

A Domain-Specific Modeling for Dynamically Reconfigurable Environmental Sensing Applications

Mohammad Fajar¹, Kenji Hisazumi², Tsuneo Nakanishi¹ and Akira Fukuda¹⁺

¹ Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan

² System LSI Research Center, Kyushu University, Japan

Abstract. The importance of reconfiguration service in Wireless Sensor Network (WSN) has driven many researchers to develop various ways to reduce incurred costs in software maintenance on sensor nodes. However, the lack of utilization of software engineering methodologies in previous studies can cause numerous efforts to configure the WSN for each site and modification of the system to introduce additional features. In this paper, we present a high level of abstraction model to support a reconfiguration mechanism for environmental sensing applications that enable developers to design their applications to be more flexible and reusable without relying on a specific platform. Preliminary evaluation on a case study indicated that the use of the proposed model was capable of increasing productivity in development time about 6-8 times as fast as that of the handwriting approach. Evaluation of quality of the generated codes in a simulated environment showed the reconfiguration process for some properties of tasks could be performed without decreasing the network performance significantly.

Keywords: domain specific modeling, wireless sensor network, environmental sensing, reconfiguration

1. Introduction

A wireless sensor network is a large-scale ad-hoc and multi-hop network, consisting of small devices such as sensor nodes, routers and sink node. These devices use limited resources including limited storage, computation and communication capabilities as well as severely constrained power supplies. Most of WSN applications are deployed in harsh unattended environments and difficult to be reached by human, so that the ability to update the program code installed on WSN nodes is an important feature in such system, necessary not only for correcting errors but also for being able to adapt the running software to changed environmental conditions [9]. In order to satisfy the requirements, many studies have been done in the area of WSN reprogramming such as over the air programming [5][8][10][11]. However, so far, most of these studies lack software engineering concepts; instead, they are more focused on implementation issues of applications and platform-specific solutions, which tend to be constructed one by one and possibly in a site-specific manner, without a ground view of integration and future evolution of various applications. The lack of utilization of software engineering methodologies in the works can cause a considerable amount of effort to configure the system for each site and modification of the system to introduce additional features [3]. Therefore, in this paper we present a domain-specific modeling (DSM) to develop reconfigurable environmental sensing applications. The model provides high level of abstraction to out of specific platforms and code generators to produce application codes automatically.

The outline of this paper is as follows: Section 2 explains a DSM concept briefly. Section 3 presents our proposed model including overview of the model, levels of abstraction, reconfiguration model and code generation. Section 4 shows preliminary results and our concluding remark is presented in Section 5.

⁺ Corresponding author. Tel/Fax.: + 81(92-802-3644).
E-mail address: (fajar@f.ait.kyushu-u.ac.jp).

2. Domain-Specific Modeling (DSM)

In software engineering, developers generally differentiate between modeling and coding activities. Models are used for designing systems, specifying required functionality and understanding them better, while codes are written to implement the designs. As a software engineering methodology, DSM has two basic characteristics [6]. First, raises the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules from a specific problem domain. Second, generates final products in a chosen programming language or other form from these high level specifications. The generating final products involves software generator to produce target codes automatically [1][4]. With these characteristics, the DSM offers several fundamental benefits over general purposes modeling and manual approaches, such as increased productivity, improved quality, and share expert knowledge. Fig.1 shows the basic architecture of DSM.



Fig. 1: Basic architecture of DSM

The objective of the DSM solution in this paper is to make the development of reconfigurable environmental sensing applications possible for a fundamentally larger developers and domain experts: people having little or no experience in WSN programming. The main idea is to get a situation where the developers can draw pictures of the application, define some input parameters using dialog interface, making relation between components and then generate the application codes to be executed in a target device.

3. A DSM for Reconfigurable Environmental Sensing

The model in this paper is used to develop reconfigurable environmental sensing applications. It supports star, tree and mesh architecture, TinyOS platform and also possible for different platforms. The language provides graphical components, relationships and rules to make applications development as easy as possible for software engineers or domain experts. Fig.2 shows an instance of the model with notation and its graphical symbols using problem domain such as sensing, data collection and node icons.

