*International Journal of Distributed Sensor Networks*, 5: 262–282, 2009

ISSN: 1550-1329 print / 1550-1477 online
DOI: 10.1080/15501320701343869

Taylor & Francis
Taylor & Francis Group

# The Design of an Environment for Monitoring and Controlling Remote Sensor Networks

IOANNIS CHATZIGIANNAKIS, GEORGIOS MYLONAS, and SOTIRIS NIKOLETSEAS

Computer Technology Institute (CTI) and Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

*In this work we present the design of jWebDust, a software environment for monitoring and controlling sensor networks via a web interface. Our software architecture provides a range of services that allow to create customized applications with minimum implementation effort that are easy to administrate. We present its open architecture, the most important design decisions, and discuss its distinct features and functionalities. jWebDust will allow heterogeneous components to operate in the same sensor network, and the integrated management and control of multiple such networks by defining web-based mechanisms to visualize the network state, the results of queries, and a means to inject queries in the network.*

**Keywords** Monitoring; Management; Multiple Sensor Networks; Heterogeneous; Virtual Sensor Networks; Tinyos; Java

## 1. Introduction

Wireless sensor networks are very large collections of small-sized, low-power, low-cost sensor devices that collect and disseminate quite detailed information about the physical environment. Large numbers of sensors can be deployed in areas of interest and use self-organization and collaborative methods to form a sensor network. The flexibility, fault tolerance, high sensing fidelity, low cost, and rapid deployment characteristics of sensor networks help to create many new and exciting application areas.

This wide range of applications is based on the use of various sensor types (i.e., thermal, acoustic, magnetic, etc.) in order to monitor a wide variety of conditions (e.g., temperature, object presence and movement, humidity, pressure, noise levels, etc.) and report them to a (fixed or mobile) control center. Thus, sensor networks can be used for important applications, including

(a) military applications (such as forces and equipment monitoring, battlefield surveillance, targeting, nuclear, biological and chemical attack detection),
(b) environmental applications (such as fire detection, flood detection, precision agriculture),

This work has been partially supported by the IST Programme of the European Union under contract number IST-2005-15964 (AEOLUS). Also, by the Programme PENED under contract number 03ED568, co-funded 75% by European Union - European Social Fund (ESF), 25% by Greek Government - Ministry of Development - General Secretariat of Research and Technology (GSRT), and by the Private Sector, under Measure 8.3 of O.P. Competitiveness - 3rd Community Support Framework (CSF).

Address correspondence to Ioannis Chatzigiannakis, CTI, and Department of Computer Engineering and Informatics, University of Patras, Patras, 26500, Greece. E-mail: ichatz@cti.gr

(c) health applications (like telemonitoring of human physiological data) and
(d) home applications (e.g., smart environments).

A possible categorization of the applications for wireless sensor networks is based on the notification strategy of the authorities, i.e., the way that the authorities are updated on the monitoring state. For example, in a museum, it is important to report only when emergency situations arise, such as an incendiary fire. On the other hand, in habitat monitoring, for instance, continuous monitoring of the physical environment is required so that scientists can gather information over a long period of time. Therefore, depending on the actual application, the following services are required [6]:

(i) Periodic Sensing (the sensor devices constantly monitor the physical environment and continuously report their sensors' measurements to a control center),
(ii) Event driven (to reduce energy consumption, sensor devices monitor silently the environment and communicate to report when certain events are realized) and
(iii) Query based (sensor devices respond to queries made by a supervising control center).

In light of these categories of applications and services, we present here the design of jWebDust, a software environment that will allow the implementation of customized applications for wireless sensor networks, as it:

(i) provides a wide range of services,
(ii) minimizes the overall implementation effort,
(iii) considerably reduces the needs for network administration, and
(iv) makes data monitoring and network management available via the Web.

In this article, we present the most important design decisions in jWebDust, and discuss its distinct features and functionalities. By implementing an application for sensor networks we mean that the end user can select a subset from the available features through the web interface of the system, that best suit his/her needs, or modify and extend the system by adding new components. In this sense, jWebDust is able to deal with several kinds of applications (size, functionality, etc.).

Generally, jWebDust differentiates the system into two main groups: the networked sensor devices (from now on referred to as motes) that operate using TinyOS, and the rest of the network (e.g., control centers, database server, etc.) that is capable of executing Java code. Both system groups use an open architecture implementing a component-based architecture. The standardized component interface and the exchange of data over broadly used protocols provide increased portability. This implies that the system can be used over different machine architectures as well as OS and server technologies.

We define and implement a simple Mote Discovery Service, which plays a crucial role in the execution of applications for wireless sensor networks as it significantly reduces the overall network administration. More specifically, this service keeps track of the motes that participate in the wireless sensor network and their technical characteristics (e.g., type of sensors attached to each device, available power, etc.). During the setup phase of the network, the motes report to the control center of the network and get registered in jWebDust's database without further human interaction. In this sense, the time required by the administrator to register the devices that make up the network and their hardware characteristics is greatly reduced, especially in the case where the network is comprised of heterogeneous devices [8, 18, 30]. Furthermore, in some cases, it is possible to deploy additional motes in order to extend the lifetime of the network and/or increase the area of

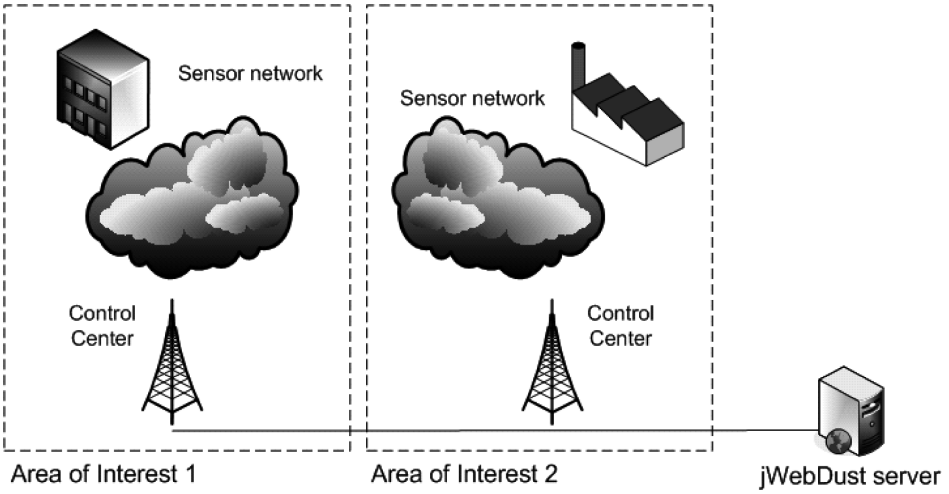


**Figure 1**. Multiple wireless sensor networks form a single, unified, virtual sensor network.

surveillance while the network is in operation [8]. In such cases, the discovery service improves the scalability of the applications that are based on jWebDust, since the need for network administration remains unaffected as the size of the sensor networks increases.

A distinct feature of jWebDust is its ability to manage multiple wireless sensor networks, each with a different control center, under a common installation (e.g., see Fig. 1). This is done by introducing the notion of a virtual sensor network that somehow, hides the actual network topology, and allows the user to control the motes as if they were deployed under a single, unified, sensor network. This abstraction significantly reduces the overhead of administering multiple networks. Furthermore, the idea of a unified, virtual sensor network allows the integration of totally heterogeneous sensor networks, i.e. not only regarding different kinds of sensors attached to the motes of the network, but also different kinds of CPU architectures, etc., [12, 30].

