Volume 1, 2025, Pages 140-150 *Journal of Computer and Communication Networks*
| Regular Article | Open Access

A Sensor Query and Tasking Language Approach to Event Driven Programming in Sensor Networks

Kanev Boris Lisitsa
Faculty of Science and Engineering, University of Liverpool, UK.
borislisitsa@hotmail.com

Article Info
Journal of Computer and Communication Networks
https://www.ansispublications.com/journals/jccn/jccn.html

https://doi.org/10.64026/JCCN/2025014

Received 16 April 2025
Revised from 30 May 2025
Accepted 25 June 2025
Available online 10 July 2025
Published by Ansis Publications

Corresponding author(s):
Kanev Boris Lisitsa, Faculty of Science and Engineering, University of Liverpool, UK.
Email: borislisitsa@hotmail.com

Abstract – Sensor Query and Tasking Language (SQTL) is a computational scripting language that is used to enhance the communication between middleware and sensor applications of a sensor network. It provides versatility and a compact script for receiving sensor data and utilizing hardware characteristics, event management, and coordination of the events between the connected sensor nodes. In this paper, the necessity of SQTL as a part of the system that is critical for the work of the sensor networks is discussed. In the sensor applications, SQTL is a programming interface to the SINA middleware so that sensor messages can be easily connected and scripted in simple and concise ways. The programming language has procedural and object-oriented components and offers basic building blocks for accessing the hardware of the sensors, identifying the location, communication, and handling events. It is important to note that SQTL has the capacity to handle events in parallel and thus the sensor nodes can respond to messages, timers and timeouts. The advanced SQTL wrapper using XML syntax enables message transmission and execution between the sensor nodes effectively with better resource utilization. This paper reviews the constructs of SQTL, SEE and the constructs available in SEE and describes their role and importance in executing program and managing resources. Lastly, it provides an illustration of two sample applications that show how SQTL can be used in real life scenarios such as distributing maximum temperature detection and harmonized car tracking.

Keywords – Information Driven Sensor Querying, Sensor Execution Environment, Constrained Anisotropic Diffusion Routing, Sensor Query and Tasking Language.

I. INTRODUCTION

A sensor model is made up of several sensor nodes [1] in which every node is connected to other nodes using a wireless connection. These nodes employ multi-hop to effectively communicate nodes, which are far in terms of space. Sensor nodes have limited processing and memory resources. Each node is equipped with a versatile central processing unit (CPU) for executing computations and a limited quantity of storage capacity for storing programme code and data. All user-to-user communication within the sensor model is directed using the gateway node [2]. Given that sensors are often not linked to a permanent substructure, they rely on batteries as their primary source of power. Consequently, the conservation of power is a key factor in the strategy of a sensor model [3]. Therefore, minimizing the amount of message traffic between sensors is of utmost importance.

A sensor node is equipped with one or several detectors that are interconnected with the corporeal environment. Common samples of sensors are PIR, light, and temperature sensors, which can detect and measuring events, such as the presence of an item, in their immediate surroundings. Each sensor functions as an independent data source, producing records with various attributes including the sensor's id and position, the kind of detector, and the reading value. The schemas of sensor data from multiple nodes of the same kind are identical, and together they constitute a disseminated table. The sensor model may be seen as a vast disseminated database structure with several tables containing various kinds of sensors. Noise may be present in sensor data, and it is often likely to get more precise outcomes by combining data from many sensors [4]. Therefore, precises or averages of unprocessed detector data are more valuable for sensor utilizations compared to individual sensor interpretations [5]. For instance, in the context of monitoring the level of a hazardous chemical in a certain area, a

potential query involves calculating the average of sensor recordings within that zone, and notifying once it exceeds a predetermined threshold.

Using declarative queries is the recommended method for communicating with a sensor network. Instead of using Turing-complete programming languages to write application-specific procedural code, we propose that sensor network applications should be designed to be data-driven. This means that we can simplify the functionality of many applications by using a common interface that supports expressive queries. This study focuses on analyzing questions that have the specific structure shown in **Fig. 1**. However, the development of an appropriate query language for sensor models is a topic that will be addressed in future study. In addition, we expand the template to include nested inquiries, whereby the fundamental query block seen in **Fig. 1** might be present inside the HAVING or WHERE clause of an extra block of query. The following are the query template's fundamental semantics: The disseminated relation of the sensor type is defined by the FROM clause; aggregates and ascertains from detector records are specified by the SELECT clause; a predicate is used to filter sensor records in the WHERE clause; different partitions of sensor records are created by the GROUP BY clause based on specific attributes; and groups are eliminated by the HAVING clause. It is important to note that join queries may be created by providing various relations in the FROM clause.

```
SELECT {attributes, aggregates}
    FROM {Sensordata S}
                WHERE {predicate}
    GROUP BY {attributes}
HAVING {predicate}
    DURATION time interval EVERY time span
```

Fig 1. Query Template.

Sensor frameworks are represented as disseminated databases in research projects like Dataspace and Device Database System. Data retrieval is performed using SQL3-like query languages. Nevertheless, while SQL is a declaratory language, it does not include procedures to carry out coordination duties among sensor nodes. While the study in Gadget Database System has expanded SQL to include support for monitoring jobs, known as long-running queries, it still lacks the ability to describe precise coordination between sensor nodes. The data flow pattern in this situation is contingent upon the internal processing mechanism of the database. An efficient approach involves delegating the task of gathering data from each sensor node and dispensing all the data to the frontend node, as seen in [6]. Nevertheless, this approach is not feasible due to the following reasons. When the number of nodes rises, a reply subsidence at the frontend becomes a bottleneck and negatively impacts the total presentation. Furthermore, sensor nodes mostly depend on wireless communication to facilitate their interaction with one another. Nodes that are far from the frontend node may be unable to establish direct communication with the frontend owing to their restricted transmission power. Users are restricted from customizing the relationship between nodes according to their interests when they are limited to using just the provided query language.

