

# Operating System Fundamentals for the EYES Distributed Sensor Network

Stefan Dulman and Paul Havinga  
University of Twente, Department of Computer Science  
Enschede, the Netherlands  
{dulman, havinga}@cs.utwente.nl

*Abstract*—One of the problems related to sensor networks built up from large number of nodes having limited capabilities is the operating system running inside each node. It must provide a lot of functionality and allow easy development of applications, but has very restricted resources (like memory, processing capabilities, and energy). Some of the features are also related to the general nature of the network and applications we are building: it is a distributed system, and made up from several types of mobile sensors. In this paper, we have identified the main requirements of such an operating system and propose an architecture that satisfies them. The results of this study are the first steps in our project and provides the basics for all future work.

*Keywords*—sensor network, distributed operating system, mobile sensor nodes

## I. INTRODUCTION

The vision of ubiquitous computing requires the development of devices and technologies that can be pervasive without being intrusive. Small sensors with communicating capabilities and low processing power tend to become more and more familiar devices nowadays. The basic component of such a smart environment will be a small node with sensing and wireless communications capabilities, able to organize itself flexibly into a network for data collection and delivery. Realizing such a sensor network presents many significant challenges, especially at the architectural, protocol, and operating system level. Major steps forward are required in the field of communications protocol, distributed data processing, and application support.

Although sensor nodes will be equipped with a power supply (battery) and embedded processor that makes them autonomous and self-aware, their functionality and capabilities will be very limited. However, deployed in larger numbers they can perform more complex tasks. Therefore, collaboration between nodes is essential to deliver smart services in a ubiquitous setting.

Several projects have been completed in this field [1][5], of which TinyOS developed at Berkeley University [3] is particularly dedicated to sensor nodes. Each of them tries

to give solutions to applications with very specific initial conditions. In the EYES project (European project, IST-2001-34734) [2] we address problems such as distributed information processing, wireless communication and mobile computing. We are also building a prototype sensor network to test new algorithms concerning the previous mentioned topics and also include research on reliability and security.

No matter how large a sensor network is, there are several "basic" operations that need to be performed by each node: network detection and configuration, acquisition, processing and delivery of collected data, etc. This paper presents our approach on designing an operating system that is able to provide these basic functionalities under the extremely lean conditions. On top of these ones there will be possible to develop more complex local or distributed applications.

## II. EYES SENSOR CHARACTERISTICS

There may be different types of sensors in a network, with different levels of functionality and different roles. Some nodes in the system may execute autonomously to provide networking and system services or control various information retrieval and dissemination. Other nodes have less functionality, basically just gather data, and communicate through a master node. Therefore, we intend to use two kinds of sensor nodes to do our experiments: a high-end sensor node that is capable of performing significant computation, and a low-end node (see Figure 1), that will have the basic functionality of the system, but whose processing capabilities are very limited. The operating system we are designing concerns the low-end nodes and takes into consideration the limited resources available.

The main processor used in the project is MSP430F149, produced by Texas Instruments [8]. It is a 16-bit processor and it has 60 Kbytes of program memory and 2 Kbytes of data memory. It also has several power saving modes. A node is also equipped with an auxiliary serial EEPROM memory of 8 Megabits (used for application and data storage). The access time of this memory is low, so it will have

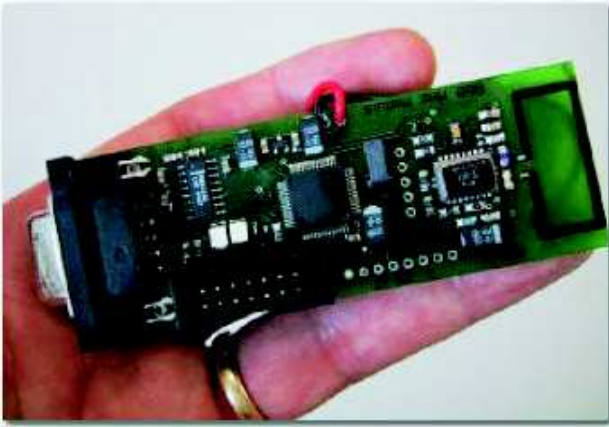


Fig. 1. EYES sensor node

a function similar to a secondary storage (e.g. a hard-disk drive) in a modern computer.

The communication function is realized by a TR1001 hybrid transceiver [6] that is very well suited for this kind of application: it has low power consumption and a small size. It is easy to use and integrates many aspects related to the communication mechanism in one single chip.

### III. EYES OPERATING SYSTEM REQUIREMENTS

Probably, the expression "operating system" is not the best description of the software we are developing. Still, we will be using this term because we are trying to offer as many facilities as the limited resource environment allows. Our system should come as close as possible to a fully featured operating system.

