DSML4TinyOS: Code Generation for Wireless Devices

Hussein M. Marah
International Computer Institute, Ege University
Izmir, Turkey
hussein.marah@gmail.com

Raheleh Eslampanah
Electric and Electronics Engineering Department,
Izmir University of Economics
Izmir, Turkey
raheleh.eslampanah@ieu.edu.tr

Moharram Challenger
International Computer Institute, Ege University
Izmir, Turkey
moharram.challenger@ege.edu.tr

ABSTRACT

There are various operating systems and programming languages for programming the low power wireless devices in the Internet of Things (IoT). This heterogeneity makes the process of programming these devices time-consuming and complex. In our running study, we aim to deploy Model-driven Engineering (MDE) techniques in order to increase the level of abstraction to deal with this complexity. To this end, our purpose is to provide a platform-independent modeling framework for the development of IoT programs from developers' domain models. This will be realized by developing various platform specific modeling environments for different IoT operating systems and their programming languages. In this paper, we present DSML4TinyOS; a Domain-specific Modeling Language for TinyOS with which the developers can generate architectural code for low power wireless devices in nesC language. The meta-model, graphical concrete syntax, constraint checking rules, and model to text transformation rules of DSML4TinyOS are introduced and a case study is presented for the evaluation of the proposed DSML.

KEYWORDS

Model-driven Engineering, Domain-specific Modeling Language, Wireless Sensor Network, TinyOS, Code generation

1 INTRODUCTION

Wireless Sensor Networks (WSN) can be defined as a network of devices (mostly sensors) which are connected using IEEE 802.15 protocol (for low power WPAN) to perform information gathering and monitoring. WSNs have the potential to be used in a wide range of domains, such as health care, building, infrastructure, agriculture and security [13][14]. These communication networks can be used in Internet of Things where a path of sensors can collect information from a large area and send it through sink node(s) to the Internet via gateways, see Figure 1 [14].

A WSN is a complicated system for communication where a considerable number of devices (i.e., sensors) collect data and send it through channels of wireless communication [3]. Although, advances in the developing applications for low-power and low-cost micro-controllers play a vital role for implementing the WSN, the scarcity of specialized developers of these application who possess the required knowledge and the enough experience for such systems, limits the ability of implementing such network systems simply and quickly [9]. Also, there are many operating systems (such as TinyOS [13], ContikiOS [8], RIOT [2], FreeRTOS [23]) and

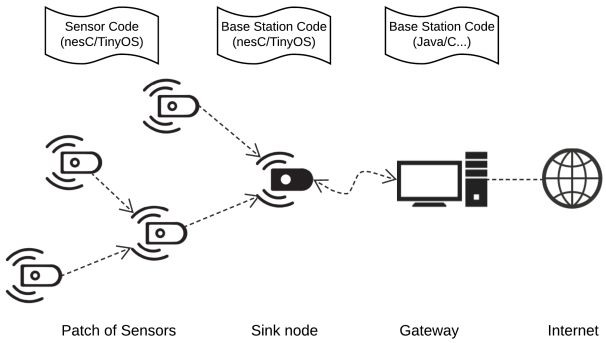

Figure 1: The typical Wireless Sensor Network Architecture

programming languages for implementing these systems. This heterogeneity makes the programming of these devices even more time-consuming and complex.

As a possible solution, in our running study, we intend to deploy and use Model-driven Engineering (MDE) approach and its techniques in order to increase the abstraction level in the process of software development for these IoT systems, see Figure 2. To this end a Platform Independent Modeling (PIM) environment, called DSML4WSN, will be provided to model the WSN based communication for IoT systems independent of any platform, operating system, or programming language. According to MDE principles, the PIM level models can be automatically transformed to the Platform-specific Models (PSM), such as DSML4TinyOS in Figure 2. PSMs provide a modeling platform to design the model of problem domain using platform-specific concepts and generate the architectural code in the target platform.