In jWebDust, the information collected by the control centers of all operating sensor networks is stored in a single, central, relational database. Based on this database server, jWebDust provides a web-based, user-friendly interface that targets both scientific as well as other less technically-trained personnel. The user interface is customizable and also allows the designer to present the information in different ways and offers extendable statistics based on the needs of the application. Furthermore, the component-based architecture that is used in designing jWebDust offers high adaptability. This software environment, which is currently under development, features an open interface through which added functionality can be implemented and integrated at later stages, probably carried out by the sensor network administrator. Since jWebDust is still under development, we did not include a comparative evaluation of our software in this work.

Finally, the design presented here is an extended version of the work presented in [9] and [10].

## 2. Related Work

Although wireless sensor networks have attracted a lot of attention from researchers at all levels of the system hierarchy (from the physical layer and communication protocols up to the

application layer), software environments that provide the necessary tools and operations to allow the implementation of a wide range of applications are relatively few. Examples of such software include TinyDB [38], Mote-VIEW [24], Scatterviewer [33], and MSR Sense [25].

TinyDB is an application that allows multiple concurrent queries, event-based queries, and time synchronization through an extensible framework that supports adding new sensor types and event types. The central idea of TinyDB is to provide an SQL-like interface to the programmer that makes the wireless sensor network look like an RDBMS. A tree-based routing scheme is used for multihop communication inside the network while data is stored in the motes' memory. TinyDB also produces a visualization of the network's topology. In addition, the TASK [7] (Tiny Application Sensor Kit) is built on top of TinyDB in order to further simplify application deployment and development. The kit includes a server that acts as a proxy for the sensor network on the internet, a relational database where readings from the sensor network are stored, and a frontend that helps to choose, record, and visualize motes' metadata. In contrast to the database schema used in TASK, our approach is more detailed and can be easily extended to the final application's functionality needs (e.g., adding new sensor types, mote types, etc.).

Mote-VIEW [24] is an application that provides tools to the user to visualize results from a sensor network, combined with a data logger that runs on the sensor network gateway. The data logger constantly listens to readings arriving from the network through a control center attached to the gateway and stores them in a relational database. The motes in the sensor network continuously poll their sensors for readings at a sampling rate specified by the network administrator and communicate them to the gateway using a simple multi-hop protocol. The user can check readings from the motes' sensors on the fly, see a visualization of the network's topology, produce graphs from selected motes' readings, check the status of the motes. Mote-VIEW is part of the MoteWorks software suite provided by Crossbow.

ScatterViewer [33] is an application designed for the ScatterWeb sensor platform, with characteristics similar to MoteView, plus the useful capability of replacing the motes' firmware over the air and the option of running on an embedded platform. Another application environment similar in many ways to MoteView and ScatterViewer is Monsense [28].

Microsoft's MSR Networked Embedded Sensing Toolkit (MSR Sense) [25] is a software suite (currently under development) that helps users to collect, process, archive, and visualize data from a sensor network. These data come from applications running on the motes, that are independent of MSR Sense. It has useful features like an extension to Excel 2003 (Senscel) to import, visualize, and process sensor data.

A subject considered in this work, buffering and disconnected operation of sensor network control centers, is analyzed and used in Hourglass [36]. The need for such features arises from the fact that sensor networks often connect to the Internet through wireless links, that can suffer from occasional or periodical downtimes. We need to ensure that no commands to and results from the sensor network are lost.

Dynamic resource discovery, relative to the discovery protocol we use, is discussed and used in [37], Global Sensor Networks [1, 22] and Hourglass. By the term dynamic resource discovery we refer to the capability of the (heterogeneous) sensor network to know what capabilities (services) the motes in the network offer. We use a custom encoding protocol versus an XML encoded protocol for the sake of simplicity.

[1, 22, 36] use peer-to-peer concepts to enable inter-sensor network communication. Other environments for managing wireless sensor networks that, to some extent, use peer-to-peer concepts, are CarTel [16] and MetroSense [13].

There are a number of examples of sensor networks interfacing through the Web. The sensor deployment at Great Duck Island [15] features a Web interface, through which users can select from a set of sensors and visualize readings from the motes of the network for specified periods of time and intervals. The interface of the deployment at the James Reserve [21] is a similar example, through which live pictures from the sensor field can also be viewed. The MoteLab [23] at Harvard university is a sensor node deployment consisting of a large number of wired sensor nodes. Users can reprogram nodes and view sensor reading using a web interface. Also, examples of systems that make use of web-based map services (like Google Maps) to administer a wireless sensor network (though running on different levels) are SensorScope [34] and SenseWeb [32].

The rest of the article is organized as follows. In Section 3, we present the high-level design of jWebDust. In Section 4, we discuss about the communication among motes in the sensor network. The protocol through which queries are distributed over the network and the dissemination of sensor readings that match these queries back to the control center is presented in Section 5. Section 6 presents the services offered by the wireless sensor networks, while in Section 7 we refer to the ways a user interested in using a wireless sensor network can utilize jWebDust. We conclude and discuss possible future directions in Section 8.

## 3. The Architecture of jWebDust

jWebDust is designed on a component-based architecture with several diverse design goals in mind. This version of the architecture design is focused on specifying a set of components by grouping them considering certain aspects of desired functionality, with emphasis on autonomy, reliability, and availability. At a high-level, the components that make up jWebDust are organized using the and availability. At a high-level, the components that make up jWebDust are organized using the N-tier application model. We distinguish the following five tiers:

(i) the Sensor Tier that consists of one or more wireless sensor networks deployed to areas of interest,
(ii) the Control Tier that corresponds to the control centers where the wireless sensor networks report the realization of events,
(iii) the Data Tier responsible for storing the information extracted from the wireless sensor network(s),
(iv) the Middle Tier that is responsible for processing the data to generate statistics and other meaningful information and
(v) the Presentation Tier that interfaces the information with the final user in an easy way based on the capabilities of the user's machine.

The five tiers that make up jWebDust are shown in Fig. 2.

The component-based architecture enables the jWebDust system to gain control over all critical resources, required by the implemented functionality (connection to database, communication with a server, code execution). The autonomy of the components from the rest of the operating environment makes the system independent from the machine architecture, allowing it to be executed over favored machine architectures using any best-of-breed server technologies.

### *3.1. The Sensor Tier*

Naturally, the sensor tier is the foundation of any application for sensor networks. The motes are usually scattered in the area of interest and form a sensor network as shown in

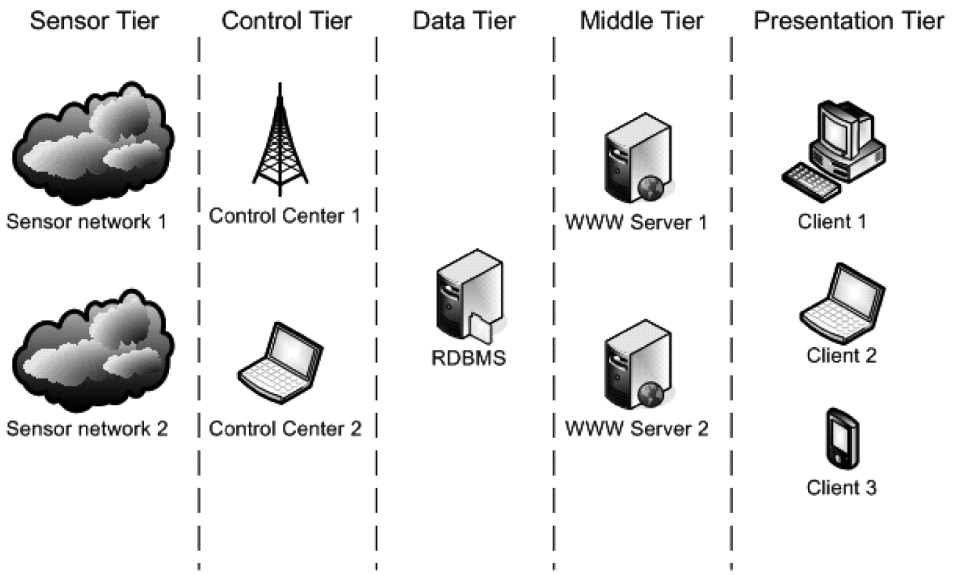


**Figure 2.** N-tier Architecture.

Fig. 4. Each of these scattered motes has the capability to collect data and route data back to the control center and the end users. Data are routed to the control center by a multi-hop architecture as shown in Fig. 4. Then, the control center communicates with the other tiers of the system possibly via internet.

