QueryAgent : A General Query Processing Tool for Sensor Networks

Weisong Shi, Sivakumar Sellamuthu, Kewei Sha, and Loren Schwiebert
Wayne State University
{*weisong, siva, kewei, loren*}*@wayne.edu*

Abstract

Sensor networks are promising in many applications; however, we have not yet seen its wide acceptance and deployment. We envision that a big obstacle to solving this dilemma is the lack of an easily usable tool for applications scientists to manage and use sensor networks. In this paper, we take an initial step to tackle this problem by developing a general tool to fill the gap between applications and sensor network protocols, which would also pave the way for the wide acceptance and deployment of sensor networks. Our approach includes two components: a general network programming interface *abstracted from common usage patterns, and* a transparent query to interface converter *which will ease the burden on application scientists significantly. Using this automatic converter, application scientists need to use a high level SQL-like language for plugging and playing wireless sensor network operations. These two components will be integrated and implemented into a general query processing tool called* QueryAgent*. This tool also maps efficient sensor network protocols to different requirements of end-users in an intelligent way.*

1 Introduction

As new fabrication and integration technologies reduce the size and cost of micro-sensors and wireless sensors, we will witness another revolution that facilitates observation and control of our physical world [1, 15], just as networking technologies have done for the ways individuals and organizations exchange information. Although sensor networking is promising, we have not yet seen its wide acceptance and deployment. Thus, we need to revisit the two motivations for wireless sensor networks: "technology push" and "application demand pull".

On one hand, research on wireless sensor networks, including sensor system design, energy-efficient routing protocols, data aggregation, and other optimizations to extend the lifetime of sensors, has been a hot topic. These protocols have continually evolved over the past five years; however, most published results are based on computer simulation, which requires extensive assumptions regarding operating conditions and other parameters. Therefore, these published results are either not sufficiently convincing or too theoretical to be used in a real deployment of sensor networks. For example, several GPS-based localization algorithms have been proposed, but none of them can be used in waste containment system monitoring where sensors are deployed underground [17]. Therefore, we believe much of the current research is too abstract to be accepted by application scientists.

On the other hand, application scientists have already realized the importance of using wireless sensor networks, such as the environment engineering community [4], but it is non-trivial, if not impossible, for them to deploy a wireless sensor network. This is very similar to the dilemma in parallel computing, where it is very difficult for application scientists to parallelize their applications to use clusters or parallel computers. We believe this is the main obstacle to the success of wireless sensor networks, and it is the responsibility of the sensor networks community to fill this gap over the next five years.

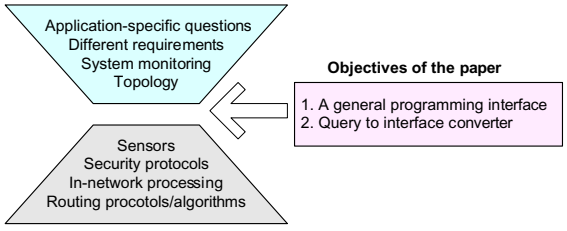

Figure 1. The gap between applications and underlying sensor networks, and the objectives of the paper.

Figure 1 shows the hourglass situation of wireless sensor networks. The *objective* of this paper is developing a general query processing tool to fill the gap between applications and sensor networks, and pave the way for wide acceptance and deployment of sensor networks. Motivated by the invention of TCP/IP protocols on the Internet, our approach to address this gap consists of two components, from the

Proceedings of the 2004 International Conference on Parallel Processing Workshops (ICPPW'04)
1530-2016/04 $20.00 © 2004 **IEEE**

bottom up, including: *a general network programming interface* and *a transparent query to interface converter*. The contributions of this paper are two-fold:

- *Abstract a general network programming interface for wireless sensor networks*. Like the connection-oriented and connectionless services provided by the transport layer of the TCP/IP network suite, the general programming services abstracted here will serve the high layer services in much the same way.
- *Fill the gap between high level queries from application scientists and the lower level programming semantics provided by underlying sensor networks,* which will ease the burden on application scientists significantly. We believe this tool is the first step toward opening the door to widespread success of wireless sensor networks.

The remainder of this paper is structured as follows. In Section 2, the design of the general programming interface, which abstracted form the usage patterns of sensor network applications, is depicted first. Section 3 outlines the design details of the QueryAgent. Several examples of QueryAgent are listed in Section 4. Section 5 compares our work with related efforts. Finally, the current status and a summary are listed in Section 6.