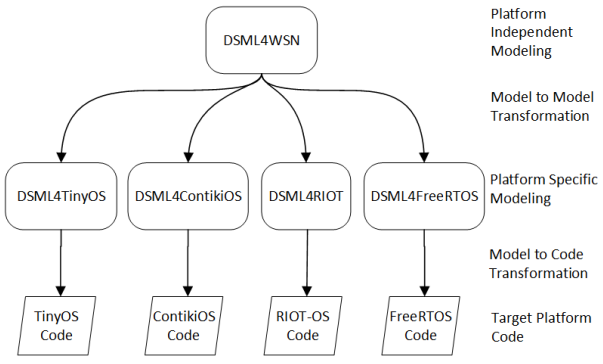

Figure 2: Overview of the proposed MDE approach

CEUR-WS.org/Vol-2245/mde4iot_paper_3.pdf

In this paper, we present the DSML4TinyOS, a generative modeling language for development of WSN programs in TinyOS framework (nesC language). TinyOS is a component-based operating system for embedded devices, especially low-power and low-memory wireless devices [12][11] to implement low power IoT systems. Using DSML4TinyOS, the IoT developers can model the communication system of their low power wireless devices and generate the architectural code for TinyOS framework.

To this end the abstract syntax of DSML4TinyOS is presented with a metamodel. The concrete syntax is provided by mapping the concepts in a metamodel with graphical notations using Sirius framework [26]. Also the graphical concrete syntax is empowered by some constraints checking to control the domain rules on the developers' instance models. This will result in improving domain models which in turn leads to high quality code generation. Finally, the instance models are transformed automatically to the target code in nesC. This development process requires less time and effort and can lead to less errors comparing to manual development process.

The rest of this paper is organized as follows: Section 2 briefly discusses TinyOS. The syntax and semantics of the proposed language, DSML4TinyOS, are presented in Section 3. A case study is used in Section 4 to evaluate the proposed DSML. The related work is reported in Section 5 and the paper is concluded in Section 6.

2 TINYOS

TinyOS is a lightweight, flexible, free and open source operating system which began as a project in University of California, Berkeley [13][14]. TinyOS has three main layers that work together to make the operating system efficient: application layer (TinyOS/nesC), services layer (actuating, sensing and communication) and finally the hardware layer.

The main programming language for TinyOS is nesC. In fact, nesC is a programming language for networked embedded systems. Although, it is based on C, its own characteristics distinguish it from C language. The nesC is designed for devices that have low memory. Many applications can be installed inside 16KB of memory and the whole core OS can fit in about 400 bytes [13].The features of nesC focus on the notion of Components that encapsulate a certain set of services, specified by Interfaces. Figure 3 shows the structure of TinyOS applications in nesC.

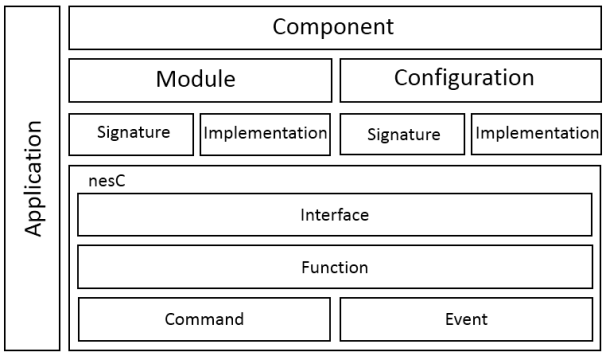

Figure 3: Structure of TinyOS applications

The Application connects Components by wiring [14]. A Component contains Interfaces, and it can use the same Interface type any time as long as each instance has a different name. There are two types of Components; Module and Configuration. The Components may use or provide Interfaces which respectively can have events and commands as shown in Figure 3.

