A QoS-Driven Self-Adaptive Architecture For Wireless Sensor Networks

Ahmed Jemal, Riadh Ben Halima

A QoS-Driven Self-Adaptive Architecture For Wireless Sensor Networks

Ahmed JEMAL[1,2,3] Riadh BEN HALIMA[1,2,3]

[1]*University of Sfax, ReDCAD Laboratory, B.P. 1173, Sfax-Tunisia*
[2]*CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[3]*Univ de Toulouse, LAAS, F-31400 Toulouse, France*
jmlhmd@gmail.com, riadh.benhalima@enis.rnu.tn, khalil@laas.fr

***Abstract*— Recently, Wireless Sensor Networks (WSN) have become increasingly used to perform distributed sensing and convey useful information. These kinds of environments are complex, heterogeneous and often affected by unpredictable behavior and poor management. This fostered considerable research on designs and techniques that enhance these systems with an adaptation behavior. In this paper, we focus on the self-adaptation branch of the research and give an overview of the current existing approaches. We also analyze the collected approaches and we summarize their common and individual characteristics. Then, we describe our proposed approach to adapt running WSN applications while adopting the autonomic control loop [1]; MAPE: Monitoring, Analysis, Planning, and Execution. Differently from other approaches, where adaptation is generally performed by simply re-deploying another version of application, we focus on the distinction between three different levels of adaptation. We define a sensor level (level1) composed of terminal leaf nodes, a cluster head level (level2) that is an elected node with collection capability and a base station level (level3) which is an enhanced capabilities node that can be a computer or a mobile smart phone. This makes our system able to provide quick adaptation to multiple context parameter changes and to deal with multiple users requirements changes in order to preserve energy consumption efficiency, and maintain system lifetime durability. To illustrate our approach, we study the Smart Home Health Care (SHHC) system over the AZEM simulator which is an enhanced version we developed of AvroraZ. This case study enables us to show the feasibility and the efficiency of our approach for self-adapting WSNs.**

***Keywords*—WSN, Self-adaptation, Monitoring, Reconfiguration, Energy and Mobility, Simulator, AZEM**

I. INTRODUCTION

A Wireless Sensors Network (WSN) is a set of wireless interconnected sensor nodes that are able to monitor environment and give measures and alerts on indicators in order to process their analyze [2]. WSN applications are highly scalable and the number of connected nodes can vary from hundreds to thousands at runtime. They provide a fine-grain monitoring and represent a key feature in Quality of Service (QoS) management. The WSNs show their importance in many domains such as medical field, industrial and military. Also, it enables managing smart buildings and assisting ambient living applications. However, the environments where they are deployed are in perpetual change.

We notice that nodes are battery operated, and generally we cannot recharge them periodically. Consequently, the energy monitoring and management have to be mastered in order to better operate these long-lived applications [3]. This imposes several requirements, compared to traditional information systems. So, we need to dynamically adapt the behavior of executed applications at runtime, and a solution that halts applications and proceeds to reprogram nodes from scratch which is not suitable in much cases.

Context changes and user requirements are the two major triggers for adaptation in order to enhance the functioning and manage degradations. In order to enable such dynamic adaptation, WSNs must have an autonomous behavior in order to react to context/resources and user requirements changes.

In this paper, we present an autonomic enabled-architecture according to the IBM control loop [1]. It provides primitives and mechanisms to adapt WSNs applications in order to increase the WSNs life duration, to manage the mobility, and to cope with context and requirements changes. The autonomic loop is based on four phases, namely: monitoring to collect data, analysis to identify degradation, planning to plan for adaptation actions and execution to enact them. We elaborate a MAPE pattern (see figure 3) that is deployed in different levels of the sensors network, including the cluster leafs level that is composed of terminal sensor nodes, the cluster head level, and the base station level.