This research aims to address the emerging need to manage and coordinate the sensor networks for applications that include ecological monitoring, structures, and production. The goal of this study is to review the literature on SQTL to understand the extent to which it can be useful in enhancing communication, job distribution, and resource utilization among the sensor nodes. It may be possible as a result of developing a more extensive understanding of the possibilities of SQTL to enhance the effectiveness of sensor networks, which can cause faster response times, decreased resource consumption, and greater expansiveness. The information derived from such conclusions is very valuable for the improvement of sensor networks, which contributes to the development of more reliable and cost-effective solutions for numerous practical applications.

The remainder of the article is arranged in the following manner: Section II presents a review of previous works done in the event-driven programming for sensor networks. Section III discusses the features, wrapper and implementation ecosystem, language constructs, and SEE-provided primitives in Sensor Query and Tasking Language (SQTL). Section IV discusses SQTL sample applications in maximum temperature detection, and coordinated vehicle tracking. Lastly, Section V summarizes the article, and recommends future research directions.

II. RELATED WORKS

Amato et al. [7] describe a COUGAR technique that treats sensor models as disseminated databases where users ask declarative queries of the network, which are then transformed by a query processor of front-end into an effective query plan for in-network dispensation. In the same vein, De Vera et al. [8] contend that sensor networks should be mainly regarded as virtual databases. They propose that query optimization should be carried out using data-centric routing techniques inside the network. The research by So and Brush [9] focuses on the effective calculation of collective replies to queries inside a network. The ACQUIRE technique described in this work aligns with the database viewpoint and may be seen as a data-centric routing apparatus that offers enhanced query enhancement for handling certain types of queries: intricate, one-time questions for simulated data.

Nasser and Chen [10] introduce and analyses Directed Diffusion, a data-centric protocol specifically designed for efficiently handling persistent inquiries. Directed Diffusion is a networking technique where identified data is first sent around the framework via flooding, however there are options to optimize for localized requests. The sources that have appropriate data then respond with the suitable data stream. Simon [11] analyze the effect of combination on reducing the energy expenses of data-centric methods in their study. The IDSQ (Information Driven Sensor Querying) and CADR (Constrained Anisotropic Diffusion Routing) approaches described by Chu, Haußecker, and Zhao [12] are also relevant to our investigation. In IDSQ, the sensors are queried regarding relevant information using a criteria that considers both the amount of information gained and the cost of connection. On the other hand, CADR directs a query to the best possible destination by following a gradient of information gain across the sensor network.

A system that closely resembles ACQUIRE is the rumor-routing system lately developed by Sadagopan, Krishnamachari, and Helmy [13]. Their method is interesting because it starts mobile agents that move randomly over the network by employing sources that contain events, which in turn creates event-paths. Similar to ACQUIRE, the queries produced by the sink/querier are mobile agents, which display arbitrary walks. When a query agent comes across an event-path, it utilizes that data to effectively navigate to the setting of the event. Rumour directing is a method used to reduce the expense of interest-flooding in Directed Dispersion when physical data is not accessible. However, rumour routing is not particularly designed for handling sophisticated one-shot questions for simulated information, as ACQUIRE does, and it does not provide any update constraints. Furthermore, multiple sources may participate in data replication, and in the event of rumor routing, each source may start a random walk.

In such instances, the process of disseminating rumours may not be energy-efficient. An SQL-inspired technique is the most prevalent method used for querying sensor networks. This enables a straightforward and precise approach to querying at the application level, using a simple and declarative syntax. COUGAR [14], SINA [15], and TinyDB [16] are examples of solutions that use this strategy. A portion of the research in this field has focused on pure sensor dataset structures, which primarily offer an efficient query routing and processing disseminated dataset solution suitable for resource-inhibited sensor models. **Fig. 2** displays a sample TinyDB query, taken from reference [17].

```
SELECT AVG (volume), room FROM sensors
                WHERE floor = 6
GROUP BY room
HAVING AVG (volume) > threshold
SAMPLE PERIOD 30s
```

Fig 2. TinyDB Query Form.

COUGAR and TinyDB are specifically intended for use in straightforward data collecting applications, with limited support for basic network selection and aggregation tasks that rely on simple arithmetic operations. Both platforms provide a query language similar to SQL, which includes capabilities for handling temporal data and data streaming. **Fig. 3** displays the graphical user interface of TinyDB, showcasing an example query. TinyDB surpasses COUGAR in energy conservation by using frequency-based sampling for query responses and implementing an energy-efficient routing mechanism to facilitate node communication. COUGAR uses a structure consisting of leader nodes to collect data from the backend and provide responses to queries.

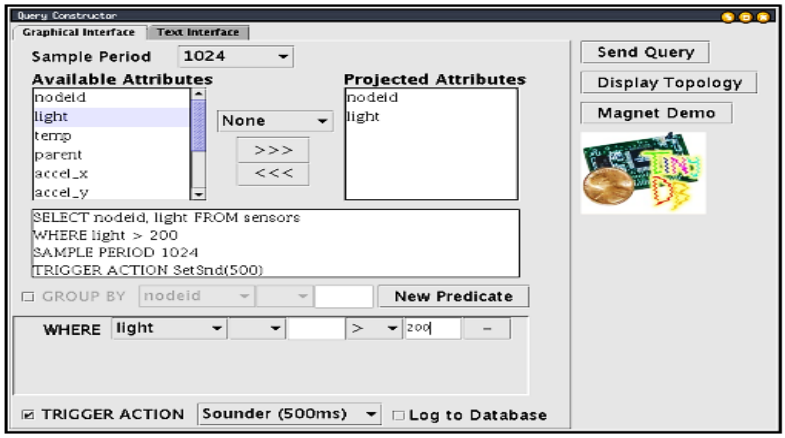

Fig 3. TinyDB Graphical user Interface Showing a Query Example.

