RR_Data*);
2. Query Response (*QR_Data*);
3. Emergence Report (*ER_Data*).

When a *SensorNode* detects an abnormal event in *RegularProcess*, e.g. smoke, it will immediately generate and send an *ER_Data* packet to the *ClusterHeader* containing the information related to the abnormal event without waiting for the rest of consecutive measurements, e.g. 10 times. *SensorListenerProcess* in *ClusterHeader*, uses a socket named *Pscok_ER* defined in *EmergencyThread* to transmit the emergency report by calling *AlarmCall* PT_Thread_Call.

The *QR_Data* packet is only transmitted to *SinkNode* when the sink asks *ClusterHeader* for the current measurement aspects defined in *QueryData*, e.g. temperature, which belongs to respective cluster. To implement this feature, *Sensor_Port* numbered '32900' is opened by *QueryProcesses* in *SensorNode* and *ClusterHeader* to accept instant data queries. *CurrentMeasurement* is sent back to *ClusterHeader* when a client connection is initiated. Then, *ClusterHeader* aggregates minimum, maximum, and average of the measurements from *SensorNodes* and constructs *QR_Data* to send the *SinkNode* via previously opened *ServerCon_Query*.

The resulting modeling artifact paves the way to do some analysis such as formal validation and verification of the system model based on the domain rules which can lead to less logical errors in the development of the system. Furthermore, the generation mechanism can be defined over this systematic modeling to generate the architectural code which can reduce the cost and number of errors for the embedded software development of IoT systems.

6 Related Work

Some MDE studies are provided in the literature to simplify the design, development, deployment and configuration of WSN and IoT systems. As there is considerable attention of researchers on applying MDE approaches on WSNs and IoT, two surveys ([11, 15]) studied systematic mapping of this domain to provide organized view of existing MDE approaches for WSN and IoT.

According to the above-mentioned surveys, among the reported studies, different MDE-based languages developed between the years 2007 and 2015. Furthermore, the modeling motivation of most of these studies is code-generation [11]. Among these studies, the code generation in nesC language for TinyOS is considered. However, none of these studies address Contiki OS.

LwiSSy is a Domain-specific Language (DSL) [6] to model Wireless Sensor and Actuators Network (WSAN) systems. LwiSSy allows the separation of responsibilities between domain expertise and network expertise. It also involves separation of structure, behavior, and optimization concerns by multiple views. In the study of [22], a model-driven architecture (MDA) is proposed which composes platform independent modeling (PIM), platform-specific modeling (PSM), and transformation rules for WSAN application development. PIM helps the domain experts to develop applications without knowing the programming details on WSAN platforms. PSM allows network experts to focus on the specific characteristics of their area of expertise without the need of knowing each specific application domain. Doddapaneni et al. [7] proposed a framework to model separately the software components and their interactions, the low-level and hardware specification of the nodes, and the physical

