Programming a Sensor Network in a layered middleware architecture

Michele Albano
Instituto de Telecomunicações - Aveiro
Portugal

Stefano Chessa
University of Pisa
and
ISTI-CNR
Italy

Abstract

Wireless Sensor Networks (WSNs) have become a mature technology aimed at performing environmental monitoring and data collection. Nonetheless, harnessing the power of a WSN presents a number of research challenges. WSN application developers have to deal both with the business logic of the application and with WSN's issues, such as those related to networking (routing), storage, and transport. A middleware can cope with this emerging complexity, and can provide the necessary abstractions for the definition, creation and maintenance of applications. This work discusses the problems related to the development of such a middleware and surveys the state of the art on the field.

1. Introduction

In the last few years, hardware and software innovations have been leading Wireless Sensor Networks (WSNs) from the research labs to deployments in real contexts. A WSN application is a distributed application that is built on a large number of low-cost, low-power, battery-powered sensors (Akyildiz et Al., 2002; Baronti et Al., 2007; Chessa, 2009). Sensors are spread in an environment (*sensor field*) without any predetermined infrastructure and cooperate to execute common monitoring tasks which usually consist in sensing environmental data and monitoring a variable set of objects. The sensed data are collected by a sink (a node that can communicate with both the WSN and an external network), when the sink is connected to the network. The sink, which could be either static or mobile, is in turn accessed by the external operators to retrieve the information gathered by the network.
Distilling a given high-level behavior from a set of sensors is a challenging problem, since the application has to deal with its own business logic, and with the issues that naturally arise when WSNs are taken into account, such as network formation, data transport and data management, security and energy saving. Dealing with these issues can be done either explicitly,

thus adding complexity to the applications, or implicitly by means of a middleware, that is a software layer that abstracts from common issues of the WSNs.
A middleware would let the developers to focus on the applications' business logic. On the other hand, there is no unique way to define which issues belong to the WSN, and which ones are part of the business logic. In general, this depends on the politics/mechanisms dichotomy, and hence on the level of abstraction that is provided by the middleware. Even a minimal middleware can provide benefits to the application developer, nonetheless it presents research challenges.
The level of abstraction provided by a WSN middleware inherently depends on the mechanisms that are used by the middleware to implement the high-level primitives. The analysis of state-of-the-art mechanisms presented in this work is developed into a structured vision of the mechanisms, that were organized into three layers (Programming Abstraction layer, Data Management layer, and Network layer) whose mechanisms can interact with each other or can be used directly by the application, plus a set of Dependability mechanisms that is orthogonal to the layers and that comprises mechanisms that are used by all the layers and by the user applications alike.

2. Organizing middleware mechanisms into layers

Current research papers agree that one of the critical points to leverage on the potential usefulness of WSNs is the possibility of abstracting common WSNs problems by means of convenient middleware, but literature is not coherent when defining what a middleware is (Albano et Al., 2007-1; Bai et Al., 2009; Chu & Buyya, 2007; Hadim & Mohamed, 2006; Mottola & Picco, 2008; Rahman, 2009; Romer et Al., 2002; Rubio et Al., 2007; Sugihara et Al., 2008; Tong, 2009; Wang et Al., 2008).
To this purpose we focus on the goals that a middleware has to achieve. The main purpose of middleware is to support the development, maintenance, deployment and execution of applications, filling in the gap between the application layer and the hardware, operating system and network stacks layers. In the case of WSNs, this can include mechanisms for formulating high-level sensing tasks, communicating this task to the WSN, merging/aggregating the sensor readings, and reporting them. The actual analysis of state-of-the-art middleware presented a variety of different techniques and approaches used to address the aforementioned goals. In fact, WSN systems offer functionalities that can be collectively called "middleware" but that are very different from each other. For example, different middleware offer:

- low-level mechanisms that operate at the datum granularity, such as data centric storage (Albano et Al., 2010; Bruck et Al., 2005)
- abstract mechanisms that hide the single datum, like database-like systems (Amato et Al., 2010; Madden et Al., 2003)
- service oriented architectures, for example the ZigBee standard (Baronti et Al., 2007)
- platforms for mobile agents, for example AFME (Muldoon et Al., 2006) or Agilla (Fok et Al., 2009)

This analysis of the state-of-the-art mechanisms identifies three layers (Programming Abstraction layer, Data Management layer, and Network layer), and the mechanisms are categorized in terms of this structure. The mechanisms can interact with each other or can be used directly by the application. A set of mechanisms for Dependability is also identified, and it can be

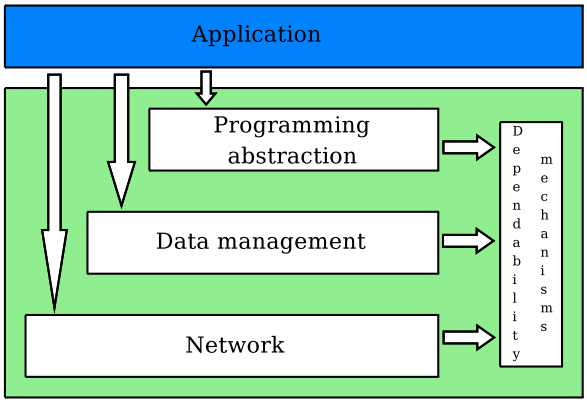


Fig. 1. Middleware is structured as a set of layers between the hardware and the application.

used at different levels for different purposes. Figure 1 provides a graphical representation of this structure.

Programming Abstraction layer comprises the mechanisms that model the behavior of the whole WSN, and that provide the user application with techniques to abstract the WSN details. For example, a user application can perceive a WSN as a relational database (Amato et Al., 2010; Madden et Al., 2003), as a publish/subscribe system (Albano & Chessa, 2009-1), as a service oriented architecture (Baronti et Al., 2007), or as a platform for mobile agents (Muldoon et Al., 2006).

The Data Management layer lets the user perform *store* and *retrieve* operations on data, either storing them on a specific sensor or set of sensors, or selecting the set of sensors from a metadatum of the data to be stored or retrieved, as in Data Centric Storage systems (Albano et Al., 2010; Ee et Al., 2006; Ratnasamy et Al., 2003).

The Network layer features explicit send and receive operations, and lets the user application control the transmissions performed at a finer grain. This layer nonetheless performs some abstraction, since it hides the routing protocol used, hence it lets the invoker specify the goal of the routing process, be it a sensor or a coordinate in the sensor field. This work does not provide details on Network layers, since it focuses on the high-level issues of WSN middleware.

The Dependability mechanisms regard the techniques used to create reliable primitives on the layers, and are used by all the layers alike.

A user application can rely on one of these layers, depending on the level of abstraction required, by addressing the whole WSN using the abstraction provided by the Programming Abstraction layer, or by performing explicit Data Management specifying which data are stored and retrieved, or it can be based on network level send/receive operations.

As an example of interaction of these layers, the Publish/Subscribe system described in (Albano & Chessa, 2009-1) is based on a Data Centric Storage layer provided in (Albano et Al., 2010). The DCS mechanisms such as (Ratnasamy et Al., 2003) used geographical routing such as (Karp & Kung, 2000) (Network layer) to transport data and queries to proper sensors. Erasure coding (Dependability mechanisms) was used to guarantee data availability in (Albano & Chessa, 2009-2), and to optimize access to data in (Dimakis et Al., 2006).

3. Data Management layer

The final goal of a WSN is to gather data from the environment and to route it to data consumers, and the Data Management layer is responsible for controlling dataflows and managing the exchange of data between nodes, towards the data consumers, with the option of caching the data into the WSN before transferring them out of the WSN. The current paradigm considers that data exit the WSN via special sensors, called *sink*s. The sinks are gateways that are connected to both the WSN and an external network, like the internet.
Data can reach the sink in three ways:

1. **local storage**: data are stored on a set of sensors that depends on the sensor that produced the data,
2. **external storage**: data are sent to the sink as soon as they are produced,
3. **in-network storage**: produced data are sent to a set of the sensors that depends on some characteristics of the data.

3.1 Local and External Storage

Local storage is a Data Management paradigm that prescribes data to be stored on a set of sensors that depends on the sensor producing them. The most common implementation of this paradigm stores data on the sensor producing them. When the sink wants to access the data, it must send a request to the sensors that stored them. This approach presents some limitations. The first is that, if some sensors detect a lot of events, their related sets of sensors become burdened by storing more data and hence they deplete their resources earlier (battery, memory, etc). A second problem is that the sink does not usually know in advance which sensor is producing data it is interested into, and hence it is necessary to broadcast a request to contact every node when retrieving data.
External storage is another approach where data are sent to the sink as soon as they are produced. Main drawbacks of this approach are that data can not be pre-processed and aggregated with other data. Moreover, if there are more than one sinks, data must be duplicated and sent to each sink. Finally, the sink must be always connected to the WSN, or the sink would miss data produced while it is away.

