# MaD-WiSe: Programming and Accessing Data in a Wireless Sensor Networks

G. Amato, P. Baronti, and S. Chessa, *Member, IEEE*

***Abstract* — MaD-WiSe is a wireless sensor network database designed to perform in-network distributed query processing and to manage acquired data. This paper briefly presents the MaD-WiSe system and its query processing model based on data streams. Then, it focuses on the MaD-WiSe graphical user interface, which supports query definition and injection and query results collection.**

***Keywords* — Wireless Sensor Networks, Databases, Stream Data Processing.**

## I. Introduction

ACCESSING and processing data produced in a wireless sensor network [1] using a database-like approach [2]-[4] has several advantages. Sensors can be deployed in the physical environment once for all and applications that manipulate their data can be created, refined, and modified afterwards without any physical intervention on the sensors themselves. In fact, data management activity performed in the network is remotely controlled by interactively issuing queries, expressed in a high level language, which specify what data are of interest for a certain task, and how they should be manipulated. Changing the behavior of the data management activity in the network corresponds to execute actions like stopping a query execution and/or formulating a new one.

We have proposed a new database approach to data management in sensor networks (called MaD-WiSe) that attempts to overtake some of the typical limitations of systems recently proposed in this field. More specifically our approach addresses the following issues:

- in-network distributed processing of queries that relate data acquired by different nodes;
- execution of temporal aggregates;
- offering opportunities for network topology and data statistics aware query optimization;

Work funded in part by the European Commission in the framework of the "SatNEx" NoE project (contract N. 507052), and by MIUR in the framework of the Italian projects named ”IS-MANET” and ”VICOM”.

Giuseppe Amato is with the Istituto di Scienza e Tecnologie dell'Informazione del CNR, 56100 Pisa, Italy (phone: +39 050 3152906, fax: +39 050 3152811, e-mail: amato@isti.cnr.it).

Paolo Baronti is with the Istituto di Scienza e Tecnologie dell'Informazione del CNR, 56100 Pisa, Italy (phone: +39 050 3152887, fax: +39 050 3152811, e-mail: baronti@isti.cnr.it).

Stefano Chessa is with the Istituto di Scienza e Tecnologie dell'Informazione del CNR, 56100 Pisa, Italy, and with the Department of Computer Science, University of Pisa, via Buonarroti 2, 56100 Pisa, Italy (phone: +39 050 3152887, fax: +39 050 3152811, e-mail: chessa@isti.cnr.it).

For this purpose we employed a data model based on data streams [5] with very fine-grained granularity: data produced by a single transducer can also be modeled as a single data stream. The algebra of our query processor is composed of operators that take data streams as input and produce data streams as output. A query is represented as a combination of operators of the query algebra connected by data streams. Data streams can connect operators executed on different nodes, offering the opportunity of real distributed query processing (a query can be processed across several nodes in which different parts are independently executed). In our approach time is divided into epochs, and queries are repetitively executed in every epoch considering data produced in current epoch. Contrary to most of existing approaches, in an epoch several samples can be acquired by the same transducer depending on the query formulation.

The above model allows us to (in-network) process queries that relate data produced by different nodes, process temporal and spatial aggregates, and opens up new opportunities for more effective query optimization techniques that take into account network topology, transducer statistics, and costs.

In this paper we briefly describe the MaD-WiSe system and the query processing model, and we focus specifically on the user interface that we have designed to formulate, send and monitor query execution in a wireless sensor network.

## II. The MaD-WiSe System

The MaD-WiSe system allows the interaction with a sensor network as a relational database. It consists of two main parts. The first one runs on the sensors and is responsible for implementing the actual data acquisition, data processing and data forwarding activities requested in the network. The second part is a Graphical User Interface (GUI) application that runs on a PC connected to the sink and allows a user to program, by means of queries, the sensor network and to view the corresponding results.

