Declarative Interface for In-network Actuation on Wireless Sensor-Actuator Networks

Asanka Sayakkara*, M.D.J.S. Goonetillake† and Kasun De Zoysa‡
*asanka.code@gmail.com, †jsg@ucsc.cmb.ac.lk, ‡kasun@ucsc.cmb.ac.lk
University of Colombo School of Computing,
No. 35, Reid avenue,
Colombo 7, Sri Lanka.

***Abstract*—Declarative SQL queries are a way of abstracting out the underlying complexity in Wireless sensor-actuator networks (WSAN). Current SQL syntax to perform actuation tasks misleads the user because of their artificial way of expressing the actuation tasks thus violating the purpose of the declarative nature. Particularly with respect to distributed in-network actuation tasks where multiple sensor nodes and actuator nodes are involved, the user may be required to write multiple queries based on the current actuation query syntax causing a negative impact on the performance. We address these issues in existing declarative layer with modifications to the virtual data table of the abstraction and thus a new syntax is suggested and implemented.**

I. Introduction

Wireless sensor networks(WSN) [10] consists of small wireless enable embedded devices called motes in large numbers. They are used for tasks such as environmental monitoring and tactical surveillance[13], [15]. Motes are capable of taking readings from their built-in sensors and transmitting to a specified location for further processing and analyzing. By the nature of their applications, motes have to be low cost and should be able to run on battery power making them resource constrained in all aspects including memory capacity and CPU power. While motes are capable of sensing the environment, various real world applications raise the requirement of controlling the environment by performing actions based on sensor readings. To this end, actuators such as switches and regulators can be connected to motes to perform actuation tasks. Such networks are known as wireless sensor-actuator networks (WSAN)[1].

Due to hardware limitations of motes it is necessary to write low level application codes carefully managing the memory and saving battery power as much as possible for a longer service life. One approach of making the WSN users life easier is high-level programming abstractions which hide the low level hardware details. Database abstraction for WSNs [3] like Cougar [16] and TinyDB [12] are such abstractions where the whole network is represented as a virtual database table and each column of the table is mapped to a sensor type available in the network.

In such implementations, the data of the network is acquired by running SQL queries over the virtual table. SQL enables the user to specify which data to be acquired from the virtual table in a declarative way without bothering how the data is acquired. However with respect to complex sensing associated with actuating requirements, current declarative interface for sensor networks exhibit serious weaknesses making them unusable for such scenarios. Moreover, there's no easy or declarative mechanism to obtain meta level information on actuators in WSAN which is important for actuator monitoring purposes. We illustrate a scenario with respect to a greenhouse cultivation [4] for the purpose of pinpointing the weaknesses in the current approach.

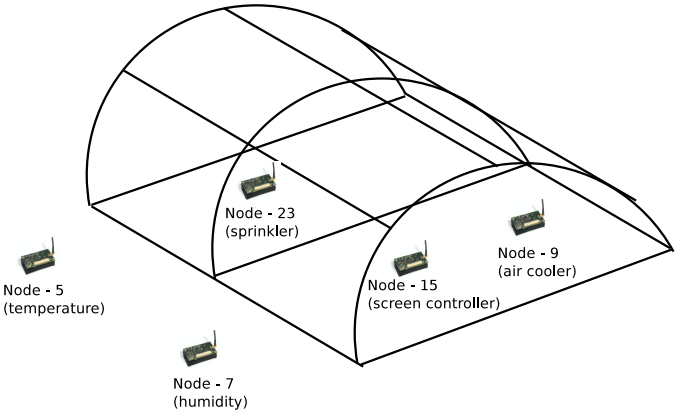

Fig. 1. For example a greenhouse with multiple sensor and actuator nodes. The actuators include an air cooler, a screen controller and a sprinkler which are fixed inside the greenhouse while the sensors include a temperature sensor and a humidity sensor which are placed outside the greenhouse. Only the nodes relevant to the queries are given for simplicity.

