# Towards Declarative Query Scoping in Sensor Networks

Daniel Jacobi[1], Pablo E. Guerrero[1], Khalid Nawaz[1],
Christian Seeger[1⋆], Arthur Herzog[1⋆], Kristof Van Laerhoven[2], and Ilia Petrov[1]

[1] Databases and Distributed Systems Group,
[2] Embedded Sensing Systems,
Dept. of Computer Science, Technische Universität Darmstadt, Germany

**Abstract.** In the last decade, several large-scale wireless sensor networks have been deployed to monitor a variety of environments. The declarative nature of the database approach for accessing sensor data has gained great popularity because of both its simplicity and its energy-efficient implementation. At the same time another declarative abstraction made its way into mainstream sensor network deployments: user-defined groups of nodes. By restricting the set of nodes that participate in a task to such a group, the overall network lifetime can be prolonged. It is straightforward to see that integrating these two approaches, that is, restricting a query's scope to a group of sensor nodes, is beneficial. In this work we explore the integration of two such database and scoping technologies: TikiDB, a modern reincarnation of a sensor network query processor, and Scopes, a network-wide grouping mechanism.

## 1 Introduction

Wireless Sensor Networks (WSNs) have been suggested as a potent solution to monitor large areas with sensing hardware, distributed over hundreds of nodes that jointly form an ad-hoc multi-hop network. Each node in such a WSN is equipped with a limited amount of processing and battery resources, communication capabilities to transmit its information to neighboring nodes in the network, and a set of sensors, which can observe the local environment.

Operating a sensor network originally implied writing code in a procedural programming language like C (or variants of it) for TinyOS [9]. This code has to deal with low-level issues such as interrupts, network unreliability and power consumption. In contrast, declarative operation is simpler than writing procedural code: it allows the user to focus on what needs to be done, without thinking how to achieve it, and thereby reduces the system complexity.

In sensor networks, declarative data access was investigated by Gehrke [1,19] and Madden [11,12]. Their systems offer a query processor-like interface to the

⋆ Supported by the German Research Foundation (DFG) within the research training group 1362 *Cooperative, Adaptive, and Responsive Monitoring in Mixed Mode Environments*, and the *Center for Advanced Security Research Darmstadt* (CASED).

sensor network. The vast amount of work in this context shows general consensus that sensor data access should be declarative.

Another abstraction that has made its way into mainstream sensor network deployments is node grouping [3]. Having its origin in event-based systems [6], the idea of scoping a WSN was published in [16], and related implementations appeared in [18,13,15]. Most of these systems do acknowledge the importance of a declarative operation by providing a simple syntax to define the node groups.

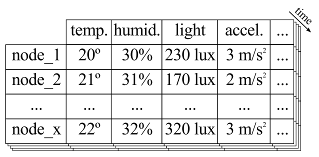
The intuitiveness of the database and scoping approaches makes their combination an ideal integrated solution for query management in sensor networks. To the best of our knowledge no work has explored this intersection. This paper presents a number of extensions that this integration enables, their implementation and evaluation.

The rest of the paper is structured as follows. In the next section we review related sensor network research with distinct attention to declarative query processing and management of node groups, describing specific components of the two approaches and their relevant implementation details. In Section 3, a number of extensions are presented that naturally emerge when using both systems together. An initial evaluation is provided in Section 4 studying the behavior of the system. We summarize our main findings in Section 5, together with future directions of research.

## 2 Related Work

A significant amount of work has been carried out in the last decade to address the topics of query processing and node group management. In the following two subsections we review these topics. We then describe techniques to constrain a query's span in the network which are, to a certain extent, comparable to the approach presented in this paper.

### 2.1 From Cougar to TikiDB



| | temp. | humid. | light | accel. | ... |
|---|---|---|---|---|---|
| node_1 | 20º | 30% | 230 lux | 3 m/s$^2$ | ... |
| node_2 | 21º | 31% | 170 lux | 2 m/s$^2$ | ... |
| ... | ... | ... | ... | ... | ... |
| node_x | 22º | 32% | 320 lux | 3 m/s$^2$ | ... |

Fig. 1: TinyDB's `sensors` virtual data cube

The earliest systems to provide a declarative interface for accessing sensor data were Cougar [1,19] and TinyDB [11,12]. In TinyDB, the sensor nodes compose a global, virtual table called `sensors`, which has sensor types as columns (e.g., temperature, humidity), and nodes as rows. Each record is virtually updated by each node at a frequency specified by the user, effectively forming a virtual data cube over time, as illustrated in Fig. 1.