2 General Network Programming Interfaces

After a brief description of the usage patterns in sensor networks in this section, we propose a set of general interfaces.

2.1 Usage Patterns of Sensor Networks

The main function of a sensor network is to monitor and gather data from the sensor field. Generally, there are three usage patterns of a sensor network to collect data: active querying (pull), passive monitoring (push), and a combination of the two. In the case of active querying, each time data is needed, the sink generates a query message to ask for data from the sensor network. Sensors with corresponding data generate a reply message and route it back to the sink or gateway. For example, if the sink wants the current temperature of an area that is monitored by several sensors, it sends a query to these sensors asking for the temperature. In the passive monitoring mode, however, the sensors are gathering readings at all times and periodically report these readings to the sink. For example, a sensor used to monitor the humidity of some area and required to report the humidity once an hour, or report interesting events such as when the humidity exceeds a pre-defined maximum. This is also known as event driven sensor usage. In addition, a combination of both push and pull is used in some scenarios. When the sink sends queries to ask for specific sensor readings, these queries are active for a period of time during which sensors having relevant data for that query report these readings to the sink. For example, sensors collect the wind speed in some area. Under normal weather conditions, the sink queries the wireless sensor network for the wind speed as needed. If the weather becomes violent, the sink may send out a query asking the sensors to report the wind speed every 30 minutes for two days. The difference between active querying and passive monitoring lies in the time interval during which the query is valid. In active querying, the query is satisfied after the reply is sent. On the other hand, the query will last for a long time in passive monitoring during which time reply messages are sent repeatedly. With hybrid monitoring, the query duration varies; sometimes the query lasts a long time and other times it quickly becomes invalid.

2.2 General Programming Interfaces

From the application perspective, we are interested in just gathering data from the sensor network, and we do not care how the sensor network distributes the request to the sensors. Consequently, the whole sensor network can be viewed as a dynamic distributed database from which the application gets the data of interest. Conversely, from the viewpoint of a lower-level layer such as the network layer, the research focus is how to make routing more energy efficient and perform load balancing. These layers are separate, and it is inconvenient to require the application layer to know details of lower layers in order for the lower layers to provide optimal support for the application layer. Instead, it is better to define an interface that specifies the information the application level should provide to collect data from the sensor field, and offers sufficient information to the network layer to choose suitable protocols. We have several objectives in designing this interface. First, it should be general, which means the interface can be used by a wide variety of applications. For example, it can be used for both active querying and passive monitoring. Second, it should be flexible; the interface supports different application requirements through different types of algorithms. For example, when an application needs very accurate data, the interface supports the collection of highly accurate data. Third, it should be intelligent, i.e., for different communication models, the interface selects the most suitable protocols to achieve the goal of energy efficient and extend the lifetime of the sensor network. The benefits obtained from a clearly defined general interface between the application layer and lower layers are as follows:

- The interface masks the gap between the application layer and the lower layers, building a general platform for communication.
- It can make the application independent of the sensor network. The application can view the sensor network

as a dynamic database and get the data it is interested in without understanding the details of how messages are routed.

- The lower layers are also independent of the application. These layers can focus on the design of energy efficient routing or query protocols.
- The interface can direct the design of the low-level routing or query protocol, i.e., the low-level protocol is designed to support the requirements of the interface.
- The interface can also be augmented to support other functions such as error control.

The interface between the application layer of the sensor network and a lower layer protocol is defined to support several communication models. We can classify the communication models in sensor networks into four types: Unicast, Area Multicast, Area Anycast, and Broadcast. Using this interface, the application needs to know only where to direct the query and the specific type of communication model in the generated query. On the other hand, the lower layers take care of forwarding the query and routing the reply back to the sink by selecting a suitable routing protocol for the communication model specified by the general interface.

The four communication models are abstracted to fit the characteristics of the data source. The difference among the four communication models lies in the granularity of the area of the data source. In the unicast model, the data source of a query is an individual sensor, so the communication is point-to-point between the sink and that sensor. Area multicast is interested in the data from a certain area, so it routes the query to all sensors in a certain area, and then all the sensors in the area generate a reply message to the sink. Alternately, area anycast is also interested in readings in a certain area, so it routes the query to a specific area and at least one sensor in this area sends a reply message to the sink. Finally, in the case of broadcast, the query message is routed to every sensor in the network, and all sensors with corresponding data return a reply to the sink. These four communication models can be used in both active querying and passive monitoring. We propose APIs for both.