A. An Actuation Scenario

In a greenhouse environment it is particularly crucial to manage the environment effectively by controlling the environmental parameters such as temperature, light, humidity, CO_2, ambient pressure and wind flow. There are actuators to control these parameters and these may include screen controllers/carton sliders to protect the cultivation from direct sunlight by regulating the amount of sunlight entering to the greenhouse, heaters and air coolers to maintain humidity and temperature, fans to maintain the wind flow and also actuators to inject CO_2 to influence photosynthesis.

In order to obtain the real benefit of deploying WSAN in this context the end user should be able to monitor and thus to control the environmental parameters (which may subject to dynamic changes) through controlling relevant actuators

accordingly. By the nature of the application, the network may have three types of motes which are motes with sensors only, motes with actuators only and motes with both sensors and actuators.

Consider as such a scenario where a WSAN deployed for a greenhouse to monitor the condition of the field continuously and to perform actions when necessary. Assume that this network employs a TinyDB [12] like abstraction to monitor the network since TinyDB provides more facilities for WSAN. If climate sensors (say motes 5 and 7) fixed outside the greenhouse sense an outside temperature greater than $35\,^{\circ}\mathrm{C}$ and humidity level greater than 45, the user requires to switch on an air cooler, a screen controller and a sprinkler attached to motes 9, 15 and 23 respectively which are fixed inside the greenhouse. Figure 1 depicts this greenhouse environment.

To check the relevant sensor readings of the relevant nodes, user has to send Query-1 on TinyDB abstraction layer. If this query generate results from motes 5 and 7, that means the sensors of our concern have the suitable values to perform the actuation. Based on that results, the user has to send actuation queries as shown in Query-2, 3 and 4 targeting at each node and their actuators accordingly.

Query-1

```
SELECT nodeid
FROM sensors
WHERE (nodeid=5 OR nodeid=7)
  AND temperature>35 AND humidity>45
ONCE;
```

Query-2

```
SELECT nodeid
FROM sensors
WHERE nodeid=9
OUTPUT ACTION air-cooler-on()
ONCE;
```

Query-3

```
SELECT nodeid
FROM sensors
WHERE nodeid=15
OUTPUT ACTION screen-ctrl-on()
ONCE;
```

Query-4

```
SELECT nodeid
FROM sensors
WHERE nodeid=23
OUTPUT ACTION sprinkler-on()
ONCE;
```

TinyDB's query language follows conventional SQL syntax to acquire data from the virtual database table with some additions to support exclusive requirements in sensor networks as explained in detail in the following section.

B. Problem Of Actuation Queries

We identify three main issues in current actuation queries in the declarative interface. Firstly when performing the actuation with existing queries, user has to write *SELECT* queries with the *OUTPUT ACTION* clause. Due to this new clause, the query does not return any data requested by the *SELECT* query and instead calls the low level function to perform the actuation. However when writing or reading *SELECT* queries, semantically it represents displaying of data relevant to the stated columns with respect to a database table. However, these actuation queries performed by enhanced *SELECT* queries are not intended to display or return any data thus confusing the query writer. This is due to the misuse of 'SELECT nodeid' when the exact node id is defined in the WHERE clause as in Query-2, 3 and 4. Therefore the user does not receive the true advantage of declarative interface when the syntax is in this type of an artificial way.

Secondly, when multiple sensors and actuators are to be involved in an actuation task as the given scenario, the user may have to write number of queries for checking different sensor conditions before actuating different actuators. This multiple query usage leads to time delays as human intervention is required to issue each query from a sequence of queries. Moreover if the user forgets/misses to issue a query from the corresponding query sequence the expected control outcome cannot be achieved. As such multiple query usage is not recommended if time delays between sensing and actuating is required to be as minimum as possible for time critical tasks. Additionally multiple queries results in more power usage of nodes reducing the service life of the network. These issues become worse when the number of nodes in the network grows in large numbers.

Finally since the existing data model of declarative interface does not provide an abstraction over actuators, they have to be activated by calling low level functions. Therefore existing data model does not provide any easy mechanism to find the status of actuators if actuation tasks have to be done based on the current status of actuators. In this paper we provide a solution to overcome the issues associated with existing declarative layer. The rest of this paper is organized as follows. In Section II, we thoroughly analyze different actuation query syntax and their weaknesses in available declarative interface mainly based on TinyDB [12]. The Section III introduces our solution approach addressing those issues while Section IV and V evaluate and discuss this approach. Finally we provide details of related works in Section VI before concluding the paper.