[3] should not be confused with node clustering, a technique to subordinate nodes to a master according to their physical proximity

Users, in turn, specify a SQL-like query to extract sensor data from the network. Consider for example a sensor network equipped with environmental monitoring nodes. A user, e.g., interested in determining whether air humidity exceeds a threshold of 20%, provides the query to the right. The `SELECT` clause defines a projection on `humidity`, i.e., the result set consists of a table with one column and one row per network node. The `WHERE` clause reduces the set of results by filtering temperature tuples that don't fulfill the specified condition. The results are delivered to the base station at a 30-second sample rate (as specified with the `SAMPLE PERIOD` clause), and the query lifetime is 3 days.

```
SELECT humidity
   FROM sensors
   WHERE humidity > 20%
   SAMPLE PERIOD 30s
   FOR 3d
```

When users are interested in aggregated values instead of collecting all individual rows, the query can include the desired aggregation and the system performs the computation, as shown in the query to the right. The result of this query is a single row per sample period, based on the non-aggregated rows, with the average temperature. Performing the aggregation inside the network has the benefit of lowering the communication costs. TinyDB's data aggregation framework [11] implements this mechanism, and can be extended with customized functions beyond the traditional average, max and min functions.

```
SELECT AVG(temperature)
   FROM sensors
   SAMPLE PERIOD 30s
```

This style of declarative interaction, i.e. issuing queries through the network via one of its nodes, makes the system flexible to be used and reconfigured without requiring any changes to the individual node's code. TinyDB, alas, has not been kept up-to-date with the evolution of neither sensor hardware platforms nor TinyOS, the operating system on which it worked. Therefore we developed our own system, called TikiDB, which works on Contiki's Rime protocol stack [4,5]. TikiDB behaves just as TinyDB: it exhibits a tree establishment protocol over which query dissemination, data collection and aggregation functions operate.

### 2.2 Management of Node Groups

Initial sensor network architectures assumed that all sensor nodes participate in a single global task. It soon became evident that in many scenarios, a sensor network could be used for multiple purposes [17]. The idea of creating groups of nodes evolved naturally [16]: by restricting the set of nodes that participate in each task, communication costs are reduced, which translates into a prolonged overall network lifetime. Variations of this idea emerged almost concurrently, e.g., for groups of physically nearby nodes [18], logical groups [13], and also under the name of roles [15].

In our group we have built Scopes [10], a framework in which a group of nodes (called *scope*) can be declaratively defined by specifying a membership condition that nodes must satisfy.

An important feature of the framework is the possibility to relate scopes to each other in a hierarchy. A scope's definition specializes that of its parent

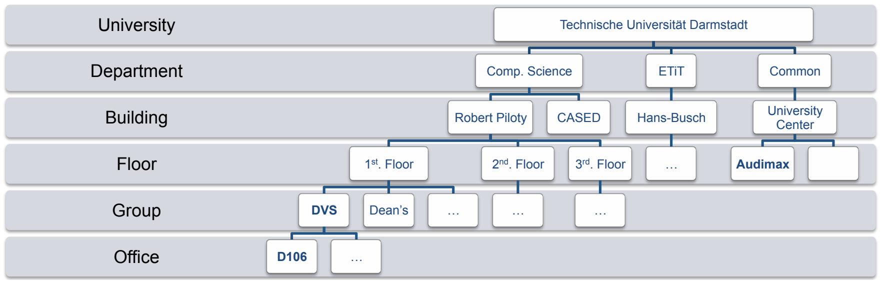


Fig. 2: A hierarchy of scopes for facility management applications at TUD.

