EMBEDDED SYSTEMS

HANDBOOK

Edited by

RICHARD ZURAWSKI

Taylor & Francis
Taylor & Francis Group
Boca Raton London New York

A CRC title, part of the Taylor & Francis imprint, a member of the
Taylor & Francis Group, the academic division of T&F Informa plc.

Published in 2006 by
CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-10: 0-8493-2824-1 (Hardcover)
International Standard Book Number-13: 978-0-8493-2824-4 (Hardcover)
Library of Congress Card Number 2005040574

Library of Congress Cataloging-in-Publication Data

Embedded systems handbook / edited by Richard Zurawski.
p. cm.
Includes bibliographical references and index.
ISBN 0-8493-2824-1 (alk. paper)
1. Embedded computer systems--Handbooks, manuals, etc. I. Zurawski, Richard.

TK7895.E42E64 2005
004.16--dc22 2005040574

T&F informa
Taylor & Francis Group
is the Academic Division of T&F Informa plc.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

33

Architectures for Wireless Sensor Networks

S. Dulman,
S. Chatterjea,
T. Hoffmeijer,
P. Havinga,
and J. Hurink
University of Twente

The vision of ubiquitous computing requires the development of devices and technologies that can be pervasive without being intrusive. 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. Building such a sensor network presents many significant challenges, especially at the architectural, protocol, and operating system level.

Although sensor nodes might be equipped with a power supply or energy scavenging means and an embedded processor that makes them autonomous and self-aware, their functionality and capabilities will be very limited. Therefore, collaboration between nodes is essential to deliver smart services in a ubiquitous setting. New algorithms for networking and distributed collaboration need to be developed. These algorithms will be the key for building self-organizing and collaborative sensor networks that show emergent behavior and can operate in a challenging environment where nodes move, fail, and energy is a scarce resource.

The question that rises is how to organize the internal software and hardware components in a manner that will allow them to work properly and be able to adapt dynamically to new environments, requirements, and applications. At the same time the solution should be general enough to be suited for as many applications as possible. Architecture definition also includes, at the higher level, a global view of the whole network. The topology, placement of base stations, beacons, etc. is also of interest.

In this chapter, we will present and analyze some of the characteristics of the architectures for wireless sensor networks. Then, we will propose a new dataflow-based architecture that allows, as a new feature, the dynamic reconfiguration of the sensor nodes software at runtime.

33.1 Sensor Node Architecture

Current existing technology already allows integration of functionalities for information gathering, processing, and communication in a tight packaging or even in a single chip (e.g., Figure 33.1 presents the EYES sensor node [1]). The four basic blocks needed to construct a sensor node are (see Figure 33.2):

Sensor platform. The sensors are the interfaces to the real world. They collect the necessary information and have to be monitored by the central processing unit (CPU). The platforms may be built in a modular way such that a variety of sensors can be used in the same network. The utilization of a very wide range of sensors (monitoring characteristics of the environment, such as light, temperature, air pollution, pressure, etc.) is envisioned. The sensing unit can also be extended to contain one or more actuation units (e.g., to give the node the possibility of repositioning itself).

Processing unit. Is the intelligence of the sensor node will not only collect the information detected by the sensor but will also communicate with the server network. The level of intelligence in the sensor node will strongly depend on the type of information that is gathered by its sensors and by the way in which the network operates. The sensed information will be preprocessed to reduce the amount of data to be transmitted via the wireless interface. The processing unit will also have to execute some networking protocols in order to forward the results of the sensing operation through the network to the requesting user.

Communication interface. This is the link of each node to the sensor network itself. The focus relies on a wireless communication link, in particular on the radio communication, although visible or infrared light, ultrasound, etc. means of communications have already been used [2]. The used radio transceivers can usually function in simplex mode only, and can be completely turned off, in order to save energy.

Power source. Owing to the application areas of the sensor networks, autonomy is an important issue. Sensor nodes are usually equipped with a power supply in the form of one or more batteries. Current studies focus on reducing the energy consumption by using low-power hardware components

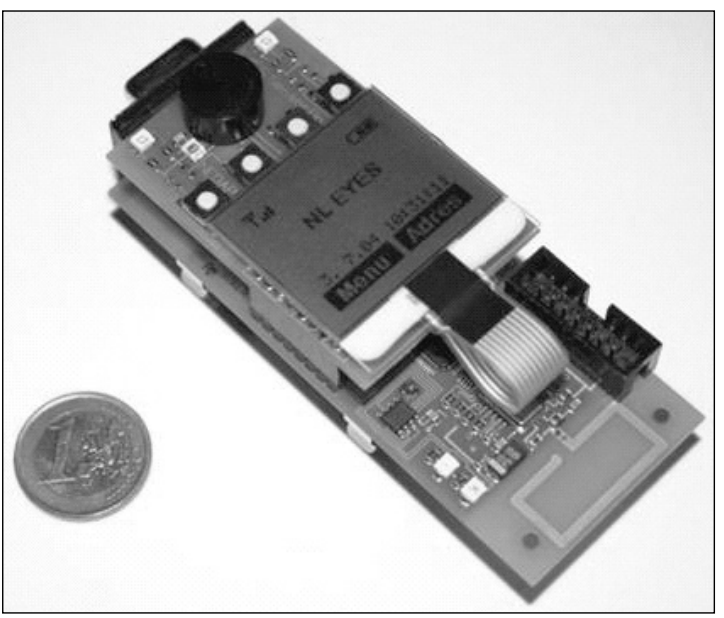

FIGURE 33.1 EYES sensor node. (From EYES. Eyes European project, http://eyes.eu.org. With permission.)

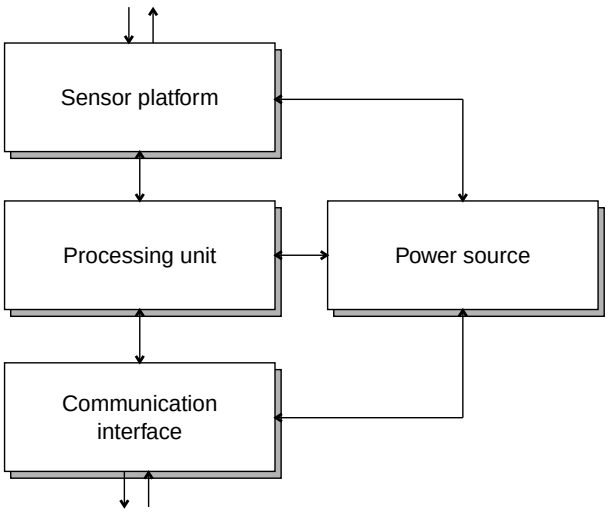

FIGURE 33.2 Sensor node components.

and advanced networking and data management algorithms. The usage of such energy scavenging techniques for sensor nodes might make possible for the sensor nodes to be self-powered. No matter which form of power source is used, energy is still a scarce resource and a series of trade-offs will be employed during the design phase to minimize its usage.

Sensor networks will be heterogeneous from the point of view of the types of nodes deployed. Moreover, whether or not any specific sensor node can be considered as being part of the network only depends on the correct usage and participation in the sensor network suite of protocols and not on the node's specific way of implementing software or hardware. An intuitive description given in Reference 3 envisions a sea of sensor nodes, some of them being mobile and some of them being static, occasionally containing tiny isles of relatively resource-rich devices. Some nodes in the system may execute autonomously (e.g., forming the backbone of the network by executing network and system services, controlling various information retrieval and dissemination functions, etc.), while others will have less functionality (e.g., just gathering data and relaying it to a more powerful node). Thus, from the sensor node architecture point of view we can distinguish between several kinds of sensor nodes. A simple yet sufficient in the majority of the cases approach would be to have two kinds of nodes: high-end sensor nodes (nodes that have plenty of resources or superior capabilities; the best candidate for such a node would probably be a fully equipped PDA device or even a laptop) and low-end nodes (nodes that have only the basic functionality of the system and have very limited processing capabilities).

The architecture of a sensor node consists of two main components: defining the precise way in which functionalities are needed and how to join them into a coherent sensor node. In other words, sensor node architecture means defining the exact way in which the selected hardware components connect to each other, how they communicate and how they interact with the CPU, etc.

A large variety of sensor node architectures have been built up to this moment. As a general design rule, all of them have targetted the following three objectives: energy efficiency, small size, and low cost. Energy efficiency is by far the most important design constraint because energy consumption depends on the lifetime of the sensor nodes. As the typical scenario of sensor networks deployment assumes that the power supplies of nodes will be limited and not rechargeable, a series of trade-offs need to be made to decrease the amount of consumed energy. Small size of the nodes leads to the ability of deploying lots of them to study a certain phenomenon. The ideal size is suggested by the name of one of the first research projects in the area: SmartDust [4]. Very cheap sensor nodes will lead to rapid deployment of such networks and large-scale usage.