In the field of BBQ, Murasawa [18] used a particular model that relied on time-varying multivariate Gaussian distributions. We will now provide a description of this model. It is substantial to note that our technique is relevant to any

model, regardless of its complexity. More or less intricate models may be used as well. The query processor of the new models does not need any modifications and may use existing code that interacts with and retrieves specific data from the sensor network. **Fig. 4** demonstrates our fundamental structure with an instance. Users input SQL questions to the dataset, which are then converted into statistical calculations based on the framework. The inquiries consist of error tolerances and target confidence limits, which indicate the level of uncertainty that the user is ready to accept. These limits will be easily understood by technical and scientific users, since they are the same as the assurance limits used to communicate findings in most technical disciplines (see the graph in the top right corner of **Fig. 4**). In this instance, the user seeks approximates for the sensor readings of nodes 1 through 8, with a precision of 0.1 degrees Celsius and a confidence level of 95%. According to the model, the system determines that the optimal approach to respond to the question with the desired level of certainty is to get the battery voltage data from detectors 1 and 2, and the humidity data from detector 4.

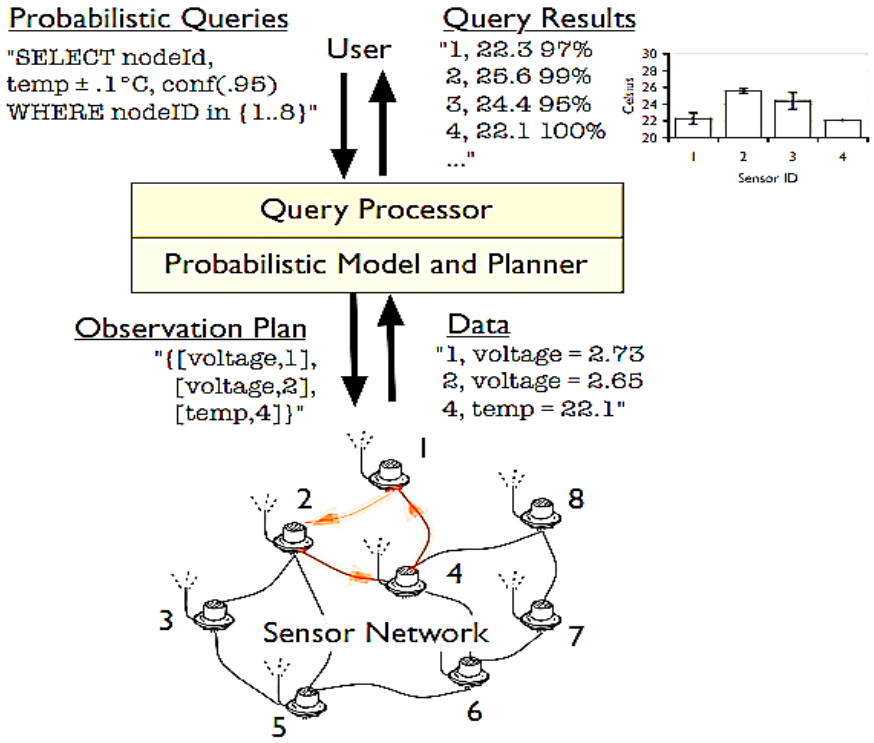

Fig 4. Sensor Network Architecture for Model-Based Querying [19].

Rather than being a solution that is limited to sensor databases, SINA is a complete middleware platform for sensor networks. It provides support for scripting in the SQTL (Sensor Query and Tasking Language) language as well as SQL-like queries. This programming language offers fundamental elements for accessing sensor hardware, facilitating conversation, and managing events. SQTL scripts may be distributed via the model during runtime. The SINA data architecture exhibits more flexibility compared to the sensor database systems discussed before. It uses an associative spreadsheet to store data, using ascribe-based naming to define individual cells. A certain number of cells are predetermined, however more cells may be added as needed. SINA's ability to adapt to different network topologies is very advanced, since it can effectively address challenges such as the movement of the querying (sink) node.

This study presents an architecture that enables the tasking and querying of sensor models. The creation of the appropriate Sensor Execution Environment (SEE) and SQTL is the fundamental concept of the architecture. Other than representing a sensor model as a disseminated dataset with passive nodes, we represent it as a disseminated collection of cooperating nodes with active, programmable capabilities. This enables the nodes to collaborate with each other so as to achieve a certain goal. Subsequently, the nodes assume an active and independent state. Nodes may be configured to enhance their interactivity, hence improving the efficiency of executing user requests.

III. SENSOR QUERY AND TASKING LANGUAGE

SQTL serves as the scheduling gateway among sensor utilizations with the SINA middleware inside the architecture. The language is a procedural scripting language that is specifically intended to be both versatile and concise. It has the ability to comprehend basic declarative query queries. Furthermore, it offers sensor hardware access functions such as get Temperature and turn on, as well as location-aware capabilities like is Neighbor and get Position. It also includes communication primitives such as tell and execute. Furthermore, it offers an event management architecture that is ideal for sensor model utilizations in which nodes routinely handle asynchronous events, including messages being received or timer-triggered events. A programmer can define an event processing block in a corresponding way by utilizing the “upon” construct.

SQTL currently supports three sorts of events: (1) events that occur when a sensor node receives a message, (2) events that are triggered regularly by timers, and (3) occurrences that are produced by a timer expiring. Receive, every, and expire are the SQTL keywords that describe these kinds of occurrences, correspondingly. A SQTL message, which includes a script, is designed to be understood and performed by any nodes in the model. To direct a script to a particular recipient or a set of recipients, it is necessary to enclose the message in a SQTL Wrapper. This wrapper serves as a header of a message that specifies the sender, the recipients, a specific utilization executing on the recipients, and the parameters for the application. The SQTL wrapper utilizes the syntax of XML (Extensible Markup Language) to build an application layer header. This header has the ability to establish complex addressing schemes for ascribe-based names. **Table 1** provides a summary of the typical fields included in SQTL wrappers.