scope: member nodes will be a subset of its parents'. As an example, consider the development of facility management applications at the Technische Universität Darmstadt. Fig. 2 presents one such simple scope hierarchy. The topmost scope, representing the entire university, is split into departments, which in turn are split geographically into buildings, then floors, and so on, until fine granularity scopes are achieved, e.g. D106 being Prof. Buchmann's office. The statements below show how to declaratively express scope `Office_D106` as subscope of `DVS` (Fig. 3), as well as `Temp_Office_D106`, which picks temperature nodes in the given office (Fig. 4).

```
CREATE SCOPE Office_D106
   AS ( ROOM = 'D106' )
   SUBSCOPE OF DVS;
```

Fig. 3: Scope definition for office D106

```
CREATE SCOPE Temp_Office_D106
  AS ( EXISTS SENSOR 'TEMPERATURE'
       AND TEMPERATURE < 20C )
  SUBSCOPE OF Office_D106;
```

Fig. 4: Specialization for temperature nodes

A node's membership to a scope might change over time, hence a timely reevaluation at each node is required. The Scopes framework implements mechanisms to correctly deal with the membership, and provides automatic maintenance against network dynamics (nodes leaving and joining and unreliable communication).

In addition to reliably notifying nodes about their membership, Scopes enables a bidirectional communication channel between a scope's creator (called *scope root*) and the scope *members*. The framework resorts to specific routing algorithms that can be chosen to better fulfill the application needs. A naïve implementation simply floods messages throughout the network. An energy-efficient protocol was presented in [10], which uses controlled flooding to disseminate top level scope definitions through the network, and a converge-cast routing tree for relaying data back to the sink. The used tree topology makes this protocol a good candidate for implementing query processing functions, therefore we concentrate on it in this work.

### 2.3 Node Set Reduction

Semantically, a query is answered by extracting data from the `sensors` table as specified by the `FROM` clause, effectively addressing all nodes. Internally, however, queries do not always necessarily need to be spread across the entire network. Consider a query which requires temperature readings greater than or equal to a certain threshold 'x'. Clearly, nodes with values lower than 'x' can abstain from participating in the query other than for forwarding purposes. TinyDB reduces the set of nodes that participate in answering such queries by using a semantic routing tree (or SRT for short). SRTs are overlays on traditional routing trees that, similar to database indices, are used to locate nodes that have data relevant to the query. An SRT is created over constant values, e.g. temperature and humidity, with the statement:

```
CREATE SRT th_index ON sensors (temperature,humidity) ROOT 1
```

The statement creates an SRT named `th_index` rooted at node 1; nodes then discover the range of temperature and humidity values their children have. That information is used later to determine whether a query must be forwarded downwards or not. To a minor extent, the definition of an SRT resembles those of scopes. The flexibility of scope definitions, as shown in the previous subsection, goes far beyond such indices.

Other techniques exist that reduce the set of nodes participating in a query. Dubois-Ferrière and Estrin investigated a multi-sink scenario, and proposed to partition the network using Voronoi scopes [2]. Gupta et al. [8] investigated an approach to suppress nodes that are in close enough proximity such that they might contribute the same or similar data, effectively assuming node redundancy. These techniques complement the approach presented in this paper, and can be applied a posteriori. When resource constraints are not an issue, the approach for mobile phones and smart objects from Frank et al. [7] can be also used.

In the next section we present and exemplify a number of extensions that emerge from the integration of declarative query processing and network scoping.

## 3 Integrating Queries with Scopes

As introduced earlier, the declarative approach to querying a sensor network employs a `SELECT-FROM-WHERE-GROUP BY` statement. The user is normally bound to refer to the whole network in these queries: the `FROM` clause is used with the `sensors` table [4]. Scoping a sensor network, on the other hand, enables the definition of dynamic data sources.

Therefore, it results natural to extend the query semantics by adding the possibility to use these scopes as sources instead of the entire `sensors` table. This section describes TikiDB's extensions to the query processing data model by resorting to example queries for illustration purposes.