The SHHC is used to illustrate our approach. It is a smart building that includes many equipment and sensors. Our purpose is to increase the system efficiency and rationalize the energy consumption while minimizing the message loss rate that results from sensors mobility. Both aspects (Energy & Mobility) are monitored thanks to monitors that we propose in our self-adaptive approach and simulated through our simulator: AZEM. AZEM[1] is an extended version of AvroraZ that we developed in order to consider Energy and Mobility monitoring and management. Compared to other simulators which simulate a static or pre-programmed scenario, it enables interactivity during the simulation. For instance, it allows modifying the sensor position during the simulation process.

The paper is organized as follows: Section 2 gives taxonomy of WSNs where we focus on adaptive middleware. The adaptation levels of WSNs are addressed in this section. In Section 3, we detail our self-adaptive approach and give details about our MAPE architecture. In Section 4, we present the AZEM class diagram and underline the brought extensions. In Section 5, we present the case study, and the carried out experiments that show the feasibility and the efficiency of our approach. The last section concludes the paper.

[1] http://www.redcad.org/members/benhalima/azem/

II. SURVEY ON SENSOR NETWORK MIDDLEWARE

In this section, we examine the state of the art in WSN middlewares. We classify the related works into two main classes regarding their ability or not to adapt their states to the environment constraints and user requirements.

A. Non-adaptive middleware

A middleware is called non-adaptive if it does not support the architecture reconfiguration at runtime. In this case, in order to operate a reconfiguration, we need to (i) halt the current running application, (ii) reprogram and (iii) redeploy the entire application which is difficult to perform. In this class, we distinguish three solutions namely database-based, in which applications use sql-like requests to extract sensing values from network, event-based, in which sensing values are pushed to the application following pre-programmed thresholds, and the hybrid which corresponds to the integration of both solutions.

1. Database-based solutions

The middleware that adopts a database-based solution considers the entire network as a virtual database. It offers an easy interface that allows users to query the sensors network in order to extract data.

Cougar [4], MaD-WiSe [5] and TinyDB [6] belong to this subclass of middleware and are generally used for limited-data collection applications. These middlewares assume that sensors are largely homogeneous and have consistent and homogenous data types which are not always the case. However, these middlewares become unsuitable for networks integrating rich sensors such as camera and microphone.

2. Event-based solutions

The event-based middleware is a direct application of Message-Oriented classic Middleware (MOM) initially dedicated for distributed systems, to the context of WSNs. Mires [7] is an example of that middleware. It provides an asynchronous event-based communication model suitable for sensor networks applications. It is built on TinyOS [8] using NesC programming language [9]. It adopts a component-oriented programming approach while using a communication infrastructure based on the publish/subscribe model. This allows decoupling producers and consumers of events. However, this middleware has some deficiencies. Indeed, we notice that it does not consider the heterogeneity of sensors or their data.

3. Hybrid solutions

Such middleware subclass includes both database and event-based mechanisms. For instance, DSWARE (Data Service middleWARE) [10] integrates this subclass. It uses a database-based solution designed for sensor networks based on event detection. It supports a group-based decision making and a centralized storage. It ensures reliable data, mainly when the failure rate of messages exchange is high.

We note that this subclass of middlewares does not take into account neither the heterogeneity of sensors, nor the resource constraints or the network topologies. Therefore, it is not appropriate for complex applications. In addition, this kind of middleware does not consider scalability, mobility of nodes, security and availability. Also, we point out that this subclass of middlewares does not handle adaptability. In other words, this kind of middleware cannot continue to meet application requirements when environment constraints change or user requirements evolve. However, some work began to consider the property of adaptability; they present only a simple tuning and adjustment. SINA [11] is an example of a database-based middleware for sensor networks that facilitates querying and monitoring network nodes. It includes two low-level mechanisms for sensors managing. The first level is the hierarchical clustering of sensors for energy conservation and scaling. The second level ensures their identification based on attributes and makes easy the access to data. Nodes are sensitive to two types of language used by the middleware: an SQL-like query language as for TinyDB and Cougar, and a procedural scripting language called LQTS that can be sent to nodes and executed there through the SEE (Sensor Execution Environment). That's what makes SINA a flexible middleware.