33.1.1 Mathematical Energy Consumption Model of a Node

In this section, we present a basic version of an energy model for a sensor node. The aim of the model is to predict the current energy state of the battery of a sensor node based on historical data on the use of the sensor node and the current energy state of the battery.

In general a sensor node may consist of several components. The main components are: a radio, a processor, a sensor, a battery, external memory, and periphery (e.g., a voltage regulator or debugging equipment and periphery to drive an actuator). In the presented model we consider only the first four components. The external memory is neglected in this stage of the research since its use of energy is rather complex and needs an own energy model if the memory is a relevant part of the functional behavior of the sensor node and not just used for storage. The periphery can be quite different and, thus, can not be integrated in an energy model of a sensor node in a uniform way.

For the battery we assume that the usage of energy by the other components is independent of the current energy state of the battery. This implies that the reduction of the energy state of the battery depends only on the actions of the different components. Furthermore, we do not consider a reactivation of the battery by time or external circumstances. Based on these assumptions, it remains to give models for the energy consumption of the three components radio, processor, and sensor.

The base of the model for the energy consumption of a component is the definition of a set S of possible states $s_1, \ldots, s_k$ for the component. These states are defined such that the energy consumption of the component is given by the sum of the energy consumptions within the states $s_1, \ldots, s_k$ plus the energy needed to switch between the different states. We assume that the energy consumption within a state s_j can be measured using a simple index t_j (e.g., execution time or number of instructions) and that the energy needed to switch between the different states can be calculated on the basis of a state transition matrix st, where st_{ij} denotes the number of times the component has switched from state s_i to state s_j. If now P_j denotes the needed power in the state s_j and E_{ij} denotes the energy consumption of switching once from state s_i to state s_j, the total energy consumption of the component is given by

$$E_{\text{consumed}} = \sum_{j=1}^{k} t_j P_j + \sum_{i,j=1, i\neq j}^{k} \mathrm{st}_{ij} E_{ij} \tag{33.1}$$

In the following, we describe the state sets S and the indices to measure the energy consumption within the states for the radio, processor, and sensor:

Radio. For the energy consumption of a radio four different states need to be distinguished: *off*, *sleep*, *receiving*, and *transmitting*. For all these four states the energy consumption depends on the time the radio has been in the state. Thus, for the radio we need to memorize the times the radio has been in the four states and the 4×4 state transition matrix representing the number of times the radio has switched between the four states.

Processor. In general, for a processor, four main states can be identified: *off*, *sleep*, *idle*, and *active*. In sleep mode the CPU and most internal peripherals are turned off. It can be awaked by an external event (interrupt) only, on which idle state is entered. In idle mode the CPU is still inactive, but now some peripherals are active, such as the internal clock or timer). Within the *active* state the CPU and all peripherals are active. Within this state multiple states might be identified based on clock speeds and voltages. We assume that the energy consumption depends on the time the processor has been in a certain state.

Sensor. For a (simple) sensor we assume that only the two states *on* and *off* are given and that the energy consumption within both states can be measured by time. However, if more powerful sensors are used, it may be necessary to work with more states (similar to the processor or the radio).

The energy model for the complete sensor node now consists of the energy models for the three components radio, processor, and sensor plus two extra indicators for the battery:

- For the battery only the energy state E_{old} at a time t_{old} in the past is given.
- For each component, the indices I_j characterizing the energy consumption in state s_j since time t_{old} and the state transition matrix st indicating the transitions since time t_{old} are specified.

Based on this information an estimate of the current energy state of the battery can be calculated by subtracting from E_{old} the sum of the consumed energy for each component estimated on the base of Equation (33.1).

Since the energy model gives only an estimate of the remaining energy of the battery, in practice it may be a good approach to use the energy model only for limited time intervals. If the difference between the current time and t_{old} gets larger than a certain threshold, the current energy state of the battery should be estimated on the base of measurements or other information available on the energy state and E_{old} and t_{old} should be replaced by this new estimate and the current time. Furthermore, the indices characterizing the states and the state transition matrix are reset for all the components of the sensor node.

33.2 Wireless Sensor Network Architectures

A sensor network is a very powerful tool when compared to a single sensing device. It consists of a large number of nodes, equipped with a variety of sensors that are able to monitor different characteristics of a phenomenon. A dense network of such small devices, will give the researcher the opportunity to have a spatial view over the phenomenon and, at the same time, it will produce results based on a combination of various sorts of sensed data.

Each sensor node will have two basic operation modes: initialization phase and operation phase. But, the network as a whole will function in a smooth way, with the majority of the nodes in the operation mode and only a subset of nodes in the initialization phase. The two modes of operation for the sensor nodes have the following characteristics:

Initialization mode. A node can be considered in initialization mode if it tries to integrate itself in the network and is not performing its routine function. A node can be in initialization mode, for example, at power on or when it detects a change in the environment and needs to configure itself. During initialization, the node can pass through different phases such as detecting its neighbors and the network topology, synchronizing with its neighbors, determining its own position or even performing configuration operations on its own hardware and software. At a higher abstraction level, a node can be considered in initialization mode if it tries to determine which services are already present in the network, which services it needs to provide or can use.

Operation mode. After the initialization phase the node enters a stable state, the regular operation state. It will function based on the conditions determined in the initialization phase. The node can exit the operation mode and pass through an initialization mode if either the physical conditions around it or the conditions related to the network or to itself have changed. The operation mode is characterized by small bursts of node activity (such as reading sensors values, performing computations, or participating in networking protocols) and periods spent in an energy-saving low-power mode.

33.2.1 Protocol Stack Approach

A first approach to building a wireless sensor network will be to use a layered protocol stack as a starting point, as in the case of traditional computer network. The main difference between the two kinds of networks is that the blocks needed to build the sensor network usually span themselves over multiple layers while others depend on several protocol layers. This characteristic of sensor networks comes from the fact that they have to provide functionalities that are not present in traditional networks. Figure 33.3 presents an approximative mapping of the main blocks onto the traditional OSI protocol layers.

The authors of Reference 5 propose an architecture based on the five OSI layers together with three management planes that go throughout the whole protocol stack (see Figure 33.4). A brief description of the layers included: (1) the physical layer, which addresses mainly the hardware details of the wireless communication mechanism, such as the modulation type, the transmission and receiving techniques, etc.; (2) the data-link layer is concerned with the Media Access Control (MAC) protocol that manages communication over the noisy shared channel; (3) the network layers manages routing the data between the nodes, while the transport layer helps to maintain the dataflow; (4) finally, the application layer contains (very often) only one single user application.

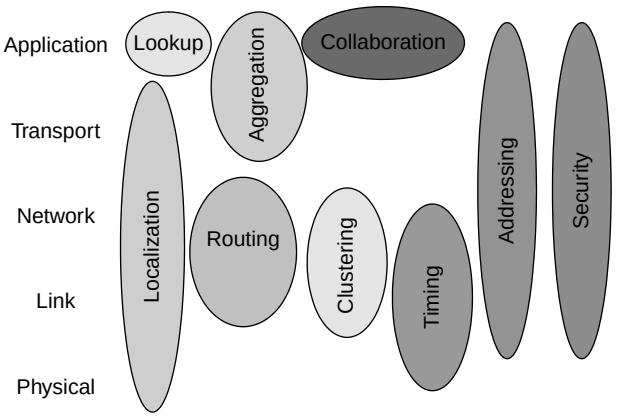

FIGURE 33.3 Relationship between building blocks and OSI layers.

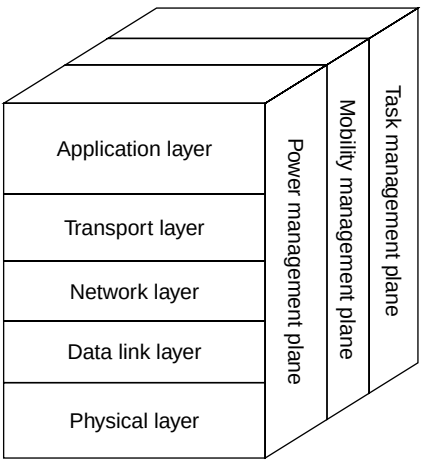

FIGURE 33.4 Protocol stack representation of the architecture. (From Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. *IEEE Communication Magazine*, 40, 102–114, 2002. With permission.)

In addition to the five network layers, three management planes have the following functionality: the power management plane coordinates the energy consumption inside the sensor node. It can, for example, be based on the available amount of energy, allow the node to take part in certain distributed algorithms or to control the amount of traffic it wants to forward. The mobility management plane will manage all the information regarding the physical neighbors and their movement patterns as well as its own moving pattern. The task management plane coordinates sensing in a certain region based on the number of nodes and their placement (in very densely deployed sensor networks, energy might be saved by turning certain sensors off to reduce the amount of redundant information sensed).