Table 1. Arguments that the SQTL Wrapper's Actions Use

Argument	Meaning
Sender	An SQTL message wrapper's sender
Receiver group **criteria**	Potential recipients indicate by the next two sub-arguments Sub-argument of receiver to designate receiver group; value may be either NEIGHBORS or ALL-NODES Sub-argument from the recipient outlining the recipients' selection criteria
Application-id	Unique ID for every utilization within the same model of sensors
Num-hop	The distance in hops from gateway node
language	Indicate the language employed in the content.
content	A payload that holds return values, a message, or a program
With (optional) **Parameter** **Type** **Name** **Value**	Multiple program parameters that are sent from the sender to the recipient Sub-argument/repeatable of with Parameter`s data type Parameter`s name Parameter`s value

SQTL Features

SQTL incorporates the functionalities of both object-adapted and technical programming languages. The majority of primitives offered by the SEE inside the SQTL language are implemented as classes. Each node is furnished with a SEE that handles the reception of messages, analyses all incoming SQTL messages, and executes the appropriate action based on the message type. SEE examines the receiver dispute of a dispatch and determines, depending on its value, either to transmit the dispatch to the next destination.

Dispatches containing the ALL_NODES group sub-argument will be sent to all sensor nodes in the model, whereas messages containing the NEIGHBOURS group sub-argument will only be sent to the immediate neighboring nodes of the sender. The receiver's characteristics recorded in its datasheet will be compared with an attribute-based name, which is presented as a list of ascribe-value pairs in the criterion field. The SEE algorithm only considers a message if the properties of the node meet the specified requirements. The act of matching a message with its intended recipient(s) upon arrival is referred to as late binding and is explained in [20]. After injecting a SQTL script from node at Front-end (a specialized node directly linked to the model) to a single or multiple nodes, the script has the capability to propagate itself to further nodes to fulfil the given goal. A tell dispatch is formed and sent back to the asking node—typically the upstream node where the script originated—after an outcome is provided at each distinct node.

Application programmes, specifically SQTL programmes produced at the Frontend to meet user necessities, can instantiate these classes. This allows them to access various system resources, such as individual sensing devices or groups of sensors with the same functionality in a node. It is significant to understand that preventing direct access to these physical sensor devices by apps enables the operating system to seamlessly combine data from several sensors of the same kind. For instance, a sensor node may have many motion detection sensors. Typically, the sensors are oriented in various orientations to optimize the coverage of the detecting area. Creating an instance of this motion detector class will provide a reasonable entity that signifies a collection of all detectors inside this class. The outcomes from each detector may be condensed and sent to the calling utilization when the programme triggers a specified function to get the outcome. Nevertheless, the language lacks both the ability to inherit system classes and the capability to create user-defined classes. The event handling system, which is an integral component of the language, receives messages from other nodes and detection outcomes of from represented logical gadgets. The output of sensor gadgets is also influenced by a programme coded in a traditional procedural manner. The ultimate outcome will thereafter be sent to the Frontend.

SQTL Wrapper and Implementation Ecosystem

As stated earlier, a comprehensive SQTL programme, enclosed in a SQTL wrapper created at the Frontend, will be disseminated across a single or multiple nodes. A set of precise activities is established to meet the criteria for transmitting these programmes among the sensor and Frontend nodes, as well as among the nodes themselves. Actions need the inclusion of action parameters, which are referred to as parameters.

At the sensor node, a Sensor Event Executor (SEE) analyses all incoming SQTL messages and executes the corresponding action for each defined action type in the messages. SEE examines the: recipient parameter and determines the appropriate destination for each message depending on the value in this field. Messages with the keyword "ALL-NODES" within their: cluster subparameters would be broadcasted to all nodes within the model. On the other hand, messages with the keyword "NEIGHBOURS" will only be transmitted to the immediate neighbouring nodes. Messages with specified recipient node identities in the: group will only be reputed by the corresponding target node. In addition, the sender may indicate the recipient`s feature in the: criteria subparameter if they are unsure about the specific nodes that will obtain the message at the time of sending.

An approach to define criteria is by using attribute-value pairs. The usage of criteria will establish a link between the criteria and the real properties of the receiver, facilitated by the SEE of the headset. SEE only allows messages to be accepted if the node can meet specific requirements. The process of late binding, which has been previously characterized as a characteristic of Associative Broadcast by Bayerdorffer, is referred to as such. Any communications that have a known application-id will be handled topically as well. In this scenario, a dispatch containing a tell action will be sent to the appropriate utilization and then handled by the application. SEE will handle all remaining activities. SEE will be stimulated to store SQTL code provided in the message's: content section in the detector`s storage area upon receiving a "install" message. After the code is allocated, the index specified by the: application-id option, nothing more will be done. The code remains dormant in the retention until SEE gets a SQTL communication with the activity "start," at which point SEE will commence the execution of this code.

The execution length may be predetermined inside the SQTL code or dynamically changed online using other action signals such as halt or suspend. When the programme is suspended, it has the capability to restart from the exact place where it was last stopped upon receiving a resume notification. The last two operations, flush and uninstall, result in the node that receives these notifications eliminating the installed code. A Storage Execution Environment (SEE) that obtains uninstall messages will wait until the presently executing utilization finishes before removing the code from storage spaces of the node. However, when a message is received, SEE will instantly stop and delete the stated SQTL utilization programme, even if it is in the midst of implementation. By using this method of executing and storing code, the amount of bandwidth required to distribute the same SQTL programmes may be significantly decreased, provided that a sensor node has sufficient memory capacity and the programme size is minimal.