3.1.1 Local storage

A number of proposals for WSN middleware are based on the local storage paradigm, since it is the most obvious paradigm to cope with discontinuous connection of the sink to the WSN.
The work describing tinyDSM (Piotrowski et Al., 2009), presented substantially a local storage Data Management layer. This middleware allows a node to ask its neighbors to replicate data, hence realizing a high-availability local storage system. When a sink queries the data, data replication assures the information to be available even if some nodes are exhausted or in sleep mode.
The middleware presented in (Dimakis et Al., 2006) proposes a local storage solution that uses a kind of encoding that enables a fully distributed computation of the code. The technique refers to a model featuring a set V_1 of k source sensors, each producing a single data packet, and a set V_2 of n storage sensors, storing the data. The encoding is based on a bipartite graph that connects each source to $O(\log k)$ randomly selected storage sensors. The encoding is performed by each storage sensor using a linear combination of all incoming data, where each incoming data are multiplied by a randomly selected factor. Each storage sensor then

stores the random factors associated to each incoming datum and the result of the linear combination. The authors show that the sink can reconstruct all k packets querying any k storage nodes with high probability. This fully distributed encoding results in a memory overhead that can be ignored only if the data to be encoded are much larger than the random factors.
Another local storage technique for fully distributed coding is based on the Growth Codes (Kamra et Al., 2006), and it implements linear coding using XOR operations. In this model the sensors give to the sink codewords obtained as the XOR of different data, and the sink performs the decoding. The goal of growth codes is coping with the "coupon collection phenomena" with random data collection, since in a pure erasure coding approach, the last few data symbols are the most difficult to get. This coding algorithm implies that for the first data to be encoded, only the original data are stored, and only after some time the encoding composes a number of data to construct the codewords to be stored. As long as the sink receives enough codewords obtained from a single datum, it is able to obtain the different data from the codewords.

3.1.2 External storage

Data Management layers implementing an external storage solution, cope with data management by sending data to the sinks as soon as they are produced. In this paradigm, data are stored and analyzed outside the WSN, hence the WSN's role is limited to data acquisition. Data Management layers of this kind are usually more resource expensive of the other Data Management layers, since they perform a large number of data transmission operations. In this kind of Data Management layer, data can be subject to a filter that decides if it has to be sent to the sink, but the filter must be loose enough not to throw away any data that can become interesting for the user application during the WSN's lifetime.
A refinement of this paradigm is the routing tree, that is used in Directed Diffusion (Intanagonwiwat et Al., 2000), that is a middleware that implements an External storage system that is reprogrammable on-the-fly using interest propagation. Data are named by meta-data that describe them, then the data consumer (a sink) disperses a message into the WSN by a broadcast to instruct nodes to send him data by a multi-hop routing tree that is set up by the interest dispersal: every node takes note about the node he received the interest from, and it sets it up as the next step in a routing process towards the data consumer. At the same time, every node starts considering data pertaining to a certain category as "interesting data" and, instead of discarding them, they send it along the gradient created by the interest dispersal towards the sink. Some Programming Abstraction layers, such as (Amato et Al., 2010; Madden et Al., 2003), employ a refined version of External storage, where data are sent towards the sink as soon as it is collected, but where it is processed while it is moving up the routing tree.
Another solution of the external storage kind is publish/subscribe, where nodes are instructed about sending data concerning interesting data when they collect them. An example is the Data Management solution adopted in Mires (Souto et Al., 2004), that sets up data collection by means of a publish/subscribe service. The main difference with routing trees, is that publish/subscribe is initiated by a node that advertises the data it can produce, and then the node is explicitly instructed to send the data to some data consumer. In contrast, routing trees are about flooding the WSN with an interest, that instructs all the nodes to send data pertaining a meta-datum to the broadcast initiator.

3.2 Data Centric Storage

Data Centric Storage (DCS) is a family of in-network storage techniques, using functions that relate different meta-data describing data to different sets of sensors. Since in WSNs the content of the data is generally more important than the identity of the sensors that gathered the data, a node producing a datum d associates a meta-datum k to d, computes a set of sensors applying a function f to the meta-datum k, and then the node sends d to the set of sensors $f(k)$ for storage. At the high-level, a sink directs a retrieve request towards a meta-datum k. This operation is realized applying the same function f to the meta-datum k to identify the set of sensors $f(k)$ that stored the data. DSWare (Li et Al., 2003-1) is an example of Programming Abstraction layer that relies on DCS to cache data into the WSN before providing them to the user application.

The seminal work described in (Ratnasamy et Al., 2003) introduced DCS, and also compared it with local and external storage. Comparing this approach to the external storage approach, the authors observed that DCS contributes to save sensors' energy and to improve network lifetime. Since sensors have limited memory capacity, the storage of all the data sensed by the WSN may result impractical, however with DCS it is possible to pre-process and aggregate data and thus reduce their size.

A number of different DCS techniques have been proposed, and they differ in the way

1. the datum is assigned a meta-datum;
2. the nodes that store the datum of a meta-datum are selected;
3. the datum is routed to/from the nodes that store it.

The assignment of a meta-datum to the datum is inherently application-dependent, and it will not be discussed in this survey. On the other hand, different DCS architectures can use different functions f_i from meta-datum k to a subset $f_i(k)$ of the sensors, and they can access different Network layers to implement routing from/to these subsets of sensors, and the rest of this section describes the state-of-the-art of DCS architectures based on these two characteristics.

3.2.1 Geographic Hash Table

The reference model of DCS in WSNs is the *Geographic Hash Table* (DCS-GHT) (Ratnasamy et Al., 2003), that constitutes the first proposal of DCS. In DCS-GHT, it is assumed that the geographic coordinates of each sensor are known, and that each datum is described by a *meta-datum* (or *name*). The set of sensors selected to store a datum is computed by means of a hash function applied to the corresponding meta-datum. This function returns a pair of geographic coordinates fitting in the area where the sensor network is deployed.

DCS-GHT exploits the primitives `store` for data storage, and `retrieve` for data retrieval. The `store` primitive takes in input a datum d and its meta-datum k. By hashing k, it produces a pair of coordinates (x, y) and uses the GPSR routing protocol (Karp & Kung, 2000) to route the pair (d, k) towards (x, y). The GPSR routing protocol is able to deliver the data to the sensor closest to the point (x, y) (this sensor is called *home node*). In principle the home node could be a sensor located exactly on the point (x, y), however the chance for this to happen is negligible. As a side effect, GPSR also identifies the inner perimeter of sensors (called *home perimeter*) enclosing (x, y) (the reader is referred to the work of (Karp & Kung, 2000) for more details). Once the home node receives the pair (k, d), it stores the pair in its memory, and, to enforce data persistence against sensors' faults, it also requests the sensors in the home perimeter to store a copy of (k, d). The `retrieve` primitive hashes the input parameter k (the

meta-datum) to obtain the coordinate (x, y), then, by means of GPSR, it sends a request for the data with meta-datum k to the point (x, y). When this request reaches the sensors in the perimeter around (x, y), they send back the data they store that correspond to k. See Figure 2 for an example of `store` and `retrieve` execution, where the data producer A stores into the WSN a datum regarding a meta-datum k, and the data consumer (the sink) asks the WSN for data regarding the same meta-datum k. Both A and the sink hash k to the same location (x, y) (represented in the figure by D), then they route their requests towards that location. In the case of A, the `store` primitive semantics involve storing a copy of its datum on all the nodes in the home perimeter around D. In the case of the sink, a `retrieve` response is generated as soon as the query reaches one of the nodes belonging to the perimeter around D.

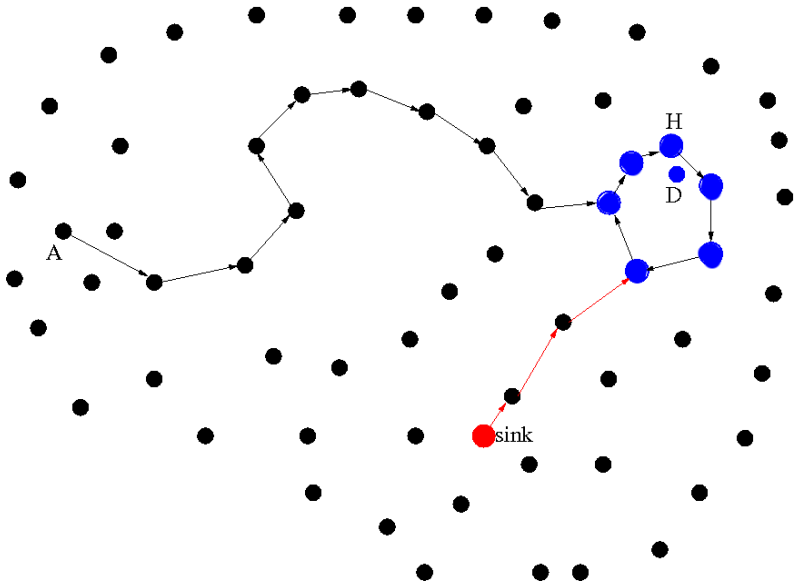


Fig. 2. A stores data, the sink retrieves them, using DCS-GHT

Although innovative, DCS-GHT presents a number of limitations when deployed on real WSNs. It assumes a uniform distribution of sensors and uniformly hashes meta-data on them. Moreover, if WSN produces a large amount of data associated to the same meta-datum, all such data will be stored by the DCS-GHT within the same home perimeter, thus overloading sensors on that perimeter. To avoid this problem DCS-GHT employs **structured replication**, that is a technique that augments event names with a hierarchy depth d and uses a hierarchical decomposition of the key space. Let us consider a single meta-datum that is hashed into a location r, and let us call r the root location for the meta-datum, and d the hierarchy depth. Let us consider the sensing area as the 0-level cell, and given an i-level cell, let us divide recursively it into 4 smaller cells, splitting each coordinate span in half. Given a hierarchy depth d, there are 4^d cells, and $4^d - 1$ mirror images of r, replicating the position of r in its d-level cell into the other cells of d hierarchy level.
For example, Figure 3 shows a $d = 2$ decomposition, and the mirror images of the root point $(3, 3)$ at every level. A node that detects an event, now stores the event at the mirror closest to its location, which is easily computable. Thus, structured replication reduces the storage cost