TinyOS is one of the major operating systems for wireless sensor networks. Due to the specific programming style of TinyOS, Component based programming, some challenges have been raised in the development of related programs. One of these challenges is the specific approach for nesC programming which is depended on Components to form an Application [11]. This challenge can be addressed by using model-driven techniques and extracting the Components and Configuration patterns and reusing them as model to text transformation templates. On the other hand, due to the limited services and primitives that sensors may introduce, such as sending or receiving packets, these services can be used repeatedly in programs. The code for these services are structurally similar in different parts of the program, which makes it very suitable for applying model-driven techniques that can lead to increase productivity and decrease the cost of development [20]. With this motivation, in this study, a generative DSML is proposed for TinyOS based WSN development.

3 THE SYNTAX AND SEMANTICS OF DSML4TINYOS

To develop a DSML for TinyOS, the first step is to design a metamodel that specifies the abstract syntax of the language [27]. After the process of analyzing and examining the structure of TinyOS and its programming language (nesC) using different examples, we could conceptualize the elements and their relations in TinyOS and represent them in the proposed metamodel. This metamodel plays the role of abstract syntax for the language. It abstracts the new language from the details of configurations and setup for the components in TinyOS, among the other abstractions. Figure 4 shows the metamodel for TinyOS. This metamodel is designed using Eclipse Modelling Framework (EMF) and encoded in Ecore format [22].

As pointed out earlier, Module and Configuration are two types of Components in TinyOS, and they have a signature (Module_Signature, Configuration_Signature) which provide or use Interfaces, although they mainly differ in their implementation. The module implementation (Module_Implementation) part consists of nesC code that has similar syntax to the C language. The Module part of the code declares variables and functions, calls functions and compiles the code, while Configuration implementation (Configuration_Implementation) part consists of nesC code (wiring code) that wires and connects Components of the Application together. The other elements are sub-elements of Module implementation (Module_Implementation) or Configuration implementation (Configuration_Implementation) Components. Particularly, functions (events and commands) are mainly defined in Module implementation (Module_Implementation) while Wiring code is only defined in Configuration implementation (Configuration_Implementation).

After designing the metamodel, the second step is developing the graphical concrete syntax (GCS) which paves the way for a