II. Declarative Queries For Actuation

This section contains the basic syntax variations of TinyDB which can be used to perform actuation tasks in different ways. When performing an actuation using TinyDB query language, we mainly identify three different approaches based on it's syntax definitions. The applicability of each of these approaches depends on the nature of actuation scenario we face. In this section we look into these three approaches to highlight their strengths and weaknesses in different actuation requirements.

A. Approach - 1

The simplest way of performing an actuation using TinyDB abstraction is a *SELECT* query with *OUTPUT ACTION* keyword which calls a low level function as the example query shown below.

Query-5

```
SELECT nodeid
FROM sensors
WHERE nodeid=9 AND temperature > 25
OUTPUT ACTION air-cooler-on()
SAMPLE PERIOD 1s FOR 10s;
```

Immediately after each node in the network receives this query it is being executed. The important fact to keep in mind regarding the above query type is that the sensor and the actuator which gets called by the function (here it is *air-cooler-on()*) should reside in the same node. This is because when a node execute a query it has access only to sensors and actuators which are connected to that particular node. Because of that reason this type of queries can refer to sensors and actuators residing in the same node. Additionally in this syntax only one function can be called with the *OUTPUT ACTION* clause making it unable to control multiple actuators in a node by a single query.

The *SAMPLE PERIOD* keyword is used in TinyDB to specify the sampling rate of data. When executing the above query at node 9, *temperature* sensor readings are taken at each second starting from the time the query received for a period of 10 seconds. If any of those readings were greater than 25, then the *air-cooler-on()* function gets called.

B. Approach - 2

In this approach we describe queries that get executed only if a particular event is occurred in the environment where the nodes are deployed. The following query is an example where the enclosed *SELECT* query gets executed only if the event named as *light-high()* is detected by the node.

Query-6

```
ON EVENT light-high():
SELECT nodeid
FROM sensors
WHERE nodeid=9
OUTPUT ACTION screen-ctrl-on()
ONCE;
```

These event does not have to be always triggered by hardware components of the node. Another declarative query like the one shown below which received to the node can signal an event causing the previous query to get executed. The *SIGNAL* clause takes the event name as a parameter which is triggered when this query is executed on the same node where the previous query is stored.

Query-7

```
SELECT nodeid
FROM sensors
WHERE nodeid=9 AND light > 40
OUTPUT ACTION SIGNAL light-high()
ONCE;
```

These event based queries can be used to set scheduled actuations which are performed according to the events occurred in the environment.

C. Approach - 3

Another way of performing an in-network actuation in TinyDB interface is the nested *SELECT* queries. This nesting is done using the *OUTPUT ACTION* clause as shown in Query-8. When this query is received to a node in the network, it immediately start executing the query. If the predicate in the first *WHERE* clause evaluates to true, then this node perform the action specified with the *OUTPUT ACTION* clause which is a nested *SELECT* query. This nested *SELECT* query is broadcasted to the network as a new query which will perform an actuation on the same or another node in the network. Distributed in-network actuation tasks can be performed by using such nested queries in TinyDB since the sensors and actuators do not have to be residing in the same node. Usage of nested *SELECT* queries still suffer from the weaknesses we have explained previously like the inability to control multiple actuators of a same node from a single query.

Query-8

```
SELECT nodeid
FROM sensors
WHERE nodeid=9 AND light > 40
OUTPUT ACTION
(
SELECT nodeid
FROM sensors
WHERE nodeid=9
OUTPUT ACTION screen-ctrl-on()
ONCE
)
SAMPLE PERIOD 10s;
```

While being used to acquire data from the sensor network, TinyDB's *SELECT* query has been used to perform actuation tasks too. It is evident that all the declarative database abstraction layers including TinyDB (designed for wireless sensor networks) have not given enough functionality to perform actuation tasks. This will affect the wireless sensor-actuator networks for which actuation tasks play a major role. Therefore providing the support to perform complex actuation scenarios while preserving the declarative nature of SQL queries in the database abstraction is an important requirement in wireless sensor-actuator networks.