Typically, as stated in the introduction, motes are wireless sensor nodes, each with its own sensors. Motes poll these sensors to collect readings according to a sensor query protocol (Sec. 5).

We are currently investigating also the possibility of using a number of wired sensors, as opposed to only wireless ones, i.e., on e mote (With a wireless transceiver) could connect to a number of wired sensors. These sensors are connected through a common wire/bus and

| | | |
|---|---|---|
| *Presentation Tier* | User interface, servlets & portlets | |
| *Middle Tier* | Provide access to Data Tier<br>Business logic | |
| *Data Tier* | Database System | |
| *Control Tier* | General Control Centers | Mini Control Centers |
| | Multiple control centers<br>Buffering & Disconnected operation<br>Communication with Sensor Tier | Network Monitoring<br>Information to sensors |
| *Sensor Tier* | TinyOS Components | Query Subsystem<br>Discovery Service<br>Monitoring Service<br>Time Synchronization |
| | Multihop Routing<br>Sensorboard Libraries<br>MAC layer, etc. | |

**Figure 3.** Outline of jWebDust's Tiers.

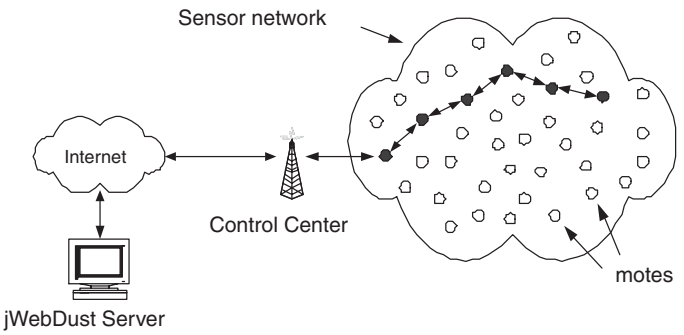


**Figure 4.** Motes forming a sensor network.

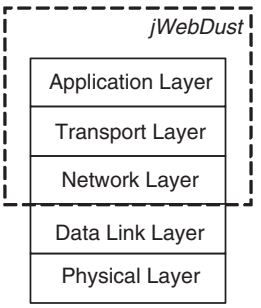


**Figure 5.** Sensor networks protocol stack.

communicates using a protocol like IEEE 1451.4. The mote communicates periodically with the sensors to collect their readings.

The protocol stack used by all the motes (see Fig. 5) is based on the TinyOS operating system and consists of the application layer, transport layer, network layer, data link layer and the physical layer. The jWebDust firmware, executed by the motes, implements the network layer and the transport layer using a tree-based routing scheme, which is described in greater details in Section 4. Additionally, the implementation of the application layer involves a protocol for distributing queries in the sensor network and disseminating the data matching these queries back to the control center; this protocol is defined more formally in Section 5.

### *3.2. The Control Tier*

The Control Tier consists of the control centers of each sensor network. We distinguish between two types of control centers:

(i) general control centers, and
(ii) mini control centers.

General control centers are responsible for the gathering of all the readings coming from the sensor networks and the forwarding of the queries from the data tier to the motes; in other words, they act as gateways between the sensor tier and the data tier. This is achieved by periodically polling the data tier for any new queries. When a new query record

is generated in the data tier, the general control center will forward it to the sensor network. Similarly, the control centers will remain idle, waiting to receive sensor readings. All received sensor readings are stored directly to the data tier.

Mini control centers are used by the network adiministrator to check on the status of the nodes inside the sensor network, e.g., check the energy levels of motes that have no connectivity to the rest of the network. Also, mini control centers can be used to provide certain information to the motes, e.g., due to the lack of expensive GPS modules, the administrator can use a mini control center to set geographical coordinates on the motes. Data mules [35] is a similar approach, but it proposes that mobile control centers are used to propagate data inside the sensor network, to make up for poor connectivity. In our design, mini control centers are used by the administrators while moving inside the deployment area. They are not used to propagate data inside the sensor network, and unlike general control centers they do not communicate with the upper tiers of the system.

On the hardware level, a typical general control center consists of one mote connected to a desktop PC or laptop along with a network connection to the database server. The mote attached to the control tier is needed in order to communicate with the sensor network. Alternatively, an embedded platform can be used (e.g., Stargate [24]). A mini control center can consist of a PDA, possibly with a GPS, with a mote attached to it in order to communicate with the sensor network.

*3.2.1. Buffering and Disconnected Operation.* One important feature of the control tier is that control centers can operate even when there is no connection to the data tier for sustained period of time. The importance of this feature is outlined by the fact that control centers themselves may have a wireless connection to the data tier and uninterrupted communication is not guaranteed: e.g., consider the case where a wireless sensor network is used for a precision agriculture application in a vineyard. Due to the nature of the application, the control center must connect to the database server through a wireless connection. This WiFi or GPRS link however, can be down from time to time, or too expensive to operate continuously.

During such a disconnected period, any new queries made by the user will not be forwarded to the sensor network, since the control center cannot be contacted. At the same time sensor readings received will not be sent to the data tier but will be stored in a temporary buffer, possibly of limited capacity. As soon as the control center establishes a connection with the data tier, all locally stored readings are forwarded to the data tier and the new queries are forwarded to the network.

*3.2.2. Multiple Sensor Networks.* Another feature of jWebDust is the support of multiple sensor networks in a way that the user perceives them as a single virtual sensor network. For each sensor network a unique ID is given to its respective control center. This sensor network ID helps to distinguish one mote from another, when they have the same mote ID but belong to different sensor networks, thus making it possible to manage different networks in a unified way. For example, two sensor networks, each one with its own control center, located in two different buildings can be administered through the same interface.