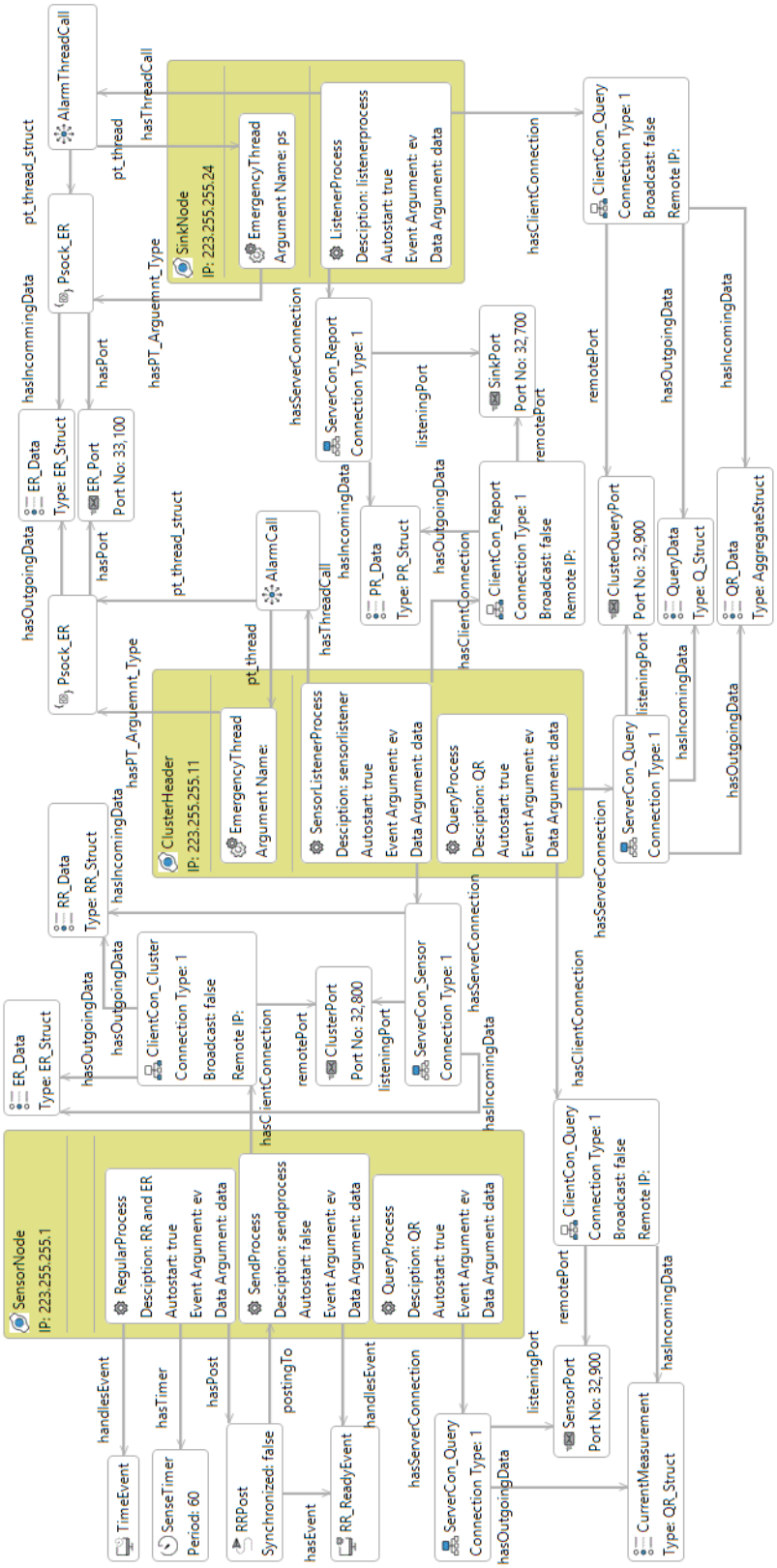

Figure 5 The instance model for the forest fire detection system.

environment where the nodes are deployed. This multi-view architectural approach [8] requires linking the models together for mapping models.

Tei et al. [25] propose a process that enables stepwise refinement to separately address data processing-related and network-related concerns. Their approach is similar to the modeling purpose of Rodrigues et al. [21] which especially takes into consideration the separation of responsibilities between domain experts and network experts. Rodrigues et al. [21] propose PIM for domain experts and PSM for network experts which can be used for the MDE of systems working on TinyOS. However, Tei et al. [25] have limited support for experts. PSM is not supported in their study and the experts simply create templates over platform independent models.

As also indicated in Section 1, our work contributes to the aforementioned noteworthy studies in the way of providing an MDE for developing IoT systems based on Contiki OS. To the best of our knowledge, currently no study addresses modelling WSNs or IoT systems according to the specifications of Contiki platform. The unique lightweight thread structure of Contiki makes it more popular in the implementation of new IoT systems and conduces the developers preferring Contiki instead of many other existing platforms such as TinyOS and SOS. Hence, providing a modeling language as proposed in this study can facilitate the efficient development of IoT systems based on this fashionable OS.

7 Conclusion

A metamodel and its supporting graphical modeling environment for the MDE of IoT systems are discussed in this paper. The metamodel includes all entities and relations required for modeling systems according to the event-driven mechanism of Contiki OS. Modeling based on the Contiki protothread architecture is also possible with using this metamodel. Moreover, a concrete syntax has been derived from this metamodel. Developers can use the proposed modeling language inside a graphical modeling environment to design the IoT systems as described in the conducted use case study. Both the modeling editor and the instance model discussed in this paper are available online with including required installation and configuration instructions at: `http://serlab.ube.ege.edu.tr/Bundles/ContikiOS_Editor.zip`.

The work discussed herein is our initial effort towards providing a full-fledged MDE environment for the development of Contiki-based IoT systems. Our next work will include design and implementation of model-to-text transformations which enable the automatic code generation from the modeled systems. This facility will be built-in for the existing Eclipse-based graphical modeling environment and hence the developers quickly achieve the required codes for deploying the designed systems on the embedded devices running Contiki OS.