B. Self-adaptive middleware

The most used middleware is service oriented. This class manages adaptation issues while using structural and behavioral reconfiguration actions. A service is defined as a modular, autonomous, and independent from execution platform and having a well-defined interface. Indeed, the service-oriented architecture (SOA) has been used to address the heterogeneity of nodes, and ensure the composition. It provides mechanisms to publish, find and bind services and promotes their reuse.

We distinguish three levels where services can be deployed namely: on the node, on the gateway and on the base station.

Firstly, the sensors are small equipment, with a small processing capacity and various modes of wireless communication, including WiFi, Bluetooth, ZigBee, Radio and Infrared. These sensors offer some features such as reading temperature, pressure, humidity, movement, etc... Many technologies are available on the market: Telos, Mica2 and its successor Micaz of Crossbow, and SunSpot from SUN micro-system are the most used.

Secondly, the gateways are collectors, and are characterized by a higher cost, more resources and provide functionality to mediate between sensors and the base station.

Thirdly, the base stations are the final collectors of information from sensors and gateways to control them. They are assimilated to traditional PCs, laptops, Palms, etc. Their role is to manage the sensor network as a whole including their administration and adaptation.

Therefore, we propose to classify the service-oriented middleware into three categories according to the level where services are deployed. The first one includes the middleware in which management services are deployed on sensors. The second includes the middleware where management services are deployed on the base station. The third category is characterized by cross-level deployed service-oriented middleware architecture. We notice the absence of work in the literature where services are deployed only on the gateway.

1. Sensor side adaptation

This first class implements and deploys adaptation primitives on sensors. It enables horizontal interaction between leaf nodes. This requires defining an appropriate structure of adaptation services which enable interaction within services deployed on other nodes. This class includes middleware such as Tiny Web Services [12] and TinyWS [13].

Tiny Web Services [12] is a middleware built on the basis of mini web services. These mini web services operate on a small version of TCP/IP, called µIP. This middleware is based on event notifications in order to save energy.

TinyWS [13] offers an embedded web services platform residing on the sensor nodes, giving them the opportunity to communicate directly without going through the gateway. It is developed on the basis of TinyOS using TCP/IP. However, this approach does not offer an abstract view of an application before its implementation.

Meanwhile, the majority of the above middlewares uses an XML based-communication that engenders an overhead and consumes a large part of the device resources.

2. Base station side adaptation

The second class implements the services at the base station level. In this class, the WSN is considered as a data source and installed applications on the base station are web service-based, in most cases. The work presented in [14] belongs to this context and provides an OSGI based service-oriented middleware deployed on the base station. Its proposed middleware communicates with sensor network by using a packet forwarder installed on the base station. It defines three types of bundle, namely communication, wrapper and application. This allows communicating different applications while using different data types from heterogeneous nodes. This middleware requires the continuous availability of data sources and does not support their accidental interruption. Moreover, the addition of a new hardware needs the intervention of the system administrator on the middleware to manage the wrapper for the added node.

3. Cross-level adaptation

The majority of the proposed middlewares deploys services on the three levels while following a cross-layer architecture.

The work of Jeremie et al. [15] adopts a SOA for the WSN applications. It proposes a layered service oriented approach: the first is dedicated to limited capacity nodes called "WSN-SOA", the second is devoted to gateway nodes and is based on Devices Profile for Web Services (DPWS) [16] and the third is used on the base station and adopts the SOA standard. So, the applications deployed on such architecture offers its end-results as web services. This solution ensures interoperability between different levels through gateways that implement a set of matching services offered as OSGI bundles. However, no adaptation is proposed to deal with the service failure. Also, WSN-SOA messages are not semantically expressive.