3.1. Levels of abstraction

For separation of concerns, we design our DSM in three levels of abstraction: logical level, physical and task allocation abstraction level. In [2] the Logical model provides required components for modeling application tasks such as sensing, sending and data collecting task as well as relationship components to connect them each other. Each task consists of a set of program instructions that run on a node. This logical model can be seen as a rooted tree with two directions. First, a direction for collecting environmental data, where data collection task as a root, processing or calculation tasks as its branches and sensing tasks as its leaf nodes. Data flow in this direction is started from leaf nodes (e.g. Temperature task) toward branch nodes (e.g. Average Calculation task) and end at the root (e.g. Data Collection task). The second direction is used for describing dissemination of new configurations. Data flow is started from root (e.g. Data dissemination task) and then goes to all branches and leaf nodes (e.g. Get task). Modeling activity in the logical level involves: defining a group of task, putting tasks to the group and decide their parameters, connecting each task to form their data flow. Fig.2 shows a logical level with some components and their relationship.

The physical model supplies required components to define a network architecture of application such as sensor nodes, intermediate and sink node. These nodes are organized into node groups. Each group consists of one or more nodes that have similar behavior, and often they have the same resources capabilities. Relationship at this level is used to form network architecture. The flow of sensing moves from sensor nodes through intermediate and ends at a sink, while the flow of disseminating is started from sink through intermediates or sensor nodes. The behavior of each node is defined by their tasks at logical level (will be decided in allocation model). Modeling work at this level involves: (1) defining node groups (sensor node, intermediate and a sink), (2) connecting each group to form a network architecture. Physical level in Fig.2

shows an example of tree architecture with two sensor node groups, two head node groups and a sink node. Our DSM generator translates this physical model as a tree with sink node as its root (Node ID=0).