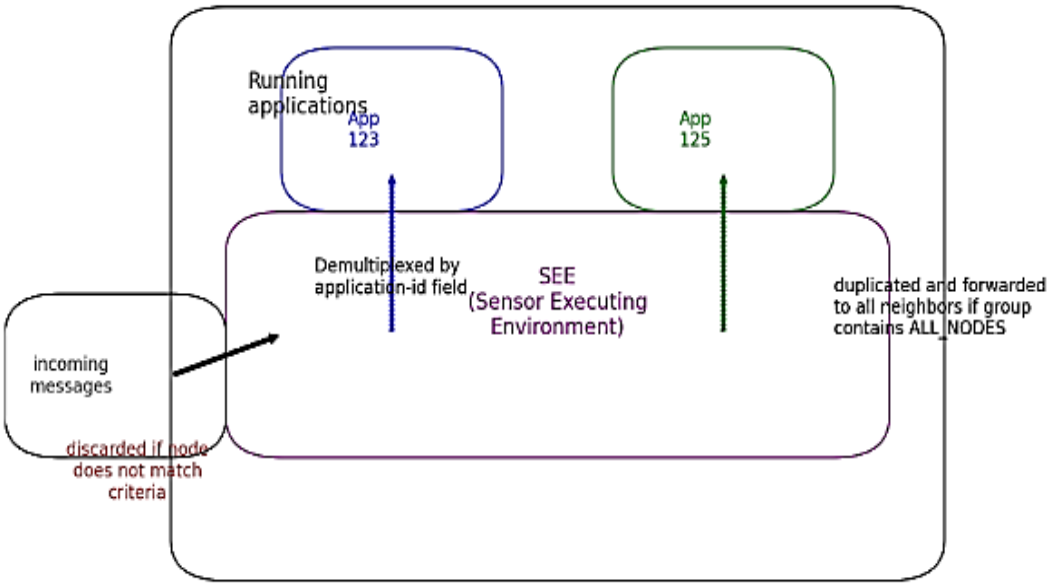

Fig 5. Sending out Messages that a Sensor Node Has Received.

Practically, some programmes that are designed for tasking will be performed just once. The implementation of these expendable programmes should be initiated by the implement action. The SQTL programme is specifically developed for sensor models that use active technologies. Once a SQTL code is inserted from the Frontend to a single and multiple nodes, it may be propagated to further detectors to fulfil a job specified within the code. Once results have been generated at every distinct node, the SQTL application will create a tell message to send the result back to the intended recipients. Typically, the recipients are the upstream nodes from which codes originated from, but they could also be any other node specified in

the code, in case lower layer routing procedures are available. It is important to note that all operations are directed towards SEE, meaning that SEE does not transmit these SQTL signals to the running apps. Except for an activity, which will be examined to see whether it necessitates any re-advancing by SEE. Once the re-advancing operation is complete, it will be sent to the provided: utilization-id.

Fig. 5 illustrates the process of dispatching approaching messages carried out by the SEE. The ability to transport programs written in languages other than SQTL is one of the many features of the SQTL wrapper. Using the: content parameter to incorporate a SQL query is one example of how the frontend might use the: language parameter to specify SQL. To execute this embedded SQL expression, however, the node must have a SQL engine built. By using SEE, the implanted SQL declaration may be sent to the SQL engine. In addition, SEE handles both the incoming and outgoing SQTL messages from any apps that are currently active. The: receiveT option specifies the target or targets to which the underlying communication mechanism will distribute outgoing messages.

SQTL Language Constructs

This section provides an overview of the primary language constructs used in SQTL. As previously stated, SQTL is supposed to resemble a technical language with lightweight object-oriented features. The language features include of arithmetic operators $(+, -, *, /)$, contrasted operators $(==, !=, <, >)$, and boolean operators (AND, OR, NOT). It also has assignments, restrictive declarations (if, then, else), loop statements (whereas), object creation (novel), and occurrence management (upon). The absence of a variable declaration block allows for the creation of variables as needed, without specifying their type. The majority of language constructs discussed above are used in a similar manner to other procedural languages. Nodes are typically configured to handle asynchronous occurrences such as message delivery or a timer-triggered event, for the majority of sensor network applications. By employing the "upon" constructs, a system analyst may define an occurence managing block in a suitable manner. SQTL currently supports three sorts of events: (1) events that occur when a sensor node receives a message, (2) events that are triggered regularly by a timer, and (3) occurrences that are produced by a timeout. These kinds of occurrences are denoted by SQTL keywords timeout, each, and receive, in that order.

SEE-Provided Primitives

The SEE supplies a variety of primitives. Depending on their purpose, they may be classified as follows: (a) primitives that access sensors, such as getTemperatureSensorQ, turnOnQ, turnOffQ; (b) primitives that communicate, such as tellO, executeO, sendQ; and (c) primitives that are aware of their position, such as isNorthOfO, isNearQ, and isNeighbor. Array and linked-list are two examples of the fundamental data structures supplied by SQTL, in addition to the system-provided primitives already mentioned. Data aggregation functions like maximum, minimum, and average on the data structures can also be used.

IV. SAMPLE APPLICATIONS

This section provides a description of two sample apps that may be used to query and allocate tasks to sensor network. All sensor nodes have identical capabilities. There are no instances of packet loss, no conflicts in communication, and all communications are symmetrical. There are no instances of sensor nodes failing while the algorithms are being performed. Due to the inherent condition of messages in sensor models, all conversations among nodes are broadcasted. Sender nodes have the ability to designate a specific recipient by using attribute matching. The lowest levels of the network are not intended to offer routing assistance. Nevertheless, programmes have the capability to monitor the sender's address and utilize reverse route forwarding to transmit outcomes back to the sender.