III. Our Self-Adaptive Architecture

In this section, we present our self-adaptive architecture which implements the MAPE pattern as shown in Figure 2.

A. Self-adaptive control loop elements

In what follows, we identify and specify the role of each element of our self-adaptive architecture.

1. The monitor:

We mean by monitor a component deployed on a sensor node and capable of monitoring network QoS parameters that we call metrics. Therefore, we need as many monitors as metrics to be supervised.

```
Interface Monitor_interface {
command int getSupervisedValue();
command void setSensingRate(int period);
event void eventNotify (MonitoringMess *msg, ErrorMess error); }
```

Figure 1. The Monitoring Interface

These metrics focus on the communication quality within the network and its life time. All these Monitors are exposed on nodes as services and respect the *Monitor_interface* presented in Figure 1. Thus, they implements *getSupervisedValue()*, *setSensingRate(int period)* and *eventNotify(MonitoringMess *msg, ErrorMess error)* methods dedicated respectively to read monitored metric, to set the rate of the sensing operation and to give the monitor the possibility to notify when an event occurs.

2. The analyzer:

In our architecture, we propose two types of analyzers. The first operates in the cluster head sensor and is called "in-network analyzer". Indeed, it interprets data from the cluster with the aim to infer the possibility of applying aggregate functions like average, maximum and compression. The second, called "out-network analyzer", is deployed on the base station. This strategic place gives it a global view of the sensor network allowing synthesizing and proposing actions to overall network reconfiguration.

3. The planner:

The planner gets analysis reports, and provides a defined sequence of actions to recover the system. It is driven by a set of pre-defined rules. Indeed, in our architecture, we provide two planners that meet the recommendations extracted from both in and out network analysis reports. Actions on cluster data optimization (number of messages exchanged) are taken into account in the in-network planning phase. We can mention an application of aggregation rules such as the sum operation or the maximum number of received data. We can also propose rules that reconfigure the overall structure of the network while following instructions from out-networks analysis. Migration plans can be planned during this phase.

4. The executor:

This is the component that ensures the enforcement of planned adaptation actions to bring the system to its stable state. According to the planned action, two types of executors are deployed. The first turns on the cluster head achieving actions like messages filtering, emission frequency adjustment, and optimization of the transmission signal frequency. These actions operate adjustments within the network. A second executor operates on the base station and provides the actions of global network reorganization, including components deployment/ redeployment/ undeployment, activation / deactivation.

B. *Self-adaptive control loop pattern*

The MAPE pattern shown in Figure 2, presents the deployment architecture of the autonomic components. In the bottom level (cluster leaf), a Monitor and an Executor are deployed on each device. Collected data from this level are sent to the cluster head's Analyzer which plans for local adaptation actions and asks Executors (belonging to cluster head and leafs) for executing them. When local adaptation is not sufficient, the base station Analyzer can perform a global analysis (based on local analysis) and plan for a global adaptation plan. Then, each local Planner receives his sub-plan including elementary adaptation actions to be performed by the device (Cluster leafs and head) Executor. This is the case when all cluster nodes have not enough energy to play the role of head. Therefore, we ask the global Analyzer and Planner (belonging to the base station) to detect the nearest node belonging to another cluster that has enough energy and can play the role of head.

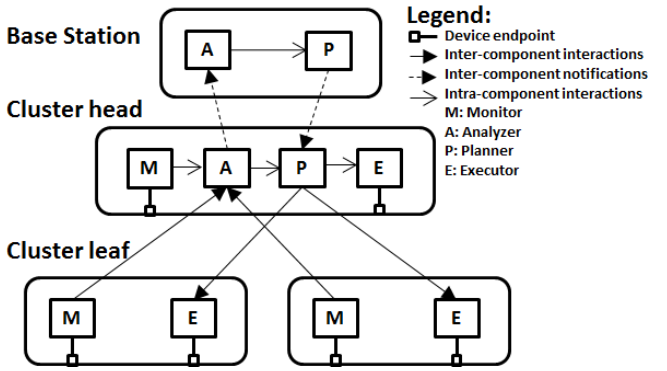

Figure 2. Self-adaptive control loop; Master/Slave Analysis and Planning modules

The legend of figure 3 is described in the following:

Device endpoint: They are two kinds of interaction between the device and the MAPE components. First, the monitor (M) interacts with the device for monitoring purposes. Second, the Executor (E) interacts with the device to enforce planned adaptation actions.