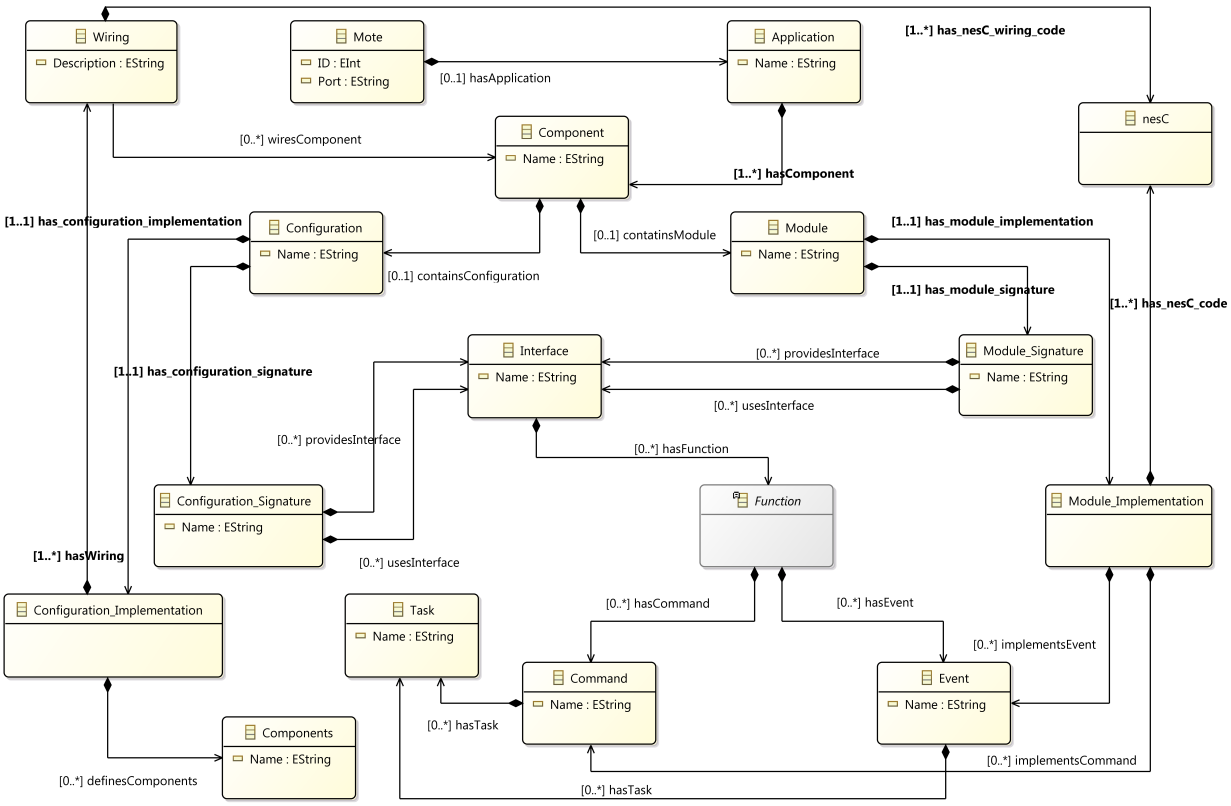

Figure 4: A domain-specific metamodel for TinyOS

graphical editor. To implement this editor, some graphical notations are selected for the elements of the metamodel, see Figure 5. The first notation, which represent a Mote, indicates to a node that the wireless program will be uploaded in. A Mote has an ID number, as an attribute, defined while installing TinyOS application, and a port number indicating the COM port that is used to get data. The Applications (in nesC language) are built out of Components containing bidirectional and well-defined Interfaces. Also, the Component and Interface notations are defined to represent the related elements in nesC Applications. Figure 5 illustrates the notations selected for different elements of TinyOS metamodel.

The graphical tool supporting DSML4TinyOS is implemented using Sirius modeling tool. It is part of an Eclipse Modelling Framework [10] that allows users to easily create their own graphical concrete syntax. In addition, to consider the static semantics in the modeling environment, domain rules, such as name conflicts, are implemented as constraints to check the developers' instance models. These constraints are implemented in AQL (Acceleo Query Language) [1]. The static semantics improve the quality of models and accordingly the quality of generated code.

The final step of developing DSML4TinyOS is writing the model to code/text (M2T) transformation rules to generate the artifacts in target platform, TinyOS. M2T transformation rules are a set of

Concept	Notation	Concept	Notation
Mote		Application	
Module		Configuration	
Components		Interface	
Module_Signature		Configuration_Signature	
Component		nesC	
Function		Wiring	
Event		Command	
Configuration Implementation		Module Implementation	
Task			

Figure 5: Concepts and their graphical notations for the TinyOS modeling environment

templates which are extracted from nesC codes. These templates are implemented in an template engine called Acceleo [2]. Acceleo is a pragmatic implementation of the Object Management Group (OMG) that uses Model to Text Language (MTL) standard. Acceleo

[1] Acceleo Query Language, https://www.eclipse.org/acceleo/documentation/aql.html