### *3.3. The Data Tier*

The components at the data tier are based on the relational database system, and the functionality and methods provided are related to the services required by the middle tier

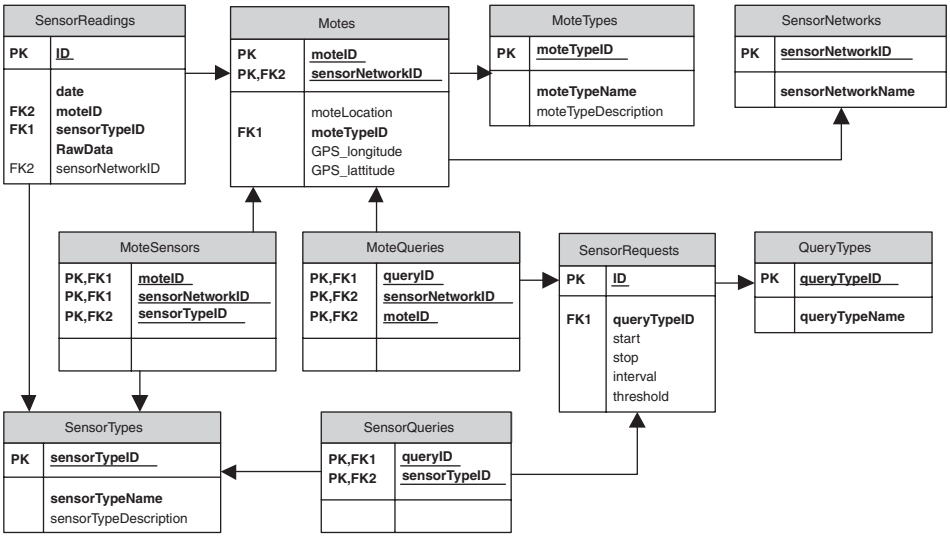


**Figure 6**. jWebDust's database schema.

and the control tier. At this stage of the implementation of jWebDust, the data tier is made up from components that use the widely available ANSI SQL or SQL89.

Our database schema can be seen in UML form in Fig. 6 and consists of ten tables that can be organized in three categories:

(i) Mote related tables that are used to store information regarding the technical characteristics of the motes,
(ii) Query related tables that keep track of active and historic queries and
(iii) Sensor readings table that stores the information received from the sensor network.

*3.3.1. Mote Related Tables.* The information related to the hardware characteristics of the motes that comprise the wireless sensor networks are organized in this category. Our goal is to support heterogeneous sensor networks, i.e., networks that consist of different kinds of motes (e.g., Telos, MicaZ motes, etc.), which in turn may have different kinds of sensors attached to them (e.g., light, pressure, humidity, temperature, etc.). Of course, motes inside the same sensor network must be able to communicate with each other (i.e., use the same communication protocol).

In order to achieve this goal we use five tables in our system's database schema. The MoteTypes and SensorTypes tables represent the cpu/radio types that are expected to be deployed in the network along with the monitors/sensors attached to them. These tables are maintained by the network administrator and keep track of the technical specifications of the hardware. The Mote table contains records for all the motes that participate in the sensor network. The Mote table has a one-to-one relation with the MoteTypes table and since each mote may support multiple sensors, it has a many-to-many relation with the SensorTypes tables; to implement such a relation we use the MoteSensors table. Finally, the SensorNetworks table is used to record the multiple sensor networks operating that are managed by this installation of jWebDust. This table has a one-to-many relation with the Motes table, since each mote belongs to only one sensor network.

*3.3.2. Query Related Tables.* This group of tables is used to hold information regarding the queries made on the network, that may address multiple sensors and/or multiple sensor types. As jWebDust offers different types of queries (for more information see Section 5) that can be made by the user to the sensor network, the QueryTypes table is used to keep track of all the available options.

The SensorRequests table represents all queries made to the sensor network and each record of the table contains the parameters that describe the query, e.g., start time, stop time, and reading interval (in the case of periodic sensing), sensor reading threshold (in the case of event-driven queries). The queryTypeID field is the type of a specific query, according to the QueryTypes table.

Each query may refer to multiple motes and multiple sensors. In this sense there is a one-to-many relation with the Mote and SensorTypes table; to implement the many-to-many relations we use the SensorQueries and MoteQueries tables respectively.

*3.3.3. Sensor Readings Table.* All the information received from the sensor networks that match a specific query are stored in the SensorReadings table. Each record represents a reading coming from a specific mote in the network concerning a single sensor. Therefore, the table is related with the SensorTypes table (to keep track of the sensor type), the Mote table (for the mote that reported the value), and the SensorRequests table (to be related with the actual query).

### *3.4. The Middle Tier*

The next category/tier of components is the Middle tier; these components make up the jWebDust logic. Typically, the middle tier is responsible for delivering structures and data to the presentation tier. In order to process the requests of the presentation tier, communication with the Data tier is required; successfully or not, the responsibilities of the middle tier are completed when the results are returned to the presentation tier.

The middle tier is made up from several components; these components can be considered as distinct applications that run on a server (also referred to as servlets). They also have autotelic functionality and are executed independently of one another, in order to process certain data structures and produce some specified output. To that extent, the components contribute to a simpler distribution of workload. Additionally the development of the presentation tier is more efficient if the available methods of the middle tier can operate simultaneously and independently.

Each component is written based on the data structures available and the operations required by each entity. Generally, these components are organized in three groups:

(i) model,
(ii) manager, and
(iii) business logic.

The components in the model group provide a mapping to the tables in the system database. The components in the manager provide functions to access and modify the contents of the data tier using the mapping from the logic group components. For example, the management of queries is handled by a single component and the methods implemented perform action such as createQuery, deleteQuery, updateQuery, etc. All these methods accept parameters given by the presentation tier, e.g., the logic of the creation of a new query (i.e., check if a similar query already exists or if the mote provides the sensors specified in the query) is standardized through a single component, the RequestsManager. The query

component just described, along with other similar components that are part of the jWebDust system, provide a specified interface to the other components; thus if modifications to system logic are carried out, the implementation of this interface can be modified. As the interface to these methods remains unaffected, components that use them will not notice any change and thus will not require any further change. The business logic group components implements other functions such as generating reports in various reports, checking the status of the system, and creating alerts, etc.

The benefits of this component-based architecture are not limited only to the above properties. In a monolithic approach, modifying a module would result in a recompilation and redeployment of the whole application, in our case, we only need to modify certain components and redeploy only these components.

### *3.5. The Presentation Tier*

The presentation layer is basically the user interface; it is the layer most available to the end users, responsible for the collection of input and presentation of the information collected from the sensor network. The components that make up the presentation tier are based on different technologies, as shown in Fig. 2. The terms thin client and rich client are used to describe the capabilities of the presentation tier given the available resources. We characterize as rich clients the components of the presentation tier that are of rich functionality.

The jWebDust presentation layer is based on rich clients, and this seems to be the reasonable decision as an environment with high ease of use and high speed of operation is always desired. However, the addition of thinner clients will ensure the interoperability of the system through the various available machine architectures. The end users are allowed to choose among the available client solution in order to maximize the available resources. The open architecture of jWebDust system allows new components to be introduced at later stages that will cope with these issues.

## 4. Sensor Network Communication Protocol

Communication in wireless sensor networks presents many distinctive characteristics, which render the conventional and most ad-hoc routing approaches inadequate for use in these networks. Many routing schemes have been proposed specifically for sensor networks; however, only a few of them have been implemented in TinyOS. Routing protocols for TinyOS can be divided into two categories, based on the number of possible routing destinations.

The first category includes most of the implemented protocols, which use a many-to-one routing approach, i.e., every message created by motes inside the sensor network, apart from communication for routing purposes, has a single destination, the control center. In this category falls the ''standard'' multi-hop tree-based routing protocol included in the TinyOS distribution, called MultiHopRouter, and a stripped-down implementation of the DSDV protocol [26]. More recently, the EAD [3] protocol builds a special rooted broadcast tree with many leaves that is used to facilitate efficient and energy-aware data-centric routing.

The second category includes protocols that allow many-to-many routing, although some restrictions exist on the total number of possible destinations. A stripped-down implementation of AODV protocol [27] (called TinyAODV), falls into this category,

along with Flood, a simple flooding protocol, and also a simple implementation of Directed Diffusion protocol [17] (called TinyDiffusion).