Maximum Temperature Detection

This first sample demonstrates the use of SQTL to programme sensor nodes for executing a query including data aggregation. To determine the maximum temperature, we will assume that all nodes in the model have the ability to sense temperature. The SQTL framework allows for the installation and execution of mobile code. In this case, a specialized SQL engine would be copied onto the nodes. This engine would evaluate a SQL query, which is encased behind a SQTL wrapper, at the node.

Designing a SQL engine to be both efficient and general simultaneously is a non-trivial task. While it is possible to attain generality, there is a risk that the engine may adopt a centralized design, leading to the implosion issue as explained in [21]. In this scenario, it would be more advantageous to use code that describes the tasks in greater detail rather than relying on a single SQL query. **Fig. 6** illustrates a dispersed implementation of the highest temperature query written in SQTL. The programming of detector nodes prioritizes increased interactivity across nodes, rather than overwhelming a single node. Upon programme initialization, the frontend node selects an accurate node to transmit the Findmax code. Assuming the node is node A. After receiving and executing the code, node A verifies the Frontend by delivering a 'validation' communication. Node A subsequently transmits the Findmax code to its neighboring nodes, B, C, and D, who will in turn execute the same code recursively and provide validation signals back to Node A.

Coordinated Vehicle Tracking

Off-road navigation tactics have suggested the use of tracked vehicles to help in a variety of terrain circumstances, such as mud, rubble-strewn terrain, steep slopes, loose sand, snow, or any mixture of these. Prior research has shown that tracked

cars have superior efficacy in navigating challenging landscapes when compared to conventional wheeled vehicles [22]. They have many benefits for traversing diverse landscapes, including as excellent grip [23], even on slick ground.

```
(execute
: sender FRONTEND
: receiver (: group NODE (1): criteria TRUE)
:application—id 123
: language SQL
: content (SELECT Max(getTemperatureO) FROM ALL NODES)
)
                    (execute
                        : sender                    FRONTEND
                        : receiver                  (: group NODE(l) criteria TRUE)
                        :application-id                     language 123
                        : content                   SQTL
                       )
                                tel l (MESSAGE.sender, 'TRUE', 'confirm');
                                execute          (NEIGHBORS,          'TRUE',
MESSAGE.content);
                                confirmationCount = 0; // Initialize the confirmation
counter
                    // Handle incoming confirmation messages
                    upon {
                                 receive(msg) where msg. content == 'confirm' {
                                confirmationCount++;
                          }
                                timeout (500) {
                           break;
                      }
                    }
                    // Create a list to store temperature readings
                          temperatureReadings = new List();
                    // Add the current node's temperature to the list
                    temperatureReadings.add(getTempSensor(). getCurrentTemp());
```

Fig 6. SQTL Code for Utilizing a SQTL Wrapper to Find the Dispersed Highest Temperature Method.

In addition to those features, this product offers robust load support, exceptional performance, and efficient power delivery. In contrast to wheeled vehicles, this design exhibits greater durability and is capable of executing sharp turns with a reduced turning radius, while also keeping a consistent velocity on both flat and uneven surfaces. Therefore, tracked vehicles were specifically designed to efficiently perform a wide range of activities in industries like agricultural, security, and military operations. Autonomous technology for tracked vehicles has advanced in recent years, enabling its use in precision agriculture within the civilian sector, namely in broadacre agriculture. Furthermore, autonomous cars enhance both safety and productivity, while simultaneously reducing the expenses associated with skilled personnel. To achieve the development of an independent car, it is important to create a route-trailing direction scheme that effectively steers the car along a pre-established path. The route monitoring of these vehicles is crucial for several agricultural chores, such as pesticide spraying, weeding, and planting. Moreover, the agriculture sector is showing a growing inclination towards sovereign technology, occasionally referred to as robot mechanization.

Most viability investigations on sovereign agricultural machinery have focused on the harvesting of various crops and fruits, including sweet pepper, beans, sugar beetroot, cucumber, and also the use of robotic technology in paddy field

farming. Target tracking has increasingly included deep learning methods in recent years. In 2017, a new algorithm called discriminative correlation filter with channel and spatial reliability tracker was developed and proposed. This algorithm aimed to improve tracking accuracy. In 2018, another algorithm called ECO+ was proposed, which demonstrated the effectiveness of deep tracking [24]. Finally, in a later year, the SiamRPN++ algorithm was introduced. This algorithm focused on enhancing accuracy and reducing the size of the network. This system exhibits exceptional tracking precision, but, the scarcity of training data is a significant challenge, a predicament often seen in target tracking using deep learning techniques.