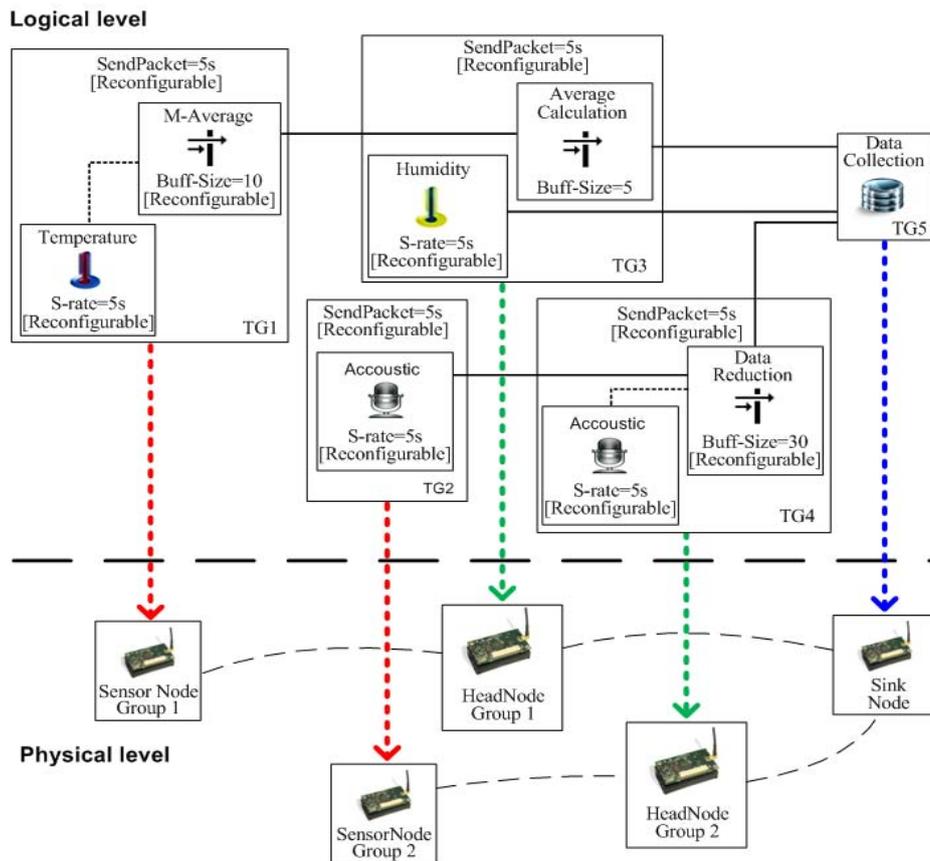


Fig. 2: An instance of environmental sensing reconfiguration model

After logical model and physical model have been defined. The next phase is task allocation. This model provides required components for making a relationship between logical components and physical model components. In this phase, developers draw vertical relationships from a group of task at logical level to a node group at physical level. For example, in Fig.2 we use a vertical relationship to allocate three tasks in TG1 (sensing temperature, moving average and sending task) to Sensor Node Group 1, this allocation means that all of nodes in the Sensor Node Group 1 perform these tasks. The sequence of execution of the tasks is determined by intra-relationship. In TG1, each node will sense temperature and put the sensed data into moving average's buffer; then sensor nodes perform moving average task to calculate the data in the buffer and finally, send the calculation result to the network.

3.2. Reconfiguration Model

This model provides the required components to prepare a reconfiguration mechanism for environmental sensing applications without reprogramming the entire codes of sensor nodes. Reconfiguration means that the software components of WSN can be swapped out for others on demand, including the values of their properties. In this paper, we specifically concentrate on reconfiguring the properties of components or tasks at the logical level. Reconfiguration of the sensor node is done when we need to change the properties of tasks that are running on sensor node. In environmental sensing applications, we have some properties that usually need to be reconfigured, among others rate (including sampling and sending rate), size (including buffer size of some processing or calculations), and window (including sliding number). Fig. 3 shows the full metamodel for the WSN reconfiguration. When applications employ them, then we will need to prepare a reconfiguration mechanism for the applications to adapt with environmental conditions or user requirements.