In the following we will give a description of the main building blocks needed to setup a sensor network. The description will follow the OSI model. This should not imply that this is the right structure for these networks, but is taken only as a reference point:

Physical layer. The physical layer is responsible for the management of the wireless interface. For a given communication task, it defines a series of characteristics as: operating frequency, modulation type, data coding, interface between hardware and software, etc.

The large majority of already built sensor networks prototypes and most of the envisioned application scenarios assume the use of a radio transceiver as the means for communication. The unlicensed industrial,

scientific, and medical (ISM) band is preferred because it is a free band designed for short-range devices using low-power radios and requiring low data-transmission rates. The modulation scheme used is another important parameter to decide upon. Complex modulation schemes might not be preferred because they require important resources (in the form of energy, memory, computation power).

In the future, the advancements of the integrating circuits technology (e.g., ASIC, FPGA) will allow the use of modulation techniques, such as ultrawide band (UWB) or impulse radio (IR), while if the sensor node is built using off-the-shelf components the choice comes down mainly to schemes such as amplitude shift keying (ASK) or frequency shift keying (FSK). Based on the modulation type and on the hardware used, a specific data encoding scheme will be chosen to assure both the synchronization required by the hardware component and a first level of error correction. At the same time, the data frame will also include some carefully chosen initial bytes needed for the conditioning of the receiver circuitry and clock recovery.

It is worth mentioning that the minimum output power required to transmit a radio signal over a certain distance is directly proportional to the distance raised to a power between two and four (the coefficient depends on the type of the antenna used and its placement relative to the ground, indoor–outdoor deployment, etc.). In these conditions, it is more efficient to transmit a signal using a multihop network composed of short-range radios rather than using a (power consuming) long-range link [5].

The communication subsystem usually needs a controller hierarchy to create the abstraction for the other layers in the protocol stack (we are referring to the device hardware characteristics and the strict timing requirements). If a simple transceiver is used, some of these capabilities will need to be provided by the main processing unit of the sensor node (this can require a substantial amount of resources for exact timing execution synchronization, crosslayer distribution of the received data, etc.). The use of more advanced specialized communication controllers is not preferred as they will hide important low-level details of information.

Data-link layer. The data-link layer is responsible for managing most of the communication tasks within one hop (both point-to-point and multicasting communication patterns). The main research issues here are the MAC protocols, the error control strategies and the power consumption control.

The MAC protocols make the communication between several devices over a shared channel possible by coordinating the sending–receiving actions as a function of time or frequency. Several strategies have already been studied and implemented for the mobile telephony networks and for the mobile ad hoc networks but unfortunately, none of them is directly applicable. Still, ideas can be borrowed from the existing standards and applications and new MAC protocols can be derived — this can be proven by the large number of new schemes that target specifically the wireless sensor networks.

As the radio component is probably the main energy consumer in each sensor node, the MAC protocol must be very efficient. To achieve this, the protocol must, first of all, make use of the power down state of the transceiver (turn the radio off) as much as possible because the energy consumption is negligible in this state. The most important problem comes from the scheduling of the sleep, receive, and transmit states. The transitions among these states also need to be taken into account as they consume energy and sometimes take large time intervals. Message collisions, overhearing, and idle listening are direct implications of the scheduling used inside the MAC protocol which, in addition, influences the bandwidth lost due to the control packet overheads.

A second function of the data-link layer is to perform error control of the received data packets. The existent techniques include automatic repeat-request (ARQ) and forward error correction (FEC) codes. The choice of a specific technique comes down to the trade-off between the energy consumed to transmit redundant information over the channel and the energy and high computation power needed at both the coder/decoder sides.

Additional functions of the data-link layer are creating and maintaining a list of the neighbor nodes (all nodes situated within the direct transmission range of the node in discussion); extracting and advertising the source and destination as well as the data content of the overheard packets; supplying information related to the amount of energy spent on transmitting, receiving, coding, and decoding the packets, the amount of errors detected, the status of the channel, etc.

Network layer. The network layer is responsible for routing of the packets inside the sensor network. It is one of the most studied topics in the area of wireless sensor networks and it received a lot of attention lately. The main design constraint for this layer is, as in all the previous cases, the energy efficiency.

The main function of wireless sensor networks is to deliver sensed data (or data aggregates) to the base stations requesting it. The concept of data-centric routing has been used to address this problem in an energy-efficient manner, minimizing the amount of traffic in the network. In data-centric routing, each node is assigned a specific task based on the interests of the base stations. In the second phase of the algorithm, the collected data is sent back to the requesting nodes. Interest dissemination can be done in two different ways, depending on the expected amount of traffic and level of events in the sensor network: the base stations can broadcast the interest to the whole network or the sensor nodes themselves can advertise their capabilities and the base stations will subscribe to that.

Based on the previous considerations, the network layer needs to be optimized mainly for two operations: spreading the user queries, generated at one or more base stations, around the whole network, and then retrieving the sensed data to the requesting node. Individual addressing of each sensor node is not important in the majority of the applications.

Due to the high density of the sensor nodes, a lot of redundant information will be available inside the sensor network. Retrieving all this information to a certain base station might easily exceed the available bandwidth, making the sensor network unusable. The solution to this problem is the data aggregation technique, which requires each sensor node to inspect the content of the packets it has to route and aggregate the contained information, reducing the high redundancy of the multiple sensed data. This technique was proven to substantially reduce the overall traffic and make the sensor network behave as an instrument for analyzing data rather than just a transport infrastructure for raw data [6].

Transport layer. This layer appears from the need to connect the wireless sensor network to an external network, such as the Internet, in order to disseminate its data readings to a larger community [7]. Usually the protocols needed for such interconnections require significant resources and they will not be present in all the sensor nodes. The envisioned scenario is to allow a small subset of nodes to behave as gateways between the sensor network and some external networks. These nodes will be equipped with superior resources and computation capabilities, and will be able to run the needed protocols to interconnect the networks.

Application layer. The application layer usually links the user's applications with the underlying layers in the protocol stack. Sensor networks are designed to fulfill one single application scenario for each particular case. The whole protocol stack is designed for a special application and the whole network is seen as an instrument. These make the application layer to be distributed along the whole protocol stack, and not appear explicitly. Still, for the sake of classification we can consider an explicit application layer that could have one of the following functionalities [5]: sensor management protocol, task management and data advertisement protocol, and sensor query and data dissemination protocol.

33.2.2 EYES Project Approach

The approach taken in the EYES project [1] consists of only two key system abstraction layers: the sensor and networking layer and the distributed services layer (see Figure 33.5). Each layer provides services that may be spontaneously specified and reconfigured:

1. The sensor 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. Communication protocols must be energy efficient since sensor nodes have very limited energy supplies. To provide more efficient dissemination of data, some sensors may process data streams, and provide replication and caching.

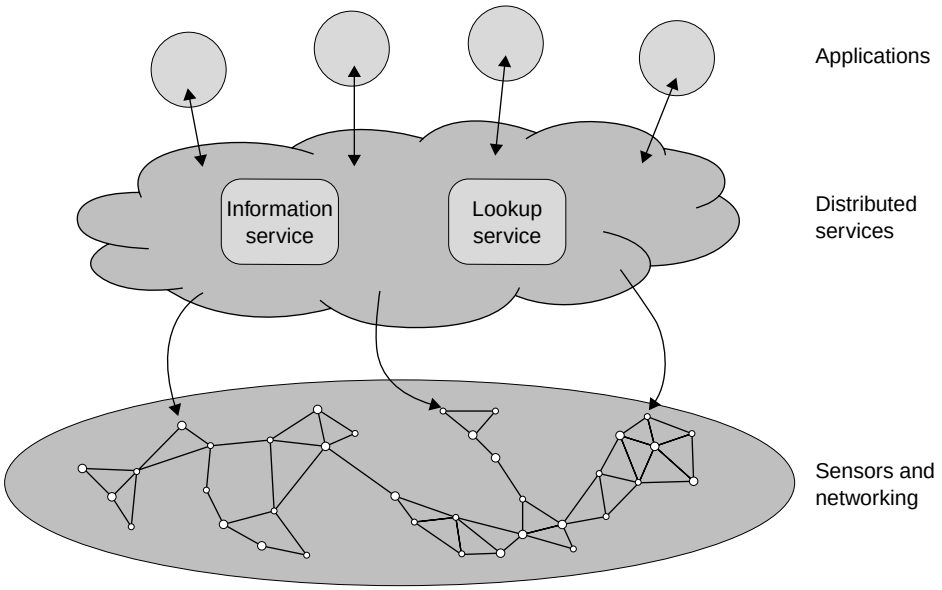

FIGURE 33.5 EYES project architecture description.