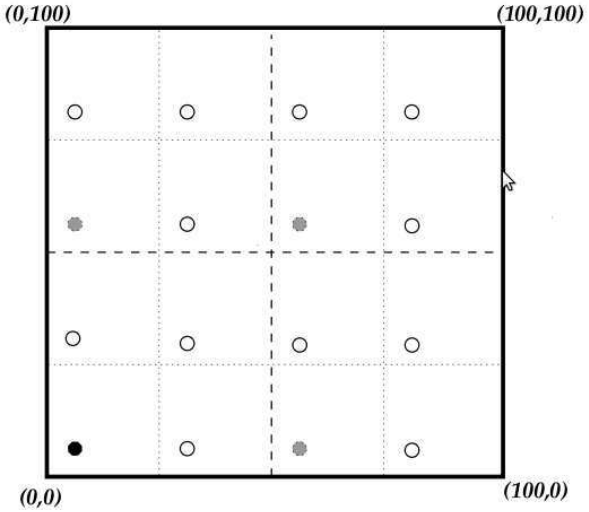


Fig. 3. Example of structured replication with a 2-level decomposition

at one node for one key with n detected events from $O(\sqrt{n})$ to $O(\sqrt{n}/2^d)$. DCS-GHT must route queries to all mirror nodes, to find all data stored into the 4^d mirrors of r.

Structured replication is efficient in limiting the quantity of data stored around a single home node, but this is not sufficient by itself to ensure load balancing, in fact the storage load can become unbalanced even if there is not an unbalance in the meta-data.

Resilient Data Centric Storage (R-DCS) (Ghose et Al., 2003) is an extension of DCS-GHT that addresses the issue of having all data of the same type stored on the same set of nodes. It divides the sensing area into zones, and each sensor can either be a *monitor node*, a *replica node*, or a *normal node*, with respect to a given event type. A normal node generates events and forwards packets, but it does not store data pertaining the given event type. Each zone has one sensor that is on *monitor mode* for each event type; the monitor node does not store data, but knows the location of replica nodes for their event type and fowards data to them. *Replica nodes*, finally, are nodes that can store data regarding a given event type.

Bottom line, R-DCS adds to the efficiency of the DCS system limiting the data transmission from the sensor producing a datum to the monitor node of its zone, and then to the closest replica node. Moreover, resiliency to failures is improved since data are not replicated locally, but instead they are located on replica nodes that are far away from each other, and hence a disaster, that would destroy all sensors close to it, can not exterminate all replica nodes for a given meta-datum.

Another variant of DCS-GHT is Rendezvous Regions (RR) (Seada et Helmy, 2004), that has mechanisms similar to DCS-GHT but, instead of directing queries towards a home node, it makes use of regions in the sensing area, and of all the sensors located into those regions. In RR the network topology is divided into geographical regions, where each region is responsible for a set of keys, with keys representing meta-data of sensed data, or services offered by sensors. The service or data provider stores information in the corresponding region, and the service or data user associates directly its query to the region. The obvious distinction

between RR and DCS-GHT is in using a rendezvous region instead of a rendezvous point. Moreover, RR is also targeted to designing geographic systems that need only approximate location information for the nodes.
Other works explore different routing mechanisms in DCS, based on multiple trees (Ee et Al., 2006), on Geographic Hash Tables over clusters of nodes (Araujo et Al., 2005), on the recursive subdivision of the WSN using K-D trees (Aly et Al., 2006; Li et Al., 2003-2), or on double rulings (Sarkar et Al., 2006).
Let us now consider the storage load on the nodes, and let us define Quality of Service (QoS) for Data Management in the WSN as the capability of the Data Management layer to guarantee that a given datum is stored on a given number of sensors, in order to provide the desired resilience to sensor faults for the datum.
The state-of-the-art mechanisms discussed so far were designed for different goals, but despite their merits, they did not take into account QoS. For example, in DCS-GHT the number of sensors storing a datum depends on the number of sensors in a perimeter, and in RR it depends on the population of a region. On the other hand, the work described in (Albano et Al., 2007-2) and (Albano et Al., 2010) presented DCS systems that do not rely on WSN topology to decide the level of redundancy of the datum. Rather, the systems enable the user application to select the number of nodes that are required to store the datum. Moreover, they select the destination coordinate for the datum using a biased hash function, to adapt the process to the sensor distribution: more meta-data are hashed into more populated regions of the WSN, and the result is that the storage load is balanced between all the sensors.

4. Programming Abstraction layer

Most applications do not need low-level access to a WSN, and a high-level perspective can hide the WSN under a traditional computer science appearance, like a database (Amato et Al., 2010; Madden et Al., 2003), or a publish/subscribe system (Albano & Chessa, 2009-1), or an agent-based platform (Muldoon et Al., 2006).
A thorough analysis of research papers showed that a coherent taxonomy is hard to build, since the approaches applied to WSN middleware design are very different and focus on different abstraction levels. For example, the Programming Abstraction layer comprises both the database approach of TinyDB (Madden et Al., 2003), and the process based approach of the virtual machine Maté (Levis & Culler, 2002). Moreover, current literature has produced taxonomies that do not agree on categories. For example, Maté has been defined as a process based approach, see Wang et Al. (2008), or as a virtual machine approach, see Rubio et Al. (2007). In this last case, the category does not really describes the way the system is programmed, but instead focuses on the underlying structure of the layer. In this survey, Programming Abstraction layers are first classified into **global entity** and **local entities** layers, then the **local entities** layers are further divided into **static local entities** and **mobile local entities**, depending on what is addressed by the user application.
The first category is **global entity**. This category is inspired by the survey of Wang et al (Wang et Al., 2008), that called it *system level abstraction*. The middleware that belong to the **global entity** category "abstract the WSN as a single virtual system and allow the programmer to express a single centralized program (global behavior)" (Wang et Al., 2008), and the WSN is considered a single virtual machine that processes the high-level program. This approach leaves "only a small set of programming primitives for the programmer while making transparent the low-level concerns such as the distributed code generation, remote data access and management, and inter-node program flow coordination" (Wang et Al., 2008). Examples of **global**

entity middleware are the database approach (Amato et Al., 2010; Madden et Al., 2003), that accesses the WSN like a single database management system, and the service oriented interface offered by the "Domain layer" of MiSense (Khedo & Subramanian, 2009), that considers the WSN like a single server that offers a set of services to the application programmer.
The second and third categories let the programmer address a number of entities interacting in the WSN, hence they are called "local approaches". Actually, these two categories differ by the identity of the entities that are considered. Category **static local entities** features programmable entities that do not change over time. Most of these approaches consider the single node as the entity that is executing the program, but this category also comprises cluster-based approaches, where the WSN is composed by a number of clusters of sensors.An example of this approach is the event-driven rule-based middleware of FACTS (Tergloth et Al., 2006), where the same application is deployed over all the sensors, and all information is represented by facts. Rules consist of a predicate over these facts and an action, and the action is triggered by the rule engine whenever the predicate becomes true. Another example of the **static local entities** approach is the virtual machine Maté (Levis & Culler, 2002), that organizes programs into small code capsules and presents a process based interface to the user application. The application code is processed on the local node, and the state of the application can not migrate on different nodes. On the other hand, the virtual machine can execute new programs that are received from the network.
The third category, **mobile local entities**, is based on programmable entities being not in a static relation with a set of real sensors. Approaches of this category consider soft entities that can migrate from sensor to sensor, moving their state with themselves. This category mainly features *mobile agent* middleware, like Agilla (Fok et Al., 2009). This middleware is based on a set of agents, that own a state and have a program flow. The agents, while executing their code, can clone and migrate to other nodes. In particular, clone and migrate operations can have a strong or weak semantics. Weak semantics means that only the code is transferred or cloned to the new node, while strong semantics means that the code and the application's state are migrated/cloned. Thus, the agent execution resumes from where it left off. This taxonomy does not consider Impala (Liu & Martonosi, 2003) as a **mobile local entities** approach, since it only enables code updates but not state migration. The middleware Envirotrack (Abdelzaher et Al., 2004), on the other hand, fits in this category of **mobile local entities**. The goal of Envirotrack is tracking objects, like a fire or a noise emitter, and the set of sensors that are sensing the event create a group to locate the object. As the object moves, the set of sensors that belong to the group, and that are executing the program, changes to follow the object.
The rest of this section reports state-of-the-art middleware that implement a Programming Abstraction layer, and divides them into the three broad categories that were described in the first part of the section, and that are summarized into Figure 4.

4.1 Global Entity

The middleware belonging to this category offer a view of the WSN as a single, centralized element that allows the developer to abstract the low-level details. The drawback is that the developer has less control on resource usage and on the algorithms used for routing and data management.

