

# Middleware for Wireless Sensor Networks

Breno Augusto Dias Vitorino, Luiz Filipe Menezes Vieira, Marcos Augusto Menezes Vieira, Vinícius Coelho de Almeida, Antônio Otávio Fernandes, Diógenes Cecílio da Silva, Claudionor Nunes Coelho Jr.  
Universidade Federal de Minas Gerais, Departamento de Ciência da Computação  
Projeto: SensorNet/CNPq

## Introduction

Lately, advances in the area of hardware and wireless communication had created a new paradigm for environment surveillance. Small, low cost devices can be installed in inhospitable environments or places of difficult access. Collaborating through message exchanging, these devices measure environmental characteristics such as pressure, temperature and luminosity, sending these data to a base station. In this way, the user has a whole perception of the environment and can take off important conclusions from these measurements. This is the wireless sensor networks (WSN) paradigm.

WSN's have attracted great interest of the scientific community in the areas of habitat monitoring, toxic substance detection, target tracking, etc. Seen the potential of these applications, an important question is how to program these networks in a simple and clear way, using the available resources efficiently. This objective is pursued by the middleware specified in this work. This software is capable to assist the WSN programming, supplying a finite state machine view of the problem. This approach is based on the event-driven programming model, which leads to great energy savings.

Abstraction from specific node characteristics - the node should ignore a program that requires specific hardware capabilities that are not available. To ease the implementation of applications, a given code should be able to run in every node in the network, despite their differences in type and number of sensors;

Energy-efficient - all the features presented above should use policies that improve network lifetime, by saving energy as much as possible. The programs should run efficiently on the node, and the middleware enforces the use of optimized binary code.

## Specification

The language offered by this middleware use the C language core. It is widely used in embedded software development, so it represents minor effort for developers to learn another syntax. The compiled C code also produces an efficient binary code, which is one of our requirements. The use of a scripting language was considered in the beginning, but then it was discarded because this language class imposes a prohibitive computational overhead when interpreting them, what invalidates the programming of infrastructure applications. Moreover, it would be complicated to compile it in an efficient form.

The pointers had been removed from then chosen language. Dynamic memory allocation is not possible because it demands a complicated memory management which can't exist with such few computational resources offered by sensor nodes. This decision means that the data structures created by the programmer will always have a maximum size defined during development.

The middleware will generative native assembly code to the target platform from a given source code. So, the middleware will have to add any additional features in the binary code directly or generating system calls to the operating system. Common features, like code mobility, will have support from the operating system.

There will be special blocking calls to implement an event-driven programming. These calls represent specific events that we wish to detect, and can be combined with and/or operators to determine more complex conditions to continue the normal program execution. This feature allows the program to be thought as a finite state machine, where transitions represent the observed events and the states their handling. While the execution is blocked, the operating system should schedule another program for execution or put the sensor node in low power mode, saving energy.

The blocking calls will access the sensing and networking resources, abstracting the operating system API. These calls will be implemented as additional modules to the operating system, with hardware-specific code. For example, if the API has a generic function to send data, the middleware will have modules to convert the data into structured packets and vice-versa.

## Features

An appropriate programming environment to express applications in WSN - this is our main motivation for developing this middleware. The challenge is creating a language that supports the peculiarities of sensor networks and generates an executable code that efficiently uses the node's resources. Some of these peculiarities are: close interaction with the environment, sensor node failure when its energy runs out, etc. The decentralized nature of this network leads to distributed algorithms as a common way to implement applications. This class of algorithms has being favored in this language;

Support for code mobility - although the decision of transmitting the code through the network has 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, users may want 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 be broadcasted to the network. New application versions could replace old ones;

Event-driven programming - most 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 into "sleep" mode to save energy. The middleware will allow programmers to execute computations in response to detected events. The set of events available may be customizable. Some kinds of events are timeouts, reception of messages, sensor readings;

Middleware	<i>c@t</i> <sup>1</sup>	<i>SensorWare</i> <sup>2</sup>	<i>TinyDB</i> <sup>3</sup>	<i>TinyGALS</i> <sup>4</sup>	<i>This middleware</i>
Category	Holistic programming	Scripting	Database	Modular programming	<b>Finite state machine</b>
Base programming language	Scheme	Tcl	SQL	C	<b>C</b>
Energy overhead	High (Broadcasts to select groups)	High (Interpret scripts)	Medium (Maintain a routing tree and parse queries)	Low (Manage message queues)	<b>Low (Computation in response to events)</b>
Semantic gap	Low (Declaration of sensor node groups)	Medium (High-level commands)	Low (SQL queries)	High (Explicit concurrency)	<b>Medium (Finite state machine model)</b>
Data delivery model <sup>5</sup>	Event-driven	Event-driven	Continuous/ Observer-initiated	Event-driven	<b>Event-driven</b>

1. D. P. Seetharamakrishnan, "c@t: A Language for Programming Massively Distributed Embedded System", *Unpublished Masters Thesis at Massachusetts Institute of Technology*, September 2002.
2. A. Boulis and M. B. Srivastava, "A Framework for Efficient and Programmable Sensor Networks", *In Proc. of OPENARCH 2002*, June 2002.
3. 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.
4. E. Cheong, J. Liebman, J. Liu, and F. Zhao, "TinyGALS: A Programming Model for Event-driven Embedded Systems," *Proc. ACM Symp. Applied Computing*, Melbourne, FL, March 2003.
5. S. Tilak, N. B. Abu-Ghazaleh, W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Network Models", *Mobile Computing and Communications Review*, 2002.

Table 1: Comparison between this middleware and other approaches