2. The distributed services layer contains distributed services for supporting mobile sensor applications. Distributed services coordinate with each other to perform decentralized services. These distributed servers may be replicated for higher availability, efficiency, and robustness. 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 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 distributed services. Communication in a sensor network is data centric since the identity of the numerous sensor nodes is not important, only the sensed data together with time and location information counts. The three main functions of the nodes within a sensor network are directly related to this:

Data discovery. Several classes of sensors will be equipped in the network. Specialized sensors can monitor climatic parameters (humidity, temperature, etc.), motion detection, vision sensors, and so on. A first step of data preprocessing can also be included in this task.

Data processing and aggregation. This task is directly related to performing distributed computations on the sensed data and also aggregating several observations into a single one. The goal of this operation is the reduction of energy consumption. Data processing influences it by the fact that the transmission of one (raw sensed) data packet is equivalent to many thousands of computation cycles in the current architectures. Data aggregation keeps the overall traffic low by inspecting the contents of the routed packets, and in general, reducing the redundancy of the data in traffic by combining several similar packets into a single one.

Data dissemination. This task includes the networking functionality comprising routing, multicasting, broadcasting, addressing, etc.

The existing network scenarios contain both static and mobile nodes. In some cases, the static nodes can be considered to form a back-bone of the network and are more likely to be preferred in certain distributed protocols. Both mobile and static nodes will have to perform data dissemination, so the protocols should be designed to be invariant to node mobility. The particular hardware capabilities of each kind of sensor node will determine how the previously described tasks will be mapped onto them (in principle, all the

nodes could provide all the previous functionalities). During the initialization phase of the network, the functionality of every node will be decided based on both the hardware configurations and the particular environmental conditions.

For a large sensor network to be able to function correctly, a tiered architecture is needed. This means that nodes will have to organize themselves into clusters based on certain conditions. The nodes in each cluster will elect a leader — the best fitted node to perform coordination inside the cluster (this can be e.g., the node with the highest amount of energy or the node having the most advanced hardware architecture or just a random node). The cluster leader will be responsible for scheduling the node operations, managing the resources and the cluster structure, and maintaining communication with the other clusters.

We can talk about several types of clusters that can coexist in a single network:

Geographical clustering. The basic mode of organizing the sensor network. The clusters are built based on the geographical proximity. Neighboring nodes (nodes that are in transmission range of each other) will organize themselves into groups. This operation can be handled in a completely distributed manner and it is a necessity for the networking protocols to work even when the network scales up.

Information clustering. The sensor nodes can be grouped into information clusters based on the services they can provide. This clustering structure belongs to the distributed services layer and is built on top of the geographical clustering. Nodes using this clustering scheme need not be direct neighbors from the physical point of view.

Security clustering. A even higher hierarchy appears if security is taken into consideration. Nodes can be grouped based on their trust levels or based on the actions they are allowed to perform or resources they are allowed to use in the network.

Besides offering increased capabilities to the sensor network, clustering is considered as one of the principal building blocks for the sensor networks also from the point of view of energy consumption. The overhead given by the energy spent for creating and organizing the sensor network is easily recovered in the long term due to the reduced traffic it leads to.

33.2.2.1 Distributed Services Layer Examples

This section focuses on the distributed services that are required to support applications for wireless sensor networks. We discuss the requirements of the foundation necessary to run these distributed services and describe how various research projects approach this problem area from a multitude of perspectives. A comparison of the projects is also carried out.

One of the primary issues of concern in wireless sensor networks is to ensure that every node in the network is able to utilize energy in a highly efficient manner so as to extend the total network lifetime to a maximum [5, 8, 9]. As such, researchers have been looking at ways to minimize energy usage at every layer of the network stack, starting from the physical layer right up to the application layer.

While there are a wide range of methods that can be employed to reduce energy consumption, architectures designed for distributed services generally focus on one primary area — how to reduce the amount of communication required and yet get the main job done without any significant negative impact by observing and manipulating the data that flows through the network [6, 10, 11]. This leads us to look at the problem at hand from a *data-centric* perspective.

In conventional IP-style communication networks, such as on the Internet for instance, nodes are identified by their end-points and internode communication is layered on an end-to-end delivery service that is provided within the network. At the communication level, the main focus is to get connected to a particular node within the network, thus the addresses of the source and destination nodes are of paramount importance [12]. The precise *data* that actually flows through the network is irrelevant to IP.

Sensor networks, however, have a fundamental difference compared to the conventional communication networks described above as they are *application-specific networks*. Thus instead of concentrating on

which particular node a certain data message is originating from, a greater interest lies in the data message itself — what is the data in the data message and what can be done with it? This is where the concept of a data-centric network architecture comes into play.

As sensor nodes are envisioned to be deployed by the hundreds and potentially even thousands [8], specific sensor nodes are not usually of any interest (unless of course a particular sensor needs to have its software patched or a failure needs to be corrected). This means that instead of a sensor network application asking for the temperature of a particular node with ID 0315, it might pose a query asking, "What is the temperature in sector D of the forest?"

Such a framework ensures that the acquired results are not just dependent on a single sensor. Thus other nodes in sector D can respond to the query even if the node with ID 0315 dies. The outcome is not only a more robust network but due to the high density of nodes [13], the user of the network is also able to obtain results of a higher fidelity (or resolution). Additionally, as nodes within the network are able to comprehend the meaning of the data passing through them, it is possible for them to carry out application-specific processing within the network thus resulting in the reduction of data that needs to be transmitted [14]. In-network processing is particularly important as local computation is significantly cheaper than radio communication [15].

33.2.2.1.1 Directed Diffusion

Directed Diffusion is one of the pioneering data-centric communication paradigms developed specifically for wireless sensor networks [6]. Diffusion is based on a publish/subscribe API (application–programming interface), where the details of how published data is delivered to subscribers is hidden from the data producers (sources) and publishers (sinks). The transmission and arrival of events (*interest* or *data* messages) occur asynchronously. Interests describe tasks that are expressed using a list of attribute-value pairs as shown below:

```
// detect location of seagull
type = seagull
// send back results every 20ms
interval = 20ms
// for the next 15 seconds
duration = 15s
// from sensors within rectangle
rect = [-100,100,200,400]
```

A node that receives a data message sends it to its Filter API, which subsequently performs a matching operation according to a list of attributes and their corresponding values. If a match is established between the received data message and the filter residing on the node, the diffusion substrate passes the event to the appropriate application module. Thus the Filter API is able to influence the data which propagates through the network from the source to the sink node as an application module may perform some application-specific processing on the received event, for example, it may decide to aggregate the data. For example, consider a scenario in an environmental monitoring project where the user needs to be notified when the light intensity in a certain area goes beyond a specified threshold. As the density of deployed nodes may be very high, it is likely that a large number of sensors would respond to an increase in light intensity simultaneously. Instead of having every sensor relaying this notification to the user, intermediate nodes in the region could aggregate the readings from their neighboring nodes and return only the Boolean result thus greatly reducing the number of radio transmissions.

Apart from aggregating data by simply suppressing duplicate messages, application-specific filters can also take advantage of named-data to decide how to relay data messages back toward the sink node and what data to cache in order to route future interest messages in a more intelligent and energy-saving manner. Filters also help save energy by ensuring that nodes react appropriately to incoming events only if the attribute matching process has proven to be successful.

Diffusion also supports a more complex form of in-network aggregation. Filters allow *nested queries* such that one sensor is able to trigger other sensors in its vicinity if the attribute-value matching operation

is successful. It is not necessary for a user to directly query all the relevant sensors. Instead the user only queries a certain sensor which in turn eventually queries the other relevant sensors around it if certain conditions are met. In this case, energy savings are obtained from two aspects. First, since the user may be geographically distant from the observed phenomenon, the energy spent transmitting data can be reduced drastically using a triggering sensor. Second, if sampling the triggered (or secondary) sensor consumes a lot more energy than the triggering (initial) sensor, then energy consumption can be reduced greatly by reducing the duty cycle of the secondary sensor to only periods when certain conditions are met at the initial sensor.

33.2.2.1.2 COUGAR

Building up on the same concept, that processing data within the network would result in significant energy savings, but deviating from the library-based lower-level approach, that is, as used by Directed Diffusion, the COUGAR [10, 16] project envisions the sensor network as an extension of a conventional database thus viewing it as a device database system. It makes the usage of the network more user-friendly by suggesting the use of a high-level declarative language similar to SQL. Using a declarative language ensures that queries are formulated independent of the physical structure and organization of the sensor network.