Databases

The middleware in this category model the whole WSN as a distributed database system. The user formulates data requests using a SQL-like query language, that includes syntax for

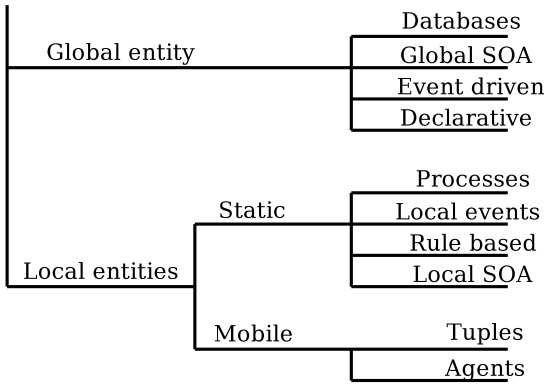


Fig. 4. Programming Abstraction layers.

specifying sample rates as well as query duration. The high level query is translated into a set of data acquisition, data processing and data transfer operations that are carried out by the nodes in the WSN. Query optimization evaluates the various alternatives of task allocation over the sensors, to choose the one that minimizes energy consumption. Examples of the Database approach are TinyDB (Madden et Al., 2003), Cougar (Yao & Gehrke, 2002), MaD-WiSe (Amato et Al., 2010), SINA (Shen et Al., 2001), and Senceive (Hermann & Dargie, 2008). TinyDB (Madden et Al., 2003), Cougar (Yao & Gehrke, 2002) and MaD-WiSe (Amato et Al., 2010) are all based on a pure database paradigm. They essentially provide a distributed database solution tailored on resource-constrained sensor networks, focusing on efficient query routing and processing. TinyDB (Madden et Al., 2003) uses a SQL-like language with extensions for query duration and sample rates. Queries are expressed over a single `sensors` table that represents all WSN sampled data. Moreover, TinyDB supports spatial aggregation operators and data filtering. Query dissemination is done via Semantic Routing Trees (SRTs), that are routing trees built from the sink.

Cougar (Yao & Gehrke, 2002) shares a number of features with TinyDB. Nodes are modelled as Abstract Data Types (ADTs), and the queries can be addressed toward either single nodes or sets of sensors that satisfying a particular condition, like the physical location of the sensors.

MaD-WiSe (Amato et Al., 2010) is a distributed database system that supports in-network query processing, and query optimization is performed on streams that abstract data sampling, by means of transformation rules based on heuristics that consider query execution plans. Query processing is based on streams that abstract data channels between operators of a query algebra and drive their pipelined behavior (computation and aggregation is carried out on flowing records with almost no need of storage). Operators include selections, projections, spatial aggregates as well as unions and joins. Currently, the ability to perform joins between streams is unique to MaD-WiSe and permits in-network processing of data obtained from different sources.

SINA (Shen et Al., 2001) uses an attribute-based naming scheme in order to facilitate the data-centric characteristics of sensor queries and it allows hierarchical clustering of nodes in order to facilitate scalable operations within sensor networks. The middleware design is based on the creation of clusters of nodes, that cooperate between themselves to orchestrate sensing tasks. The WSN as a whole is considered a collection of logical datasheets. Each cluster of

nodes is related to a datasheet, that is made up of cells, each of them representing a sensor attribute, that can be a single value or a time series. Each cell is unique, and each sensor maintains the whole datasheet. The SQL-like primitives of SINA can be used to issue queries in sensor networks. However, SINA does not provide schemes to hide the faulty nature of both sensor operations and wireless communication, leaving to the application layer the responsibility to provide robustness and reliability for data services.
Senceive (Hermann & Dargie, 2008) is similar to the previous approaches, but based on a graphical interface to define the operations to be performed for data gathering, in terms of SQL-like queries. The WSN is accessed from a special server, that is also a sink to the WSN. Query processing is performed combining all the queries that are active on a sensor. The resulting command is sent to the sensor to retrieve data, and the same command can be sent to a set of nodes using multicast of broadcast routing to save bandwidth and energy. Data are routed towards the data sink using routing trees similar to Directed Diffusion ones (Intanagonwiwat et Al., 2000). Data Storage is realized running a MySQL instance on the server, that is also the access point to the WSN. Data are then stored and processed on the server, and then delivered to the application. The database is also used to store the configuration for the WSN and the middleware.

Global Service Oriented Architectures

Middleware offering Global Service Oriented Architectures model the WSN as a single server that offers a set of services to the application programmers.
In particular, MiSense (Khedo & Subramanian, 2009) is a service-oriented component-based middleware that supports distributed sensor applications with various performance requirements. MiSense copes with application complexity by imposing a structure on top of the component model in the form of composability restrictions and by offering service-specific interfaces to the rest of the system. MiSense breaks up the middleware design into self-contained, interacting components in order to resolve the tension between the optimization requirements for specific scenarios and the need for flexibility and reusability for developing energy efficient WSN applications. The layered approach allows programmer to use different levels of abstraction during application design, and the upper layer of the middleware, the Domain layer, models the WSN as a single server that offers a set of services to the application programmer, and allows the programmer to address the WSN by abstracting the low-level details. The middleware is in charge of taking decisions on communication protocols, network topology organization, sensor operation modes and other functions typical of WSNs, to adapt the middleware to network changes.

Event Driven Global Programming

Abstract Task Graphs (Bakshi et Al., 2005; Mottola et Al., 2007) (ATaG) is a middleware that offers a dataflow programming model with a graphical composition language. It is based on a data-driven program flow and a mixed imperative-declarative specification. It allows developers to declare graphically the data flow and connectivity of virtual tasks, and to specify the functionality of tasks using an imperative language. The application developer produces the declarative part of the ATaG program using a GUI and a description of the target deployment in the form of an annotated network graph (ANG). A code generator analyzes the ATaG program, determines the I/O dependencies between tasks and data objects, and generates code templates for the abstract tasks and data. The programmer populates the code templates with application-specific code. The compiler then interprets the program annotations in the context

of the ANG, and generates configuration files for each node to customize the behavior of that node based on its role in the system. Finally, compile-ready code is generated for each node in the network.
MagnetOS (Barr et Al., 2002) is a distributed, power-aware, adaptive operating system, that targets ad hoc and sensor networks. This middleware implements a virtual machine that considers network-wide energy management as the primary concern in WSNs, and it provides a unified single system image of a Java virtual machine across the nodes that comprise a WSN. MagnetOS optimizes energy consumption, avoids hotspots and increases system longevity by transparently partitioning applications into components and dynamically placing these components on nodes within the WSN. Invocation of methods rely on RMI, and routing is based on the AODV protocol (Perkins & Royer, 1999).
MacroLab (Hnat et Al., 2008) lets the user write a single system-wide program in a C-like language, such that the program manages and processes the data as they are collected from the environment. Then the system generates a number of different versions of the programs, from completely centralized (using an external storage Data Management layer) to completely distributed (using a local storage Data Management layer). A cost analyzer computes which version of the program is the most energy efficient and deploys it to the sensors. During program execution, information about the energy cost of the current deployment is constantly collected. Should the system find out that another version of the program would be more beneficial, it will deploy it to the sensors on the run to enhance the WSN efficiency.

Declarative Systems

The Regiment (Newton et Al., 2007) system consists of a high-level language for WSN programming, and of a compiler that translates a global program into a node-level code. Regiment allows the programmer to look at the WSN as a set of spatially-distributed data streams, that may be defined by topological or geographic relationships between nodes. The middleware provides primitives for in-network data processing and region manipulation. In particular, Regiment calls *deglobalization* the process executed by its compiler, that transforms the network-wide representation of the program into a node-level, event-driven program. The process maps region operation into associated spanning trees that establish region membership and permit in-network data aggregation.
The Smart Messages (Borcea et Al., 2004) middleware addresses high-end sensors equipped with several MBs of memory, and it enables the programmer to reason in terms of Spatial Programming (SP), a space aware programming models, that is used to program an unkown number of volatile embedded systems in order to execute a user-defined application in a certain geographical area. The SP runtime system maintains a mapping between spatial references and the nodes they refer to. The mapping concerning each application is kept into a table, that is persistent during the application execution. Smart Messages (SMs) are actually migratory execution units, with code and data sections, and a lightweight execution stack. The SP program is translated into a SM program, then the nodes cooperate to support the SM execution by providing virtual machines.

4.2 Local Entities

The middleware that feature the **Local Entities** approach, offer to the user application a system composed by a number of interacting entities, and mechanisms to orchestrate their interactions to pursue the application goal.

The advantage of this approach with respect to the **Global Entity** one is that it provides a higher degree of resource control to the developer, and its disadvantage is that the application developer copes with more complexity, since the developer is allowed a glance at the underlying WSN structure.

4.2.1 Static Local Entities

The middleware that offer a Programming Abstraction layer of the **Static Local Entities** kind, adopt the traditional view of considering either a single sensor or a set of sensors as the recipient of the program. The sets are defined at application startup, and the state of the single applications that run on the entities can not migrate to different entities.