[4] other tables can be used, called *materialization points*, but these can't be used to specify node sets

## 3.1 Data Model Extensions

```
SELECT temperature, humidity
   FROM Office_D106
   WHERE humidity > 20%
   SAMPLE PERIOD 30s
```

The simplest way to restrict the set of nodes that participate in a query is by replacing the `sensors` keyword with the name of the scope to be used. Consider the query to the right, where the scope `Office_D106` is defined as in Fig. 3. In this case, the result set consists of tuples generated at nodes which are members of the specified scope (and which meet the `WHERE` clause).

```
SELECT light
   FROM Office_D106, Office_D108
   WHERE light > 200 lux
   SAMPLE PERIOD 30s
```

When multiple scopes are to be used as sources, they can be listed separated from each other by commas as in the query to the right. Here, a user is interested in light sensor values from nodes in both offices. It is easy to observe that the notation stands for union between node groups, instead of an implicit cross join between these.

```
SELECT AVG(temperature)
   FROM Office_D106
   WHERE humidity > 20%
   SAMPLE PERIOD 30s
```

Data aggregation also matches very well with scopes. Aggregating data from a scope's nodes over an epoch is possible by specifying the desired attribute and the aggregation function, as exemplified to the right. The temperature values of all nodes being member of `Office_D106` are then averaged and presented to the user at the base station.

```
SELECT node_type,
       MIN(battery_voltage)
   FROM ComputerScience
   GROUP BY node_type
   SAMPLE PERIOD 30s
```

Results can also be grouped by common attributes such as node type or room. In this case, the root node will deliver a result set with one row for each unique value of the grouping attribute (e.g. node type) together with the aggregated value. The query to the right assumes an attribute `node_type`, which takes values according to the node's properties such as processor and sensors available. It illustrates how to find the lowest battery level for each node type at a large scope, `ComputerScience`.

```
SELECT SUBSCOPE OF DVS,
       AVG(light)
   FROM DVS
   GROUP BY SUBSCOPES OF DVS
   SAMPLE PERIOD 30s
```

Lastly, a more powerful aggregation operation exploits the hierarchical relation between a scope and its subscopes. When a user is interested in an aggregated value for each of the subscopes of a particular scope, he can use the clause `SUBSCOPE OF`. With this, the result set includes one row for each subscope of the specified scope, together with the aggregated value. Consider the query to the right: the grouping element are those subscopes of `DVS`, that is, each of the offices that belong to it (cf. Fig. 2). Note that the user might not know a priori what the subscopes of a scope are, or if there exist any at all. The system takes care of looking them up and managing the aggregated values independently from each other.

### 3.2 Design Considerations

Each of the introduced queries require a corresponding mapping to the operations offered by the network scoping interface. We now describe the design considerations and implementation details of the aforementioned operations, as well as modifications to the underlying scoping layers to correctly support these query processing operations.

**Architecture.** The system follows a layered architecture, as depicted in Fig. 5. Users write and submit queries to TikiDB through a connected node. TikiDB is in charge of parsing the query and allocating the necessary timers and other resources for its execution. Scopes is used for creating the node groups and maintaining them against network dynamics, as well as for disseminating the queries towards scope members and transporting data flowing back to the scope root. We have implemented this stack entirely using Contiki [4], an operating system running threads on a C-based event-driven kernel.

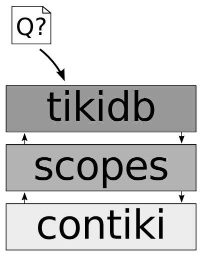


Fig. 5: System layers

**Query Dissemination.** TikiDB's procedure for query injection uses the communication channel offered by Scopes to disseminate the query specification. In this way, indeed, only nodes that are members of the scope are notified (e.g., nodes circled in green in Fig. 6). The strict layering employed by the Scopes framework avoids notifying intermediate nodes, which only forward data messages to their destination (cf. node 2). This feature is necessary in scenarios where security and privacy are of utmost importance, since the network may contain multiple scopes from different users and/or companies. The drawback of this approach, however, is that such nodes cannot be used to perform aggregation: when data flows from producer nodes towards the root. In this work we have extended the Scopes framework to enable the user to specify whether this behavior is desired or not.

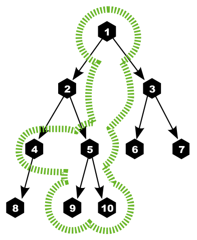


Fig. 6: Query dissemination

## 4 Preliminary Evaluation

In this section we describe a preliminary evaluation of our approach regarding reliability and power efficiency.