Each query from the application layer should call one of the four communication models in the interface and each routing protocol should support at least one communication model in the interface. For each communication model, we define a set of APIs. Communication occurs between the application layer and the low-level layers by calling the APIs defined in the general interface. An example query of the temperature of one sensor is described as follows. First, the application sends a `unicast(QID,tmptr,sensorID)` message to the general interface. This interface returns a suitable routing protocol for this query. Second, the API named `start_unicast(QID,PName,tmptr,sensorID)` is called. A message is sent to low-level layer; actual processing for this query starts at the lower-level layer. Third, `listen_unicast(QID,data)` is issued to wait for the data from the sensor network. Finally, the application calls API `finish_unicast(QID,data)` and receives the expected data, e.g., in the case of active querying this API will be called immediately after the reply is received and in the case of passive monitoring this API is called when data for this query is not needed any longer. We also define the APIs to control and manage the sensor network. Here we just define three of them `Turnoff(QID)`, `Turnon(QID)`, and `Move(QID,direction,value)`. More APIs for control functions will be added in the future.

Table 1 presents a more detailed description of the general interface. These APIs are suitable for all three types of sensor network usage patterns. Different communication models are suitable for use in different scenarios. Unicast is most suitable when the application is interested in readings from specific sensors. Area multicast is a proper choice when the application is interested in one specified area. For instance, when the application wants to collect data on the concentration of one poisonous gas in some dangerous area (*Select concentration from WSN where* $S.x \geq xlow$ *and* $S.x \leq xhigh$ *and* $S.y \geq ylow$ *and* $S.y \leq yhigh$), it can send a query to all sensors in that area and get a reply from all of them. In this case, point-to-point routing is no longer suitable. Moreover, it is possible that several sensors collect the same information for the same area for redundancy and fault tolerance. So, Area anycast is perfect in this scenario to save the energy. Sometimes, the application does not know exactly where the target is, thus it has to transmit the query to all sensors in the network. The broadcast communication model is useful in this case. Finally, control information is also used to control the sensor network such as to turn off some sensors to save energy. When a sensor is mobile, the application can control the movement of sensors by sending control information. Control information is more useful as sensors become more powerful.

From the above analysis, we observe that each communication model has a suitable scenario for its use, while it is not suitable for some other scenarios. These four communication models are enough to support the requirements of most applications. Thus, we define a general interface to support these four types of communication.

3 QueryAgent: An Automated Converter

The general interfaces defined in Section 2 provide an abstract interface between applications and the underlying network; however, it is still difficult for application users to use the interface directly. Therefore, we propose to build an

API	Description
PName unicast(QID,intst,dstn)	The application layer wants to get data from one sensor. The interface returns a suitable routing protocol name.
Data listen_unicast(QID,data)	The application layer listens to get the data from the sensor network.
Boolean start_unicast(QID,PName,intst,dstn)	Ask low-level layer to start the unicast process.
Data finish_unicast(QID,data)	Explicitly finish the unicast process.
PName area_multicast(QID,intst,dstn)	The application layer wants to get data from some specified area. The interface returns a suitable routing protocol name.
Data listen_area_multicast(QID,data)	The application layer listens to get the data from the sensor network.
Boolean start_area_multicast(QID,PName,intst,dstn)	Ask low-level layer to start the area multicast process.
Data finish_area_multicast(QID,data)	Explicitly finish the area multicast process.
PName area_multicast(QID,intst,dstn)	The application layer wants to get data from at least one sensor in the specified area. The interface returns a suitable routing protocol name.
Data listen_area_anycast(QID,data)	The application layer listens to get the data from the sensor network.
Boolean start_area_anycast(QID,PName,intst,dstn)	Ask low-level layer to start the area anycast process.
Data finish_area_anycast(QID,data)	Explicitly finish the area anycast process.
PName broadcast(QID,intst,dstn)	The application layer wants to get data from all sensors. The interface returns a suitable routing protocol name.
Data listen_broadcast(QID,data)	The application layer listens to get the data from the sensor network.
Boolean start_broadcast(QID,PName,intst,dstn)	Ask low-level layer to start the broadcast process.
Data finish_broadcast(QID,data)	Explicitly finish the broadcast process.
Turnoff(QID)	Turn off the sensor.
Turnon(QID)	Turn on the sensor.
Move(QID,direction,value)	Move the sensor to another location.