Conventional database systems use a *warehousing* approach [17] where every sensor that gathers data from an environment subsequently relays that data back to a central site where this data is then logged for future processing. While this framework is suitable for historical queries and snapshot queries, it cannot be used to service *long-running* queries [17]. For instance, consider the following query:

Retrieve the rainfall level for all sensors in sector A every 30 sec if it is greater than 60 mm.

Using the warehousing approach, every sensor would relay its reading back to a central database every 30 sec regardless of whether it is in sector A or its rainfall level reading is greater than 60 mm. Upon receiving all the readings from the sensors, the database would then carry out the required processing to extract all the relevant data. The primary problem in this approach is that excessive resources are consumed at each and every sensor node as large amounts of raw data need to be transmitted through the network.

As the COUGAR approach is modeled around the concept of a database, the system generally proceeds as follows. It accepts a query from the user, produces a query execution plan (which contains detailed instructions of how exactly a query needs to be serviced), executes this plan against the device database system, and produces the answer. The query optimizer generates a number of query execution plans and selects the plan that minimizes a given cost function. The cost function is based on two metrics, namely resource usage (expressed in Joules) and reaction time.

In this case, the COUGAR approach selects the most appropriate query execution plan that pushes the selection (rainfall level >60 mm) onto the sensor nodes. Only the nodes that meet this condition send their readings back to the central node. Thus just like in Directed Diffusion, the key idea here is to transfer part of the processing to the nodes themselves, which in turn would reduce the amount of data that needs to be transmitted.

33.2.2.1.3 TinyDB

Following the steps of Directed Diffusion and the COUGAR project, TinyDB [11] also proclaims the use of some form of in-network processing to increase the efficiency of the network and thus improve network lifetime. However, while TinyDB views the sensor network from the database perspective just like COUGAR, it goes a step further by pushing not only selection operations to the sensor nodes but also basic aggregation operations that are common in databases, such as MIN, MAX, SUM, COUNT, and AVERAGE.

Figure 33.6 illustrates the obvious advantage that performing such in-network aggregation operations have compared to transmitting just raw data. Without aggregation, every node in the network needs to transmit not only its own reading but also those of all its children. This not only causes a bottleneck close

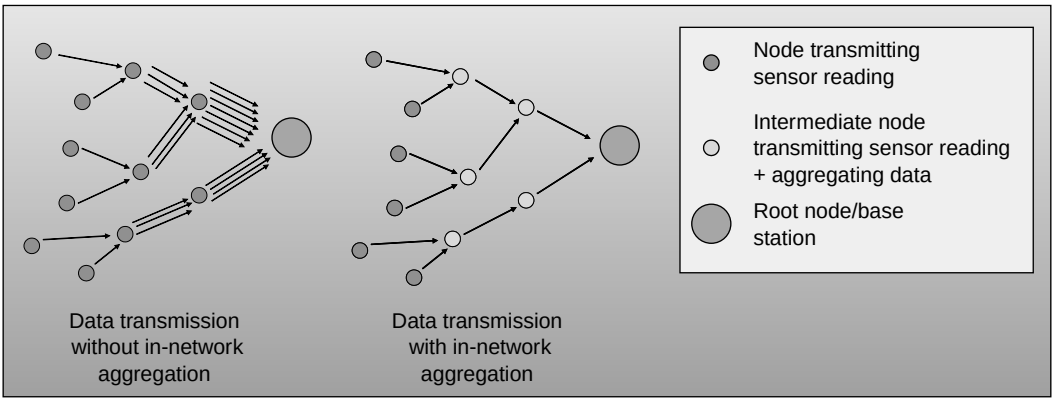

FIGURE 33.6 The effect of using in-network aggregation.

to the root node but also results in unequal consumption of energy, that is, the closer a node is to the root node, the larger the number of messages it needs to transmit, which naturally results in higher energy consumption. Thus nodes closer to the root node die earlier. Losing nodes closer to the root node can have disastrous consequences on the network due to network partitioning. Using in-network aggregation, however, every intermediate node aggregates its own reading with that of its children and eventually transmits only one combined result.

Additionally, TinyDB has numerous other features, such as communication scheduling, hypothesis testing, and acquisitional query processing, which makes it one of the most feature-rich distributed query processing frameworks for wireless sensor networks at the moment.

TinyDB requires users to specify queries injected into the sensor network using an SQL-like language. This language describes what data needs to be collected and how it should be processed upon collection as it propagates through the network toward the sink node. The language used by TinyDB varies from traditional SQL in the sense that the semantics supports queries that are continuous and periodic. For example, a query could state: "Return the temperature reading of all the sensors on Level 4 of the building every 5 min over a period of 10 h." The period of time between every successive sample is known as an *epoch* (in this example it is 5 min).

Just like in SQL, TinyDB queries follow the "SELECT - FROM - WHERE - GROUPBY - HAVING" format that supports selection, join, projection, aggregation, and grouping. Just like in COUGAR, sensor data is viewed as a single virtual table with one column per sensor type. Tuples are appended to the table at every epoch. Epochs also allow computation to be scheduled such that power is minimized. For example, the following query specifies that each sensor should report its own identifier and temperature readings once every 60 sec for a duration of 300 sec:

```
SELECT nodeid, temp
FROM sensors
SAMPLE PERIOD 60s FOR 300s
```

The virtual table sensors is conceptually an unbounded, continuous data stream of values that contain one column for every attribute and one row for every possible instant in time. The table is not actually stored in any device, that is, it is not materialized but sensor nodes only generate the attributes and rows that are referenced in active queries. Apart from the standard query shown above, TinyDB also supports event-based queries and lifetime queries [18]. Event-based queries reduce energy consumption by allowing nodes to remain dormant until some triggering event is detected. Lifetime queries are useful when users are not particularly interested in the specific rate of incoming readings but more on the required lifetime of the network. So the basic idea is to send out a query saying that sensor readings are required for say 60 days. The nodes then decide on the best possible rate at which readings can be sent given the specified network lifetime.

Queries are disseminated into the network via a routing tree rooted at the base station that is formed as nodes forward the received query to other nodes in the network. Every parent node can have multiple child nodes but every child node can only have a single parent node. Every node also keeps track of its distance from the root node in terms of the number of hops. This form of communication topology is commonly known as tree-based routing.

Upon receiving a query, each node begins processing it. A special acquisition operator at each node acquires readings from sensors corresponding to the fields or attributes referenced in the query. Similar to the concept of nested queries in Directed Diffusion, where sensors with a low sampling cost are sampled first, TinyDB performs the ordering of sampling and predicates. Consider the following query as an example where a user wishes to obtain readings from an accelerometer and a magnetometer provided certain conditions are met:

```
SELECT accel, mag
FROM sensors
WHERE accel > c1
AND mag > c2
SAMPLE INTERVAL 1s FOR 60s
```

Depending on the cost of sampling the accelerometer and the magnetometer sensors, the optimizer will first sample the cheaper sensor to see if its condition is met. It will only proceed to the more costly second sensor if the first condition has been met.

Next we describe how the sampled data is processed within the nodes and is subsequently propagated up the network toward the root node. Consider the following query:

Report the average temperature of the fourth floor of the building every 30 sec.

To service the above query, the query plan has three operators: a data acquisitional operator, a select operator that checks if the value of floor equals 4, and the aggregate operator that computes the average temperature from not only the current node but also its children located on the fourth floor. Each sensor node applies the plan once per epoch and the data stream produced at the root node is the answer to the query. The partial computation of averages is represented as {sum, count} pairs, which are merged at each intermediate node in the query plan to compute a running average as data flows up the tree.

TinyDB uses a *slotted* scheduling protocol to collect data where parent and child nodes receive and send (respectively) data in the tree-based communication protocol. Each node is assumed to produce exactly one result per epoch, which must be forwarded all the way to the base station. Every epoch is divided into a number of fixed-length intervals that is dependent on the depth of the tree. The intervals are numbered in reverse order such that interval 1 is the last interval in the epoch. Every node in the network is assigned to a specific interval that correlates to its depth in the routing tree. Thus for instance if a particular node is two hops away from the root node, it is assigned the second interval. During its own interval, a node performs the necessary computation, transmits its result and goes back to sleep. In the interval preceding its own, a node sets its radio to "listen" mode collecting results from its child nodes. Thus data flows up the tree in a staggered manner eventually reaching the root node during interval 1 as shown in Figure 33.7.

33.2.2.1.4 Discussion

In this section we do a comparison of the various projects described above and highlight some of their drawbacks. We also mention some other work in the literature that has contributed further improvements to some of these existing projects. Table 33.1 shows a list comparing some of the key features of the various projects.