The GUI application has two distinctive characteristics:

- it allows the user to graphically draw a query;
- it leverages on the ability of the sensor software of being *dynamically reprogrammable*.

The sensor software offers a set of functionalities that can be activated upon reception of command messages. By sending appropriate command messages, the GUI may request a specific sensor to execute operators of the query algebra and to manage data streams. Operators of a query

can be distributed on different sensors enabling in-network distributed query processing. By means of command messages the GUI may also request that the currently running query be stopped and replaced with another. In this way it is possible to change the behavior of the sensors on the basis of user needs without physically acting on them (i.e. without uploading a new firmware on the sensors). A user can do this simply interacting with the GUI. After starting a certain distributed query and observing the data coming back from the sensors, the user can decide to replace the current query with a new one. With a few mouse clicks he can define the new distributed query and instruct the GUI to send the appropriate messages to the sensor network. Next section briefly introduces the sensor architecture and the data model, and Section IV presents the GUI application.

## III. The Sensor Architecture

We implemented our system on the mica2 motes [6]. These sensors were developed at UC Berkeley and are widely used in academic environments. They are equipped with an 8 MHz microcontroller, 4 KB of data memory, 128 KB of instruction memory, a radio, and several transducers for light, temperature, acceleration and others. The operating system is TinyOs [7], [8] which is simple and offers basic hardware abstraction functionalities.

The sensor side of the MaD-WiSe system is organized into three layers, as depicted in Figure 1. The arrows indicate *use* relations among the layers. The layers interact through well defined interfaces and are autonomous with respect to each other. Each layer can be replaced with a new (different) implementation provided it complies with the existing interfaces.

The Network layer sits on top of the standard MAC layer of TinyOS. It provides 2 types of communication services to the above layers. It offers both a connectionless and a connection-oriented service. At network startup a distributed protocol assigns a tuple of virtual coordinates to each sensor which is used by a multi-hop geographic routing protocol [9]. The network layer also implements an application-driven energy efficiency protocol for the connection-oriented service [10].

The Stream System Layer offers abstraction mechanisms for data access by means of data streams. It can be thought of as the equivalent of a file system on a sensor network, the main difference being that, in the latter, data is continuously produced as a consequence of acquisition from transducers or processing. The basic concept is the *stream*: a unidirectional data channel implemented over the connection-oriented service of the Network layer. It carries a flow of records (in the simplest case each record contains a sensed data or a combination of data sensed by different transducers). A stream is implemented as a finite size queue that holds recently acquired records. The Stream System offers functionalities to create/remove streams as well as read and write records from/to existing streams.

There are three types of streams: *local*, *remote*, and *sensor* streams. A local stream is local to a sensor in the

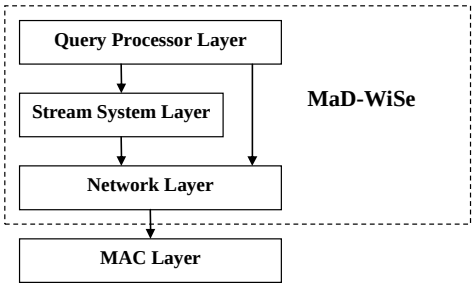

Fig. 1. Software layers.

sense that writing to and reading from the stream can only be requested by code running on the sensor. A remote stream is a data channel between two distinct sensors: writing to a remote stream happens on one sensor while reading from the stream happens on the other sensor. A sensor stream is directly connected to a transducer to carries-out data originated from it. Readings are only possible on this type of streams, given that the stream is automatically fed by the associated transducer.

The Query Processor Layer implements the query processor of a distributed database over the Stream System. It can be programmed remotely to take part in a distributed query execution. Queries are defined in terms of *operators* connected by *streams*. Streams play the role of relations (tables) and operators manipulate them similarly to relational algebra operators. However there are some fundamental differences. Tables are (more or less) static collections of records while streams are flowing records. Correspondingly, operators do not act on static relations but process records on-the-fly when they arrive. We have adapted some of the relational algebra operators to fit our data model.