Process based

These middleware allow the developers to write applications in separate, small modules. The system injects and distributes the modules throughout the network using tailored algorithms, aiming at minimizing overall energy consumption and resource use.
Solutions in this category include Maté (Levis & Culler, 2002), ASVM (Levis et Al., 2005) and DAViM (Michiels et Al., 2006) that offer explicitly a virtual machine to the user application for the program execution. For example, Maté (Levis & Culler, 2002) is a byte code interpreter that runs on TinyOS. The user code of the application is broken into capsules of 24 byte-long instructions. Each capsule comprises a version number for its code, and the capsules are disseminated throughout the network such that every time a sensor receives a newer version of a capsule, the contained code is saved and then the capsule is forwarded to the sensor's neighbors. Maté does not have to buffer packets nor to store large data because it uses a synchronous model that begins execution in response to an event such as a packet transmission or a timeout. The synchronous model makes application-level programming simpler and less prone to bugs than dealing with asynchronous event notifications, but it limits the expressiveness of the programming model.
Another Process based approach is given by Contiki (Dunkels et Al., 2004), that is a lightweight operating system that supports dynamic loading and replacement of individual programs and services. Contiki is considered a Process based approach since, even though it is built around an event-driven kernel, it also provides preemptive multi-threading.Contiki is implemented in C and it has been ported to a number of micro-controller architectures. This operating system includes mechanisms to reduce energy consumption, and the total size of compiled code fits in 4KB RAM. Contiki has the ability to load and unload individual programs at run-time, and its programs use native code, and can therefore implement low level device drivers without loss of execution efficiency.
Another middleware of the Process based kind is MiLAN (Heinzelman, 2004), that lets programmers to fine-tune the network by setting QoS parameters on the basis of application requirements, that are set through a standard API. The benefits that can be drawn from MiLAN are here considered like a support to the operating system, helping the application to manage low-level mechanisms.
Impala (Liu & Martonosi, 2003) is a middleware designed to be used in the ZebraNet project, that aims at implementing surveillance systems for wildlife environments. Impala novelty relies in its approach of updating at rutime the application that is being executed on the sensors. Applications are modular in order to enable small updates that require little power during transmission. Even though Impala has been defined in a number of surveys as a "Mobile

Agents" approach, the only migration that can happen is about the code being updated with a new program, hence this work considers it a **static local entities** approach.

Event-based programming

Another approach to WSN middleware is based on the notion of events. There, the application specifies interest in certain state changes of the real world (basic events). Upon detecting such an event, the middleware sends a so-called event notification towards interested applications. The application can also specify certain patterns of events (compound events), such that the application is only notified if occurred events match these patterns. In (Yoneki & Bacon, 2005), a reasonably sophisticated set of event operators for describing event patterns in sensor networks has been produced. A crucial limitation of this solution is the complexity that is involved in implementing user applications.

DSWare (Li et Al., 2003-1) provides data-centric and group-based services for sensor networks. A realtime service handles the individual sensor reports, computing correlations among different sensor observations, with the goal of correctly capturing the characteristics of occurred events. The event service supports confidence functions which are designed on data semantics, including relative importance of sub-events and historical patterns. The event service enables partial detection of critical events, and it can also be used to differentiate between the occurrences of events and false alarms. Data are cached into the network using Data Centric Storage (see Subsection 3.2) and an SQL-like script language is used to subscribe events from a set of sensors.

Abstract Regions (Welsh & Mainland, 2004) is a middleware composed by a family of spatial operators that capture local communication within the regions of the network, which may be defined in terms of radio connectivity, geographic location, or other properties of nodes. Abstract Regions provides interfaces for identifying neighboring nodes, sharing data among neighbors, and performing reductions on shared variables. In addition, Abstract Regions exposes the trade-off between the accuracy and resource usage of communication operations. Applications can adapt to changing network conditions by tuning the energy and bandwidth usage of the underlying communication substrate. On the other hand, the group identity is static and set at application start-up and hence this approach can not perform state migration between nodes.

SensorWare (Boulis et Al., 2003) is a middleware that offers good flexibility to the development, at the expense of increased responsibility for the programmer. SensorWare provides a language model to implement distributed algorithms, providing a way to share the resources of a node among many applications and users that might concurrently use the distributed algorithm. The WSN is viewed as executing a set of collaborating programs in a corresponding set of nodes. The sensing, communication, and signal-processing resources of a node are exposed to the control scripts that orchestrate the dataflows to assemble custom protocols and signal processing stacks. SensorWare is also responsible for the dynamic deployment of the distributed algorithms into the WSN, dynamically programming the nodes. The approach of SensorWare is to allow nodes to program their peers, so that the user does not have to worry about deploying the distributed algorithm (because the information on how the algorithm unfolds lies within the algorithm), and the nodes save communication energy because they interact with their immediate neighbors and not with the sink through multi-hop routes.

The Mires middleware (Souto et Al., 2004) is a more pragmatic publish/subscribe solution that has been designed and implemented on top of TinyOS using nesC. It lets the applications specify interests in certain state changes of the real world. Upon detecting such an

event, a node sends a so-called event notification towards interested applications. Mires adopts a component-based programming model using active messages to implement its publish/subscribe-based communication infrastructure.
TinyOS (Levis et Al., 2004) is one of the most popular operating systems for networked embedded devices. It is component-based, event-driven and highly configurable, and it does not provide dynamic memory allocation. The programming model of TinyOS is based on the concepts of tasks, events and commands. The task is a low priority code piece that is run while the processor is not requested by event handlers. Components communicate via commands, sent to lower level components, and events, raised to upper level components. Events can preempt tasks and other events. This concurrency mechanism is inherited from the nesC language, used to implement the operating system. A TinyOS application is composed by a component definition file (`.comp`) and a wiring description file (`.desc`). The wiring description file defines dependencies between components through the channel interface connecting components.
Marwis (Wagenknecht, 2008) is a middleware based on Contiki (Dunkels et Al., 2004) (executed on the sensors) and Linux (executed on computers managing the WSN), that aims at managing WSNs composed by different kinds of sensors. Sensors are divided into smaller sensor subnetworks (SSNs), each containing only sensors of one type, then a wireless mesh network (WMN) operates as a backbone for the SSNs and as a gateway to the WSNs. A code updater running on the sensors takes care of code replacement, and some sensors called mesh nodes (MNs) contain a gateway to the WSN and a database of the sensors' status and sensed data, and are used to export the sensed data to the external user applications.

Rule-based systems

A rule-based system considers the application as composed by a program that has to be run on a node, and that is executed whenever a condition is verified. The middleware FACTS (Tergloth et Al., 2006), for example, is both event-driven and rule-based, and it combines these paradigms to perform energy saving. The same application is deployed to all the nodes, and it comprises a set of actions and a set of conditions (rules) for the actions to be executed. All data are defined as "facts", and the rules consist of a combination of a predicate over these facts and an action. The action is triggered by the rule engine whenever the predicate becomes true.
Escape (Russello et Al., 2008) is a framework that is used to simplify application development and deployment, and it promotes the reuse of code. The framework is component-based and it is aimed at the development of sense-and-react applications that combine the use of sensors and actuators. The central component of the framework is a publish/subscribe service broker that manages subscriptions, and that generates data routing towards the data subscriber. The novelty of the approach is the orchestration of two more components, the *service layer* and the *policy manager*. The service layer offers all the services that are orthogonal to the publish/subscribe system, like data collection, routing, and encryption. The policies are enforced on the services offered, to ensure the correct behavior of the middleware. Policies can be used to specify which actions are to be associated with the broker operation, and to coordinate sensors and actuators' operations.

Service Architectures for Static Local Entities

A different approach to WSN middleware is given by the ZigBee standard (Baronti et Al., 2007; ZigBee, 2005). This is a short-range multi-hop wireless protocol constructed over IEEE

802.15.4. At the Network layer it features an inherently node centric behavior, but it offers service-oriented mechanisms at the application level. Since it offers very general services, it does not deal with data management and collection, and it has a high degree of complexity and a big footprint. The ZigBee specification includes mechanisms aimed at limiting the sensors duty cycle, which however are configurable at network creation and that can not be adapted dynamically. ZigBee defines a framework under which the programmers develop applications in terms of Application Objects (APO). Each ZigBee device can host up to 240 APOs, which exploit the services offered by ZigBee which include data transmission, binding, discovery services, and security services. Each APO in the network is uniquely identified by combining its endpoint address and the network address of the hosting device. In the most simple setting, an APO consists of a limited set of attributes which can be accessed from remote APOs using simple *get*, *set*, and *event* transactions. An application profile is the specification in a standard format of the behavior of an application, whose execution may involve several ZigBee devices. An application profile describes a set of devices and clusters and defines the kind of data service. The basic services offered by ZigBee are device and service discovery, binding of devices, network management functions to manage connections/disconnections in a ZigBee network, and security management at network level and device level.

SMEPP Light (Vairo et Al., 2008) is written in NesC and runs on top of TinyOS, and it was inspired by the European Project SMEPP (Albano et Al., 2007-1; SMEPP, 2010), that aimed at creating a secure, service-oriented middleware for embedded peer-to-peer systems. SMEPP Light is a version of SMEPP that is tailored for WSNs, and it supports a subset of SMEPP's primitives. In particular, SMEPP Light does not support full-fledged services, but on the other hand it organizes the sensors into groups and provides eventing mechanisms, based on the directed diffusion paradigm (Intanagonwiwat et Al., 2000), for query dissemination and data collection. A sensor requests events from sensors belonging to the same group and creates a routing tree rooted in the subscriber. Two levels of security are provided. Network security uses two keys that are set at compile time, while group level security uses three keys, and it is based on a masterKey. The masterKey is known in advance by all the sensors that can get into the group, and it is used to restrict the access to the group. For energy management purposes, each group defines a duty cycle that imposes to each node a period of activity followed by a period of inactivity, and each data subscription can add to the activity periods, prescribing to some nodes to sample data from the environment and to relay the data events.

The service approach to WSN was developed also in the direction of web services. Open Sensor Web Architecture (OSWA) (Chu & Buyya, 2007) aims at making various types of web-resident sensors and instruments, discoverable, accessible and controllable via the World Wide Web. The novelty of this approach resides in the efforts that have been made in overcoming the obstacles related to the heterogeneity of the different sensors and instruments that were targeted by sensor web projects.

RESTful (Yazar & Dunkels, 2009) is a web service based middleware, that lets the developer interact with a REST based web service when querying a node. The middleware bases its energy-serving strategies on the X-MAC (Buettner et Al., 2006) protocol, modified to be session-aware. Multi-hop communication is implemented at application level, using a REST call on each communication hop.