The main characteristics of the operating system are: small size, power awareness, distribution and reconfiguration. To these main features several other specific requirements are added that are described below.

- The small code size of the system is necessary because the given memory is small. At first sight, the memory constraints are very severe and it seems quite impossible to compact a fully operational OS within these limits. It is possible, however, if only necessary functionalities are implemented and they are modified to suit our application.
- Power awareness is an obligatory condition for our system because the low-end nodes are battery operated. The batteries are monitored so the microprocessor can find out at any time how much power is still available. So we treat energy as a special resource, and take certain measures when it reaches a certain (low) level (such as signal this event in the network, and reorganize the tasks it is willing (and able) to perform). The microprocessor we use was chosen also from the point of view of power consumption. It can be set to function in a very low power mode when there are no more operations to execute. However, the pe-

ripherals remain active so an external event (such as an interruption request from the radio component) can wake up the system.

- The network as a whole must behave as a distributed system because a central server will not be available. Still, the nodes have to be able to organize themselves into clusters and elect cluster leaders. This has several advantages, e.g. the single point of failure disappears and the communication load on closest nodes to the central server is no longer a problem.
- Failure of network nodes will be regarded as a "regular" phenomenon due to the large number of nodes with finite energy (possibly deployed in a harsh environment) and also due to the mobile nature of the nodes. The network will have the ability to reconfigure itself and to support several classes of failures (starting with crash-failures up to Byzantine behaviors)[4][7].

### IV. EYES OPERATING SYSTEM CHARACTERISTICS

The general principles described in the previous section leave space for several approaches. We have chosen the following specific guidelines in our design.

One modality to achieve small size of the code and, in the same time, to solve the limited available energy problem is to design the system to operate on an event-driven basis. This way, the code can be structured in modules, which will be executed as responses to external events. When there are no more external events to take care of, the system can enter the power-saving mode. To allow this, we must make sure that the code is written in a non-blocking manner.

For example, the programmer should not use loops to scan a certain input port, waiting for a value. Instead, at the time a new value arrives, it will generate an interrupt that will wake the microprocessor from the power-saving mode. The operating system will translate it to an event and then pass it to the appropriate process. From this small description we can find out one of the properties of processes: they should perform their computations, return a value and then enter the sleep mode.

To make things easier and to have a control over the size of the code, a task mechanism is used. A task defines a certain block of code that runs to completion. Each software module can ask for several tasks to be performed. A task scheduler maintains the execution of tasks. It can be implemented as a simple FIFO schema [3] or a more advanced one allowing priority and deadline driven executions for real-time applications. The interrupts are also seen as tasks that are scheduled to be executed.

The distributed nature of our system requires at least two basic facilities: resource management and a remote

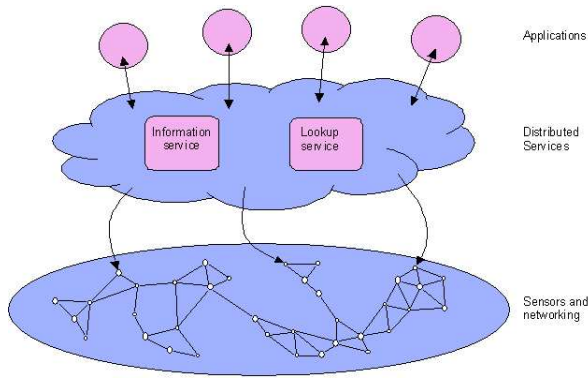


Fig. 2. EYES architecture overview

procedure call mechanism. By resource management we understand the operations of resource request, declaration and allocation. Resource requests appear in the case that a node has to perform a specific computation but does not have enough memory, energy, or even speed. A query will be submitted to the system (its neighbors) and the ones that can do the computation will respond (resource declaration and allocation occurs). The remote procedure call will be the mechanism that allows the requester to perform the needed computation using other's node resources.

The system must be robust to node failures. The routing algorithm, implemented also in the operating system, will be fault-tolerant. Applications built on top of it can rely on it but have to deal with failures themselves. This is necessary because even if all the nodes work without failures, due to their mobility, network configuration can change during a transmission leading to errors.

