MIDDLEWARE FOR WIRELESS SENSOR NETWORKS

Breno A. D. Vitorino[1], *Luiz F. M. Vieira*[1], *Marcos A. M. Vieira*[1], *Vinícius C. de Almeida*[1], *Antônio O. Fernandes*[1], *Diógenes C. da Silva*[2], *Claudionor N. C. Júnior*[1]

[1]Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
[2]Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais
Avenida Antônio Carlos, 6627, Belo Horizonte, Minas Gerais – Brasil
{vitorino, lfvieira, mmvieira, makish, otavio, coelho}@dcc.ufmg.br, diogenes@ufmg.br

ABSTRACT

Wireless Sensor Networks operate with hundreds or thousands of tiny embedded computers, which have restricted power, energy, and computation capabilities and communicate over wireless means. As these networks enable several applications, the programmers need an easy and clear way to develop over these sensor nodes. Our work proposes a middleware that will reach this objective.

The middleware is a software layer that sits between the operating system and applications. It will meet many requirements of wireless sensor networks, and also incorporate energy-efficiency policies in every of its features. Some of them are local communication, code mobility, event-driven operation, and a high-level programming language. Project decisions had been taken analyzing recent studies on this area.

1. INTRODUCTION

Tiny embedded devices compose wireless sensor networks (WSN). These devices can communicate with each other by wireless means (radio, infrared, etc.), and have memory, energy and computational power restrictions, due to its limited size. They should work autonomously, collecting useful data from the environment with specialized sensors. This information will be routed to a common gateway, a more powerful computer device.

Some interesting applications arise from these devices, therefore referred as sensor nodes, such as natural environment surveillance, dangerous substance detection, and traffic monitoring. Their purpose cannot be implemented simply by collecting and forwarding data to the gateway, but with distributed algorithms. Sensor nodes have to cooperate with each other in order to achieve the correct result.

These applications were hard-coded in assembly or have the support of an operating system, such as TinyOS[1]. Programmers need an easier and clear way to develop those applications, disrespecting the individual characteristics of the nodes. A middleware for WSN would surely fill this objective.

A middleware is a software layer between the operating system and the applications, which disciplines the development and introduces abstractions that greatly improve the application construction. Our goal is to build a middleware that incorporates the best practices for programming WSN.

Our work is part of SensorNet project, which will study architecture, protocols, management and applications for WSN.

The rest of this paper is organized as follows. Section 2 discusses the latest approaches on middlewares for WSN. Section 3 presents the requirements met by our work. Section 4 shows the proposed architecture, and Section 5 describes our project status and next steps.

2. RELATED WORK

The needs for a middleware for this kind of embedded system are well known. Although their characteristics can be clearly identified, the exact architecture to cover all the requirements is an open research[2].

Maté[3] builds a virtual machine above TinyOS, and provides code mobility at a low transmission cost. Besides, the programming model states that all operations are done on stacks, similar to Java bytecodes[4], so it also offers memory protection. It was designed to hide synchrony problems associated to nesC language[5], and offer a better way to program the sensor nodes. All these achievements are explained by dealing with scripts rather than binary code.

Although the scripts impose an energy overhead in their execution due to translation, they make harder security checks and smaller code size possible, comparing to pure binary executables. Sensorware[6] uses the same idea, but with the Tcl core[7]. This work, however, does not meet our restricted memory requirements. It was developed to iPAQ 3670 architecture, which is far beyond the state-of-art nodes, such as mica motes[3].

Another approach is to write one program that will be correctly compiled and distributed to our sensor nodes, before deployment. This was done in c@t language[8], which deploys a way to aggregate similar functions of the sensor nodes in a logical manner. In this way, sensors interact with each other and are addressed as groups, which is very reasonable.

One could consider the network as a distributed database, and so create SQL-like statements to query this network. Projects like DSWare[9], TinyDB[10], and Cougar[11] chose that view. It facilitates data collection and evaluation, as SQL is very easy to understand. Statements include event identification, detection rate, duration of the query, and region specification. Those middlewares' challenges reside on detecting energy-efficient paths to satisfy the queries and how the network accomplishes these queries. Even the former characteristic can be changed to support real-time constrains.

The problem of a database-like view of the WSN is restricting the number of applications that can be deployed. General processing offers more flexibility, so that a programmer can define complex interactions between nodes in order to solve more elaborated problems like target detection.