The choice of the routing protocol for a sensor network depends on the type of application used and has a great impact on the system's overall performance. For a survey of efficient data propagation protocols, see [5]. For a focused study of energy efficiency aspects, see [4].

We plan to use in jWebDust a communication protocol based on a simple BFS (Breadth First Search) scheme, which constructs a tree, with some extensions that aim toward reducing the energy dissipation of the motes and assuring the overall connectivity of the network. More specifically, an extension that uses a mechanism for varying the transmission range [2]. The implementation of such a mechanism is based on the ability of TinyOS to adjust the transmission range of the motes, e.g., in the MICA platform via the Potentiometer component. All information from the sensor network is routed toward the control center, which is the root of the routing tree.

The main idea is to periodically check whether a satisfactory number of nearby motes is active. This is done by periodically broadcasting ''hello'' messages from the control center and flooding the network. If the motes maintain network connectivity, this message will reach all the motes of the network, at a certain time point. There is a software component in each mote responsible for maintaining a table holding the id's of all mote's neighbors. By maintaining a flag in each mote, which should be up if any neighbors have sent ''hello'' messages over some period of time and down if no such message has been received, motes can decide on whether to modify their transmission range in order to overcome network connectivity issues or not. The concept of a varying transmission range protocol (VTRP) and a study of ways of modifying the transmission range are described in [2].

Mini control centers, on the other hand, do not need a multihop routing protocol to communicate with their nearby nodes, since they do not propagate data from other nodes. They use broadcasting to find their neighbors and once a sensor node is in the neighbor list of the control center it can be contacted directly. Communication between individual motes and mini control centers is done using a simple query and command protocol.

## 5. Sensor Query and Data Dissemination Protocol

In this section we discuss the protocol for sending queries to the motes of the network in more detail. First, we categorize all possible queries using two criteria;

(i) the motes they are targeted to and
(ii) the way information regarding the queries are reported to the control center.

The first category includes mote-specific and attribute-based queries, while the second category includes periodic sensing, event-driven, and query-based queries. These two categories are overlapping and the actual sensor queries are combinations of these categories. jWebDust will give the users the capability to send to the sensor network all these types of queries.

In the case of a Mote-based query, we are interested in targeting specific motes by using their IDs. In this case, the protocol will send one packet per each mote included in the query. An example of such a query is the following: ''Give me humidity readings from mote 5.'' On the other hand, attribute-based queries are not targeted to specific sensors but instead to the whole sensor network. In this case, only those motes of the network that match the specific attribute will respond to the query. An example of an attribute-based query is ''give me the temperature readings from all sensors that have light readings over 200.'' Therefore, in order to reach all motes they are flooded inside the sensor network.

Periodic-update queries pose another time constraint along with the start and stop time constraints, regarding the time of reporting to the sink. They specify an interval time period between successive query reports. An example of a periodic update query is ''give me the temperature readings every 125 msec.'' Event-driven queries do not pose such specific time constraints, but instead use the notion of events in order to define the time to report back to the sink. An example of such an event is ''when temperature reaches 100°C.''

Examples of possible queries to a sensor network include ''give me the temperature and humidity readings from motes where light reading is over 200 starting from 10:15 till 13:30'' and ''give me the light readings from mote 4 when temperature reaches 30°C.'' Thus, we have four types of possible queries, attribute-based periodic update queries, attribute-based event-driven queries, mote-specific periodic update queries, and mote-specific event-driven queries.

The format of the messages exchanged can be seen in Fig. 7a-d. The standard packet size in TinyOS is 36 bytes, leaving 29 bytes when using standard single hop communication and in most cases less when using a multihop protocol. The space left is sufficient for sending all types of queries however. Timestamps occupy 5 bytes, while mote IDs consist of 2 bytes and sensor type IDs of 1 byte. Events and attribute constraints can be expressed in the same way, occupying 4 bytes, 1 byte for the type of sensor ID, 1 byte for the relation used (equal, greater than, etc.), and 2 bytes for the value used in the constraint. There is also another type of message used to stop specific queries, the cancel-query message, which consists only of the IDs of the queries that are to be canceled. The simple format of this

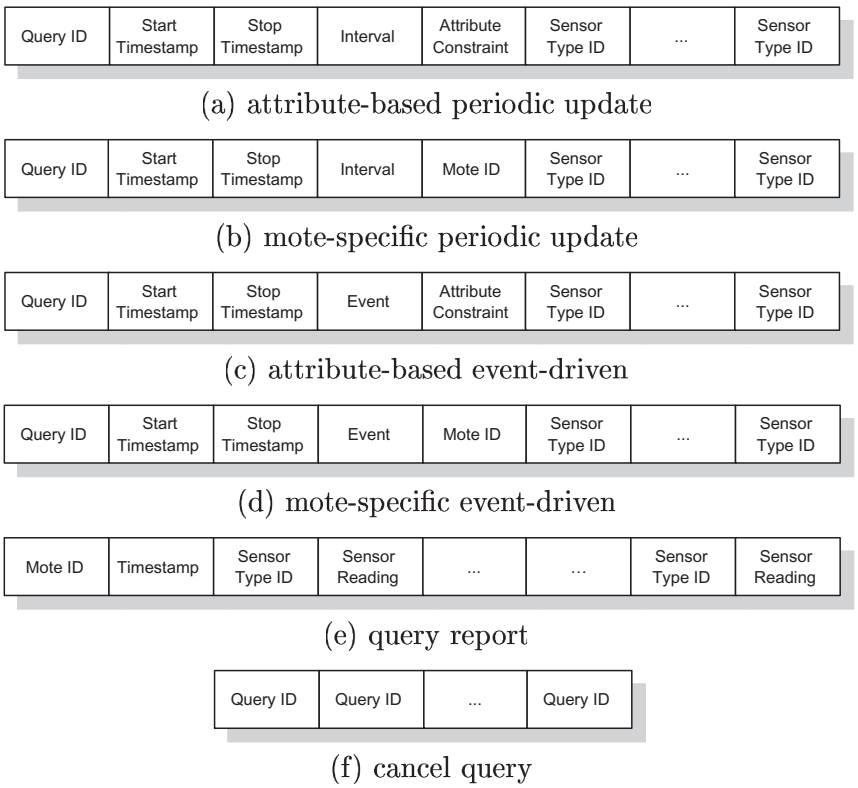


**Figure 7**. The format of the messages used for query and data dissemination.

message is shown in Fig. 7f. We note that all packets are transmitted without requiring an acknowledgement, in order to save energy.

The format of the report sent back to the sink by the motes can be seen in Fig. 7e. Because of the fact that a query can poll many sensors in the same mote and one packet might not be sufficient for sending the readings from all the requested sensors, more than one packet can be sent by motes answering a query back to the sink. The sequential messages will contain the readings that did not fit in the previous ones.

## 6. Sensor Network Services

### *6.1. Mote Discovery Service*

One of jWebDust's features is that every mote is able to register itself and its distinct features to the system, thus giving a clear view of the network and removing the need for jWebDust to initiate a mote discovery process periodically. Our goal is to provide the user with a detailed view of the properties of each mote inside the (possibly heterogeneous) sensor network, in order to take full advantage of the motes' capabilities. Other existing applications do not provide such detailed information, e.g., what sensors does a mote in the network carry, and thus do not cope well with the case of having motes with different sensor boards attached to them. As an example, consider the case where we want to have motes collecting temperature readings on the one side of the sensor network and on the other side motes with a different kind of sensor board collecting humidity readings. The use of different kinds of sensor boards could also be justified in terms of cost or because we want to take advantage of older equipment.