Table 1. Our general network programming interfaces for wireless sensor networks.

application framework called *QueryAgent* to intelligently match high level queries, a subset of SQL, with the underlying programming interfaces proposed in Section 2. This framework encapsulates all the low level details of sensor networks for applications, which are interested in query results only. Next, an overview of the QueryAgent is described, followed by a depiction of a subset of SQL semantics supported by the QueryAgent, the details of design, and supported query types.

3.1 Overview

QueryAgent acts as a bridge to connect application scientists and underlying wireless sensor networks. We propose and develop this tool to maximize the needs of application scientists and also to support lower level programming semantics provided by the underlying sensor network. QueryAgent is a general tool that receives application scientist's SQL type queries (see Section 3.2) for the sensor network. After getting queries, QueryAgent will efficiently parse and map them to appropriate APIs, which in turn call the corresponding implementation provided by the underlying sensor network, as shown in the right box of Figure 2. After obtaining the results from the network, QueryAgent performs another round of filtering and aggregation (optimization), and then presents the results to end users in a user friendly way.

3.2 A Subset of SQL Queries

From the perspective of application scientists, the whole sensor network is just a dynamic database that provides interesting information. So it is natural to choose a declarative language, which is easy to master and use for non computer science experts, as the input of QueryAgent. We define the query language as a subset of standard SQL used in relational databases. Motivated by the seminal work on Thinly [13], we decide to support the following format in QueryAgent.

```
SELECT {aggregates(attr),(attr)}
 FROM [quantifiers] sensor [as Alias]
 [WHERE (predicate)]
 [IN (regiontype)]
 [HAVING (predicate)]
 [GROUP BY (attributes)]
 [DURATION time]
 [INTERVAL time]
 [TRIGGER function]
```

This type of modeling an SQL statement is the same as that in traditional SQL. Join and logic operationx can be done by using any of the AND, OR, and NOT operators. Aggregates used are SUM, MAX, MIN, AVG, and COUNT. To compare mathematically, we use larger than ($>$), less than ($<$), equal ($=$), larger than or equal to ($>=$), and less

than or equal to (<=). Other than the normal operators, we also define quantifiers like ANY, SOME, and ALL. Another set operation, IN, is defined for use in the communication model `area_anycast()`. When this anycast routing protocol is used only one sensor in a region will reply.

3.3 Design of QueryAgent

Our design of QueryAgent consists of six modules: *Graphical User Interface (GUI)*, *SQL Parser*, *API Selector*, *Data Manager*, *Cache History*, and *Intelligent Agent*, as shown in Figure 2. These six modules are interoperable and can coordinate with each other. Each query is given an ID so it can be handled easily. Results obtained from the sensor network for each query can be clearly and correctly presented to end users.

- **GUI** A graphical user interface is the foremost thing that is needed for the application users to do their work with ease. A GUI allows users to enter their SQL queries and get results from the GUI, so users do not have to worry about the underlying complex programs or their interfaces.

- **SQL Parser and API Selector** A traditional SQL parser takes the SQL commands as input and retrieves information from the database. Instead, QueryAgent parses the command and give its output to the API selector to map into the APIs that can correctly support the query. For example,

  ```
  SELECT temperature FROM sensor
          WHERE location = 10,10
  ```

  This statement tells the SQL parser that this command is a SELECT statement and that we wish to retrieve information from the sensor network. The WHERE clause allows restrictions to those sensors that meet the specified condition(s). QueryAgent sends the output to the API given the destination location and retrieves the temperature, so the API handles the message creation by appropriately creating request packets.

- **Data Manager** The Data Manager module actually does the data aggregation if needed and does the computation for some queries. APIs distribute the information from the packets to the QueryAgent. This module collects this information and returns it to the GUI after modifying the result according to the user requirements. For example,

  ```
  SELECT temperature FROM all sensors
  ```

  In this example, the Broadcast API is chosen and all sensors return their temperature value to the API. The API sends these values to the QueryAgent. The *Data Manager* collects temperatures from all sensors and places them in local storage for offline queries later. They also store the values it obtained with the *Cache History* module. Later the *Intelligent Agent* module will make use of these values.

- **Cache History** The Cache History module is used to store all the SQL queries and their query IDs for a particular period of time, so that the Data Manager can reuse the cached data (or information) for further similar queries. They also store results from the API for later optimization.