### 4.1 Simulation Setup

We have evaluated the integration of Scopes with TikiDB through simulations. For this purpose we use Contiki's COOJA/MSPSim, which is a simulator providing a very accurate emulation of the node's hardware [14]. This enables the usage of exactly the same binary for simulations as for real hardware. While COOJA offers emulation for several hardware platforms, we chose to use Telos nodes since we plan to evaluate the system in our Tmote Sky testbed.

The energy consumed by the nodes was measured using the power profiling mechanism [3] provided by Contiki. The execution times were measured for different components (e.g., radio and processor) on the nodes while they were awake. This information, along with the current energy consumption obtained from the Tmote Sky datasheet, was used to compute the energy consumed by the nodes. In our experiments, initial energy assigned to all nodes was equal, unless otherwise stated.

```
SELECT INTERNAL_VOLTAGE
  FROM scope_x
  SAMPLE PERIOD 128s
```

Fig. 7: Query $Q_1$

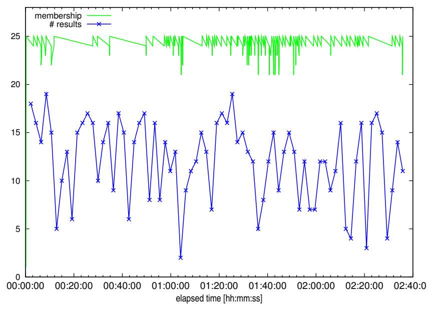
```
SELECT ROOM, AVG(TEMPERATURE)
FROM scope_x
SAMPLE PERIOD 128s
```

Fig. 8: Query $Q_2$

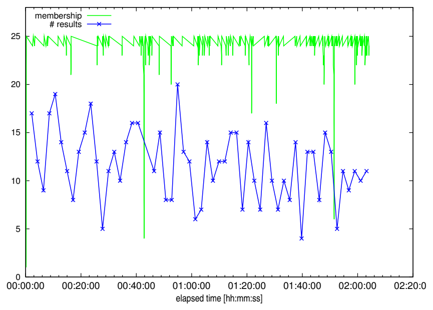
The used network topology was a uniform, 100-node grid of 10 nodes by side. Transmission range was adjusted so that each node can communicate with its four direct neighbors. Despite having used transmission success rates of 100%, radio communication is subject to collisions (we discuss this issue later). We tested the system with two different scopes, which covered 25% of the network (hence 25 nodes were members of it). The first scope, $S_1$, covered nodes distributed uniformly throughout the network, while for the second scopes, $S_2$, we chose those in the upper left corner. These scope definitions were used in combination with two queries (presented in Fig. 7 and 8) by replacing `scope_x` with the respective scope name ($S_1$ or $S_2$). Query $Q_1$ simply requests the internal voltage reading. Despite its simplicity, this query puts the network protocols under stress given the network size (and the respective tree height). Query $Q_2$ is slightly more complex in that it requires aggregation over an epoch. Radio messages are smaller, however more processing delay is present.

### 4.2 Simulation Results

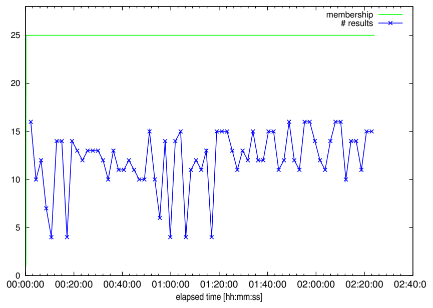
In Fig. 9 we present the reliability results for the aforementioned combination of scope definitions and queries. The x axis represents elapsed time. The plain (green) curve shows node membership, while the line with crosses (blue) represents the number of received results at the scope root (the positions of the crosses indicate the start of a new epoch). The plots show results for the first 60 epochs (~2.5 hours).