The operation of the discovery service relies on the correct programming of the motes; when installing the application firmware on the mote, the actual hardware characteristics are passed as parameters to the service. Each sensor type available to each mote and its mote type are represented by an integer, in correspondence to the integer IDs used in the data tier concerning the mote and sensor types (see Section 3.3).

Based on this information, the discovery and registration of every mote in the network is achieved using a straight-forward and quite simple protocol. When a mote is initially powered on, and after its startup and initialization phase completes, the discovery protocol sends out to the control center a short message containing the mote's ID, type and list of available sensors. The format of this message can be seen in Fig. 8. By sending this message, the mote is eventually registered in jWebDust's data tier along with an amount of useful information when the control center receives the message. Note that after sending the message, the mote will wait for an acknowledgement message from the control center. This is done to make sure that the registration message will reach the control center with a certain period of time. If such an acknowledgment message is not received within the given time period (that can be adjusted depending on the expected network size and communication medium parameters), the mote repeats the transmission by sending the short registration message. This process is repeated until an acknowledgement is properly received, i.e., the mote is registered in the control center.


**Figure 8**. The format of the discovery message.

A discussion on dynamic service discovery is done in [37, 22, 36]. In all three cases the use of an XML approach versus custom discovery messages encoding is proposed, i.e., the messages of the discovery protocol are formatted in XML. This approach offers a very good solution to the problem of heterogeneity and service discovery in wireless sensor networks, but has the drawback of requiring large messages (as analyzed in an example in [37], using ASCII XML requires 181 bytes, although using a binary encoding of XML (Binary XML) requires 20 bytes), which is not a good choice in TinyOS based sensor networks. Furthermore, we make the assumption that all motes in the network are programmed with jWebDust firmware, and thus all motes use the same format for the discovery messages.

### *6.2. Node Status Monitoring Service*

Furthermore, we would like to know whether a mote is still ''alive,'' if it has turned off its radio and is ''asleep,'' in case an energy-saving scheme is in operation [8,11], or if communication is disrupted. Thus, the discovery service will send a short message to the control center when a certain time period elapses, stating that the specific mote is ''up.'' This report time period is specified by the user and can also be adjusted by jWebDust's interface without the need to redeploy the network. Similarly, when a mote decides to change its state and ''sleep,'' it sends out a notification to the control center. Currently, we are not using sleep-awake schemes.

However, if a reading from some mote of the network reaches its control center, it is as if an ''alive'' message has been sent from the mote. So, these messages could be sent only on periods of activity in order to save energy. In fact, the ''alive'' message is formatted in the same way as sensor readings messages, except that it contains just the readings of the energy levels of the mote (if available). Energy level readings are registered in the system database as another type of sensor readings.

The correct operation of this service offers jWebDust the ability to make queries concerning specific sensors of the motes in the network, since it is aware of the sensing capabilities of each mote in the sensor network. Apart from getting a more precise view of the overall sensor network's capabilities, this process also leads to potential energy savings, since otherwise we might have to poll every sensor in each mote, in order to get readings from every operating mote in the network.

### *6.3. Time Synchronization Service*

Time synchronization of the motes is a significant part of any sensor network, as outlined in [39]. Many applications need some kind of collaboration among the motes in the network, which in turn means some kind of acceptable time synchronization. An example of the need for precise time synchronization is an application for the surveillance of a field and detection of intruders, that combines readings from several motes in order to compute the exact location of the intruder. Time synchronization can also be used to coordinate sleep-awake schemes in the sensor network (motes decide when to go to sleep based on their synchronized local clocks).

A time synchronization scheme called RBS (Reference Broadcasts Synchronization) is described in [19]. RBS employs a reference mote that transmits a reference packet to the receivers listening to it. When each listener receives the reference packet, it records the time of the reception according to its local clock and then exchanges its observation with other receivers. The result of this procedure is that the receivers can now form a relative (to the reference mote) local timescale. With some simple extensions to this general scheme (that

applies to a single broadcast domain) RBS can also be used in multihop sensor networks. RBS has been implemented for TinyOS.

Another approach is used by TinyDB, which uses a tree based (many-to-one) routing scheme. The general idea is to take advantage of the transmission hierarchy already present in the network, since every mote has selected another mote as its parent in order to send a message back to the sink. Synchronization is made possible by timestamping the messages transmitted from the control center (e.g., when sending out queries). Every mote that receives or forwards such a message updates its system clock with the timestamp found in the message.

The TPSN time synchronization scheme for sensor networks and its implementation for TinyOS is described in [14]. This method provides more accurate time synchronization than the above, since it performs a pair wise time synchronization between motes that belong to different hierarchical levels. TPSN follows a model where every mote maintains a local clock that is synchronized with respect to a reference mote (e.g., the control center). The protocol operates in two phases, the level-discovery and the synchronization phase.

These approaches have all unique characteristics and result in different results in terms of time accuracy, energy consumption and number of messages needed. The choice of the time synchronization scheme, as in the case of routing schemes, depends largely on the application. For the applications we plan to use jWebDust with we do not need time accuracy beyond the millisecond scale, so we will use a simple time synchronization scheme, like the one used in TinyDB.

## 7. Developing Applications with jWebDust

Developing applications for jWebDust involves three steps; installing and configuring jWebDust's components, deploying a sensor network and tuning of the system to meet the users' specific needs. As can be seen in Fig. 3, jWebDust consists of several tiers and includes components such as the motes' firmware control tier software etc., that need to be configured for the whole system to function properly. Let us now assume that we want to setup a light and temperature monitoring application over an office building. We here note that all tiers from the control tier and up, can be installed and run on the same system or they can be installed in different systems.

The deployment of the sensor network involves programming the motes with a certain firmware and placing them in the desired field (i.e., in our example, the various offices of the building). The firmware installed in the motes implements the query processing subsystem, sensing and routing. When programming the motes, the users must supply them with unique mote IDs and also with the respective sensor type IDs of their sensor boards. After the motes' deployment, users may have to modify their firmware in special cases, e.g., if an alternative routing scheme is to be used in the sensor network or if they want to add a new type of sensor queries.

The next step is to install the control center in the control tier and configure its connection to jWebDust's database. The control center firmware plays the role of the gateway between the sensor network and the control tier, forwarding messages from one tier to the other. The control software must also be configured with a sensor network ID (recall that jWebDust can function with multiple sensor networks).

Moving on to the data tier, the user must setup jWebDust's database (e.g., PostgreSQL [29]). The types of motes and sensors used in the network must be inserted into the database. Some scripts are provided to the user for creating a database with the standard

types (Mica motes and some of their sensor boards). For the middle tier of jWebDust's architecture, a web server capable of execute Java code must be installed and configured (e.g., Resin [31], JBoss [20]).

Regarding the presentation tier, i.e., the user interface, our goal was to provide a highly customizable user interface based on Java servlets and portlets, which gives users the ability to check the state of the sensor network from a variety of devices (PCs, PDAs, cellular phones) along with ease of use. The user can issue queries and do network maintenance tasks (such as adding new networks) by the use of a web-based interface provided, that can be modified to meet specific needs and restrictions (such as those imposed e.g., by using a PDA to check the network status).

Each servlet and portlet is developed independently of one another, and is stored in a servlet and portlet container (the application server). These components communicate with the data tier through the middle tier. Their result is markup code to be displayed by a web browser.