[2] Acceleo M2T Language, https://www.eclipse.org/acceleo/

generate.mtl | MultihopOscilloscopeC.nc | MultihopOscilloscopeAppC.nc

```
[template public generateMS(ms : Module)]
#include "Timer.h"
[if (ms.has_module_signature.Name<>'invalid')]
Module [ms.has_module_signature.Name/] @safe() {
[generateModuleSignature(ms)/]
}
[/if]
implementation {
[generateModuleImplementationEvent(ms)/]
[generateModuleImplementationCommand(ms)/]
}
[/template]
[template public generateModuleSignature(gms : Module)]
[for (interface : Interface | gms.has_module_signature.usesInterface)]
[if (interface.Name<>'invalid')]
uses interface [interface.Name/];
[/if]
[/for]
[/template]
[template public generateModuleImplementationEvent(gmie : Module)]
[for (event : Event | gmie.has_module_implementation.implementsEvent)]
[if (event.Name->oclIsInvalid()._not())]
    event void [event.Name/] {
  /**
   * event content
   */
[if (event.Name='message_t* Receive.receive()')]
    [generateModuleImplementationEventTask(gmie)/]
[/if]
 }
[/if]
[/for]
[/template]
```

Figure 6: Part of the M2T transformation in Acceleo

has been used to perform code generation from the instance model in DSML4TinyOS. Figure 6 and 7 show respectively the excerpt of the code generation rules in Acceleo, and the generated code for an instance model that will be discussed later in the next section.

generate.mtl | MultihopOscilloscopeC.nc | MultihopOscilloscopeAppC.nc

```
#include "Timer.h"
Module MultihopOscilloscopeC @safe() {
uses interface Boot;
uses interface StdControl as RoutingControl;
uses interface Send;
uses interface Receive as Snoop;
uses interface Receive;
uses interface AMSend as SerialSend;
uses interface SplitControl as SerialControl;
uses interface CollectionPacket;
uses interface RootControl;
uses interface Queue<message_t *> as UARTQueue;
uses interface Pool<message_t> as UARTMessagePool;
uses interface Timer<TMilli>;
uses interface Read<uint16_t>;
uses interface Leds;

}
implementation {
    event void Boot.booted() {
  /**
   * event content
   */
 }
    event void RadioControl.startDone() {
  /**
   * event content
   */
 }
    event void SerialControl.startDone() {
  /**
   * event content
   */
```

Figure 7: An excerpt of the generated code

4 CASE STUDY: FIRE DETECTION SYSTEM

There are many applications using WSN in different areas. One of the useful applications is Fire Detection Systems [28][6] which can be needed in different environments. In our previous work [1], a fire detection system has been implemented for libraries using an IoT system with a WSN. The system uses wireless sensors programmed in TinyOS. The study is extended with multihop communication and RPL protocol (IPv6 Routing Protocol for Low-Power and Lossy Network) to collect and send data from one sensor to another until reaches the final destination, which is the sink node, connected to the computer (the gateway). To this end, the application used in all sensors is 'MultihopOscilloscope' [17][4]. On the other hand, a Java application on the computer processes the data and triggers an alarm or calls back another IoT devices, if the temperature exceeds a threshold.

In this study the fire detection system is used as a case study to evaluate DSML4TinyOS. To this end, the system is modeled and developed using the proposed language. The TinyOS system for fire detection system is designed as an instance model confirming to TinyOS metamodel in DSML4TinyOS, see Figure 8. The instance model describes all the elements, attributes and relations required for specifying the fire detection system in nesC. The model is checked by constraints implemented in DSML4TinyOS to gain a semantically correct model.

The resulting instance model was used to generate code by applying the transformation rules provided in DSML4TinyOS. The generated code contains the essential elements of the application, and it has the main structure of the TinyOS application; see Figure 7 for excerpt of the generated code. The generation process includes two files. (MultihopOscilloscopeC and MultihopOscilloscopeAppC). The first file is for Module contents while the second file is for Configuration which contains the wiring code. The generated artifacts are syntactically and semantically error-free. Also, as they are generated automatically, they save developer time considerably. The developer can add the delta code to have fully functional software.

5 RELATED WORK

There are some MDE studies in the literature for WSN development to ease the design, development, and deployment of WSN systems. Also, we can say that there is considerable interest and orientation of researchers on applying MDE approaches on WSNs, especially in the scope of IoT [5] [16]. In general, non of these studies address the development of IoT systems with WSNs specifically.