4.2.2 Mobile Local Entities

The middleware in the **mobile local entities** category do not focus the programmer's attention on physical nodes or on the whole WSN. Instead, they consider virtual nodes as the target

of the programs, so that the actual node that is executing the data collection or the data processing algorithm is changed at runtime. This paradigm is an advanced kind of node coordination, where the identity of the computation is not anymore in the sensor that is performing the task, but it is instead a virtual entity that is associated temporarily with one sensor or a set of sensors, thus it can change over time to follow the application logic or a physical event. This last kind of execution is aimed at applications performing target tracking.

Tuple Spaces

The coordination needed in WSNs has attracted the attention of the Coordination paradigm community (Carriero & Gelernter, 2005). More specifically, different coordination models and middleware based on the Linda abstract model (Gelernter, 1985) have appeared in the area of WSNs. Linda can be considered one of the most representative coordination languages. It is based on a shared memory model where data are represented by elementary data structures called tuples, and the memory is a multiset of tuples and takes the name of "*tuple space*". Examples of this class of middleware are TinyLime (Curino et Al., 2005) and TeenyLime (Costa et Al., 2006). For example, in TinyLime, a new operational scenario is assumed, with the goal of providing contextual information, not requiring multi-hop communication among sensors, placing reasonable computation and communication demands on the sensors. Sensors are sparsely distributed in the environment, not necessarily able to communicate with each other, and a set of mobile base stations (laptops) move through space accessing the data of sensors nearby. Each base station owns a tuple space and federated tuple spaces can be established in order to communicate and synchronize several base stations.

Tuple Channels

An alternative to tuple spaces is the proposal based on the use of tuple channels (Díaz et Al., 1997) in order to carry out communication and synchronization among the involved WSN nodes. A tuple channel is a FIFO structure that allows one-to-many and many-to-one communication of data structures, represented by tuples. Several advantages can be obtained from the use of channels with respect to shared memory models:

1. Architectural expressiveness: like messaging, using channels to express the communication carried out within a distributed system is more expressive than using shared data spaces, since with a shared data space it is difficult to identify which components exchange data with each other.
2. Channels support data streams in a natural way: the application programmer does not have to deal with head and tail tuples as is necessary in a tuple space based approach to implement information streams.
3. Channel interconnection provides great flexibility for the definition of complex and dynamic interaction protocols: sensor data dissemination can be achieved elegantly, allowing for data redirection, data aggregation and redundant data elimination.

A representative of Tuple Channels middleware is TCMote (Díaz et Al., 2005). This middleware is designed to support an operational setting based on a hierarchical architecture of sensing regions, each one governed by a *region leader* with higher capabilities (power, memory, processing ability) than the rest of the region's sensors. A region leader owns a tuple channel space, which stores tuple channels used to carry out communication and synchronization between the region's sensors and the leader in a single-hop way. Data is consumed when

moved through the tuple channels, contributing to dealing with the data-centric characteristics of sensor queries. In addition, tuple channels can be dynamically interconnected through the use of predefined and user-defined connectors, to define new topologies.

Mobile Agents

In the traditional client/server-based computing architecture, data produced by multiple sources is transferred to a destination, whereas in the mobile agent based computing paradigm, a task-specific executable code traverses the relevant sources to gather the data. Mobile agents can be used to reduce the communication cost, by moving the processing function to the data rather than bringing the data to a central node.

Recently, mobile agents have been proposed for efficient data dissemination in WSNs. Some proposals are Agilla (Fok et Al., 2009), MAWSN (Chen et Al., 2006) and actorNet (Kwon et Al., 2006). We discuss the first one as the representative of this category of middleware. Agilla facilitates the rapid deployment of adaptive applications in WSNs by allowing the programmer to create and inject mobile agents, which can migrate across the WSN performing application-specific tasks. Mobile agents can move or clone themselves to desired locations in response to changes in the conditions of the environment. Each node maintains a local tuple space (in fact, can also be considered of the "tuple space" category), and different agents can coordinate through local or remote operations on these tuple spaces. Code allocation is performed using the tuple spaces, allowing an agent to tell Agilla that it is interested in tuples matching a particular template.

Agent Factory Micro Edition (AFME) (Muldoon et Al., 2006) is a middleware featuring the mobile agent approach, that is a version of Agent Factory (O'Hare, 1996) middleware for computationally constrained devices.AFME runs on top of Java 2 Micro Edition (J2ME) and it implements a framework where mobile agents operate under the BDI (Belief-Desire-Intention) paradigm to perform decisions, and where the mobile agents can migrate between devices of different capabilities, for example between personal computers and sensors. Agent design is decoupled into core behaviors, that are constant characteristics of the agent, and platform dependent behaviors, that are changed every time the agent migrates between different devices. Agent communication is agnostic, in the sense that an agent interacts without directly referencing the device of its peer, hence it does not have to know in advance if its peer is running on a personal computer or a sensor. When an agent is created, it is assigned an unique identifier, then communication is addressed by means of the unique identifier, that is resolved to an agent and then to a device, to forward the message appropriately.

Envirotrack (Abdelzaher et Al., 2004) is an object-oriented middleware that aims at providing an interface to the application programmer geared towards tracking the physical environment. Sensors which detect certain user-defined objects in the physical environment form groups, one around each object. A network abstraction layer associates a context label with each such group to represent the corresponding tracked object in the computing system. A context label is the logical address of a virtual host which follows the tracked object in the physical environment. Programs can be attached to context labels to perform context-specific computation. The programs are executed on the sensor group of the context label.

Aware (Gil et Al., 2007) is similar to Envirotrack (Abdelzaher et Al., 2004), since it has the same goal of supporting tracking applications, but it aims also to provide seamless communication between a network of entities with high capabilities (computers and robots, linked by Ethernet and 802.11) and the WSN. Aware's basic premise is to divide sensors into groups that are located around a certain environmental condition, that characterize the physical event to be

tracked. The conditions that define the physical event are distributed epidemically in the WSN, then each group of sensors elects a group leader. The system supports the definition of multiple copies of the same physical event (two fires burning at the same time) and it allows the physical event to join (the fires joined into one big fire) and to split (the main fire ignited another fire down a hill) and the sensor group identifiers are managed accordingly to stay consistent with the semantics of the group.

5. Dependability mechanisms

Dependability mechanisms in WSNs are becoming increasingly important, since they are useful for different goals, ranging from network coding (Alon et Al., 2000) to in-network data storage (Kamra et Al., 2006). For example, Data Management layers based on DCS are agnostic to the way that the nodes actually encode the data to perform their storage.

Every kind of dependable data storage must be based on some kind of redundancy on the data that are stored, to be able to reconstruct the data if/when some nodes fail. Two representatives of Dependability mechanisms are the pure replication and the *erasure coding* of data.

Pure replication: Most current approaches adopt pure replication, that implies the replication of the whole data, sometimes in conjunction with the deployment of the copies in regions of the network that are far away (Ratnasamy et Al., 2003), to maximize the lifetime of the data in front of destructive events.

In this kind of scenario, a useful approach to improve the efficiency of data management is the generation of Index Systems (Ganesan et Al., 2005) to manage a small quantity of data at a time.

Erasure coding: Beginning with the work of Shannon (Shannon, 1948) in 1948, a number of redundancy techniques have been designed and employed in very different areas (CD, storage (Alon et Al., 2000), etc).

Erasure coding (Barsi & Maestrini, 1973) consists in encoding a datum into a set of redundant fragments that guarantees the survival of the datum in front of the loss (erasure) of a limited number of fragments. In particular, given a datum d and m keys, the *n out of m* coding of d consists in m fragments (one for each key), with the property that d can be reconstructed from any subset of n fragments, provided the keys used to construct the fragments are known. These codes exploit a set of $m = n + r$ keys to encode a datum d of size L symbols into a set of m fragments of size $\sim \frac{L}{n}$, with the property that d can be reconstructed if up to r fragments are lost and up to $\lfloor \frac{r-e}{2} \rfloor$ fragments are corrupted. Examples of erasure codes are Reed Solomon codes (Plank, 1997), and RNNS (Barsi & Maestrini, 1973).

In (Rodrigues & Liskov, 2005), the authors studied the use of erasure codes in peer-to-peer networks with frequent changes in peers' membership. Previous comparisons (Weatherspoon & Kubiatowicz, 2002) mostly argue that erasure coding is the clear victor, due to huge storage savings for the same availability levels (or conversely, huge availability gains for the same storage levels). The work of Liskov et al, on the other hand, argues that while gains from coding exist, they are highly dependent on the characteristics of the nodes that comprise the overlay. In fact, when a peer leaves the network the fragments it stores are lost, and to reconstruct them (in order to restore the desired level of redundancy) it is necessary first to reconstruct the original data by reading a given number of available fragments. This and the extra complexity can out-weight the benefits of erasure codes in terms of data availability.

Nonetheless, these techniques can lead to improvements in various aspects of WSNs. First of all, they reduce the storage overhead for DCS (Dimakis et Al., 2006), to reduce transport costs for the datum (Albano & Gao, 2010), they increase the robustness of the system because

the system can use the redundancy properties of the erasure coding to recover the datum if a packet gets lost, without having to ask it again to the WSN (Albano & Chessa, 2009-2).