III. Our Approach

In our attempt to solve the identified issues in declarative interface, we begin with the current data model (Table I) and build a better syntax support for actuation queries. In this process we consider requirements of distributed in-network actuation scenarios in order to come up with a solution that could be applicable in all such cases.

TABLE I
TRADITIONAL SENSOR TABLE

nodeid	*temperature*	*humidity*
1	25	40
2	28	39
3	27	NULL
4	NULL	41

TABLE II
ENHANCED TABLE WITH ACTUATORS INTEGRATED

nodeid	*temperature*	*humidity*	*air_cooler*	*sprinkler*
1	25	40	NULL	NULL
2	28	39	off	off
3	27	NULL	on	NULL
4	NULL	41	off	on

A. Virtual Data Table

Table I shows an example virtual table of a database abstraction over a wireless sensor-actuator network. According to the table, there are four nodes in the network and each node has two sensor types, temperature and humidity. The values of those sensors shown in the table are acquired in a particular instance of time by a *SELECT* query. If a node in the network does not have a particular sensor type the corresponding cell in the table will contain a NULL value.

Now consider each node in the network is equipped with two actuator types namely an air cooler and a sprinkler which are controlled electronically. Since the actuators in the network are not represented in the table, still the virtual table will look like Table I while the user have to control actuators using low level function calls in *OUTPUT ACTION* clause of *SELECT* query. Moreover, there's no way for the user to find the current status of each actuator in the network.

To overcome the problems associated with the current mechanism and thus to provide the benefits of declarative interface on in-network actuation we understand the importance of enhancing the virtual table of the database abstraction with actuator types. As such the status of each actuator type in the network is represented as an attribute in the virtual table in addition to sensor types as shown in Table II. Most significantly this enables user to access actuators in each node just as the way sensors are accessed. Consequently when the user wants to know the current status of a particular actuator or a set of actuators in the network, simply *SELECT* queries as given below will do the job.

Query-9

```
SELECT nodeid, air_cooler, sprinkler
FROM sensors
ONCE;
```

Query-10

```
SELECT nodeid
FROM sensors
WHERE air_cooler='on'
ONCE;
```

Query-11

```
SELECT count(*)
FROM sensors
WHERE sprinkler<>NULL
ONCE;
```

B. Actuation Using Updates

Since sensor attribute values in the virtual table represent real time acquired data of the sensors, they can not be altered by the user. However actuator attribute values in the virtual table are alterable and this enables the actuator status to be changed according to the user requirements. This situation opens the door to control actuators in the network by applying updates on the virtual table using conventional *UPDATE* query syntax without using low level function calls as explained previously. However since these queries are executed over a WSAN, it is necessary to have the syntax support to specify the sampling rate of sensor /actuator attributes just as in the existing approach.

For instance if the user wants to turn the air cooler in node 2 to 'on' state only if the temperature reading of that node is greater than 20, a query like the one shown in Query-12 can be used.

Query-12

```
UPDATE sensors
SET air_cooler='on'
WHERE nodeid=2 AND temperature>20
ONCE;
```

When this query is sent to all the nodes in the network, each node will evaluate the predicates in *WHERE* clause. If a node reveals that those predicates are true on it, it can turn the air cooler actuator connected to it to 'on' state. This illustrates a *local in-network* actuation scenario.

C. Distributed In-network Actuation

When we need to perform in-network actuation tasks where multiple sensor nodes and actuator nodes are involved in a single actuation, simple queries like the Query-12 can not be used. For instance consider the requirement that we need to turn the air cooler to 'on' state in both nodes 2 and 4 only if node 1 is having a temperature reading greater than 20. We consider the join of multiple table aliases can be used to address this requirement with the actuation criteria specified in *WHERE* clause of the *UPDATE* query as given in Query-13.