Inter-component interactions: They describe the interactions between MAPE components belonging to different levels. For instance, M (from cluster leaf) sends monitoring date to A (from Cluster head).

Inter-component notifications: They describe the notifications exchanged between the same MAPE components belonging to different levels. For instance, P (from Base station that has the global plan) notifies P (from Cluster head) about the local plan to perform in its cluster.

Intra-component interactions: They describe the interactions between MAPE components of the same level.

IV. AZEM: A NEW NETWORK SIMULATOR

When developing or researching in WSN domains, it is better to use a WSN simulator for many reasons. On the one hand, it is so expensive to experiment WSN on real world which consists of tens or hundreds of sensor nodes and gateways. On the other hand, distributed nature of WSNs makes debugging of applications a very difficult task. So, using a WSN simulator can help researchers and developers of WSN applications to fetch and correct majority of bugs and code issues before performing deployment on real world.

However, choosing a simulator for WSN is a specific application task that focuses on a deployment environment and a set of modules ensuring the WSN functionalities. This task relies also on the capabilities required by application, namely the reusability, the availability, the performance, the scalability, the language and operating system dependency and techniques allowing base station application to communicate with sensor applications. WSN simulators are classified onto three categories regarding the different levels of the system. First, we distinguish simulators allowing network protocol experimentation. The most widely used one is ns-2 [17]. Second, we mention simulators that rely on the WSN operating system level. The most widely used one is Tossim [18]. Third, we recognize simulators that model accurately the hardware of sensor platforms. This class of simulators is called "instruction set level". The leader of that class is Avrora [19] that has been extended by AvroraZ [20] which enables accurate simulation of IEEE 802.15.4 based protocols. We have chosen the latest one to simulate the NesC codes deployed in the cluster head and leaf nodes. This is an open source and scalable sensor network simulator. It scales to networks of up to 10,000 nodes and can handle as many as 25 nodes in real-time.

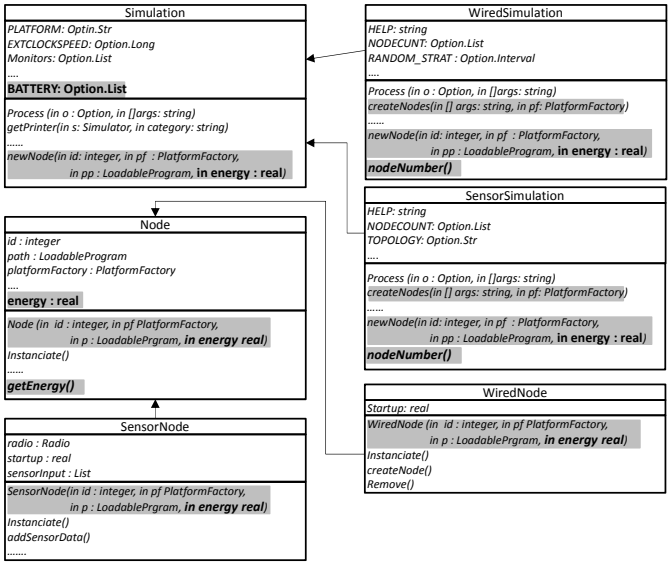

Figure 3. AZEM Class Diagram

AvroraZ is language and operating system independent. Our choice is argued by the nature of our experiments. In fact, we are more interested in simulating dynamic network adaptation rather than performance. We have to interact with sensor network by sending adaptation commands with response to sensing readings. We have also to deal with different types of node with different types of operating system and thus AvroraZ is an appropriate tool. However, this simulator suffers from the inability to separately specify the value of energy of each node at design time. Further, it is unable to dynamically change the geographical position of a node at runtime. We have modified the actual version of