As mentioned earlier, Directed Diffusion was a pioneering project in the sense that it introduced the fundamental concept of improving network efficiency by processing data within the sensor network. However, unlike COUGAR and TinyDB it does not offer a particularly simple interface, flexible naming system, or any generic aggregation and join operators. Such operators are considered as application-specific operators and must always be coded in a low-level language. A drawback of this approach is that

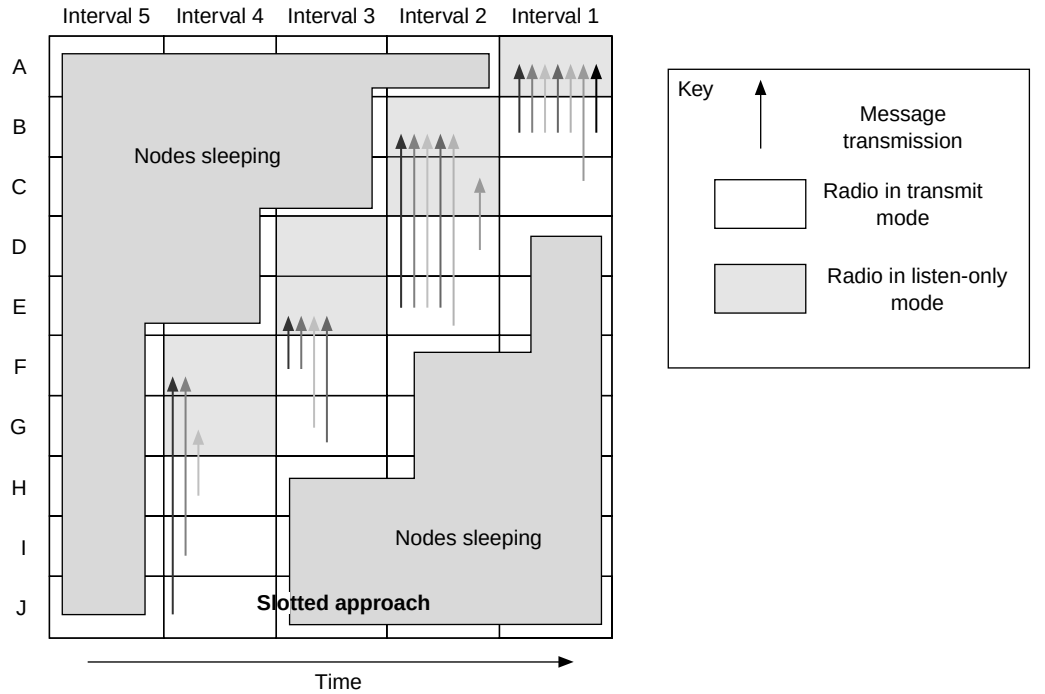

FIGURE 33.7 Communicating scheduling in TinyDB using the slotted approach [19].

TABLE 33.1 Comparison of Data Management Strategies

	Directed Diffusion	COUGAR	TinyDB
Type	Non-database	Database	Database
Platform	iPAQ class (Mote class for micro-diffusion)	iPAQ class	Mote class
Query language	Application specific, dependent on Filter API	SQL-based	SQL-based
Type of in-network aggregation	Suppression of identical data messages from different sources	Selection operators	Selection, aggregation operators and limited optimization
Crosslayer features	Routing integrated with in-network aggregation	None	Routing integrated with in-network aggregation; communication scheduling also decreases burden on the MAC layer
Caching of data for future routing	Yes	No	Yes
Power saving mechanism while sampling sensors	Yes — nested queries	None	Yes — acquisitional query processing
Type of optimization	None	Centralized	Mostly centralized — metadata is occasionally copied to catalogue

query optimizers are unable to deal with such user-defined operators as there are no fixed semantics. This is because query operators are unable to make the necessary cost comparisons between various user-defined operators. A direct consequence of this is that since the system is not able to handle optimization tasks autonomously, the arduous responsibility of placement and ordering of operators is placed on the

user. This naturally would be a great hindrance to users of the system (e.g., environmentalists) who are only concerned with injecting queries into the network and obtaining the results — not figuring out the intricacies of energy-efficient mechanisms to extend network lifetime!

While the COUGAR project specifically claims to target wireless sensor networks [20,21], apart from the feature of pushing down selection operations into the device network, it does not demonstrate any other novel design characteristics that would allow it to run on sensor networks. In fact, the COUGAR project has simulations and implementations using Linux-based iPAQ class hardware that has made them take certain design decisions that would be unsuitable for sensor networks. For instance, unlike Directed Diffusion [14] and TinyDB [18], COUGAR does not take the cost incurred by sampling sensors into consideration during the generation of query execution plans. It also does not take advantage of certain inherent properties of radio communication, for example, snooping, and also fails to suggest any methods that could link queries to communication scheduling. Additionally, the usage of XML to encode messages and tuples makes it inappropriate for sensor networks given their limited bandwidth and high cost of transmission per bit.

Among the various query processing systems currently stated in the literature, TinyDB seems to be the one that is the most feature packed. The TinyDB software has been deployed using Mica2 motes in the Berkeley Botanical Garden to monitor the microclimate in the garden's redwood grove [22]. However, the initial deployment only relays raw readings and does not currently make use of any of the aggregation techniques introduced in the TinyDB literature. While it may have approached the problem of improving energy efficiency from several angles it does have a number of inherent drawbacks the most significant being the lack of adaptability. First, the communication scheduling mentioned above is highly dependent on the depth of the network that is assumed to be fixed. This makes it unable to react to changes in the topology in the network on-the-fly that could easily happen if new nodes are added or certain nodes die. Second, the communication scheduling is also directly dependent on the epoch that is specified in every query injected into the network. With networks expected to span say hundreds or even thousands of nodes, it is unlikely that environmentalists using a particular network would only inject one query into the node at any one time. Imagine if the Internet was designed in a way such that only one person was allowed to use it at any instant! Thus methods need to be devised to enable multiple queries to run simultaneously in a sensor network.

Although TinyDB reduces the number of transmissions greatly by carrying out in-network aggregation for every long-running query, it keeps on transmitting data during the entire duration of the active query disregarding the temporal correlation in a sequence of sensor readings. Reference 23 takes advantage of this property and ensures that nodes only transmit data when there is a significant enough change between successive readings. In other words, sensors may refrain from transmitting data if the readings remain constant.

Another area related to the lack of adaptability affecting both COUGAR and TinyDB has to do with the generation of query execution plans. In both projects the systems assume a global view of the network when it comes to query optimization. Thus network metadata is periodically copied from every node within the network to the root node. This information is subsequently used to work out the best possible query optimization plan. Obviously, the cost of extracting network metadata from every node is highly prohibitive. Also query execution plans generated centrally may be outdated by the time they reach the designated nodes as conditions in a sensor network can be highly volatile, for example, the node delegated to carry out a certain task may have run out of power and died by the time instructions arrive from the root node. In this regard, it is necessary to investigate methods where query optimizations are carried out using only local information. While they may not be as optimal as plans generated based on global network metadata, it will result in significant saving in terms of the number of radio transmissions. Reference 24 looks into creating an adaptive and decentralized algorithm that places operators optimally within a sensor network. However, the preliminary simulation results are questionable since the overhead incurred during the *neighbor exploration* phase is not considered. Also there is no mention of how fast the algorithm responds to changes in networkdynamics.

33.3 Data-Centric Architecture

As we previously stated, the layered protocol stack description of the system architecture for a sensing node cannot cover all the aspects involved (such as crosslayer communication, dynamic update, etc.). In this section we address the problem of describing the system architecture in a more suited way and its implications in the application design.

33.3.1 Motivation

The sensor networks are dynamic from many points of view. Continuously changing behaviors can be noticed in several aspects of sensor networks, some of them being:

Sensing process. The natural environment is dynamic by all means (the basic purpose of sensor networks is to detect, measure, and alert the user of the changing of its parameters). The sensor modules themselves can become less accurate, need calibration or even break down.

Network topology. One of the features of the sensor networks is their continuously changing topology. There are a lot of factors contributing to this, such as: failures of nodes or the unreliable communication channel, mobility of the nodes, variations of the transmission ranges, clusters reconfiguration, addition/removal of sensor nodes, etc. Related to this aspect, the algorithms designed for sensor networks need to have two main characteristics: they need to be independent on the network topology and need to scale well with the network size.

Available services. Mobility of nodes, failures, or availability of certain kinds of nodes might trigger reconfigurations inside the sensor network. The functionality of nodes may depend on existing services at certain moments and when they are no longer available, the nodes will either reconfigure themselves or try to provide them themselves.

Network structure. New kinds of nodes may be added to the network. Their different and increased capabilities will bring changes to the regular way in which the network functions. Software modules might be improved or completely new software functionality might be implemented and deployed in the sensor nodes.