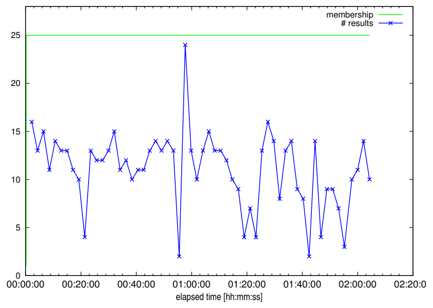
(a) $S_1$, $Q_1$



(b) $S_1$, $Q_2$



(c) $S_2$, $Q_1$



(d) $S_2$, $Q_2$

Fig. 9: Result delivery reliability for $S_1$, $S_2$ combined with $Q_1$, $Q_2$

The first aspect to consider is the scope membership (green curve), since nodes that do not become scope members will not generate tuples to contribute to the result. Both scope definitions cover 25 nodes; we observe that while $S_2$ remains stable over time from the beginning of the test run till the end, $S_1$ shows a slight variability. This is expected, since $S_1$ spans the whole network, while $S_2$ covers a concentrated, smaller fraction of it. In general, however, scope membership remained high, which is to be attributed to the reliability with which *administrative* messages are sent by the Scopes framework.

Given this almost ideal node membership, we consider the amount of received results at the scope root. At first sight, it seems surprising that only around 50% of the results arrive at the root. This suboptimal outcome, however, is due to a number of issues. With query $Q_1$, messages get longer as nodes are closer to the root. This triggers the message fragmentation function; the Scopes framework, in turn, sends *data* messages without requesting acknowledgements. Occupying the broadcast medium for longer periods of time clearly increases the probability of message collisions. A packet loss near the root implies losing a big part of the results. With query $Q_2$, on the other side, in-network processing keeps individual nodes busier for longer times. This causes results to sometimes arrive out of order at their next hop, which eventually discards them (out-of-order messages arrived 3% of the time).

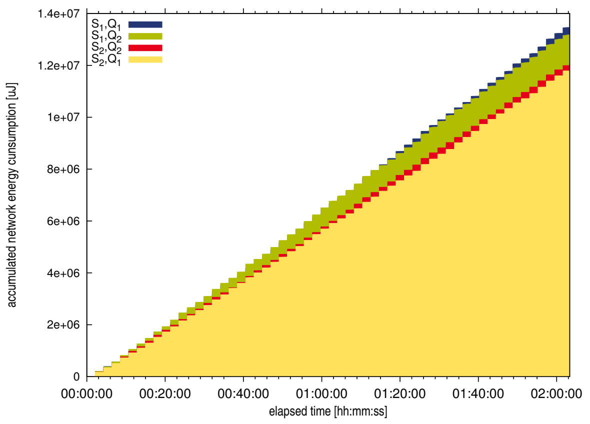


Fig. 10: Power efficiency for the tested queries

In Fig. 10, we present the results for power efficiency. The plot presents time on the x axis, and accumulated network energy consumption on the y axis. The plot contains energy consumption measurements for the first ∼55 epochs. Here it is observed that the queries involving scope $S_1$ consume 14% more energy than the queries involving $S_2$. This was expected because nodes in $S_1$ are spread through the network, which requires spending more energy to collect results. Also, it is noted that the two queries executed on the same scope do not vary drastically (the observed absolute difference was $< 2\%$). As expected, $S_1$,$Q_1$ has required more energy than $S_1$,$Q_2$, since the latter performs in-network aggregation, therefore minimizing message size. However, the lower energy consumption shown by $S_2$,$Q_1$ compared to $S_2$,$Q_2$ is contradictory to our expectations: the query executing aggregation requires more energy than the query without it. We speculate that a possible explanation could be that the savings in communication costs are lower than the extra costs for computing aggregates, since in $S_2$, member nodes are closer to the scope root, thus requiring less communication. A detailed investigation of this issue is a matter of future work.

## 5 Conclusions and Future Work

In this paper we have proposed an approach to declaratively specify the set of nodes that participate in a query. While, in sensor networks, declarative interfaces to query processing as well as for node group management had already been investigated, the integration of these two is a promising approach which can further improve the usability of sensor networks for non-experts.

We have shown the benefits of this approach by integrating two of such systems. On one side, TikiDB, which is a modern reincarnation of a distributed query processor developed on the Contiki operating system. TikiDB manages query aspects such as query parsing, data acquisition, data aggregation and filtering. On the other side, Scopes, which is a distributed node grouping system that efficiently handles network dynamics and enables a bidirectional communication channel between a scope's root node and its members.