AvroraZ for this purpose and we propose a new version that makes possible to specify separately the energy level of each node and to modify nodes positions at runtime. We call our version AZEM (**A**vrora**Z** + **E**nergy + **M**obility). The AZEM class diagram is shown in Figure 3. So, we extracted the AvroraZ diagram class and added new features. We highlight our code extensions in gray.

We added attributes and methods, and modified existing methods in order to enable energy and mobility management. For instance, we add the attribute *energy* in the *Node* class and initiate it on the constructor in order to save and follow the energy consumption. The detailed class diagram is available at http://www.redcad.org/members/benhalima/azem/.

V. Illustration

The illustration is based on the smart home health care case study that is detailed in the following.

A. The Smart Home Health Care

Figure. 4 WSN in Smart Home Health Care

The wireless sensor equipment is often used in healthcare applications while applying many types of portable sensors. These sensors are placed on the human body surface or just at its proximity. They are composed of small devices like heart rate, blood pressure, temperature, glucose sensors to measure the biophysical patient state, as illustrated in Figure 4.

To be deployed comfortably on human body and for a long period without noising their usual life days, these devices must be small enough and battery powered. Thus, energy consumption must be managed properly and must be optimized so that the battery replacement occurs at periodic time with a maximum number of sensors battery replacements at the same time.

In addition, healthcare equipment must collaborate with WSN deployed on the patient home in order to transmit captured data to the home gateway using multi-hop relay. The gateway sends collected data to an application deployed on a medical server using Internet in order to be studied and to carry out the patient global health supervision. On critical state of the patient medical and nursing personals can indicate some urgent treatment to be performed remotely on the patient like injecting "Insulin". The gateway receives commands from the medical server application and notifies the actuator to perform necessary actions.

B. The Case Study Experimentation

To experiment our approach, we propose to maintain patient connectivity using an optimal transmission frequency range when moving from one gateway scope to another. We suppose that two elder patients are living together in the same home composed of two floors. As described in Figure 5, each patient disposes of smart sensors deployed on his body that send collected data to a gateway through a clustered WSN. We deploy the first gateway in the first floor and the second one in the second. If one of the patients moves from one floor to another, the WSN connectivity may be interrupted. Thus, two alternatives can occur. First, sensors are using a high transmission frequency, thereby connectivity can be maintained. Nevertheless, sensors will consume more energy. Second, some messages will be lost. We choose this second case and enable the WSN self-adaptation in order to avoid messages loss. Therefore, it will increase the system efficiency and rationalize the energy consumption. We enable an adaptation action that connects sensors to the available gateway range. This will minimize the message loss rate that results from sensors mobility.

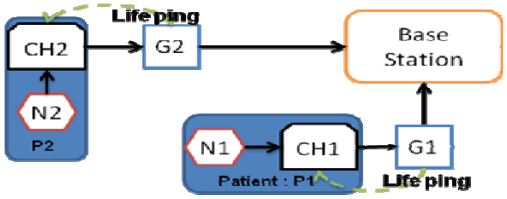

Figure 5. The Initial architecture (before moving)

Figure 5 shows the initial configuration. The node *N1* reads a sensed value, and then transmits it to the cluster head *CH1* that forwards it to the gateway *G1*. *CH1* sends also its sensed value to *G1*. Finally, *G1* transmits data to the Base Station. *N2*, *CH2* and *G2* perform same transmission actions regarding the patient *P2*.