However, portlets, unlike servlets, produce only part of the markup code of a web page. Thus, they are meant to be aggregated within the context of a portal page, that is made up from a number of different windows (while servlets are displayed usually in a separate web page or frame). Each window represents a portlet and each portlet is an independent application (e.g., an application could be to show the energy levels of motes for a specified time interval). The user selects which applications she wants and the application server produces the corresponding result combining the output from the portlets.

Regarding jWebDust's differences with existing solutions, e.g., moteView, jWebDust will provide some extra flexibility in the way the final user utilizes a wireless sensor network, i.e., more ways to query the system, and to the way the application interfaces with the user, i.e., a customizable web-based interface.

### *7.1. A Sample Application – Precision Agriculture*

Precision agriculture, in general, is about monitoring a set of parameters in various areas of a farm and managing the responding situations, which can have an effect on crop yields. The basic idea is that a large farm consists of a number of smaller areas, each with its own specific parameters, that can be dealt with independently. Essentially, this can lead e.g., to a more effective use of pesticides, which in turn leads to savings for the farmer and better quality for the farm's yield.

For our sample application, consider the case where we want to monitor the conditions inside a vineyard. Mr. Georganopoulos, a retired telephone company officer, is not an expert in computers but wants to monitor the conditions in his vineyard himself every day, without going to the vineyard (which is a good 200 Km away from his residence).

Mr. Georganopoulos is interested in monitoring only temperature, moisture, and energy levels of the motes, and does not wish to change the queries issued to get these results. After issuing the queries, he needs a simple layout to quickly check on the conditions inside the vineyard. The administrator after setting the sensor network in the vineyard, chooses the respective portlets to be displayed everytime the user logs in to the system via his browser. A sample interface is seen on Fig. 9. Such an interface will be easy to customize and also available everywhere through a simple browser and not just a specific application client.

Another example of how jWebDust differs from existing solutions is that it will allow the user to have a heterogenous sensor network. For instance a possible situation is that

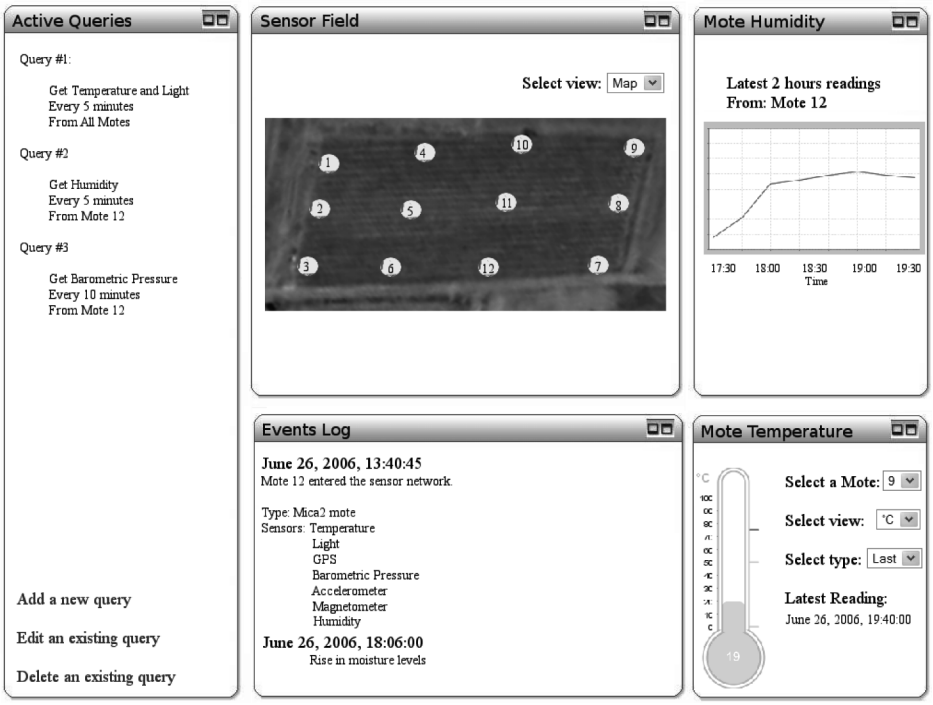


**Figure 9**. A sample interface for vineyard monitoring using portlets.

Mr. Georganopoulos decides to expand his vineyard and chooses to use different sensor boards than the ones he is currently using, or maybe even different sensor nodes. Heterogeneity will probably be a feature that the network administrator will appreciate.

## 8. Concluding Remarks

We have presented the architecture and the design of the main components of jWebDust, and discussed several distinct features along with their implementation and functionality. The main strengths of jWebDust are:

- Provides an environment to package and manage the plethora of lower level protocols with minimum implementation and administration effort.
- Allows to implement new functionality that can be easily integrated with the rest of the architecture at all levels of the hierarchy (i.e., sensor level, control centers level, database level, user-interface level) to best suit the application needs.
- Supports multiple sensor networks (i.e., physically separated) with multiple/separate control centers and allows to handle them as a single virtual sensor network, even if more than one sensor share the same ID.
- Handles sensor networks comprised of devices with heterogeneous characteristics. Real world sensor networks will rarely be homogeneous.
- Supports Disconnected/Mobile Control Centers by taking special care of long disconnections of the control centers, and the sensor networks attached to these

centers, from the upper parts of the architecture (i.e., database, web, etc.), so that information is not lost.
- Many users can simultaneously query, monitor, and visualize the execution of the wireless sensor network through a Web-based user interface.

We plan to continue the development of jWebDust by implementing the set of features described in this work, improving them and introducing new ones. For example, we wish to address the security issues uniformly over the application environment. Another issue we are considering is the implementation of sleep-awake strategies that will further increase the lifetime of the network as sensors enter sleep mode to reduce energy consumption. We also plan to make a comparison of jWebDust and other similar software environments, when the software reaches a more stable level. Finally, apart from implementing the user interface using portlets, we also plan to use Google Earth and Google Maps to build some extra functionality.

## About the Author

**Sotiris Nikoletseas** is an Assistant Professor at the Computer Engineering and Informatics Department of Patras University, Greece and Director of the SensorsLab at CTI. His research interests include Algorithmic Techniques in Distributed Computing (focus on sensor and mobile networks), Probabilistic Techniques and Random Graphs, and Algorithmic Engineering. He has coauthored over 100 publications in Journals and refereed Conferences, several Book Chapters and a Book on the Probabilistic Method, while he has delivered several invited talks and tutorials. He has served as the Program Committee/ General Chair of many Conferences (DCOSS, IPDPS, MSWIM, WEA) and as Editorial Board Member of several Journals. He has co-initiated international conferences on sensor networking (DCOSS, ALGOSENSORS). He has coordinated externally funded European Union R&D Projects related to fundamental aspects of modern networks.

**Ioannis Chatzigiannakis** obtained his B.Eng. from the Department of Electronics Engineering and Department of Computer Science of the University of Kent at Canterbury. He obtained his Ph.D. from the Department of Computer Engineering & Informatics of the University of Patras. He is currently Adjunct Faculty at the Computer Engineering & Informatics Department of the University of Patras (since October 2005). He is the Director of the Research Unit 1 ''Foundations of Computer Science, Relevant Technologies and Applications'' (since July 2007). He has coauthored over 70 publications that have appeared in international journals and refereed international conferences of the ACM, IEEE and EATCS, as well as four chapters in books by major publishers. His main research interests include distributed and mobile computing, wireless sensor networks, algorithm engineering and software systems. He has participated in R&D projects funded by the European Union and the Greek government (including ALCOM-FT, FLAGS, AEOLUS, FRONTS, WISEBED, AUDIS, PENED, PYTHAGORAS). He has received a young researcher grant from Nokia. He periodically acts as a scientific reviewer for the Swiss National Science Foundation, Italian Research Foundation and Enterprise Estonia. He has also served as a consultant to major Greek computing industries He is the Secretary of the European Association for Theoretical Computer Science since July 2008.