In [15], a framework based on Simulink, State-flow and Embedded Coder is proposed. The goal of this framework is to generate code of the application for WSN operating systems. In this platform an application is modeled in a high level of abstraction, the code is generated automatically after it is simulated using realistic topologies for the network. The authors claim that after the phase of modeling and simulation, the framework can generate the code of the applications from the simulated model for different target operating systems.

A modeling framework was proposed in [7] which allows the developers to model the architecture of WSN software and the specification of the low-level hardware of WSN nodes separately. The framework can use the designed models in respect to generating

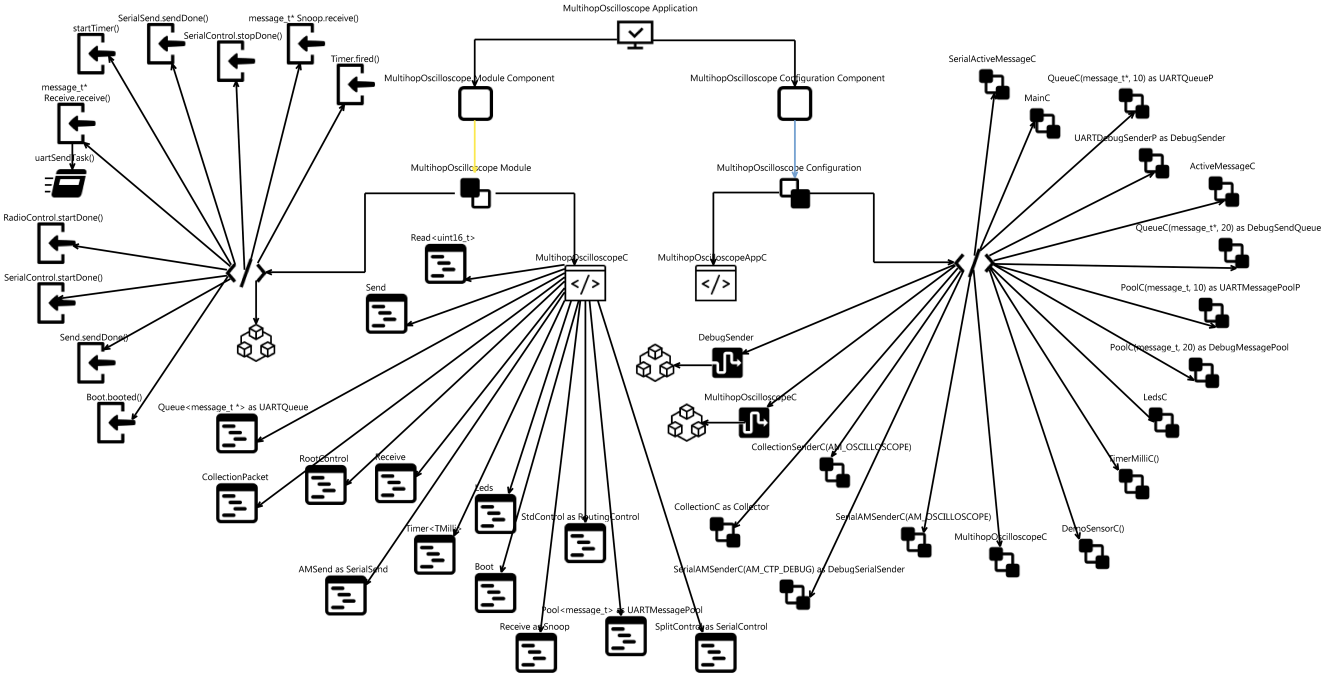

Figure 8: The instance model for the fire detection system

code for analysis purposes. The framework employs a multi-view architectural approach to model separately software Components and their interactions as well as the physical environment where the WSN devices (nodes) are actually implemented.