- **Intelligent Agent** The Intelligent Agent module optimizes query handling. For example, the query forwarding proposed in the IndirectQuery protocol [16] supports query caching, query prediction, and query prefetching based on query patterns. For example, considering a sensor network where all sensor's locations are static, then a query like

  ```
  SELECT id, temperature FROM sensor
            WHERE Location = 10,10
  ```

  will always return the same results if the temperature of this node does not expired. As a matter of fact, we found that in many sensors applications, different variables have different consistency and timeliness requirements [17], which is very useful in reducing the number of messages and routing hops in wireless sensor networks. Therefore, exploiting the difference between consistency and timeliness of different values is the job of the *Intelligent Agent* module.

3.4 Supported Queries

Based on the complexity of queries, QueryAgent is designed to support the following three query types:

- *Simple queries*: When the query is simple like "what is the temperature of node X?" then the SQL statement can be 'SELECT temperature FROM sensor WHERE id=X", which is converted into a broadcast query to the sensor network. The sensor node with id=X sends back a reply to the query. Similarly, ''what is the id of the sensor at location 10,10?" is a simple SQL query – "SELECT id FROM sensor WHERE Location=10,10", which is converted into a point-to-point query. The sensor at the specified location responds to the query with a reply message. These are non-aggregate queries.

- *Complex queries*: Complex queries have sub-queries like "What is the pressure of the region with the highest temperature?", which is stated as "SELECT pressure FROM sensor WHERE temperature=(SELECT

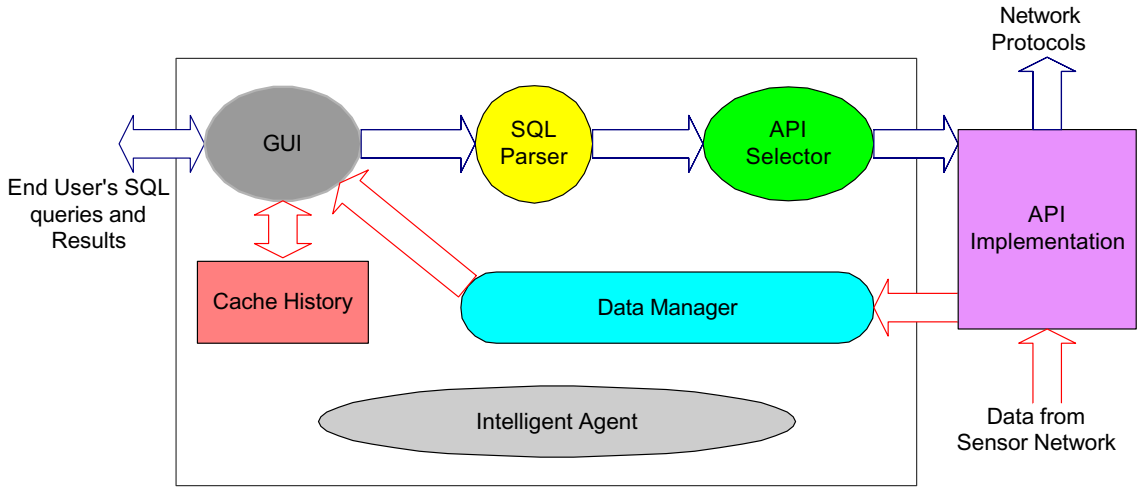

Figure 2. General design of QueryAgent.

max(Temperature) FROM sensors)". This is converted into a broadcast message. After the sensors return their temperature and pressure readings, the result is computed and returned to the application.

- *Event driven queries*: These type of queries require an advanced SQL to return readings ON EVENT. They are continuous queries like `SELECT avg(temperature) FROM sensor WHERE id=X FOR time=100`, which is mapped to an API that may use Directed Diffusion [8] as the routing protocol.

In Section 4, we discuss in more detail mapping the queries to the APIs.

4 Examples of Using QueryAgent

Given a reference implementation of the proposed programming interface as defined in Section 2, the API selector module does the mapping of SQL queries to the API. In this section, we use several examples to show the machinery of the QueryAgent tool. The API is selected according to the type of queries. Queries in our approach can be generally defined as *Active*, *Passive*, and *Event Driven queries*.