6. Conclusions

This paper considers the issues that arise when designing a middleware for WSNs, and it reviews the state of the art on solutions and basic technologies by means of a layered view. In this work, the solutions representative of the different layers are organized into a taxonomy. More specifically, the upper layer, also called the Programming Abstraction layer, has the most complex structure since it encapsulates many functions aimed at very different goals. At the bottom layer, namely the Data Management layer, this survey presents the low level mechanisms enabling the functionalities of the higher layer. Specifically we placed at this layer the data storage mechanisms. Finally, a discussion of the dependability mechanisms concludes the chapter.
The authors of this survey are positive that this work will be useful to future middleware developers, since decomposing a middleware in a coherent way can help to cope with the complexity that naturally arises when designing a complex system.

7. References

T. Abdelzaher , B. Blum , Q. Cao , Y. Chen , D. Evans , J. George , S. George , L. Gu , T. He , S. Krishnamurthy , L. Luo , S. Son , J. Stankovic , R. Stoleru , A. Wood: EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), p.582-589, March 24-26, 2004

I.F. Akyildiz et Al.: A survey on sensor networks. In:IEEE Communications Magazine vol. 40, n. 8 (2002)

M. Albano, A. Brogi, R. Popescu, M. Diaz, A. Dianes: Towards Secure Middleware for Embedded Peer-to-Peer Systems: Ob jectives & Requirements. In: RSPSI 2007, Innsbruck, Austria

M. Albano, S. Chessa, F. Nidito, S. Pelagatti: Q-NiGHT: Adding QoS to Data Centric Storage in Non-Uniform Sensor Networks. In: IEEE MDM 2007, Mannheim, Germany (2007)

M. Albano, S. Chessa, F. Nidito, S. Pelagatti: Dealing with Non Uniformity in Data Centric Storage for Wireless Sensor Networks. In: to appear on IEEE Trans. on Parallel and Distributed Systems (2010)

M. Albano, S. Chessa: Publish/subscribe in Wireless Sensor Networks based on Data Centric Storage. In: CAMS 2009, workshop of COMSWARE 2009, Dublin, Ireland, ISBN 978-1-60558-525-3

M. Albano, S. Chessa: Distributed Erasure Coding in Data Centric Storage for Wireless Sensor Networks. In: IEEE ISCC '09, Sousse, Tunisia, pp.6, 5-8 July 2009

M. Albano, J. Gao: In-Network Coding for Resilient Sensor Data Storage and Efficient Data Mule Collection. In: Proceeding of Int. Workshop on Algorithms for Sensor Systems (Algosensors 2010), Bordeaux, July 5th, 2010

N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. P. Stern: Scalable secure storage when half the system is faulty. In: Automata, Languages and Programming, pages 576-587 (2000)

M. Aly, K. Pruhs, P.K. Chrysanthis: Kddcs: a load-balanced in-network data-centric storage scheme for sensor networks. In: Proceedings of CIKM, p. 317-326 (2006)

G. Amato, S. Chessa, and C. Vairo: MaD-WiSe: A Distributed Stream Management System for Wireless Sensor Networks. In: Software Practice & Experience, 40 (5): 431-451 (2010)

F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, C. Mitidieri: CHR: a Distributed Hash Table for Wireless Ad Hoc Networks. In: Proc. of the 25th IEEE ICDCSW'05 (2005)

L.S. Bai, R.P. Dick, P.A. Dinda: Archetype-based design: Sensor network programming for application experts, not just programming experts. In: ACM IPSN 2009, p. 85-95

A. Bakshi, V.K. Prasanna, J. Reich, D. Larner: The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems. In: Proceedings of International Conference on Mobile Systems, Applications, and Services, p. 19-24, June 2005

P. Baronti et Al.: Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards. In: Computer Communications, 30 (7) 1655-1695 (2007)

R. Barr, J.C. Bicket, D.S. Dantas, B. Du, T.W.D. Kim, Danny, B. Zhou, E.G. Sirer: On the need for system-level support for ad hoc and sensor networks. In: ACM SIGOPS Operating Systems Review, vol. 36, n. 2, April 2002

F. Barsi and P. Maestrini: Error Correcting Properties of Redundant Residue Number Systems. In: IEEE Transactions on Computers, Vol. C-22 (3) 307-315 (1973)

C. Borcea, C. Intanagonwiwat, P. Kang, U. Kremer, and L. Iftode: Spatial programming using smart messages: Design and implementation. In: Proceedings the 24th International Conference on Distributed Computing Systems (ICDCS 2004), March 2004

A. Boulis, C.C. Han, and M.B. Srivastava: Design and implementation of a framework for efficient and programmable sensor networks. In: MobiSys '03: Proceedings of the 1st ACM international conference on Mobile systems, applications and services (2003)

J. Bruck, J. Gao, and A. Jiang: MAP: Medial axis based geometric routing in sensor networks. In: ACM/IEEE MobiCom, p. 88-102 (2005)

M. Buettner, V. Yee, E. Anderson, and R. Han: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: SenSys 2006.

N. Carriero, D. Gelernter: Coordination Languages and their Significance. In: Communications of the ACM, 35(2):97–107, 1992. March 2005.

M. Chen, T. Kwon, Y. Yuan, V.C.M Leung: Mobile Agent Based Wireless Sensor Networks. In: Journal of Computers, 1(1):97 (2006)

S. Chessa: Sensor Network Standards. Book chapter in: Wireless Sensor Networks: A Networking Perspective, Wiley-IEEE, ISBN: 978-0-470-16763-2, p. 407-431, September 2009

X. Chu, R. Buyya: Service Oriented Sensor Web. In: Sensor Networks and Configuration: Fundamentals, Standards, Platforms and Applications, Springer-Verlag, pp. 51-74 (2007)

P. Costa, L. Mottola, A.L. Murphy, G.P. Picco: TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In: Proceedings of the 1st International Workshop on Middleware for Sensor Networks (MidSens' 06), Melbourne, Australia, November 2006

C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, G.P. Picco: Mobile data collection in Sensor Networks: the TinyLime Middleware. In: Pervasive and Mobile Computing, vol 1, n 4, December 2005, p. 446-469 (2005)

M. Díaz, B. Rubio, and J.M. Troya: The tuple channel coordination model. In: Proceedings of ICSE'97 workshop on Software Engineering for Parallel and Distributed Systems, p. 95-106, Boston, May 1997

M. Díaz, B. Rubio, J.M. Troya: A Coordination Middleware for Wireless Sensor Networks. In: Proceedings of the IEEE International Conference on Sensor Networks (SENET' 05), pages 377–382. Montreal, Canada. IEEE Computer Society Press, August 2005

A.G. Dimakis, V. Prabhakaran, and K. Ramchandran: Decentralized erasure codes for distributed networked storage. In: IEEE Transactions on Information Theory, vol. 52, n. 6, p. 2809-2816, June 2006

A. Dunkels, B. Gronvall, T. Voigt: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the First IEEE Workshop on Embedded Networked Sensors (2004)

C.T. Ee, S. Ratnasamy, and S. Shenker: Practical Data-Centric Storage. In: USENIX NSDI Conference, San Jose, CA, May 2006

C.-L. Fok, G.-C. Roman, C.Lu: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. In: ACM Trans. Auton. Adapt. Syst., vol. 4, n. 3, p. 1–26 (2009)

D. Ganesan and B. Greenstein and D. Estrin and J. Heidemann and R. Govindan: Multiresolution storage and search in sensor networks. In: Trans. Storage, vol. 1, n. 3, pag. 277-315 (2005)

D. Gelernter: Generative Communication in Linda. In: ACM Transactions on Programming Languages and Systems, 7(1), 80–112, 1985.

A. Ghose, J. Grossklags, and J. Chuang: Resilient data-centric storage in wireless sensor networks. In: IEEE Distributed Systems online, 4(11), November 2003

P. Gil, I. Maza, A. Ollero, P. Marrón: Data Centric Middleware for the Integration of Wireless Sensor Networks and Mobile Robots In: Proceedings of 7th Conference on Mobile Robots and Competitions, ROBOTICA 2007. April 2007

S. Hadim, N. Mohamed: Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks. In: 1st IEEE International Conference on Communication System Software and Middleware (Comsware 2006)

W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo: Middleware to Support Sensor Network Applications. In: IEEE Network Magazine Special Issue, pp. 6–14, January 2004.

C. Hermann and W. Dargie: Senceive: Middleware for a wireless sensor network. In: IEEE 22nd international conference on advanced information networking and applications (AINA 2008), GinoWan, Okinawa, Japan, March 24-28, 2008

T.W. Hnat, T.I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse: MacroLab: A Vector-based Macroprogramming Framework for Cyber-Physical Systems. In: Proc. of 6th ACM Conf. on Embedded Networked Sensor Systems (SenSys'08), Raleigh, NC, November 2008

C. Intanagonwiwat, R. Govindan, D. Estrin: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM '00), p. 56-67, Boston, August 2000.

A. Kamra, J. Feldman, V. Misra, and D. Rubenstein: Growth codes: Maximizing sensor network data persistence. In: Proceedings of ACM SIGCOMM'06, p. 255-266, Pisa, Italy (2006)

B. Karp, H.T. Kung: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In: MobiCom 2000, p. 243-254 (2000)

K.K. Khedo, and R.K. Subramanian: A Service-Oriented Component-Based Middleware Architecture for Wireless Sensor Networks. In: IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.3, March 2009

Y. Kwon, S. Sundresh, K. Mechitov, G. Agha: ActorNet: An Actor Platform for Wireless Sensor Networks. In: Proceedings of the IEEE International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, May 2006

P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler: TinyOS: An operating system for wireless sensor networks. In: Ambient Intelligence. Springer-Verlag, 2004.