Query-13

```
UPDATE sensors AS sen, sensors AS act
SET act.air_cooler='on'
WHERE (act.nodeid=4 OR act.nodeid=2) AND
  (sen.nodeid=1 AND sen.temperature>20)
ONCE;
```

Two aliases for the *sensors* table have been taken as *sen* and *act*. We use the *sen* alias to specify the criteria of the sensor nodes while the *act* alias to specify the criteria of actuator

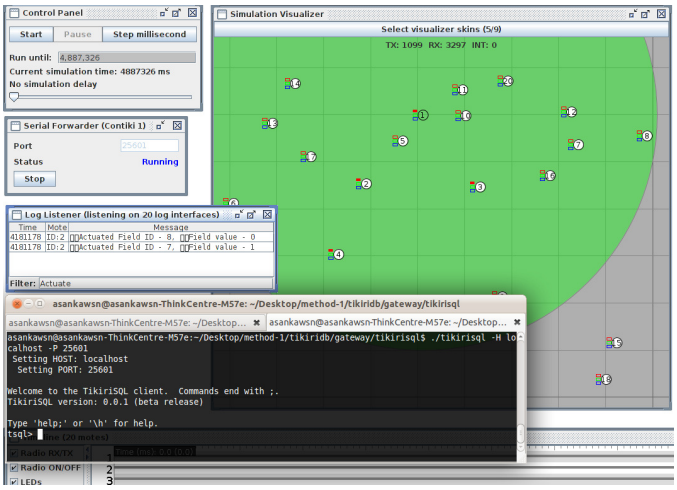

Fig. 2. Prototype implementation based on TikiriDB[9] on Contiki OS[6]. Commandline client accepts SQL queries, parse it and send to the WSAN while the sensor-actuator nodes execute the query packets received over the network.

nodes. This enables distributed in-network actuation tasks with different complex actuation criteria using a single *UPDATE* query unlike the existing actuation mechanism with *SELECT* queries. The grammar definition of this *UPDATE* query syntax is shown below.

```
UPDATE <table> [AS <alias>]
    {, <table> [AS <alias>]}*
SET [<alias>.]<attribute>=<value>
    {, [<alias>.]<attribute>=<value>}*
[
WHERE
 [<alias>.]<attribute>{<|>|=|<=|>=}
    <value>
 { {AND|OR}
   [<alias>.]<attribute>{<|>|=|<=|>=}
   <value>
 }*
]
[ SAMPLE PERIOD <seconds>
  [FOR <nrounds>] ] | [ONCE]
```

IV. Experiments and Results

For evaluating the proposed in-network actuation mechanism we mainly considered three aspects. First aspect is the applicability. There our concern is whether our proposed actuation mechanism is able to cater for any type of actuation requirement that can occur in wireless sensor-actuator networks. Second aspect is the comparison of the declarative nature of the proposed syntax in writing queries for complex actuation tasks. The third aspect is the impact on the performance when using the proposed mechanism.

A. Applicability

To evaluate the applicability of the new syntax for actuation we consider mainly three types of actuation scenarios that can occur in a wireless sensor-actuator network. Those are hybrid sensor-actuator nodes scenario, a pair of sensor nodes and actuator nodes scenario and multiple sensor nodes and actuator nodes scenario. Our goal is to show that each of these actuation scenarios can be addressed by the suggested syntax.

1) Hybrid sensor-actuator node scenario: In this case both sensors and actuators of our concern are in the same node of the network. For instance, consider the node 2 in the virtual table showed in Table II. We need to turn its air cooler and sprinkler to 'on' state if its temperature reading is greater that 25 and humidity value is lower than 45.

We can perform this actuation by using a query like the one given in Query-14.

Query-14

```
UPDATE sensors
SET air_cooler='on', sprinkler='on'
WHERE nodeid=2 AND
  temperature>25 AND humidity<45
ONCE;
```

2) A Pair of sensor node and actuator node scenario: The simplest distributed in-network actuation scenario is the involvement of two nodes where the required sensors are in one node and the required actuators are in another node. For instance we need to switch off the sprinkler of node 4 only if the node 1 has a temperature value greater than 20. We perform this actuation task by using the Query-15.

Query-15

```
UPDATE sensors AS sen, sensors AS act
SET act.sprinkler='off'
WHERE act.nodeid=4 AND
  sen.nodeid=1 AND sen.temperature>20
ONCE;
```

3) Multiple sensor nodes and actuator nodes scenario: In this case we consider actuation requirements where multiple sensor nodes and actuator nodes are involved in an actuation task. For instance, we need to switch off sprinkler in node 4 and switch on air cooler in node 2 only if all the nodes in the network which are having temperature sensors report a value greater than 25. We perform this actuation task as shown in Query-16.