During query processing the time is divided into epochs. In an epoch a transducer can be asked to acquire several values. The concept of epoch has been introduced to process blocking operators such as temporal aggregation operators. For instance, if the epoch is set to 10 minutes we can process queries asking for the average temperature during periods of ten minutes.

Defining a query for the query processor means defining what activities must be carried out by each sensor in the network. Among the activities we distinguish: data acquisition from local transducers, data processing and data forwarding. All these activities are expressed through streams and operators. Operators are active entities that take some inputs and produce an output. Inputs and output take the form of streams (we can think of streams as the means to connect operators). In more concrete terms an operator reads a record from its input stream(s), performs calculations/tests on the basis of its defining properties and the input record(s) and writes a (possibly) modified version of the inputs to the output stream.

## IV. The GUI Application

The GUI application is implemented in Java and runs on a PC connected to the sink. The purpose of the GUI is to let the user interact with the sensor network, graphically defining queries and viewing query results.

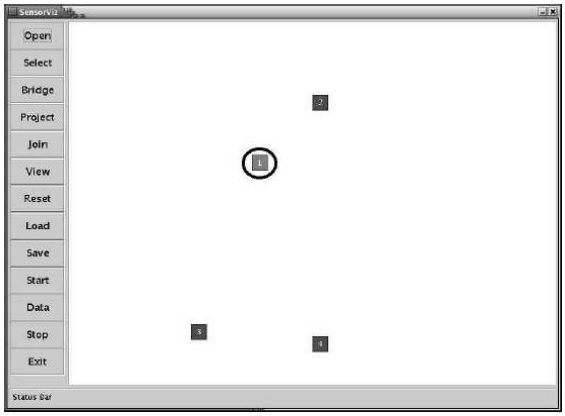


Fig. 2. GUI main frame.

Figure 2 illustrates a snapshot of the main application frame. As can be seen, three areas are identifiable: the sensor canvas (the largest, white, area), a tool bar on the left and a status bar on the bottom.

The sensor canvas is used to display the sensors. Each sensor is represented by a blue square containing its numeric id. The sink is distinguishable by its dark gray color (it appears circled in Figure 2). Clicking on a sensor square selects it, turning its color to red. Selecting a sensor is important since the interface buttons defining the query operate on the currently selected sensor. Since the position of the sensors is acquired from the sensors themselves we have an accurate representation of the network topology, including communication range.

The first 5 buttons of the tool bar allow the user to open new streams and connect existing streams through operators (i.e. they define queries). The next button (View) pops up a frame (the query canvas) showing what operators and streams are currently defined on the selected sensor, and depicts them graphically arranged in a tree (the query tree). Operators are represented by circles while streams appear as sticks (thick lines) connecting operators (see Figure 4 for a sample query and the query canvases of the sensors involved). In other words the query canvas illustrates what data sensing and data processing activities are currently defined on the sensor. The graphic tree-like representation of the query changes dynamically as new query objects are created so the user is visually aided in the query construction process. Colors and labels help in identifying the different stream types and operators.

The actual interaction with the sensor hardware begins when button Start is pressed. This instructs the application to send any command messages required to let the actual sensors open streams and set up operators as graphically defined by the user (i.e. it starts the query). In practice the GUI sends messages to the sink, which is physically connected to the PC. The query processor on the sink will relay these messages to the sensor network using the connectionless service from the Network Layer.

Button Data is only meaningful when a query is running and pops up a frame containing a text area where data records arriving from the sink are displayed one by one. By this functionality the user can monitor the query results.

The Open button allows the creation of a new stream.