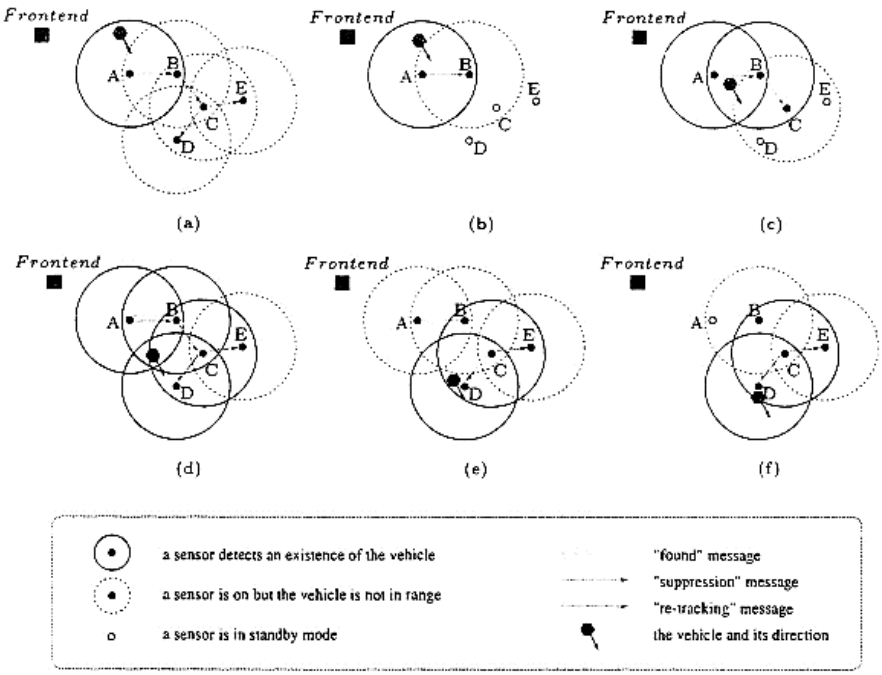

Fig 7. (a) A Detects the Approaching Car; (b) C, D and E Temporarily Halt Their Detection Actions While B Continues to Trail; (c) the Car Approaches B and C Resumes its Detector Operation; (d) C and D Sense the Car and E Reactivates His Device; (e) the Car Moves Out of Detection Areas a and B; and (f) A Discontinues Its Detection Operation.

Here, we demonstrate the proficiency of SQTL in assigning tasks to a detector model. We use a synchronised version of car trailing as an illustrative sample. The primary objective of car trailing in a detector model is to precisely determine the whereabouts of a designated vehicle and closely observe its motion. An effective method for monitoring the movement of a car is to allow all sensors inside a model to independently perform tracking utilizations without any cooperation. The sensors detect the presence of the car and provide data about the vehicle's movements to the Frontend. While this approach is effective regardless of the vehicle's location, all sensors must use energy and dispensation capacity to identify the presence of the car. On the other hand, we have created a network in SQTL that utilises the organization between sensor nodes to effectively monitor the movement of cars while also conserving limited network resources. This method is based on three extra assumptions: Initially, a vehicle is equipped with an active tag that can be noticed by detectors. Furthermore, the detecting array of detectors is equivalent to their gearbox range. Furthermore, it is possible that a desired car may or may not be inside the range of model detection when the utilization begins, meaning that it may enter the domain of the sensor model at a later point in time. The SQTL method is shown with an example in **Fig. 7**. A SQTL message sent by the Frontend to one network node causes the message to spread throughout the whole network.

Lastly, every node inside the network initiates its motion detection operations. Node A is the first node to detect the presence of the car when it enters the sensing region. Subsequently, A will transmit 'suppression' signals to the whole system (**Fig. 7(a)).** Other nodes that get the notification will turn on their motion sensors and put them in standby mode. After completing this action, A additionally transmits a 're-trailing' message only to its immediate neighbouring nodes, which in this scenario only includes B. The receipt of the 're-trailing' message prompts B to initiate the restarting process of its sensor once again. A then responds to the frontend (**Fig. 7(b)),** which is the source of the SQTL programme. When the car enters the detection area of B, B must also transmit a 're-trailing' message only to its immediate neighbours, and replies the dispatcher (A). Currently, C resumes the operation of its signal sensor upon receiving messages from B **Fig. 7(c)).** In **Fig. 7**(**d),** the car approaches points C and D, and the identical sequence of events takes place at D and C as it did at points B and A. In **Fig. 7(e),** despite the car moving outside the range of A and B, it continues to track for a certain duration and must also transmit all responses from D and C back to Frontend. In **Fig. 7(f)**, the retracting period of A has elapsed, causing it to cease detecting the car. However, B still gets a 're-trailing' communication from C, prompting B to endure its detection operation. As shown from this example utilization, the act of assigning a sensor network to achieve a certain objective collectively and

independently may be accomplished using the event-motivated capabilities of SQTL, which may not be readily achieved by methods only in accordance with SQL.

V. CONCLUSION AND FUTURE SCOPE

Sensor Query and Tasking Language (SQTL) is significant for the development and enhancement of sensor systems. Based on the analysis of SQTL's procedural scripting paradigm and its efficient means of event handling, it can be concluded that SQTL significant in ensuring that the sensor nodes can effectively communicate and coordinate themselves even when they are spread out in different areas. We investigate the nature of SQTL as a model, with major focus on the efficiency and flexibility of the model. This makes it possible to develop short and efficient scripts that can perform complicated operations in sensor networks. However, when SQTL is integrated with the Sensor Execution Environment (SEE), the importance of SQTL rises to another level because the nodes in the sensor can perform the tasks with the exact amount of resources. Subsequently, a study of the demonstration projects that have been carried out by SQTL such as the distributed maximum temperature discovery and the coordinated vehicle tracking constitute concrete evidence of the effectiveness of the technology as well as its applicability in the real world. Therefore, the continuous improvement and development of SQTL still possesses strong potential for the development of sensor networks in the future. Researchers and professionals may leverage SQTL's features to experiment with the limits of response, robustness, and speed, and to find novel ways of improving the sensor systems. The nature of SQTL makes it very suitable for handling new problems and creating adaptive solutions in complex and diverse sensor networks.

CRediT Author Statement

The author reviewed the results and approved the final version of the manuscript.

Data Availability

The datasets generated during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interests

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Funding

No funding was received for conducting this research.

Competing Interests

The authors declare no competing interests.