Most wireless sensor network architectures currently use a fixed layered structure for the protocol stack in each node. This approach has certain disadvantages for wireless sensor networks. Some of them are:

Dynamic environment. Sensor nodes address a dynamic environment where nodes have to reconfigure themselves to adapt to the changes. Since resources are very limited, reconfiguration is also needed in order to establish an efficient system (a totally new functionality might have to be used if energy levels drop under certain values). The network can adapt its functionality to a new situation, in order to lower the use of the scarce energy and memory resources, while maintaining the integrity of its operation.

Error control. It normally resides in all protocol layers so that for all layers the worst-case scenario is covered. For a wireless sensor network this redundancy might be too expensive. Adopting a central view on how error control is performed and crosslayer design will reduce the resources spent for error control.

Power control. It is traditionally done only at the physical layer, but since energy consumption in sensor nodes is a major design constraint, it is found in all layers (physical, data-link, network, transport, and application layers).

Protocol place in the sensor node architecture. An issue arises when trying to place certain layers in the protocol stack. Examples may include: timing and synchronization, localization, and calibration. These protocols might shift their place in the protocol stack as soon as their transient phase is over. The data produced by some of these algorithms might make a different protocol stack more suited for the sensor node (e.g., a localization algorithm for static sensor networks might enable a better routing algorithm that uses information about the location of the routed data destination).

Protocol availability. New protocols might become available after the network deployment. At certain moments, in specific conditions, some of the sensor nodes might use a different protocol stack that better suits their goal and the environment.

It is clear from these examples that dynamic reconfiguration of each protocol as well as dynamic reconfiguration of the active protocol stack is needed.

33.3.2 Architecture Description

The system we are trying to model is an event-driven system, meaning that it reacts and processes the incoming events and afterwards, in the absence of these stimuli, it spends its time in the sleep state (the software components running inside the sensor node are not allowed to perform blocking waiting).

Let us name a higher level of abstraction for the event class as *data*. Data may encapsulate the information provided by one or more events, have a unique name and contain additional information such as deadlines, identity of producer, etc. Data will be the means used by the internal mechanisms of the architecture to exchange information components.

In the following we will address any protocol or algorithm that can run inside a sensor node with the term *entity* (see Figure 33.8). An entity is a software component that will be triggered by the availability of one or more data types. While running, each entity is allowed to read available data types (but not wait for additional data types becoming available). As a result of the processing, each software component can produce one or more types of data (usually on their exit).

An entity is also characterized by some *functionality*, meaning the sort of operation it can produce on the input data. Based on their functionality, the entities can be classified as being part of a certain protocol layer as in the previous description. For one given functionality, several entities might exist inside a sensor node; to discern among them, one should take into consideration their *capabilities*. By capability

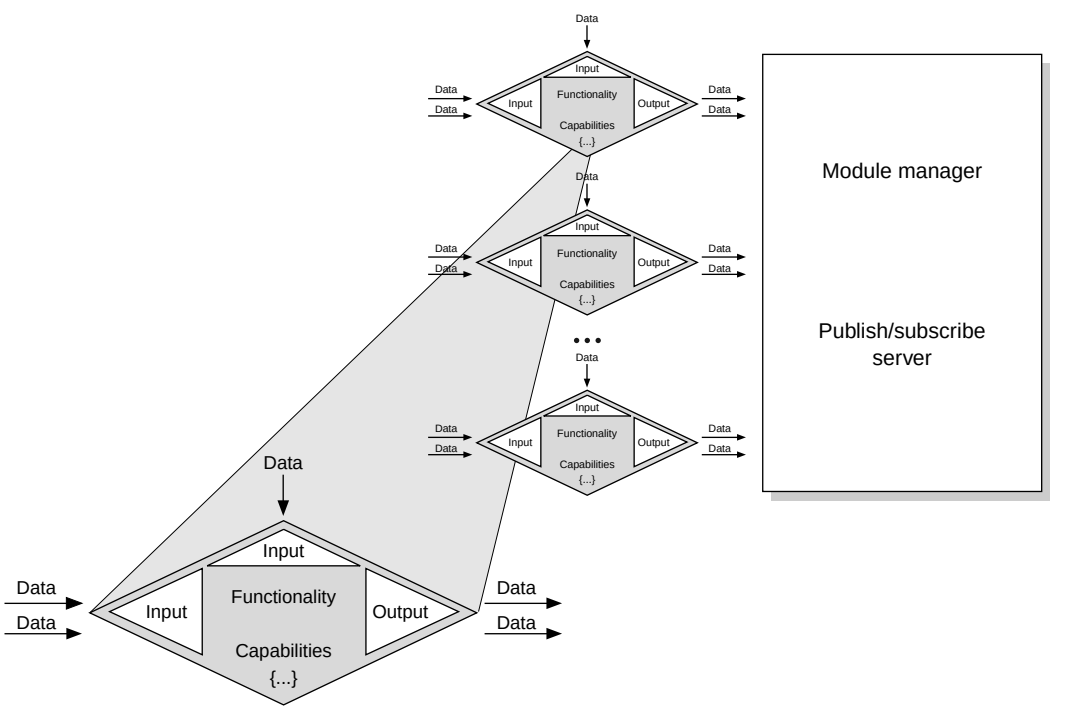

FIGURE 33.8 Entity description.

we understand high-level description containing the cost for a specific entity to perform its functionality (as energy, resources, time, etc.) and some characteristics indicating the estimated performance and quality of the algorithm.

In order for a set of components to work together, the way in which they have to be interconnected should be specified. The architectures existent up to this moment in the wireless sensor network field, assume a fixed way in which these components can be connected, which is defined at compile time (except for the architectures that e.g., allow execution of agents). To change the protocol stack in such an architecture, the user should download the whole compiled code into the sensor node (via the wireless interface) and then make use of some boot code to replace the old running code in it. In the proposed architecture we are allowing this interconnection to be changed at runtime, thus making online update of the code possible, the selection of a more suited entity to perform some functionality based on the changes in the environment, etc. (in one word allowing the architecture to become dynamically reconfigurable).

To make this mechanism work, a new entity needs to be implemented; let us call this the *data manager*. The data manager will monitor the different kinds of data being available and will coordinate the dataflow inside the sensor node. At the same time it will select the most fitting entities to perform the work and it will even be allowed to change the whole functionality of the sensor node based on the available entities and external environment (see Figure 33.9).

The implementation of these concepts can not make abstraction of the small amount of resources each sensor node has (as energy, memory, computation power, etc.). Going down from the abstraction level to the point where the device is actually working, a compulsory step is implementing the envisioned architecture in a particular operating system (in this case a better term maybe system software). A large range of operating systems exist for embedded systems in general [25, 26]. Scaled down versions with simple schedulers and limited functionality have been developed especially for wireless sensor networks [27].

Usually, the issues of system architecture and operating system are treated separately, both of them trying to be as general as possible and to cover all the possible application cases. A simplistic view of a running operating system is a scheduler that manages the available resources and coordinates the execution of a set of tasks. This operation is centralized from the point of view of the scheduler that is allowed to take all the decisions. Our architecture can also be regarded as a centralized system, with the data manager coordinating the dataflow of the other entities. To obtain the smallest overhead possible there should be a correlation between the function of the central nucleus from our architecture and the function of the scheduler from the operating system. This is why we propose a close relationship between the two concepts by extending the functionality of the scheduler with the functionality of the data manager. The main challenges that arise are keeping the size of the code low and the context-switching time.

33.3.2.1 Requirements

As we mentioned earlier, the general concept of *data* is used rather than the *event* one. For the decision based on data to work, there are some additional requirements to be met.

First of all, all the modules need to declare the name of the data that will trigger their action, the name of the data they will need to read during their action (this can generically incorporate all the shared resources in the system) and the name of the data they will produce. The scheduler needs all this information to take the decisions.

From the point of view of the operating system, a new component that takes care of all the data exchange needs to be implemented. This would in fact be an extended message passing mechanism, with the added feature of notifying the scheduler when new data types become available. The mapping of this module in the architecture is the constraint imposed to the protocols to send/receive data via, for example, a publish/subscribe mechanism to the central scheduler.