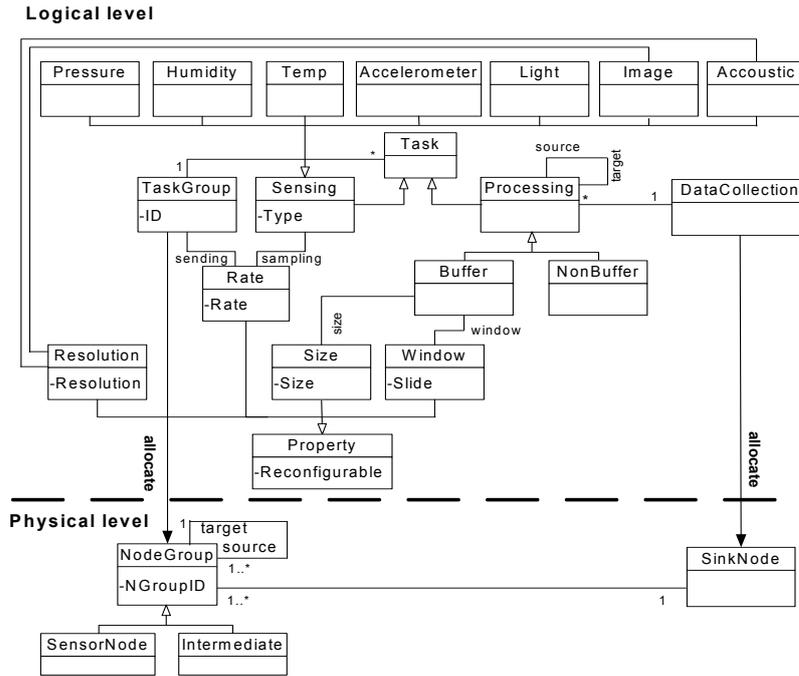


Fig. 3: Reconfigurable environmental sensing metamodel

In the reconfiguration model, each component supplies a reconfigurable option as a property, whether the components can be reconfigured or not. Developers have to use this option to prepare a reconfiguration mechanism for an application. For example, when we create a group with SendPacket task as its default task, we define the value of sending period property for the task, and then select a reconfigurable or non-reconfigurable option for the sending period. We can add another tasks to the group and repeat the procedure to decide whether their properties can be reconfigured or not. When we employ a reconfiguration model for some properties of tasks that need to be modified (e.g., buffer size), it is important to specify the behaviors of the tasks. The behaviors of a reconfiguration model are a special handling task that needs to be prepared when the properties of the tasks will be reconfigured. For example: in some environmental sensing applications, reading results of a sensor node can be stored into a buffer for pre-processing; the sensor node sends data to other nodes when the buffer is full and gets a calculation result. In this application, when we reconfigure the buffer size of calculation we should ensure the sensor node changes the expression of condition for sending data. If our target platforms support dynamic allocations, then after reconfiguring the buffer size, the new configuration should be used as a new value to trigger the sending task. Conversely, if the target platforms do not enable for dynamic allocations or it might be risky to use them (e.g. TinyOS), then the generator can prepare enough size (e.g., maximum size) for a buffer. Each time the sensor node receives a new buffer value. It must ensure the value is not greater than the possible maximum size for the buffer. Failing to deal with this kind of behavior would make the buffer overflow. In our case study (e.g., Average Calculation), the generator creates a maximum value for its buffer. When a node receives new buffer size, this value is used to change the buffer size and also will be used as a delimiter to start the sending process. If a buffer indicator reaches the delimiter, then sensor node calculates the content of the buffer and sends its result to the network. Each processing or calculation task has its own method for using the properties of the tasks, and we should handle these methods when preparing a reconfiguration mechanism.

Logical level in Fig.2 section 3.1 shows we create 5 task groups (TG1 to TG5). In TG1 we add 3 tasks, namely SendPacket, Temperature and Moving Average task. Each task can have one or more properties, such as sampling rate, sending period and buffer size. Each property has a reconfigurable value that may contain reconfigurable or non-reconfigurable options. If a property is marked as reconfigurable, it means we can change the value of the property on demand remotely. Conversely, if the property does not have a reconfigurable mark, it means the property cannot be changed remotely in the future using this mechanism.

Fig 2. Section 3.1 shows that all of properties in TG1, TG2, and TG4 can be reconfigured, while in TG3 the buffer size of Average calculation is non-reconfigurable. In this case, the value of the buffer size will not be changed in the future when sensor nodes have been deployed.

In transformation process, DSM generator reads all tasks at logical level including reconfigurable option of their properties and generates final targets with reconfiguration mechanism for the properties. It generates codes for sensor node, intermediate and sink node respectively and produces two additional functions for sensor and intermediate codes which functioning receiving configuration values and updating the new configuration. While for sink node, the generator creates an additional function for reading new configurations from application side and disseminating them to WSN.

3.3. nesC Components model and code generator

In this work, we use nesC/TinyOS as a final target of the model. Both of them are popular development environments for WSN applications. A nesC program is a collection of components. These components are defined by using module and configuration concepts. Module implementation sections consist of nesC code, declares variables and function, call functions, etc. Configuration implementation sections consist of nesC wiring code, which connects components together. Every component is in its own source file, and there is one-to-one mapping between component and source file names. For example, the file TimerMilliC.nc contains the nesC code for the component TimerMilliC. Fig.4 shows nesC main components used in this work. MainC, LedsC, ActiveMessageC, TimerMilliC, CollectionC and DisseminationC components are nesC predefined modules, while SensorC components are self predefined modules, it depends on user requirements. For data collection components we use collection tree and P2P protocol as multi-hop protocol while for dissemination we use dissemination protocol. The last component in the nesC component model is DataCollection. The component contains application codes that are generated by the generator automatically.

The final step is obtaining final application codes via code generation automatically. A nesC generator in this work has been implemented using the Meta Edit Report Language (MERL) script. The generator reads the application model and produces nesC codes for each node group, intermediate nodes and sink node. The core codes are generated into two files (nesC configuration and module file); the nesC configuration file contains wiring components and the nesC module file contains the behavior of the application. The generated solution, obtained from the corresponding model in this study and the nesC component model in the Fig.4 have been successfully compiled and tested on TinyOS simulator.

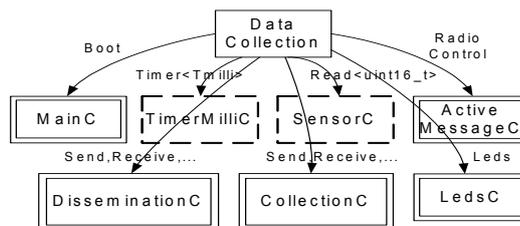


Fig. 4: nesC component model for data collection and code reconfiguration

4. Results

We developed and measured two example applications: Application-1 without reconfiguration services and Application-2 with reconfiguration support. A preliminary evaluation showed the use of our proposed model could reduce the time of development significantly, a productivity improvement of about 6-8 times over handwriting approach. (See TAB. 1)