- **Active Query**: This type of query from application scientists is used to get the data from the sensor network by actively querying them. The following query is a typical example of an active query:

```
SELECT max(humidity),id FROM sensor
 WHERE temp >
(SELECT avg(temp) FROM sensor
 WHERE location > [200,200]
        AND location < [400,400])
```

To handle this type of query, QueryAgent will call an API like `area_multicast(QID,intst,dstn)` and the corresponding listen interfaces. This API would get QueryID, interested message, and destination values specified from the query. The data of the interested message would be humidity and temperature. This API will return a suitable routing protocol name that is used to collect required data. The protocol returns the ID of the sensor with maximum humidity among the sensors that have a temperature greater than the average temperature of all sensors in a specified region.

- **Passive Query**: This type of query is used to get data from the sensor network in a continuous model at a regular interval. A query like the following is a typical example of an passive query:

```
SELECT temp,id FROM sensor
 WHERE  pressure =
 (SELECT max(pressure) FROM sensor
  WHERE location >= [200,200]
        AND location <= [900,900])
 DURATION 10000
 INTERVAL 50
 TRIGGER alert(id)
```

To handle this type of query, QueryAgent will call an API like `area_multicast(QID,intst,dstn)` and the corresponding listen interfaces. This API would get QueryID, interested message with the specified time values, and within the destination region. Interested message would have a duration of 10000 seconds and also interval of 50 seconds. Whenever a result is obtained, the `alert()` function is called. This

type of query would be mapped to a protocol that triggers sensors to send data over a period of time, so that no extra query is needed to get the data from the sensor.

- **Event Driven**: Event driven is a very popular usage pattern in applications. A representative example is object tracking. From our point of view, this is just another kind of passive querying with an irregular time interval. Information will be sent back only when certain events happen; however, the uncertainty of the events makes it non-trivial to implement this kind of query. An event-driven query is listed as follows:

```
ON EVENT rise in temperature
 SELECT temp FROM sensor
 TRIGGER alarm(temp), action(temp)
```

  To handle this type of query, QueryAgent needs to use an API like `listen_broadcast(QID,data)` to catch the event. Here the data will be temp, which represents temperature. This API tells sensors to send their temperatures back whenever there is a temperature rising. After QueryAgent gets the result, it triggers the functions `alarm()` and `action()`, which are pre-defined.

The above examples show that the job of QueryAgent is to choose all appropriate APIs for each query. As we have seen, some optimizations could be done by QueryAgent to adapt to application-specific requirements.

5 Related Work and Discussions

The proposed work builds upon a great deal of previous work in the field of wireless sensor networks in general. Instead of describing all these research results, we focus on previous work that is specifically related to our protocols, including *programming abstraction and query protocols* and *query conversion*.

Programming abstraction and query protocols. Recently, programming interfaces of wireless sensor networks have attracted a lot of attention from the research community [5, 10, 19]. Heidemann *et al.* [5] propose two types of APIs, basic diffusion APIs and filter APIs, as the interface between the application and network layers. These APIs are designed especially for directed diffusion and favor an event-driven programming model. The interface for which we aim is much more powerful and intelligent than the interface designed for only a single model. In [6], the importance of matching the dissemination algorithm to the application is demonstrated and two new implementations of the diffusion API are matched to two new classes of application. The importance of the match between algorithms and applications is revisited in [9]. They limit their scope to directed diffusion rather than providing a general interface.

MiLAN [14] is middleware between application and low-level dynamic networks. Their work intends to provide higher level abstractions of low-level concepts and to continuously control the network functionality with respect to the application's changing demands. MiLAN presents a well-defined API through which the application presents its requirements to low-level components. To the best of our knowledge, this is still an on-going project. We are not aware of any published results on how application developers benefit from this API.

TinyOS has a macro-component called `GenericComm` that provides two active message communication interfaces [10], `SendMsg` and `ReceiveMsg`, to support single hop unicast and broadcast communication. Their work focuses on a lower layer interface than our proposal, i.e., focusing on the implementation level, with the user providing detailed active messages to use the interface. Their interface is suitable for programmers working on the sensor network while our work can be used directly by application scientists. Their work complements our work. When we implement our APIs we can benefit from the interface provided by TinyOS.

State-centric and agent-based methodology and the companion software environment PIECES for collaborative signal and information processing (CSIP) algorithms are proposed by Liu *et al.* [2]. They want to mediate between a system developer's mental model of physical phenomena and the distributed sensor network platforms. They abstract common patterns in application-specific communication to some collaboration groups. We instead abstract the sensor network by the communication models instead of different groups and we propose a general interface for all applications rather than just for tracking applications.