The study in [21] proposes a Model-driven Development (MDD) approach for prototyping and optimization of WSN applications. This work provides a set of modeling languages, which describe an application at three abstraction levels, and transformation rules. The provided modeling languages transform the described models into a concrete one.

Also, the study in [18] introduces a method to build WSN applications by using the model-driven approach to separate the concerns of two levels of requirements. The first level is domain experts with no experience on WSN systems but they will contribute in applications development process, and the second level is network experts with required knowledge in WSN platform but no knowledge on the application domain. This method supports the reuse of software artifacts: an application model can be reused on various sensor platforms while a platform model can be reused for different WSN applications.

The paper [24] addresses the problem of optimizing the trade-off between various constraints like power consumption and memory usage in the process of application design. The proposed approach integrates Evolutionary Algorithms (EA) with MDD approach. Various metamodels of the system to be developed are evolved and generated, terms of evaluating the trade-off between performance value and different constraint criteria determine the process of the optimal model selection.

The study reported in [25] presents a software engineering approach that uses the models of Coloured Petri Net (CPN) as the starting point to develop protocol software for the TinyOS system. The approach depends on five steps of refinement process taking a platform-independent CPN model and progressively refining it to match the structure of the target platform. The result will end up with a refined model that enables automatic code generation.

The aforementioned studies apply MDD and MDE techniques on WSN applications for different purposes, however, none of them aim to do constraint checking and automatic code generation for TinyOS.

As one of the recent studies, in [19], the authors propose a DSL for WSN application development. However, their proposed language, SenNet, is a textual DSL. We believe that a graphical model is more comprehensible and provides an environment for to design the system in a more abstraction level. Also, our study intends to provide PIM level modeling of WSNs for IoT systems and generate PSM level models and then artifacts for different platforms. To the best of our knowledge, this is not addressed in the literature.

6 DISCUSSION AND CONCLUSION

This paper presents a MDE based development environment for WSNs applications using TinyOS. The development environment, called DSML4TinyOS, provides graphical modeling and M2T transformation rules with which the final artifacts can be generated. The proposed approach can increase the level of abstraction of TinyOS application development, ease the development process, and reduce the number of errors. These features increase the productivity and decrease the development time.

The development of DSML4TinyOS has been done in three phases: the design of the metamodel for domain (TinyOS), the development of the GCS to provide the modeling environment, and

providing the M2T transformation rules to generate applications' architectural code automatically.

Based on the evaluation done by the fire detection case study, the proposed modeling environment helps the developers to save the time in the development of the applications by automatic generation and reducing the number of errors. Even though the developers must have basic knowledge of the TinyOS and WSN (such as which Components to use and which Interfaces are to be used or to be provided in the application's Component) to implement the system, the developers do not work in code level but in the higher level of abstraction. So, for the development, the developer needs to have the domain level knowledge only, not the platform level information.

However, one challenging issue is the amount of details of modelling environment. More detailed modeling environment can lead to more code generation, and this may make the modeling environment more difficult and distract from the benefits of the initial idea of increasing the level of abstraction. On the other hand, a very abstract modeling may not give a proper productivity in the generation.

As our future work, we intend to extend this work to include PSMs for other WSN platforms. Then, by extracting the common vocabulary, a platform independent modeling environment will be developed. In this way, the developers can work in the computation level, not in the platform level.

ACKNOWLEDGEMENT

This study is realized in the scope of the Scientific Research Project No 17-UBE-002 at EGE University, Izmir-Turkey. Also, the authors would like to acknowledge Izmir University of Economics for their supports.

REFERENCES

[1] Sadık Arslan, Moharram Challenger, and Orhan Dagdeviren. 2017. Wireless sensor network based fire detection system for libraries. In *Computer Science and Engineering (UBMK), 2017 International Conference on*. IEEE, 271–276.

[2] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas C Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. IEEE, 79–80.