Query-16

```
UPDATE sensors AS sen,sensors AS act1,
    sensors AS act2
SET act1.sprinkler='off',
    act2.air_cooler='on'
WHERE act1.nodeid=4 AND
  act2.nodeid=2 AND
  sen.temperature>25
ONCE;
```

B. Declarative Nature

Moreover we evaluated the declarative nature of the suggested syntax over the existing syntax by a user survey. We randomly selected a group of 10 computer science students who have an average knowledge on SQL and acquisitional

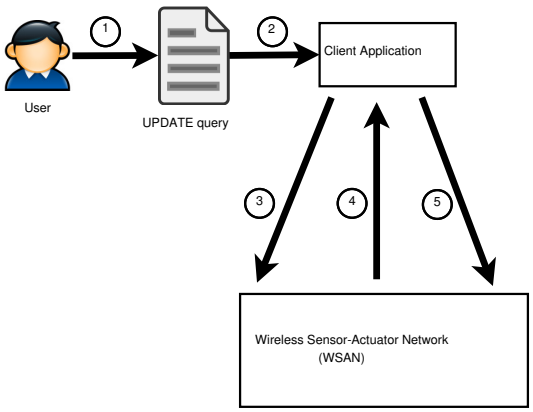

Fig. 3. Execution of an *UPDATE* query. (1) User writes an *UPDATE* query. (2) Client application accepts the query and parse it. (3) Client application asks the necessary sensor values and actuator status from the network. (4) Nodes in the network respond with requested data. (5) Client application internally process the received data, identify the nodes which should perform actuations and send the actuation commands back to the WSAN.

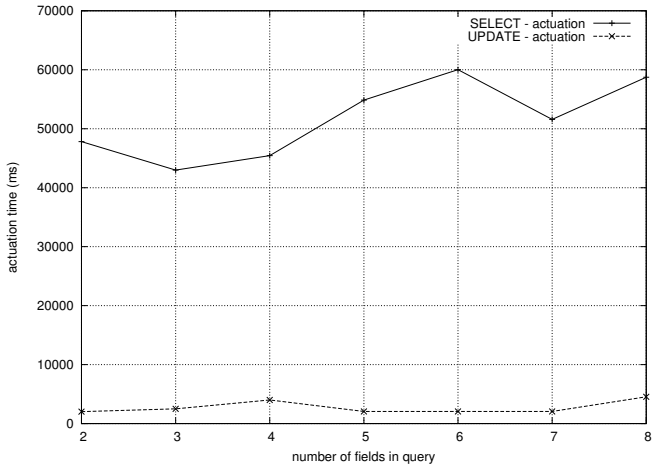

Fig. 4. Performance comparison between actuation tasks performed by using existing method (multiple SELECT queries) and and the suggested method (UPDATE queries). x-axis contains the number of fields which are involved in query while y-axis contains time to complete the actuation task.

queries and we provided them with different actuation scenarios. We asked them to write SQL queries to perform those actuation tasks in both existing and the suggested approaches. Finally we collected feedback from them regarding how they found writing queries for actuation tasks in both approaches.

It turned out that each person participated in the survey considers the existing approach of performing actuation tasks has unnecessary syntax restrictions. All the participants consider the modified virtual table with actuators as attributes is more convenient to handle than the old table. As such they have mentioned that *UPDATE* queries can be easily written to perform actuation tasks naturally without using artificially forced syntax parts of *SELECT* queries in existing approach. We further noticed that most of them have struggled to write queries using the existing syntax for the greenhouse scenario given in Section I whilst they have written with a single query for the same scenario as given in Query-17 using the proposed syntax.