P. Levis, D. Gay, D. Culler: Active Sensor Networks. In: Proceedings of the 2nh International Symposium on Networked Systems Design and Implementation (NSDI' 05), pages 29-42, San Francisco, CA, USA, March 2005.

P. Levis, D. Culler: Maté: A Tiny Virtual Machine for Sensor Networks. In: 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOSX' 02), San Jose, USA (2002)

S. Li, S. Son, and J. Stankovic: Event detection services using data service middleware in distributed sensor networks. In: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks, April 2003.

X. Li, Y. J. Kim, R. Govidan, and W. Hong: Multi-dimensional range queries in sensor networks. In: Proceedings of ACM SenSys (2003)

T. Liu, M. Martonosi: Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. In: ACM Symp. on Principles and practice of parallel programming (2003) p. 107-118

S. Madden et Al: The design of an acquisitional query processor for sensor networks. In: SIGMOD 2003, p. 491-502, San Diego, USA.

S. Michiels, W. Horré, W. Joosen, P. Verbaeten: DAViM: a Dynamically Adaptable Virtual Machine for Sensor Networks. In: Proceedings of the 1st International Workshop on Middleware for Sensor Networks (Mid-Sens' 06), co-located with the 7th International Middlewar e Conference (Middleware' 06), Melbourne, Australia, November, 2006.

L. Mottola, A. Pathak, A. Bakshi, V.K. Prasanna, and G.P. Picco: Expressing Sensor Network Interaction Patterns using Data-Driven Macroprogramming. In: Proceedings of the 3rd IEEE International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSens, colocated with IEEE PERCOM), New York (NY, USA), March 2007.

L. Mottola, G. Picco: Programming wireless sensor networks: Fundamentals concepts and state of the art. In: To appear in ACM Computing Surveys (2010)

C. Muldoon, G.M.P. O'Hare, R. Collier, M.J. O'Grady: Agent Factory Micro Edition: A Framework for Ambient Applications. In: Proceedings of Intelligent Agents in Computing Systems (IACS 2) Workshop of International Conference on Computational Science (ICCS), Reading, p. 727-734 (2006)

R. Newton, G. Morrisett, M. Welsh: The regiment macroprogramming system. In: Proceedings of International Symposium on Information Processing in Sensor Networks (IPSN 07), April 2007

G.M.P. O' Hare: Agent Factory: An Environment for the Fabrication of Multi-Agent Systems. In: G. M. P. O'Hare and N. R. Jennings (eds.), Foundations of Distributed Artificial Intelligence , Wiley Interscience, p. 449-484 (1996)

C.E. Perkins and E.M. Royer: Ad hoc On-Demand Distance Vector Routing. In: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, p. 90-100, February 1999

K. Piotrowski, P. Langendoerfer, and S. Peter: tinyDSM: A highly reliable cooperative data storage for Wireless Sensor Networks. In: International Symposium on Collaborative Technologies and Systems, Los Alamitos, USA, p. 225-232, May 2009

J.S. Plank: A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. In: Software: Practice and Experience, vol. 27, n. 9, p. 995-1012 (1997)

Md.A. Rahman: Middleware for wireless sensor networks: Challenges and Approaches. In: Seminar on Internetworking, Helsinki University of Technology, Finland, April 27th 2009.

S. Ratnasamy et Al.: Data-centric storage in sensornets with GHT, a geographic hash table. In: Mobile Networking Application (MONET), vol. 8, n. 4, p. 427-442 (2003)

R. Rodrigues and B. Liskov: High availability in DHTs: Erasure coding vs. replication. In: International Workshop on Peer-to-Peer Systems, Ithaca, NY (2005)

K. Romer, O. Kasten, F. Mattern: Middleware Challenges for Wireless Sensor Networks. In: ACM SIGMOBILE Mobile Computing and Communication Review (MC2R), 6(4):59-61, 2002.

B. Rubio, M. Díaz, and J. Troya: Programming Approaches and Challenges for Wireless Sensor Networks. In: IEEE Int. Conf. on Systems and Networks Communications (ICSNC'07), Cap Esterel, France (2007)

G. Russello, L. Mostarda, and N. Dulay. Escape: A component-based policy frame-work for sense and react applications. In: Proceedings of the 11th International Symposium on Component-Based Software Engineering, pages 212-229, 2008.

R. Sarkar, X. Zhu, J. Gao: Double Rulings for Information Brokerage in Sensor Networks. In: Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiCom06), Florence, Italy, May 2006

K. Seada and A. Helmy: Rendezvous Regions: A Scalable Architecture for Service Location and Data-Centric Storage in Large-Scale Wireless Networks. In: IEEE/ACM IPDPS 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN), Santa Fe, New Mexico, April (2004)

C. Shannon: A mathematical theory of communication. In: Bell Sys. Tech. Journal, 27:379-423, 1948

C-C. Shen, C. Srisathapornphat, C. Jaikaeo: Sensor Information Networking Architecture and Applications. In: IEEE Personal Communications:52-59, 2001

Secure Middleware for Embedded Peer-to-peer. In: http://www.smepp.org (final report published in 2010)

E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz: A Message-Oriented Middleware for Sensor Networks. In: 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2004), pages 127–134, Toronto, Canada, October 2004.

R. Sugihara, R.K. Gupta: Programming Models for Sensor Networks: A Survey. In: ACM Trans. on Sensor Networks, 4(2), March 2008, 1-28

K. Terfloth, G. Wittenburg, J. Schiller: FACTS: A Rule-based Middleware Architecture for Wireless Sensor Networks. In: Int. Conf. on Communication System Software and Middleware, New Delhi, India (2006)

S. Tong: An Evaluation Framework for middleware approaches on Wireless Sensor Networks. In: Seminar on Internetworking, Helsinki University of Technology, Finland, April 27th 2009.

C. Vairo, M. Albano, S. Chessa: A Secure Middleware for Wireless Sensor Networks. In: Proceedings of Middleware for Mobile Embedded Peer-to-Peer Systems (MIMES '08), workshop of Mobiquitous 2008, pp 1-6, Dublin, Ireland

G. Wagenknecht, M. Anwander, T. Braun, T. Staub, J. Matheka, S. Morgenthaler: MARWIS: A Management Architecture for Heterogeneous Wireless Sensor Networks. In: 6th International Conference on Wired/Wireless Internet Communications (WWIC'08), Vol. LCNS, Nr. 5031, p. 177-188, May 28 - 30 2008

M.M. Wang, J.N. Cao, J. Li, S.K. Dasi: Middleware for wireless sensor networks: A survey. In: Journal of Computer Science and Technology 23(3) (May 2008) 305-326

H. Weatherspoon, J.D. Kubiatowicz: Erasure Coding vs. Replication: A Quantitative Comparison. In: LNCS 2429, 1^{st} Int. Workshop on Peer-to-Peer Systems, p. 328 - 338 (2002)

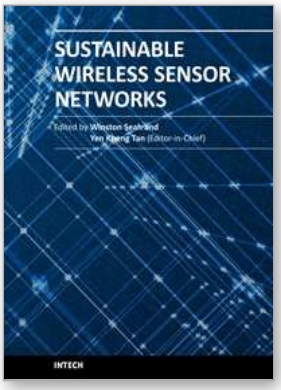
M. Welsh, G. Mainland: Programming Sensor Networks Using Abstract Regions. In: Proceedings of NSDI, 2004

Y. Yao, J. Gehrke: The Cougar Approach to In-Network Query Processing in Sensor Networks. In: SIGMOD Record vol. 31, n.3 (2002)

D. Yazar, A. Dunkels: Efficient Application Integration in IP-based Sensor Networks. In: Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings, Berkeley, CA, USA, November 2009

E. Yoneki, J. Bacon: Unified Semantics for Event Correlation over Time and Space in Hybrid Network Environments. In: IFIP International Conference on Cooperative Information Systems (CoopIS' 05), LNCS vol. 3760, pages 366-384. Springer 2005.

ZigBee Alliance: ZigBee Specification version 1.0, April 2005.



Sustainable Wireless Sensor Networks

Edited by Yen Kheng Tan

ISBN 978-953-307-297-5
Hard cover, 574 pages
Publisher InTech
Published online 14, December, 2010
Published in print edition December, 2010

Wireless Sensor Networks came into prominence around the start of this millennium motivated by the omnipresent scenario of small-sized sensors with limited power deployed in large numbers over an area to monitor different phenomenon. The sole motivation of a large portion of research efforts has been to maximize the lifetime of the network, where network lifetime is typically measured from the instant of deployment to the point when one of the nodes has expended its limited power source and becomes in-operational â€“ commonly referred as first node failure. Over the years, research has increasingly adopted ideas from wireless communications as well as embedded systems development in order to move this technology closer to realistic deployment scenarios. In such a rich research area as wireless sensor networks, it is difficult if not impossible to provide a comprehensive coverage of all relevant aspects. In this book, we hope to give the reader with a snapshot of some aspects of wireless sensor networks research that provides both a high level overview as well as detailed discussion on specific areas.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Michele Albano and Stefano Chessa (2010). Programming a Sensor Network in a layered middleware architecture, Sustainable Wireless Sensor Networks, Yen Kheng Tan (Ed.), ISBN: 978-953-307-297-5, InTech, Available from: http://www.intechopen.com/books/sustainable-wireless-sensor-networks/programming-a-sensor-network-in-a-layered-middleware-architecture

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821