**Georgios Mylonas** received his Ph.D. degree from the Department of Computer Engineering and Informatics at the University of Patras, Greece, in December 2008. He is currently working as a post-doc researcher at the Research Academic Computer Technology Institute, Patras, Greece. His research interests lie in the areas of wireless sensor networks and distributed systems and games. He has been involved in the AEOLUS,

WISEBED and ALGODES PENED research projects, funded by the European Union and the Greek Government, which focus on the algorithmic and software issues related to wireless sensor networks.

## References

1. K. Aberer, M. Hauswirth, and A. Salehi. ''The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks,'' Tech. Report, Ecole Polytechnique Federale de Lausanne (EPFL), 2006, Technical Report.
2. T. Antoniou, A. Boukerche, I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas,'' A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range,'' *37th Annual ACM/IEEE Simulation Symposium (ANSS 2004)*, 2004, pp. 43–52.
3. A. Boukerche, X. Cheng, and J. Linus, ''Energy-aware data-centric routing in microsensor networks,'' ACM Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2003) (Paris, France), 2003, pp. 42–49.
4. A. Boukerche and S. Nikoletseas, ''Energy efficient algorithms in wireless sensor networks,'' Springer Verlag, 2004, Chapter in book.
5. A. Boukerche and S. Nikoletseas, ''Wireless communications systems and networks,'' ch. in Protocols for Data Propagation in Wireless Sensor Networks: A Survey, Kluwer Academic Publishers, June 2004.
6. A. Boukerche, R. W. N. Pazzi, and R. B. Araujo, ''A supporting protocol to periodic, event-driven and query-based application scenarios for critical conditions surveillance,'' *1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004)*, 2004, pp. 137–146.
7. P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden, ''TASK: Sensor network in a box,'' In the *Proceedings of the 2nd European Workshop on Sensor Networks*, 2005.
8. I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas, ''Power conservation schemes for energy efficient data propagation in heterogeneous wireless sensor networks,'' *38th Annual ACM/IEEE Simulation Symposium (ANSS 2004)*, 2005.
9. I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, ''jWebDust : A java-based generic application environment for wireless sensor networks,'' In the *proceedings of the First International Conference on Distributed Computing in Sensor Systems (DCOSS '05)*, 2005, pp. 376–386.
10. I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, ''The architecture of a generic application environment for wireless sensor networks accessible via internet,'' To appear in *The Proceedings of the Fifth International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP '06)*, 2006.
11. I. Chatzigiannakis and S. Nikoletseas, ''A sleep-awake protocol for information propagation in smart dust networks,'' *3rd International Workshop on Mobile, Ad-hoc and Sensor Networks (WMAN 2003)*, 2003, IPDPS Workshops, p. 225.
12. E.J. Duarte-Melo and M. Liu, ''Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks,'' IEEE Global Telecommunications Conference (GLOBECOM 2002), vol. 1, IEEE, 2002, pp. 21–25.
13. S. Eisenman, N. Lane, E. Miluzzo, R. Peterson, G. Ahn, and A. Campbell, ''MetroSense project: People-centric sensing at scale,'' In. *Workshop on World-Sensor-Web (WSW 2006)*, Boulder, October 31, 2006.
14. S. Ganeriwal, R. Kumar, and M. Srivastava, ''Timingsync protocol for sensor networks,'' *1st ACM International Conference On Embedded Networked Sensor Systems (SenSys 2003) (Los Angeles, CA, USA)*, 2003, pp. 138–146.
15. Habitat monitoring on great duck island, http://www.greatduckisland.net.
16. B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden, ''CarTel: A distributed mobile sensor computing system,'' in the *Proceedings of SenSys '06*, 2006.

17. C. Intanagonwiwat, R. Govindan, and D. Estrin, ''Directed diffusion: A scalable and robust communication paradigm for sensor networks,'' *6th ACM/IEEE Annual International Conference on Mobile Computing (MOBICOM 2000)*, 2000, pp. 56–67.
18. Exploratory research: ''Heterogeneous sensor networks,'' *Intel Technology Journal: Research & Development at Intel (2004)*, http://www.intel.com/research/exploratory/heterogeneous.htm.
19. L. Girod J. Elson and D. Estrin, ''Fine-grained network time synchronization using reference broadcasts,'' *Fifth Symposium on Operating System Design and Implementation (OSDI 2002)*, 2002.
20. JBoss, and open-source application server, http://www.jboss.org.
21. Web page of the james reserve, http://www.jamesreserve.edu/.
22. S. Krco, D. Cleary, and D. Parker, ''Enabling ubiquitous sensor networking over mobile networks through peer-to-peer overlay networking,'' *Computer Communications*, vol. 28, 1586–1601, 2005
23. Motelab, an experimental wireless sensor network, http://motelab.eecs.harvard.edu/.
24. *Mote-VIEW* monitoring software, Crossbow Technology Inc., http://www.xbow.com/.
25. MSR Networked Embedded Sensing Toolkit (MSR Sense), http://research.microsoft.com/nec/msrsense/.
26. C. Perkins, ''Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers,'' *18th ACM Conference on Communications Architectures, Protocols and Applications (SIGCOMM 1994)*, 1994, pp. 234–244.
27. C.E. Perkins and E.M. Royer, ''Ad-hoc on demand distance vector (AODV) routing,'' *2nd IEEE Annual Workshop on Mobile Computing Systems and Applications*, 1999, pp. 90–100.
28. Jose Pinto, Alexandre Sousa, Paulo Lebres, Gil Manuel Goncalves, and Joao Sousa, ''MonSense - application for deployment, monitoring and control of wireless sensor networks,'' Poster in RealWSN'06.
29. PostgreSQL official web-site, http://www.postgresql.org.
30. G.J. Pottie and W.J. Kaiser, ''Wireless integrated network sensors,'' *ACM Communications* (2000).
31. Caucho Resin: Fast, open-source application server, http://www.caucho.com.
32. A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao, ''SenseWeb: Browsing the physical world in real time,'' *Demo Abstract, ACM/IEEE IPSN06*, Nashville, TN, April 2006.
33. The Scatterweb wireless sensor network platform, http://www.scatterweb.de.
34. Web page of the sensorscope project, http://sensorscope.epfl.ch.
35. R. Shah, S. Roy, S. Jain, and W. Brunette, ''Datamules: Modeling a three-tier architecture for sparse sensor networks,'' in the. *Proceedings of the First International Workshop on Sensor Network Protocols and Applications*, 2003, pp. 30–41.
36. J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, ''Hourglass: An infrastructure for connecting sensor networks and applications, 2004,'' Harvard Technical Report TR-21-04.
37. S. Tilak, K. Chiu, N. Abu-Ghazaleh, and T. Fountain, ''Dynamic resource discovery for wireless sensor networks,'' in *The Proceedings of IFIP International Symposium on Network-Centric Ubiquitous Systems (NCUS 2005)*, 2005.
38. *TinyDB*: A declarative database for sensor networks, http://telegraph.cs.berkeley.edu/tinydb/.
39. F. Zhao and L. Guibas, *Wireless Sensor Networks: An Information Processing Approach*, Elsevier Press.