Query-17

```
UPDATE sensors AS sen1,sensors AS sen2,
  sensors AS act1,sensors AS act2,
  sensors AS act3
SET act1.air_cooler='on',
  act2.screen_ctrl='on',
  act3.sprinkler='on'
WHERE
 (sen1.nodeid=5 AND sen1.temperature>35)
 AND
 (sen2.nodeid=7 AND sen2.humidity>45)
 AND
 act1.nodeid=9 AND act2.nodeid=15 AND
 act3.nodeid=23
ONCE;
```

C. Performance Impact

For evaluating the performance impact with respect to the new mechanism, we used two prototype implementations based on TikiriDB[9] which is a database abstraction implemented on Contiki OS[6]. TikiriDB has three components namely a *node application*, a *serial forwarder* and a *client application* which are shown in Figure 2. The *client application* accepts SQL queries from the user, parses them and forward to the serial forwarder. The *serial forwarder* runs on the PC which is connected to the base station of the WSAN. Its task is to accept the queries which are coming from the *client application* and forward them to the WSAN. Each node in the network runs a copy of the *node application* which executes the queries and generates results.

First prototype implementation provides the suggested *UPDATE* query syntax while the second prototype implementation provides the *OUTPUT ACTION* clause with *SELECT* query. When the user issues a *SELECT* query, it is parsed and sent to the network. Nodes respond with data which are provided to the user. When an *UPDATE* query is issued by the user, *client application* first collects the readings of the necessary sensors and status of necessary actuators. Then it processes that data to identify the nodes which have to change their actuator status according to the query. Then *client application* sends commands for those nodes to update their actuator status. This functionality of the *UPDATE* query is shown in Figure 3.

With these two prototype implementations we evaluated whether there's any performance impact when actuation tasks are performed using the suggested syntax comparing it to the existing syntax. We considered various actuation scenarios which involve different number of attributes in the virtual table. With respect to these scenarios we used queries from both existing and suggested approaches. Figure 4 shows the time to perform an actuation against the number of sensor / actuator

attributes which are involved in the actuation task. There's a significant performance difference between the two approaches since traditional approach requires human intervention to issue multiple queries to perform an actuation task while new syntax eliminate multiple query usage.

V. Discussion

Adding a list of functions to TinyDB *SELECT* query syntax is a possible way to reduce the number of queries used for an actuation task. However this option still employes low level functions within *SELECT* queries to perform actuation tasks without addressing the identified problems. Instead of adding both sensors and actuators to the same virtual table another possible approach would be to consider two separate virtual tables for sensors and actuators. However, this may lead to unnecessary complexities when performing an actuation task on a node which has both sensors and actuators. This is because the sensors and actuators of same node will reside in two tables and joins will be required to handle them. For the sake of simplicity we considered a single virtual table with both sensors and actuators.

Due to the limitations in the syntax when we need to perform an action based on an aggregated data, the *UPDATE* query alone is not applicable. Using *SELECT* and *UPDATE* queries collectively is necessary in such cases. A possible solution would be to extend the current *UPDATE* query syntax for nesting *SELECT* queries within it even though executing such queries on the network would be extremely difficult.

According to the user survey, performing actuation tasks with an *UPDATE* query is more convenient than using *SELECT* queries with low level function calls. The declarative nature of *UPDATE* syntax makes it easy to express complex actuation tasks within a single query when compared to the traditional approach where multiple queries may have to involve with system dependent function calls. Since the enhanced virtual table contains actuators as attributes, the user gets the opportunity to easily find out the current status of actuators in any given time. This also enables the user to perform in-network actuation tasks not only based on sensor values but also based on current actuator status.

As we have shown, *UPDATE* query syntax can be used to perform almost any complex actuation requirement that can arise in a wireless sensor actuator network(WSAN). The syntactic support along with the aliases can be used to coordinate multiple nodes with sensors and actuators by writing a single *UPDATE* query. Comparing to the traditional approach, this single query usage to perform an actuation has improved the performance since multiple queries take more time to execute with the additional time taken by human involvement. This enables the WSAN to respond quickly for changes in the environment.

Even though it appears like having NULL values in the virtual data table where some sensors or actuators are absent on a particular node as a drawback, it does not introduce any performance impact. When a node receives a query asking for some sensor / actuator attribute value which is not available on the node, the node sends only the values of the attributes which are available on the node. A Node does not send any messages to the *client application* to inform that the value of a particular sensor / actuator attribute is NULL. Because of this reason, having lot of NULL values in the virtual data table does not introduce any communication or any other resource overhead on the network.