Tab.1 : development time between handwriting code and proposed dsm

Application	Number of lines	Handwriting code		Proposed DSM		
		Number of functions	Time (minutes)	Number of objects	Number of relationships	Time (minutes)
Application-1	247	22	30.12	10	10	5.18
Application-2	325	27	40.26	10	10	5.48

For evaluating the quality of generated codes, we ran the codes on TinyOS Simulator (TOSSIM) [7] using two sensor nodes, one intermediate and sink node. They were connected in a multi-hop tree network. In this evaluation, the network performances between the two applications are not different significantly and the performance of application-2 is acceptable. (See TAB. II)

Tab.2: delay overhead when collecting data and performing reconfiguration

From Node (Node ID)	To Node (Node ID)	Average delay in Application-1 (ms)	Average delay in Application-2 (ms)	Delay overhead (ms)
Sensor Node(3)	Intermediate(1)	6.71925	6.86926	0.15001
Sensor Node (2)	Intermediate(1)	6.73629	6.84183	0.10554
Intermediate(1)	Sink (0)	6.83171	6.74156	0

Packet loss ratio of Application-1=0.1% and Application-2= 0.2%

5. Conclusion

We present a high level of abstraction model to support a reconfiguration mechanism for environmental sensing applications. This model enables developers to design their applications to be more flexible and reusable without relying on a specific platform. Our preliminary evaluation on a case study indicated that the use of the proposed model was capable of increasing productivity in development time about 6-8 times as fast as that of the handwriting approach. Also, evaluation of the quality of the generated codes in the simulation environment showed the reconfiguration process for some properties of tasks could be performed without decreasing the network performance significantly. For future work, the model can be improved for reprogramming mechanisms to replace part or all of the code and we also need to evaluate the proposed DSM further with various metrics.

6. References

- [1] K. Czarnecki, U. W. Eisenecker, "Generative Programming, Methods, Tools, and Applications," Addison-Wesley. 2000.
- [2] M. Fajar, K. Hisazumi, T. Nakanishi and A. Fukuda, "Applying Domain Specific Modeling for Environmental Sensing Using Wireless Sensor Network," Asian Journal of Information Technology, Vol.10,2011, pp.296-305.
- [3] M. Fajar, T. Nakanishi, S. Tagashira and A. Fukuda, "Introducing software product line development for wireless sensor/actuator network based agriculture systems," In proceedings of the AFITA ,International Conference on Quality Information for Competitive Agricultural Based Production System and Commerce, October 3-7, 2010, Bogor Agriculture University, Indonesia, pp: 83-88.
- [4] J. Greenfield et al, "Software Factories, Assembling Applications with Patterns, Models, Frameworks, and Tools," Wiley publishing Inc. 2004.
- [5] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes," in Proc. IPSN 2008, pp. 457 - 466, 22-24 Apr. 2008.
- [6] S. Kelly, J. -P. Tolvanen, "Domain-Specific Modeling. Enabling Full Code Generation," Wiley-IEEE Computer Society Press. 2008.
- [7] P. Levis, N. Lee, M. Welsh and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, Los Angeles, CA., USA., 2003, pp: 126-137.
- [8] G. Maia, D.L. Guidoni, A.L.L. de Aquino, A.A. F. Loureiro, "Improving an Over-the-Air Programming Protocol for Wireless Sensor Networks Based on Small World Concepts," Proceedings of the 12th ACM international conference on Modeling analysis and simulation of wireless and mobile systems MSWiM 09 (2009), pp. 261-267.
- [9] P.J Marron, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Roethermel, "FlexCup: A Flexible and Efficient Code Update Mechanism for Sensor Network," In Proceedings of the Third European Workshop on Wireless Sensor Networks (EWSN 2006), pp. 212-227
- [10] R. Parthasarathy, N. Peterson, WenZhan Song, A. Hurson, B.A Shirazi, "Over the Air Programming on Imote2-Based Sensor Networks," in Proc. System Sciences HICSS 2010 43rd Hawaii International Conference on (2010),

pp.1-9

- [11] S.Varma, U.S. Tiwary, R. Konakalla, "Remote reprogramming mechanism for WSN", Proc of Intelligent and Advanced Systems, ICIAS 2007, pp. 929-94.