We have proposed four extensions to TinyDB's original data model that make use of scopes to specify or reduce the set of nodes that participate in the query. The easiness in describing its syntax suggests that the constructs are applicable to many situations. Our preliminary evaluation results show a non-ideal result delivery reliability, leaving space for optimizations. Clearly, around the scope root, messages get large enough such that fragments have high probability of collision. Since data messages in Scopes are not acknowledged, data loss becomes an issue.

In the future we plan to enhance our implementation to improve reliability, reduce energy consumption, and also optimize the program size. These activities will be surrounded by extensive test runs on a real deployment to characterize and evaluate our framework in more detail.

## References

1. Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. In *Proceedings of the Second International Conference on Mobile Data Management*, January 2001.
2. Henri Dubois-Ferriere and Deborah Estrin. Efficient and Practical Query Scoping in Sensor Networks. In *Procs. of the 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 564 – 566, October 2004.
3. A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based Online Energy Estimation for Sensor Nodes. In *4th IEEE Workshop on Embedded Netwoked Sensors (Emnets-IV)*, Cork, Ireland, June 2007.
4. Adam Dunkels, Björn Gronvall, and Thiemo Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Nov. 2004.
5. Adam Dunkels, Frederik Österlind, and Zhitao He. An Adaptive Communication Architecture for Wireless Sensor Networks. In *Proceedings of Conference on Embedded Networked Sensor Systems (Sensys '07)*. ACM Press, November 2007.
6. L. Fiege, M. Mezini, G. Muehl, and A. Buchmann. Engineering Event-based Systems with Scopes. *European Conference on Object-Oriented Programming 2002*, pages 257–268, 2002.
7. Frank, Roduner, Chie Noda, and Kellerer. Query scoping for the sensor internet. In *PERSER '06: Proceedings of the 2006 ACS/IEEE International Conference on Pervasive Services*, pages 239–242, Washington, DC, USA, 2006. IEEE Computer Society.
8. Himanshu Gupta, Zongheng Zhou, Samir R. Das, and Quinyi Gu. Connected sensor cover: self-organization of sensor networks for efficient query execution. *IEEE/ACM Trans. Netw.*, 14(1):55–67, 2006.

9. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer Pister. System Architecture Directions for Networked Sensors. *SIGOPS Oper. Syst. Rev.*, 34(5):93–104, December 2000.
10. Daniel Jacobi, Pablo E. Guerrero, Ilia Petrov, and Alejandro P. Buchmann. Structuring Sensor Networks with Scopes. In *3rd IEEE European Conference on Smart Sensing and Context (EuroSSC)*, pages 40–42, Zurich, Switzerland, October 2008. IEEE Communications Society.
11. Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service for Ad-hoc Sensor Networks. *5th Symposium on Operating Systems Design and Implementation*, 36(SI):131–146, 2002.
12. Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.*, 30(1):122–173, March 2005.
13. Luca Mottola and Gian Pietro Picco. Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In Phillip B. Gibbons, Tarek F. Abdelzaher, James Aspnes, and Ramesh Rao, editors, *DCOSS*, volume 4026 of *Lecture Notes in Computer Science*, pages 150–168, San Francisco, CA, USA, June 2006. Springer.
14. Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level simulation in cooja. In *European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.
15. Kay Römer, Christian Frank, Pedro José Marrón, and Christian Becker. Generic Role Assignment for Wireless Sensor Networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, pages 7–12, Leuven, Belgium, September 2004.
16. J. Steffan, L. Fiege, M. Cilia, and A. Buchmann. Scoping in Wireless Sensor Networks: A Position Paper. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*, pages 167–171. ACM, 2004.
17. Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro P. Buchmann. Towards Multi-Purpose Wireless Sensor Networks. In *Systems Communications*, pages 336–341, Montreal, Canada, August 2005. IEEE Computer Society.
18. Kamin Whitehouse, Cory Sharp, Eric Brewer, and David E. Culler. Hood: A Neighborhood Abstraction for Sensor Networks. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications and services*, pages 99–110, Boston, Massachusetts, USA, June 2004. ACM Press.
19. Y. Yao and J. Gehrke. The Cougar Approach to In-network Query Processing in Sensor Networks. *SIGMOD record*, 31(3):9–18, 2002.