## V. EYES OS APPLICATION PROGRAMMING INTERFACE

In the EYES architecture we have defined two distinct key system layers of abstraction: the *Sensors and Networking Layer* and the *Distributed Services Layer* (see Figure 2):

- the *Sensors and Networking Layer* contains the sensor nodes (the physical sensor and wireless transmission modules) and the network protocols. Ad-hoc routing protocols allow messages to be forwarded through multiple sensor nodes taking into account the mobility of nodes, and the dynamic change of topology.
- the *Distributed Services Layer* contains distributed services for supporting mobile sensor applications. Distributed services coordinate with each other in order to perform decentralized services. We have identified two major services. The *Lookup Service* supports mobility, instantiation, and reconfiguration. The *Information Service* deals with aspects of collecting data. This service allows vast

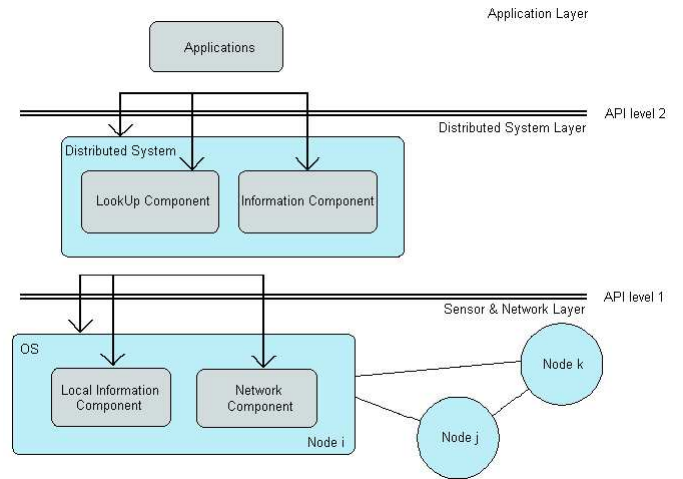


Fig. 3. EYES API overview

quantities of data to be easily and reliably accessed, manipulated, disseminated, and used in a customized fashion by applications.

On top of this architecture applications can be built using the sensor network and the distributed services.

The Application Programming Interface (API) was designed taking into consideration the previous described architecture (see Figure 3). There are two API levels (in fact there is one more not drawn in the picture - API level 0 - situated between the hardware level and the software level).

In this section we will focus on the API of the operating system (API level 1). It makes the connection between the Distributed Services Layer and the Sensors and Networking Layer. It is made up of two main blocks:

- the *Local Information Component* mainly provides access to sensor node data (get the local sensor readings, pre-processed data, etc.). Other functions of this component allow the access to the information concerning: the available resources (such as memory, energy, speed, network capabilities), status and capabilities (define/identify data types, software modules, etc.). There is also a group of functions (available only at the boot time) that allows the owner to set "vital" information for the node (calibrate the sensors, set the local variables, encryption types and keys, etc.).
- the *Network Component* manages the network functions. It provides the basic functions for transmitting (or multicasting) and receiving data packets (taking into account the security issues also). Other secondary functions retrieve information about the network (network size, neighbors, diameter, cluster leader, etc.) and traffic statistics for the node.

The *Operating System Component* provides other two groups of functions, built on top of the basic primitives described above. The first one offers information about the node localisation (relative or absolute). This can be done by querying the neighbors about the static nodes whose position is known in advance, by measuring the signal strength, etc. The second group of functions is available only at the boot time and enables the uploading of software modules to the sensor nodes. This provides an easy method of programming the network using the wireless interface (connecting hundreds of sensors one by one to a computer and downloading code in each of them is not such an easy or pleasant task).

The API level 2 sits between the application and the Distributed System Layer. The application can get the results of the sensor network (the processed data) or can ask the network to adapt and perform a specific function. The network structure and algorithms used are transparent for the end-user.

## VI. CONCLUSIONS

This paper describes the first step needed in constructing the sensor network: the definition of the minimal operating system. The characteristics defined above led to choosing the appropriate hardware components. They are also the premises for developing the top layers. We have tried to include the principal necessary features but also keep them at a minimum.

We have decided on these features after studying already existing systems but implementing fewer features than ours. The main challenges for the future development are the distributed nature of the system and the mobile nature of the nodes.

## REFERENCES

- [1] Berkley Univ. TinyOS webpage. <http://today.cs.berkeley.edu/tos/>.
- [2] EYES. EYES project webpage. <http://eyes.eu.org>.
- [3] Jason Hill. System Architecture Directions for Networked Sensors. In *Proceedings ASPLOS-IX*, pages 93–104, 2000.
- [4] N. Lynch. *Distributed Systems*. Morgan Kaufman Publishers, 1997.
- [5] pOSEK. pOSEK webpage. <http://www.isi.com/>.
- [6] RFM. TR1001 868.35 MHz Hybrid Transceiver. <http://www.rfm.com/products/data/tr1001.pdf>.
- [7] G. Tell. *Introduction to distributed algorithms*. Cambridge University Press, 1994.
- [8] Texas Instruments. MSP430F149 datasheet. <http://www.s.ti.com/sc/ds/msp430f149.pdf>.