MICROKERNEL FOR NODES OF WIRELESS SENSOR NETWORKS

Vinícius Coelho de Almeida[1], Breno Augusto Dias Vitorino[1], Luiz Filipe Menezes Vieira[1], Marcos Augusto Menezes Vieira[1], Antônio Otávio Fernandes[1], Diógenes Cecílio da Silva[2], Claudionor Nunes Coelho Jr.[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
{makish, vitorino, lfvieira, mmvieira, otavio, coelho}@dcc.ufmg.br, diogenes@ufmg.br

ABSTRACT

Wireless sensor networks, that have become possible thanks to advances in VLSI, can involve a large number of devices, with sensing and communication capability, low-power consumption and small computing power. These small embedded devices, made of thousands of integrated circuits, can be deployed, for example, at harsh environments to perform data collecting. The microkernel of these devices' microcontrollers must be energy-efficient, and still supporting concurrent execution with environment sensing.

In this work, it is presented a case study about an operating system for wireless sensor networks, in order to help to define the characteristics to be implemented in a new microkernel. The features considered worthwhile from existing operating systems for embedded devices and nodes in wireless sensor networks will be implemented.

1. INTRODUCTION

Wireless sensor networks (WSNs) [1] may be composed from thousands of nodes, which possess storing, processing, communication and sensing capacity, although with strong limitations. They must have self-configuration and adaptation mechanisms to support fault-tolerance. Moreover, they may be equipped with different sensors, given the applications' nature inside them, such as: temperature, pressure, movement, etc.

A microkernel may be defined as an operating system (OS) core. Its objective is to ease the programmer's work in developing application for a platform and to manage the resources that constitute the sensor node. The nodes' microkernel must provide to the programmer routines that allow the development of his applications.

This paper is part of SensorNet project [2], whose research is focused in architecture, protocols, management and applications in WSNs. The final goal of this work is to develop a microkernel for nodes in WSNs, using eligible concepts from existing systems. This microkernel will run in a device still being developed in that project.

In Section 2, it will be explained some features of one of the studied systems: *TinyOS* [3], and a conclusion about this OS; in Section 3, details about the proposed work and its current status are discussed.

2. TINYOS

Initially developed at Berkeley University and actively supported by a large community of users, TinyOS is an OS that provides concurrent execution for embedded networked sensors with scarce hardware resources, using Motes architecture [4]. It is an open source software platform and tool-chain designed to support concurrency intensive operations, using minimal hardware requirements.

The relevant features of TinyOS are its implementation language for applications, architecture, concurrency and communication models.

2.1. Implementation language

In order to embody the structuring concepts and execution model of TinyOS, it was designed a C extension, nesC [5]. Every component in TinyOS in v.1.0.0-1 was coded in this language, whose basic concepts are:

- Separation of construction and composition: applications are made of components, which are assembled to form most complex applications.
- Specification of component functionality by means of set of interfaces: interfaces may be provided or used by components. The provided interfaces are intended to represent the functionality that the component provides to its user; the used interfaces represent the functionality the component needs to perform its job.
- Interfaces are bi-directional: they specify a set of functions to be implemented by the interface's provider (i.e. commands) and a set to be implemented by the interface's user (i.e. events).

2.2. Component-based architecture

Every application possesses at least a configuration file and a module or implementation file. This configuration file specifies the application components set and how they invoke themselves. In the implementation file are listed the provided and used interfaces by a component.

An application uses one or more components, being possible to reuse some simpler components in order to create most elaborated ones. This creates a hierarchy of layers, where higher components originate commands to lower components, and the latter signal events to the former. Lowest-level ones represent the hardware itself.

2.3. Event-based concurrency model

Concurrency is made through the use of events and tasks, using two-level scheduling. At lowest priority level are the tasks and at highest one are the events.

Tasks are atomic in relation to other tasks, running into completion (at least that preempted by events). They can call lower components (i.e. that compose it) commands, signal higher component (i.e. that use it) events and schedule other tasks in a component. Tasks are used to perform longer processing operations, such as background data computation,

Events are generalizations of interrupt handlers, propagating processing upwards (by means of signaling other events) or downwards (by means of call of commands). They are executed when signaled, preempting execution of a task or another event.

TinyOS provides atomic sections in order to avoid data races due to concurrent updates to shared state. In an atomic section, hardware interrupts are disabled.