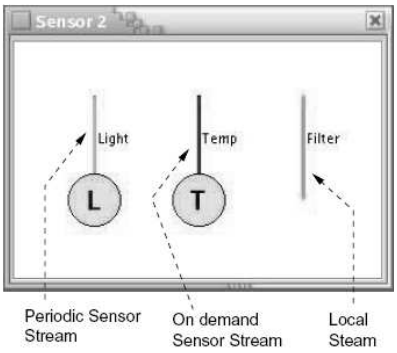


Fig.3 Creation of sensor and local streams.

Its parameters are the stream type (sensor, local, or remote) and a stream id. The stream id must be unique sensor-wide and it has two purposes: the identification of the streams on the graphical representation of the query tree (on the query canvas) and the choice of inputs and output when defining operators (see below).

When creating a sensor stream, the user must select the associated transducer (which appears as circle with a letter inside, indicating the type of transducer) and the sampling type which can be *periodic* or *on demand*. Periodic sensor stream have an associated sampling rate that indicates the period between consecutive readings from the transducer. On the contrary, an on demand sensor stream has no sampling rate since it is not sampled automatically. A reading from the transducer must be explicitly requested by the operator reading from the stream.

For a local stream there is no other information to supply besides the stream id. Figure 3 illustrates the query canvas of sensor 2 after creation of a periodic sensor stream for light (named "Light"), of an on demand sensor stream for temperature (named "Temp"), and a local stream (named "Filter").

When creating a remote stream, the user has to specify a remote sensor, the stream id used on that sensor, and a global id for the stream which must be unique network-wide (to let the two endpoints recognize the same network channel). A remote stream is graphically represented with 2 sticks: one in the query canvas of the local sensor and another in the query canvas of the remote sensor.

We now describe how operators can be requested.

A dialog for operator Select ($\sigma$) is activated by button Select. Operator $\sigma$ reads records from its input stream, evaluates a predicate on its fields and writes it unchanged to the output stream only if the predicate is satisfied. Through the Select dialog the user chooses the input and output streams from a combo box listing the available streams (not already in use by other operators). The combo boxes used for stream selection identify the streams through the symbolic ids that the user assigned them at stream creation time. The user finally selects the predicate type (one of $=$, $\neq$, $<$, $\leq$, $>$, $\geq$), the left operand (one of the fields of the input record) and the right operand (another field of the input record or a numerical constant).

Button Project pops up a dialog for defining operator Project ($\pi$). Its purpose is to read records from its input stream, select some of the fields and write a new record

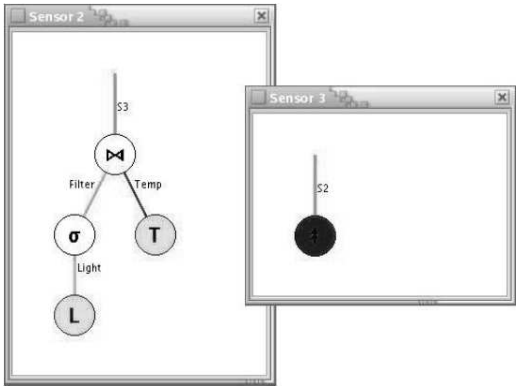


Fig 4. Creation of a ⋈ operator.

containing only the selected fields to the output stream. The dialog has combo boxes for selecting the input and output streams as well as the project fields.

Operator ⇑ (button Bridge) is a convenience operator that transfers records unmodified from its input stream to the output stream. Its main use is to pass records from a sensor stream directly to a remote stream.

A more complex operator is Join (⋈). It has 2 input streams which the user selects from combo boxes. Records from both input streams must have a timestamp field (which is always the case unless it was removed by a π operator). The purpose of ⋈ is to join records from the input streams on the basis of a common timestamp value. The resulting record contains all fields from both input records.

There are 2 implementation strategies for the Join operator. In the *sync-join* implementation one of the input streams is an on demand sensor stream and the other stream is any other type of stream (we call it the *driving stream* since it drives the join process). When a record arrives on the driving stream, the ⋈ operator asks for a record from the on demand sensor stream and makes the join. The other implementation is the *merge-join*. In this case neither of the two input streams is on demand. Records can arrive on either stream at any time. Upon receiving a record from any of the input streams it attempts to combine it with a record from the other stream.