Just like the sensors in the virtual table, the addition of actuator status does not add any extra storage requirement to nodes in the network. Current status of an actuator can be identified at run-time of a query on a particular node without a need to store the status of actuators on the memory of a node.

In our approach, client application is still a single point of failure just like the traditional approach since all the sensor and actuator coordination to perform an actuation task is done by the client application side. We are planning to improve the way an *UPDATE* query is executed in the future so that sensor-actuator coordination is performed within the network without any intermediate involvement of the client application.

VI. Related Works

With the introduction of the relational model[5] it has become the ideal way of storing data in large amounts. The *structured query language* or SQL[2] is the most widely used language for efficiently express the criteria of querying data from these relational databases. Because of the underlying complexity in wireless sensor networks, researchers have suggested the idea of abstracting WSN as a database[3], [7]. One of the earliest such implementations is Cougar[16] which had declarative queries to acquire sensor data. However, Cougar did not consider the possibility of actuators present in the network. The first implementation which supported handling actuators in addition to sensors was TinyDB[12], [11] which we discussed in the first two sections in this paper. However weaknesses in TinyDBs data model and syntax has made it difficult to use in complex in-network actuation scenarios.

According to [1], a proper coordination between nodes in the network is necessary to achieve the in-network actuation in a WSAN. In addition to the database abstraction approach there are different other techniques used to coordinate and perform in-network actuation tasks in WSANs [8], [14]. The advantage of database abstraction over these approaches is the declarative query interface which is more easier to use by non-technical users.

VII. Conclusions

In this paper we presented a new approach to perform actuation tasks in a wireless sensor-actuator network (WSAN) with the declarative database abstraction. Traditional approach lack support to perform complex actuation tasks and also the declarative nature of its syntax is questionable. The enhancements to the virtual data table with new syntax suggestions eliminate those weaknesses in the existing approach. Our evaluations show that in this new way of performing actuation tasks, queries can be easily written and actuation tasks are

performed in less time when compared with the traditional approach.

ACKNOWLEDGMENT

The authors would like to thank Chathura Suduwella and Lakmal Weerawarne for their useful suggestions in the initial work related to this research.

REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351 – 367, 2004.

[2] M. M. Astrahan and D. D. Chamberlin. Implementation of a structured english query language. *Commun. ACM*, 18:580–588, October 1975.

[3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, MDM '01, pages 3–14, London, UK, 2001. Springer-Verlag.

[4] D. D. Chaudhary, S. P. Nayse, and L. M. Waghmare. Application of wireless sensor networks for greenhouse parameter control in precision agriculture. In *International Journal of Wireless and Mobile Networks*, IJWMN, 2011.

[5] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[6] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. *Local Computer Networks, Annual IEEE Conference on*, 0:455–462, 2004.

[7] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database, 2002.

[8] M. Karpiński and V. Cahill. Stream-based macro-programming of wireless sensor, actuator network applications with sosna. In *Proceedings of the 5th workshop on Data management for sensor networks*, DMSN '08, pages 49–55, New York, NY, USA, 2008. ACM.

[9] N. M. Laxaman, M. D. J. S. Goonatillake, and K. D. Zoysa. Tikiridb: Shared wireless sensor network database for multi-user data access. *CSSL*, 2010.

[10] F. L. Lewis. Wireless sensor networks. In *Smart Environments: Technologies, Protocols, and Applications*, New York, NY, USA, 2004.

[11] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 491–502, New York, NY, USA, 2003. ACM.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.

[13] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pages 88–97, New York, NY, USA, 2002. ACM.

[14] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz. A distributed coordination framework for wireless sensor and actor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 99–110, New York, NY, USA, 2005. ACM.

[15] A. Sayakkara, W. Senanayake, K. Hewage, N. Laxaman, and K. De Zoysa. The deployment of tikiridb for monitoring palm sap production. In *Real-World Wireless Sensor Networks*, volume 6511 of *Lecture Notes in Computer Science*, pages 182–185. Springer Berlin / Heidelberg, 2010.

[16] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31:9–18, September 2002.