References

1 Vahid Khalilpour Akram and Orhan Dagdeviren. Breadth-first search-based single-phase algorithms for bridge detection in wireless sensor networks. *Sensors*, 13(7):8786–8813, 2013.

2 Lan S. Bai, Robert P. Dick, and Peter A. Dinda. Archetype-based design: Sensor network programming for application experts, not just programming experts. In *2009 International Conference on Information Processing in Sensor Networks*, pages 85–96. IEEE, 2009.

[8] ISO/IEC/IEEE 42010:2011: `https://www.iso.org/standard/50508.html`.

3 Pruet Boonma, Yuthapong Somchit, and Juggapong Natwichai. A model-driven engineering platform for wireless sensor networks. In *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 671–676, 2013.

4 Moharram Challenger, Ferhat Erata, Mehmet Onat, Hale Gezgen, and Geylani Kardas. A model-driven engineering technique for developing composite content applications. In Marjan Mernik, José Paulo Leal, and Hugo Gonçalo Oliveira, editors, *5th Symposium on Languages, Applications and Technologies (SLATE'16)*, volume 51 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–10, 2016.

5 Orhan Dagdeviren and Vahid Khalilpour Akram. An energy-efficient distributed cut vertex detection algorithm for wireless sensor networks. *The Computer Journal*, 57(12):1852–1869, 2014.

6 Priscilla Dantas, Taniro Rodrigues, Thais Batista, Flavia C. Delicato, Paulo F. Pires, Wei Li, and Albert Y. Zomaya. LWiSSy: a domain specific language to model wireless sensor and actuators network systems. In *4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 7–12. IEEE, May 2013.

7 Krishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. A model-driven engineering framework for architecting and analysing wireless sensor networks. In *Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)*, pages 1–7. IEEE, June 2012.

8 Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *International conference on Embedded networked sensor systems – SenSys'06*. ACM Press, 2006.

9 Adam Dunkels, B. Gronvall, and Thiemo Voigt. Contiki – a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. IEEE, 2004.

10 Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *4th international conference on Embedded networked sensor systems*, pages 29–42. ACM, 2006.

11 Fatima Essaadi, Yann Ben Maissa, and Mohammed Dahchour. MDE-based languages for wireless sensor networks modeling: A systematic mapping study. In *Advances in Ubiquitous Networking 2*, pages 331–346. Springer, 2017.

12 Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, September 2013.

13 Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *3rd international conference on Mobile systems, applications, and services – MobiSys'05*, page 163. ACM Press, 2005.

14 Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *ACM SIGOPS operating systems review*, 34(5):93–104, 2000.

15 Ivano Malavolta and Henry Muccini. A study on MDE approaches for engineering wireless sensor networks. In *40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 149–157. IEEE, August 2014.

16 Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

17 Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of Things: vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, September 2012.

18 Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks with logical neighborhoods. In *First international conference on Integrated internet ad hoc and sensor networks – InterSense'06*, page 8. ACM Press, April 2006.

19 Shayan Nalbandian. A survey on Internet of Things: Applications and challenges. In *International Congress on Technology, Communication and Knowledge (ICTCK)*, pages 165–169. IEEE, 2015.

20 Tobias Reusing. Comparison of operating systems tinyos and contiki. *Network Architectures and Services*, 7:7–13, 2012.

21 Taniro Rodrigues, Priscilla Dantas, Paulo F Pires, Luci Pirmez, Thais Batista, Claudio Miceli, Albert Zomaya, et al. Model-driven development of wireless sensor network applications. In *IFIP 9th International Conference on Embedded and Ubiquitous Computing*, pages 11–18. IEEE, 2011.

22 Taniro Rodrigues, Flávia C. Delicato, Thais Batista, Paulo F. Pires, and Luci Pirmez. An approach based on the domain perspective to develop WSAN applications. *Software & Systems Modeling*, pages 1–29, September 2015.

23 Douglas C Schmidt. Model-driven engineering. *IEEE COMPUTER*, 39(2):25, 2006.

24 Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*. John Wiley & Sons, Ltd, Chichester, UK, November 2009.

25 Kenji Tei, Ryo Shimizu, Yoshiaki Fukazawa, and Shinichi Honiden. Model-driven-development-based stepwise software development process for wireless sensor networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(4):675–687, April 2015.

26 Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.

27 Ning Xu. A survey of sensor network applications. *IEEE communications magazine*, 40(8):102–114, 2002.

28 Liyang Yu, Neng Wang, and Xiaoqiao Meng. Real-time forest fire detection with wireless sensor networks. In *International Conference on Wireless Communications, Networking and Mobile Computing*, volume 2, pages 1214–1217. IEEE, 2005.