An efficient naming system for the entities and the data is needed. Downloading new entities to a sensor node involves issues similar to services discovery. Several entities with the same functionality but with

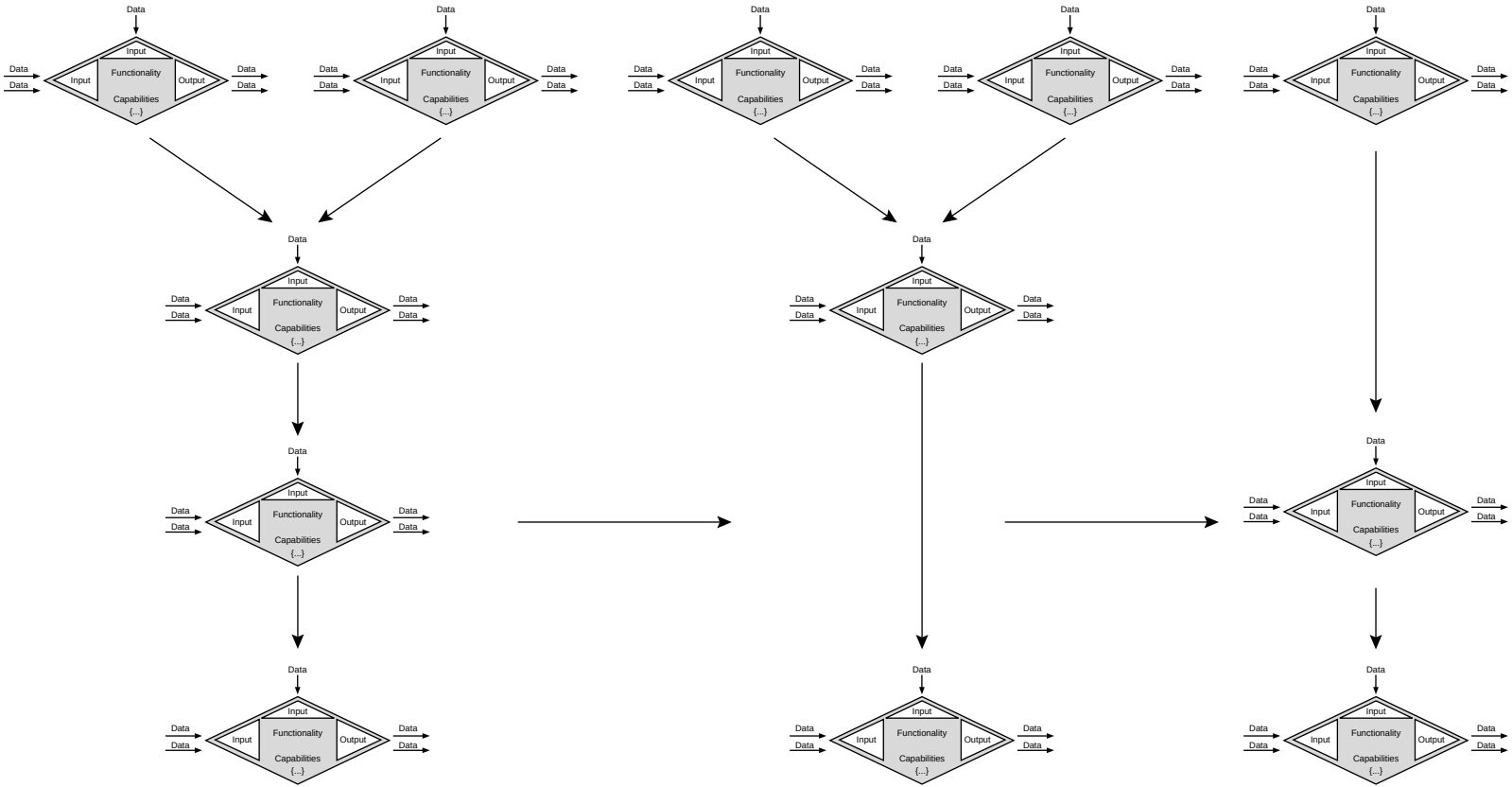

FIGURE 33.9 Architecture transitions.

different requirements and capabilities might coexist. The data-centric scheduler has to make the decision as to which one is the best.

33.3.2.2 Extension of the Architecture

The architecture presented earlier might be extended to groups of sensor nodes. Several data-centric schedulers together with a small, fixed number of protocols can communicate with each other and form a virtual backbone of the network.

Entities running inside sensor nodes can be activated using data types that become available at other sensor nodes (e.g., imagine one node using his neighbor routing entity because it needs the memory to process some other data).

Of course, this approach raises new challenges. A naming system for the functionalities and data types, reliability issues of the system (for factors, such as mobility, communication failures, node failures, security attacks) are just a few examples. Related work on these topics already exist (e.g., References 28 and 29).

33.4 Conclusion

In this chapter, we have outlined the characteristics of wireless sensor networks from an architectural point of view. As sensor networks are designed for specific applications, there is no precise architecture to fit them all but rather a common set of characteristics that can be taken as a starting point.

The combination of the data-centric features of sensor networks and the need to have a dynamic reconfigurable structure has led to a new architecture that provides enhanced capabilities than the existing ones. The new architecture characteristics and implementation issues have been discussed, laying the foundations for future work.

This area of research is currently in its infancy and major steps are required in the fields of communication protocols, data processing, and application support to make the vision of Mark Weiser a reality.

References

[1] EYES. Eyes European project, http://eyes.eu.org.

[2] Chu, P., Lo, N.R., Berg, E., and Pister, K.S.J. Optical communication using micro corner cuber reflectors. In *Proceedings of the MEMS97*. IEEE, Nagoya, Japan, 1997, pp. 350–355.

[3] Havinga, P. et al. Eyes deliverable 1.1 — system architecture specification.

[4] SmartDust. http://robotics.eecs.berkeley.edu/ pister/SmartDust.

[5] Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. A survey on sensor networks. *IEEE Communication Magazine*, 40(8), 102–114, 2002.

[6] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. Directed diffusion for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 11(1), 2–16, 2003.

[7] Pottie, G.J. and Kaiser, W.J. Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, 43(5), 51–58, 2000.

[8] Ganesan, D., Cerpa, A., Ye, W., Yu, Y., Zhao, J., and Estrin, D. Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing, Special Issue on Frontiers in Distributed Sensor Networks*, 64(7), 799–814, 2004.

[9] Estrin, D., Govindan, R., Heidemann, J.S., and Kumar, S. Next century challenges: scalable coordination in sensor networks. *Mobile Computing and Networking*. IEEE, Seattle, Washington, USA, 1999, pp. 263–270.

[10] Bonnet, P., Gehrke, J., and Seshadri, P. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*. Springer-Verlag, Heidelberg, 2001, pp. 3–14.
[11] Madden, S., Szewczyk, R., Franklin, M., and Culler, D. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing and Systems Applications*. IEEE, 2002.
[12] Postel, J. Internet protocol, rfc 791, 1981.
[13] Estrin, D., Girod, L., Pottie, G., and Srivastava, M. Instrumenting the world with wireless sensor networks. In *Proceedings of the International Conference on Accoustics, Speech and Signal Processing*. IEEE, Salt Lake City, Utah, 2001.
[14] Heidemann, J.S., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*. ACM, 2001, pp. 146–159.
[15] Pottie, G.J. and Kaiser, W.J. Wireless integrated network sensors. *Communications of the ACM*, 43(5), 51–58, 2000.
[16] Bonnet, P. and Seshadri, P. Device database systems. In *Proceedings of the International Conference on Data Engineering*. IEEE, San Diego, CA, 2000.
[17] Bonnet, P., Gehrke, J., and Seshadri, P. Querying the physical world. *IEEE Personal Communications*, 7, 10–15, 2000.
[18] Madden, S., Franklin, M.J., Hellerstein, J.M., and Hong, W. The design of an acquisitional query processor for sensor networks. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM Press, San Diego, CA, 2003, pp. 491–502.
[19] Madden, S. The design and evaluation of a query processing architecture for sensor networks. PhD thesis, University of California, Berkeley, 2003.
[20] Yao, Y. and Gehrke, J. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3), 2002.
[21] Yao, Y. and Gehrke, J. Query processing for sensor networks. In *Proceedings of the Conference on Innovative Data Systems Research*. Asilomar, CA, 2003.
[22] Gehrke, J. and Madden, S. Query processing in sensor networks. *Pervasive Computing*. IEEE, 2004, pp. 46–55.
[23] Beaver, J., Sharaf, M.A., Labrinidis, A., and Chrysanthis, P.K. Power aware in-network query processing for sensor data. In *Proceedings of the Second Hellenic Data Management Symposium*. Athens, Greece, 2003.
[24] Bonfils, B.J. and Bonnet, P. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the Second International Workshop on Information Processing in Sensor Networks (IPSN), Vol. 2634 of Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 2003, pp. 47–62.
[25] VxWorks. Wind river, http://www.windriver.com.
[26] Salvo. Pumpkin incorporated, http://www.pumpkininc.com.
[27] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., and Pister, K.S.J. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.
[28] Verissimo, P. and Casimiro, A. Event-driven support of real-time sentient objects. In *Proceedings of the Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*. IEEE, Guadalajara, Mexico, 2003.
[29] Cheong, E., Liebman, J., Liu, J., and Zhao, F. TinyGALS: a programming model for event-driven embedded systems. In *Proceedings of the 2003 ACM Symposium on Applied Computing*. ACM Press, Melbourne, Florida, 2003, pp. 698–704.