3. WSN MIDDLEWARE REQUIREMENTS

It is important to focus our middleware in some aspects, in order to solve them effectively. One of the most important project decisions is the supported class of applications. Here, we chose to ease the implementation of distributed algorithms on WSNs. This leads to an environment where programmers have to write a code that will be executed on the whole network. This idea is reasonable, as the applications need that nodes cooperate with each other in order to achieve the desired result. At the same time, we are enabling a myriad of applications to be deployed, from simple data collection to complex routing algorithms. This approach considers that localization and node's architecture is not known at development time.

To achieve the necessary interaction between nodes, we assume that one node can only communicate with its neighbors. This is reasonable since the energy cost of transmitting some data is usually the square of the distance between nodes. Beyond that, local communication is robust, scales well over the network, and does not limit the range of applications. These properties are inherent to localized algorithms, which is the focus of our middleware.

Other available feature will be code mobility. While the decision of transmitting the code through the network has an energy overhead, it is a necessity. First, some applications don't know a priori its parameters, like sensor reading frequency. Even if they were known, the users would like to change this parameter, according to current results. Another important use of code mobility is the ability to install new programs after network establishment. At any time, one node could be introduced just to forward the new code to its neighbors, which would reach all the network by broadcast. Revised applications could also replace old ones.

The middleware will support an event-driven programming. Most of applications will perform some computation just over some sensor value thresholds, and remain idle the rest of the time, when the sensor nodes can be put in "sleep" mode which saves energy. The middleware will allow programmers to execute computations in response to detected events. The set of events available will be customizable. Some kinds of events are timers, reception of messages, a specific sensor reading, etc.

4. WSN MIDDLEWARE ARCHITECTURE

The requirements described above define the middleware architecture that will support them. The proposed architecture is organized as follows:

- **Program replication management** – programs that request their replication to other nodes call a simple command, like `replicate`. This command will invoke the program replication management, that wraps the program code into a special message and send it over the network. That message also includes the program identification and version. When this message arrives at the receiver node, the program replication management handles this message and installs the program, if it is not already on the sensor node, or overwrites an older version. If there is no more memory available for programs, it will simply discard the message and advertise this decision to the sender node. One could also execute another command to uninstall a certain program from the network.
- **Local communication** – the middleware will be able to negotiate and maintain a unique address to each neighbor of a given node. That will be useful to implement routing algorithms and target tracking algorithms, for example.
- **Event detection** – every node, on its deployment, will register all the available devices with a unique name, implementing some pre-defined set of operations that are fixed for any of these devices. Our middleware will maintain these names and associate the call of these operations to their specific implementation. This approach will exhibit a uniform access for any device, while not restricting the number and types of devices available at each node.

Another concern of our middleware is to present a high-level language that will enable concepts like event-driven execution. We should use a subset of C, which is a common language for programming embedded devices and generates efficient binary code.

5. PROJECT STATUS

We initially studied the state-of-art technologies on middlewares and WSN and identified the main requirements that our software will fulfill. With this information in mind, we were able to delimit an architecture that will provide many features and still respect several constrains of the sensor nodes, including restricted memory and energy resources.

Now, we will consider the design issues of our architecture and, after that, we will implement and test the middleware for the sensor node that is being developed in SensorNet project.

ACKNOWLEDGEMENTS

This work is partially supported by CNPq.

REFERENCES

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors", *In Proc. of Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, 2000.

[2] K. Römer, O. Kasten, and F. Mattern, "Middleware Challenges for Wireless Sensor Networks", *ACM SIGMOBILE Mobile Computing and Communications Review*, pp. 59-61, October 2002.

[3] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks", *To appear in International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[4] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley, 1999.

[5] D. Culler, D. Gay, P. Levis, R. von Behren, M. Welsh, and E. Brewer, "The nesC Language: A Holistic Approach to Networked Embedded Systems", *In Proc. of Conference on Programming Language Design and Implementation of ACM SIGPLAN 2003*, 2003.

[6] A. Boulis and M. B. Srivastava, "A Framework for Efficient and Programmable Sensor Networks", *In Proc. of OPENARCH 2002*, June 2002.

[7] J. K. Ousterhout, "Scripting Higher-Level Programming for the 21st Century", *Computer Journal*, pp. 23-30, 1998.

[8] D. P. Seetharamakrishnan, "c@t: A Language for Programming Massively Distributed Embedded System", *Unpublished Masters Thesis at Massachusetts Institute of Technology*, September 2002.

[9] S. Li, S. H. Son, and J. A. Stankovic, "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks", *Master's project at University of Virginia*, May 2003.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks", *To appear in ACM SIGMOD Record*, June 2003.

[11] Y. Yao, J. Gehrke, "The cougar approach to in-network query processing in sensor networks", *ACM SIGMOD Record*, pp. 9-18, 2002. September 2002.