Welsh *et al.* [19] develop an aggregate programming model called abstract region for sensor networks to reduce the difficulty of developing sensor network applications. Applications built on abstract regions can tune resource and accuracy trade offs by using the abstract region APIs. The abstract region approach is at a lower level than our proposed general programming interface, and complements to our work.

Query conversion. Yao and Gehrke [18, 20] propose a query proxy layer that runs every sensor node. Supporting aggregate queries is also proposed by Madden *et al.* [11]. TAG [12] is designed to provide an interface to aggregate the data gleaned from the sensor network. TinyDB [13] is a query processing system for sensor networks; tightly coupled to the in-network aggregation protocol proposed in TAG [12]. Both works target mainly data aggregation and optimization, and are at a lower layer than our proposed query agent. Neither research effort mentions how to translate an SQL-style query into a sensor network query.

In [3] a model of a sensor database is defined, where

stored data is presented as relations while sensor data are presented as time series. The paper focuses on sensor query processing, taking the whole network as a sensor database. This is another way to map applications to the sensor network, but our approach is more general.

6 Current Status and Summary

Currently, we are developing the following two software components. One is a reference implementation of the abstracted APIs. In addition to implementing it in our simulator, we are also implementing it on a real test bed consists of several dozen Crossbow Motes running TinyOS [7]. The other is a prototype of QueryAgent. We expect a working prototype to be available in August 2004. Our future work includes providing a system monitoring tool with highly accurate localization and topology mapping between the real deployment a logical space, integrating with the QueryAgent. Also, application scientists will be invited to use an initial version of this tool to query sensor networks under our observation, so that we can refine the GUI as well as make QueryAgent flexible for typical queries.

References

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

[2] J. L. et al. State-centric programming for sensor-actuator network systems. *IEEE Pervasive Computing*, 2003.

[3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. *Proceedings of the Second International Conference on Mobile Data Management*, Jan. 2001.

[4] D. Estrin, W. Michener, and G. Bonito. Environmental cyberinfrastructure needs for distributed sensor networks, Aug. 2003. A Report From a National Science Foundation Sponsored Worksop.

[5] J. Heidemann et al. Building efficient wireless sensor network with low-level naming. *Proceedings of the Symposium on Operating Systems Principles*, Oct. 2001.

[6] J. Heidemann, F. Silva, and D. Estrin. Matching data dissemination algorithms to application requirements. *Proceedings of First ACM SenSys'03*, Nov. 2003.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *Proceedings the 9th ASPLOS'00*, Nov. 2000.

[8] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking(MobiCom'00)*, Aug. 2000.

[9] B. Krishnamachari and J. Heidemann. Application-specific modelling of information routing in wireless sensor networks. Tech. Rep. ISI-TR-576, University of Southern California, Aug. 2003.

[10] P. Levis et al. The emergence of networking abstractions and techniques in tinyos. *Proceedings of the First USENIX/ACM Networked System Design and Implementation*, Mar. 2004.

[11] S. Madden et al. Supporting aggregate queries over ad-hoc wireless sensor networks. *Workshop on Mobile Computing and Systems Applications*, 2002.

[12] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor network. *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[13] S. R. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. Ph.D. thesis, Department of Computer Science, University of California, Berkeley, 2003.

[14] A. Murphy and W. Heinzelman. Milan: Middleware linking applications and networks. Tech. Rep. Technical Report, University of Rochester, Nov. 2002.

[15] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM* 43(5):51–58, May 2000.

[16] K. Sha, S. Sellamuthu, and W. Shi. Load blanced query protocols in wireless sensor networks: Theory and practice. Tech. Rep. MIST-TR-2004-006, Wayne State University, Feb. 2004.

[17] W. Shi and C. Miller. Waste containment system monitoring using wireless sensor networks. Tech. Rep. MIST-TR-2004-009, Wayne State University, Mar. 2004.

[18] A. Wand and A. Chandrakasan. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record* 31(3):9–18, Sept. 2002.

[19] M. Welsh and G. Mainland. Programming sensor network using abstract regions. *Proceedings of the First USENIX/ACM Networked System Design and Implementation*, Mar. 2004.

[20] Y. Yao and J. Gehrke. Query processing in sensor networks. *the First Biennial Conference on Innovative Data Systems Research*, Jan. 2003.