Table I. Exchanged messages flow without adaptation

Node	Sent Messages	Received Messages	Energy consumption	Received messages in base station
N1	68	-	11,66 J	
N2	68	-	11,66 J	
CH1	136 = 68 (Node1 data)+68 (CH1)	68 (N1)	11,68 J	
CH2	136	68 (N2)	11,68 J	
G1	51 (life ping)	117 (CH1)	15,005 J	117
G2	51 (life ping)	136 (CH2)	15,005 J	**136**

The first experiment consists of moving *CH1* and *N1* (deployed on patient *P1*) far from G1 scope for 30 seconds and then returning them to their initial place. Even if *CH1* and *N1* reach *G2* scope, no messages will be received by *G2* because it is linked statically to *G1* and no adaptive feature is applied. As a result, a lot of messages will be lost. This experiment is ensured using those properties: Simulation time=170s, initial energy value in *G1* and *G2*=10 000 joules (generally mains powered), initial energy value in *CH1*, *CH2*, *N1* and *N2*=15 joules.

Table I summarizes the experimentation results. We focus on sent messages, received messages, consumed energy and base station received messages. We note here that the two

gateways broadcast life messages in order to initiate system deployment. These messages describe gateway properties. We show that 19 messages was lost that constitutes 7% of sent medical messages. Therefore, we need to resend these messages until their reception and as consequence we will lose more energy. In this paper, this part (resending lost messages) is not included in this experiment.

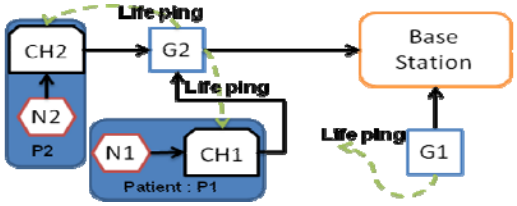

Figure 6. The adapted architecture (after moving)

In the second experiment, we apply our self-adaptive approach in the same conditions as the first experiment. In this case, the *CH1* and *N1* movement is detected using our link quality monitor deployed on the cluster head. So, the analysis and the planning are performed and an adaptation action is executed. Indeed, the adaptation action consists in connecting *CH1* to *G2* as shown in Figure 6.

Appling our self-adaptive approach allows us to recover 16 messages in 170 seconds that means 338 messages in one hour, and 8131 in one day. As a result, we recover 84 % of the lost messages compared to the first experiment.

Table II. Exchanged messages flow with adaptation

Node	Sent Messages	Received Messages	Energy consumption	Received messages in base station
N1	68	-	11,66 J	
N2	68	-	11,66 J	
CH1	136 = 68 (Node1 data)+68 (CH1)	68 (N1)	11,68 J	
CH2	136	68 (N2)	11,68 J	
G1	51 (life ping)	117 (CH1)	15,005 J	117
G2	51 (life ping)	152 (CH2+CH1)	15,005 J	**152**

In Table II, G2 receives 152 messages instead of 136 in the first experiment. As a result, we minimize the number of lost messages during sensors mobility and we rationalize the energy consumption.

VI. Conclusions

In the first part of this paper, we presented a survey of the state of the art on WSN middleware while focusing on service oriented self-adaptive approaches. Indeed, we have distinguished three classes of adaptive middleware according to the level where adaptable services are deployed namely sensor, base station and hybrid sides. In the second part, we proposed our self-adaptive architecture based on the MAPE control loop. Then, we presented the AZEM simulator which is an enhanced version we developed based on AvroraZ that enables mobility and energy management. In the third part, we study the Smart Home Health Care system, in order to illustrate our approach. This case study enables us to show the feasibility and the efficiency of our approach for self-adapting WSNs. It managed the sensors mobility and rationalized the energy consumption.

Our future work will focus on the assessment of our approach and our simulator on a large scale experiments, and the optimization of the monitoring cost while predicting a part of the sensed values.

References

[1] Jeffrey, O. K.; Chess, D. M. (2003). "The vision of autonomic computing." Computer, 36(1):41- 50.

[2] Akyildiz, I. F.; Kasimoglu, I. H. (2004). "Wireless sensor and actor networks: research challenges." Ad Hoc Networks, 2(4):351- 367.