[3] Pruet Boonma, Yuthapong Somchit, and Juggapong Natwichai. 2013. A Model-Driven Engineering Platform for Wireless Sensor Networks. *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (2013). https://doi.org/10.1109/3pgcic.2013.115

[4] Torsten Braun, Andreas Kassler, Maria Kihl, Veselin Rakocevic, Vasilios Siris, and Geert Heijenk. 2009. Multihop wireless networks. In *Traffic and QoS management in wireless multimedia networks*. Springer, 201–265.

[5] Damien Cassou, Julien Bruneau, Charles Consel, and Emilie Balland. 2012. Toward a tool-based development methodology for pervasive computing applications. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1445–1463.

[6] Zenon Chaczko and Fady Ahmad. 2005. Wireless sensor network based system for fire endangered areas. In *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*, Vol. 2. IEEE, 203–207.

[7] Krishna Doddapaneni, Enver Ever, Orhan Gemikonakli, Ivano Malavolta, Leonardo Mostarda, and Henry Muccini. 2012. 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*. IEEE Press, 1–7.

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

[9] Caglar Durmaz, Moharram Challenger, Orhan Dagdeviren, and Geylani Kardas. 2017. Modelling Contiki-Based IoT Systems. In *OASIcs-OpenAccess Series in Informatics*, Vol. 56. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Eclipse Foundation. 2018. Eclipse Modeling Framework (EMF). https://www.eclipse.org/modeling/emf/

[11] David Gay, Philip Levis, Robert Von Behren, Matt Welsh, Eric Brewer, and David Culler. 2014. The nesC language: A holistic approach to networked embedded systems. *Acm Sigplan Notices* 49, 4 (2014), 41–51.

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

[13] P.a. Levis. 2006. TinyOS: An Open Operating System for Wireless Sensor Networks (Invited Seminar). *7th International Conference on Mobile Data Management (MDM06)* (2006). https://doi.org/10.1109/mdm.2006.151

[14] Philip Levis and David Gay. 2009. TinyOS Programming. (2009). https://doi.org/10.1017/cbo9780511626609

[15] Mohammad Mostafizur Rahman Mozumdar, Francesco Gregoretti, Luciano Lavagno, Laura Vanzago, and Stefano Olivieri. 2008. A framework for modeling, simulation and automatic code generation of sensor network application. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on*. IEEE, 515–522.

[16] Pankesh Patel and Damien Cassou. 2015. Enabling high-level application development for the Internet of Things. *Journal of Systems and Software* 103 (2015), 62–84.

[17] TinyOS Release Repository. 2018. TinyOS Documentation. https://github.com/tinyos

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

[19] Aymen J Salman and Adil Al-Yasiri. 2016. Developing domain-specific language for wireless sensor network application development. In *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*. IEEE, 301–308.

[20] Douglas C Schmidt. 2006. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY*- 39, 2 (2006), 25.

[21] Ryo Shimizu, Kenji Tei, Yoshiaki Fukazawa, and Shinichi Honiden. 2011. Model driven development for rapid prototyping and optimization of wireless sensor network applications. In *Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications*. ACM, 31–36.

[22] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.

[23] Su Lim Tan and Bao Anh Tran Nguyen. 2009. Survey and performance evaluation of real-time operating systems (RTOS) for small microcontrollers. *IEEE Micro* (2009). https://doi.org/10.1109/mm.2009.56

[24] Nguyen Xuan Thang and Kurt Geihs. 2010. Model-driven development with optimization of non-functional constraints in sensor network. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*. ACM, 61–65.

[25] Vegard Veiset and Lars M Kristensen. 2013. *An Approach to Semi-Automatic Code Generation for the TinyOS Platform using Coloured Petri Nets*. Ph.D. Dissertation. Master's thesis, Bergen University College.

[26] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. 2014. Sirius: A rapid development of DSM graphical editor. In *Intelligent Engineering Systems (INES), 2014 18th International Conference on*. IEEE, 233–238.

[27] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. 2013. *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

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