The reason to provide on demand sensor streams is energy efficiency. Figure 4 illustrates the query canvases of sensors 2 and 3 after we define a ⋈ operator on sensor 2 with "Filter" as the driving stream, "Temp" as the on demand sensor stream and "S3" as the output. What happens on sensor 2 is that light is read periodically (unconditionally) but the readings are filtered by operator σ. Only if the light reading passes the filter predicate the record is written to stream "Filter" and reaches operator ⋈. Upon receiving a record on stream "Filter" (and only upon this event) ⋈ requests a temperature reading (stream Temp) and joins it with the light reading. The temperature transducer only operates when needed. This could produce significant energy savings if light rarely passes the filter predicate and the temperature transducer consumes more power than the light transducer.

Figure 4 shows that ⋈ writes on the remote stream with id "S3" with destination on the sensor 3. In fact the query

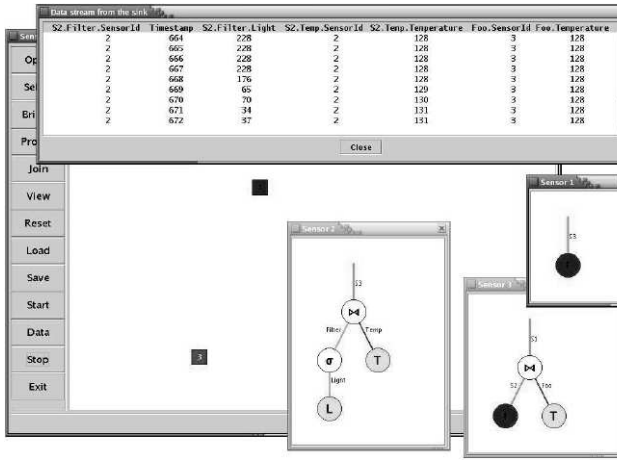


Fig 5. Query results.

canvas of sensor 3 shows the remote stream (locally called "S2"), which is attached to a fictitious "network" operator to indicate a (remote) writer for "S2".

Figure 5 shows a complex query involving three sensors, two ⋈ and one σ operators, and the respective query result on the frame activated by the Data button.

Additional operators for computing spatial and temporal aggregates can be requested as well. We omit their description given space limitations.

## V. Conclusion

We have described the MaD-WiSe system and its user interface. The MaD-WiSe system is under further development to include advanced query optimization strategies and efficient query injection.

## References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: a Survey", *Computer Networks*, vol. 38, no. 4, pp. 393–422, March 2002.

[2] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor For Sensor Networks", In *2003 Proc. SIGMOD Conf.*, pp. 491.502.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", in *2002 Proc. Symp. on Operating Systems Design and Implementation (OSDI)*.

[4] Y. Yao and J. E. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks", *Sigmod Record*, vol. 31, no. 3. Sept. 2002.

[5] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. B. Zdonik: "Aurora: a New Model and Architecture for Data Stream Management", *VLDB J.* vol. 12, no. 2, pp. 120–139, 2003.

[6] Crossbow Technology Inc. Mica2 website. http://www.xbow.com/products.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors", in *2000 Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pp. 93–104.

[8] TinyOs Community Forum. Tinyos website. http://www.tinyos.net.

[9] A. Caruso, S. Chessa, S. De, and A. Urpi, "GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks", in 2005 *Proc. IEEE INFOCOM*.

[10] G. Amato, P. Baronti, and S. Chessa, "Connection-Oriented Communication Protocol in Wireless Sensor Networks", Technical Report 2005-TR-10, Istituto di Scienza e Tecnologie dell'Informazione - CNR, Pisa, Italy, 2005.