[3] Avvenuti, M.; Corsini, P.; Masci, P.; Vecchio. A. (2007). "An application adaptation layer for wireless sensor networks." Pervasive Mob. Comput. 3(4):413-438.

[4] Bonnet, P.; Gehrke, J.; Seshadri, P. (2001). "Towards Sensor Database Systems." In *Proceedings of the Second International Conference on Mobile Data Management* (MDM '01), Springer-Verlag, London, UK, pages 3-14.

[5] MaD-WiSe project. http://mad-wise.isti.cnr.it/

[6] Madden, S. R.; Franklin,M. J.; Hellerstein, J. M.; Hong, W. (2005). "TinyDB: an acquisitional query processing system for sensor networks." *ACM Trans. Database Syst.* 30(1):122-173.

[7] Souto, E.; Guimarães, G.; Vasconcelos, G.; Vieira, M.; Rosa, N. S.; Ferraz, C.; Ferraz, C. A. G. (2004). "A message-oriented middleware for sensor networks." In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*. ACM, New York, NY, USA, pages 127-134.

[8] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E.; Culler, D. (2005). "TinyOS: An Operating System for Sensor Networks". Ambient Intelligence,pages 115-148.

[9] Gay, D.; Levis, P.; von Behren, R.; Welsh, M.; Brewer, E.; Culler, D. (2003). "The *nesC* language: A holistic approach to networked embedded systems." *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM, New York, USA, pages 1-11.

[10] Li, S.; Lin, Y.; Son, S. H.; Stankovic, J. A. and Wei, Y. (2004). "Event detection services using data service middleware in distributed sensor networks". Telecommunication Systems, 26(2-4):351–368.

[11] Chien-Chung, S; Srisathapornphat, C.; Jaikaeo, C. (2001). "Sensor information networking architecture and applications," *Personal Communications, IEEE*, 8(4):52-59.

[12] Nissanka, B.; Priyantha, A. K.; Goraczko, M.; Zhao, F.(2008). "Tiny web services: design and implementation of interoperable and evolvable sensor networks." In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, New York, NY, USA, pages 253-266.

[13] Othman, N.Y.; Glitho, R.H.; Khendek, F. (2007) "The Design and Implementation of a Web Service Framework for Individual Nodes in Sinkless Wireless Sensor Networks," *12th IEEE Symposium on Computers and Communications*, pages 941-947.

[14] Prinsloo, J.M.; Schulz, C.L.; Kourie, D.G.; Theunissen, W.H.M.; Strauss, T.; Van Den Heever, R.; Grobbelaar, S. (2006). "A service oriented architecture for wireless sensor and actor network applications." In *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (SAICSIT '06), pages 145-154.

[15] Leguay, J.; Lopez-Ramos, M.; Jean-Marie, K.; Conan, V. (2008). "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," *33rd IEEE Conference on Local Computer Networks*, pages 740-747.

[16] Devices Profile for Web Services (DPWS) specification. (2006). http://schemas.xmlsoap.org/ws/2006/02/devprof/

[17] McCanne, S.; Floyd, S. (2011). "Network simulator ns-2." http://nsnam.isi.edu/nsnam/index.php/

[18] Levis, P.; Lee, N.; Welsh, M.; Culler, D. (2003). "TOSSIM: accurate and scalable simulation of entire TinyOS applications." In *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, New York, NY, USA, pages 126-137.

[19] Titzer, B.L.; Lee, D.K.; Palsberg, J. (2005). "Avrora: scalable sensor network simulation with precise timing," *Fourth International Symposium on Information Processing in Sensor Networks*, pages 477- 482.

[20] De Paz Alberola, R. ; Pesch, D. (2008). "AvroraZ: extending Avrora with an IEEE 802.15.4 compliant radio chip model." *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. ACM, New York, NY, USA, pages 43-50.