2.4. Active Messages communication model

Active Messages (AM) is a mechanism, which allows to the sender to determine a handler that will be invoked in receiver. The data are extracted from the network and aggregated into the ongoing computation, eliminating buffering in the sending and receiving nodes. They differ from general mechanisms of Remote Procedure Call (RPC) because this computes locally on the given parameters, returning the result. Active Messages handlers execute immediately upon message arrival. This allows overlapping between local processing and application-level communication.

In TinyOS, each Active Message packet contains 36 bytes effectively transmitted with the following distribution: 2 for destination address, 1 for handler identifier, 1 for group handle, 1 for length, 29 for data payload and 2 for cyclic redundancy check (CRC).

2.5. Study conclusion

The present functionalities in an OS for WSNs depend on the application being considered. TinyOS is not designed for all applications in WSNs. Here we list some of its deficiencies:

- Lack of code mobility support: applications can't be distributed dynamically over the network.
- Explicit negotiation for data and resources: for node cooperation, the nodes must program events to be signaled at message reception.
- No kernel/user memory separation: there is not memory protection mechanism, allowing that a bad coded application overwrites system kernel memory.

In the next section, we describe our new approach.

3. PROPOSED WORK

Our microkernel for nodes in WSNs, currently in development stage, is being implemented in C language. This decision was taken in order to grant more portability across different microcontrollers, not depending on a specific architecture.

We analyzed different OSs for nodes in WSNs and embedded devices, such as: TinyOS, Bertha, SensorWare and Eyes. This allowed us to identify their eligible features and concepts, which constitute the base for the new microkernel in development. In next subsections, they are related along with their purposes.

3.1. Event-driven data delivery model

WSNs can be divided in 4 delivery models: continuous, event-driven, observer-initiated and hybrid [6]. In continuous type, the sensors will report their readings at determined rate. In event-driven, sensors will inform to the application when certain events occur. In observer-initiated, sensors will reply to a request from applications. At last, in hybrid, the later approaches can be present at the same time.

Due to the reactive behavior of the most of applications in WSNs, the event-driven model was the chosen to integrate the new microkernel. It is the most attractive because the energy expenses of this model are low, when compared to the other cited approaches.

3.2. Code mobility support

Transmission in WSNs is the process that consumes the most energy, being a good practice to minimize code mobility. However, this functionality should be supported because it allows to dynamically deploying different algorithms over the network. In addition, the nodes themselves can be programmed automatically, from a "programmer" node.

3.3. Application program interfaces separation

Nodes memory in WSNs is a very scarce resource. So it must be used parsimoniously. Some OS functionalities must be available all the time, but there are others that are application-specific. Separating these functionalities in distinct application program interfaces (API) and installing them in the nodes in a need-to-use base, the memory will be used most efficiently. For example, if a node will only read temperature, it is unnecessary and wasteful to load the movement sensing API, too. The objective of this approach is clear: load in a node only the needed functionality in order to save memory.

3.4. Operation on constrained resources

Nodes in WSNs possess storing, processing, communication and sensing capability, but they are very limited. The microkernel must consider this fact, in order to attend the specific requirements of these nodes and their WSNs. The identified demands to be attended are: low energy consumption, small computing power, fault-tolerance and self-configuration.

ACKNOWLEDGEMENTS

This work is partially supported by CNPq.

REFERENCES

[1] A. A. F. Loureiro, J. M. S. Nogueira, L. B. Ruiz, R. A. F. Mini, "Redes de Sensores sem Fio", *Curso da XXI Jornada de Atualização em Informática* – XXII Congresso da SBC, Florianópolis, 2002.

[2] SensorNet project: http://esperanca.sis.dcc.ufmg.br/

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

[4] A. Cerpa, J. Elson, M. Hamilton, J. Zhao, D. Estrin, L. Girod, "Habitat Monitoring: Application Driver for Wireless Communication Technology", *Workshop on Data Communication in Latin America and the Caribbean*, ACM Press, pp. 20-41, 2001.

[5] D. Gay, P. Levis, R. Behren, "The nesC Language: A Holistic Approach to Network Embedded Systems", to appear in *ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, 2003.

[6] S. Tilak, N. B. Abu-Ghazaleh, W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Network Models", *Mobile Computing and Communications Review*, 2002.