References

[1]. M. A. Mahmood, W. K. G. Seah, and I. Welch, "Reliability in wireless sensor networks: A survey and challenges ahead," Computer Networks, vol. 79, pp. 166–187, Mar. 2015, doi: 10.1016/j.comnet.2014.12.016.
[2]. S. Ivanov, S. Balasubramaniam, D. Botvich, and O. B. Akan, "Gravity gradient routing for information delivery in fog Wireless Sensor Networks," Ad Hoc Networks, vol. 46, pp. 61–74, Aug. 2016, doi: 10.1016/j.adhoc.2016.03.011.
[3]. G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," Ad Hoc Networks, vol. 7, no. 3, pp. 537–568, May 2009, doi: 10.1016/j.adhoc.2008.06.003.
[4]. J. Dong, D. Zhuang, Y. Huang, and J. Fu, "Advances in Multi-Sensor Data Fusion: Algorithms and Applications," Sensors, vol. 9, no. 10, pp. 7771–7784, Sep. 2009, doi: 10.3390/s91007771.
[5]. H. Y. Teh, A. W. Kempa-Liehr, and K. I.-K. Wang, "Sensor data quality: a systematic review," Journal of Big Data, vol. 7, no. 1, Feb. 2020, doi: 10.1186/s40537-020-0285-1.
[6]. J. Granjal, E. Monteiro, and J. S. Silva, "Security in the integration of low-power Wireless Sensor Networks with the Internet: A survey," Ad Hoc Networks, vol. 24, pp. 264–287, Jan. 2015, doi: 10.1016/j.adhoc.2014.08.001.
[7]. G. Amato, S. Chessa, C. Gennaro, and C. Vairo, "Querying moving events in wireless sensor networks," Pervasive and Mobile Computing, vol. 16, pp. 51–75, Jan. 2015, doi: 10.1016/j.pmcj.2014.01.008.
[8]. D. Díaz Pardo de Vera, Á. Sigüenza Izquierdo, J. Bernat Vercher, and L. Hernández Gómez, "A Ubiquitous Sensor Network Platform for Integrating Smart Devices into the Semantic Sensor Web," Sensors, vol. 14, no. 6, pp. 10725–10752, Jun. 2014, doi: 10.3390/s140610725.
[9]. H.-J. So and T. A. Brush, "Student perceptions of collaborative learning, social presence and satisfaction in a blended learning environment: Relationships and critical factors," Computers & Education, vol. 51, no. 1, pp. 318–336, Aug. 2008, doi: 10.1016/j.compedu.2007.05.009.
[10]. N. Nasser and Y. Chen, "SEEM: Secure and energy-efficient multipath routing protocol for wireless sensor networks," Computer Communications, vol. 30, no. 11–12, pp. 2401–2412, Sep. 2007, doi: 10.1016/j.comcom.2007.04.014.
[11]. H. A. Simon, "Bounded Rationality and Organizational Learning," Organization Science, vol. 2, no. 1, pp. 125–134, Feb. 1991, doi: 10.1287/orsc.2.1.125.
[12]. M. Chu, H. Haussecker, and Feng Zhao, "Scalable Information-Driven Sensor Querying and Routing for Ad Hoc Heterogeneous Sensor Networks," The International Journal of High Performance Computing Applications, vol. 16, no. 3, pp. 293–313, Aug. 2002, doi: 10.1177/10943420020160030901.
[13]. N. Sadagopan, B. Krishnamachari, and A. Helmy, "Active query forwarding in sensor networks," Ad Hoc Networks, vol. 3, no. 1, pp. 91–113, Jan. 2005, doi: 10.1016/j.adhoc.2003.08.001.
[14]. D. Niculescu, "Communication paradigms for sensor networks," IEEE Communications Magazine, vol. 43, no. 3, pp. 116–122, Mar. 2005, doi: 10.1109/mcom.2005.1404605.
[15]. S. Shekarpour, E. Marx, A.-C. Ngonga Ngomo, and S. Auer, "SINA: Semantic interpretation of user queries for question answering on interlinked data," Journal of Web Semantics, vol. 30, pp. 39–51, Jan. 2015, doi: 10.1016/j.websem.2014.06.002.
[16]. V. A. Epanechnikov, "Non-Parametric Estimation of a Multivariate Probability Density," Theory of Probability & Its Applications, vol. 14, no. 1, pp. 153–158, Jan. 1969, doi: 10.1137/1114019.

[17]. G. Chatzimilioudis, A. Cuzzocrea, D. Gunopulos, and N. Mamoulis, "A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement," Journal of Computer and System Sciences, vol. 79, no. 3, pp. 349–368, May 2013, doi: 10.1016/j.jcss.2012.09.013.
[18]. Y. Murasawa, "Measuring the natural rates, gaps, and deviation cycles," Empirical Economics, vol. 47, no. 2, pp. 495–522, Sep. 2013, doi: 10.1007/s00181-013-0747-9.
[19]. A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-based approximate querying in sensor networks," The VLDB Journal, vol. 14, no. 4, pp. 417–443, Oct. 2005, doi: 10.1007/s00778-005-0159-3.
[20]. Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," IEEE Personal Communications, vol. 8, no. 4, pp. 52–59, 2001, doi: 10.1109/98.944004.
[21]. A. Buckley et al., "LHAPDF6: parton density access in the LHC precision era," The European Physical Journal C, vol. 75, no. 3, Mar. 2015, doi: 10.1140/epjc/s10052-015-3318-8.
[22]. B. Sebastian and P. Ben-Tzvi, "Physics Based Path Planning for Autonomous Tracked Vehicle in Challenging Terrain," Journal of Intelligent & Robotic Systems, vol. 95, no. 2, pp. 511–526, Apr. 2018, doi: 10.1007/s10846-018-0851-3.
[23]. H. Antonson, S. Mårdh, M. Wiklund, and G. Blomqvist, "Effect of surrounding landscape on driving behaviour: A driving simulator study," Journal of Environmental Psychology, vol. 29, no. 4, pp. 493–502, Dec. 2009, doi: 10.1016/j.jenvp.2009.03.005.
[24]. O. Iqbal, V. I. T. Muro, S. Katoch, A. Spanias, and S. Jayasuriya, "Adaptive Subsampling for ROI-Based Visual Tracking: Algorithms and FPGA Implementation," IEEE Access, vol. 10, pp. 90507–90522, 2022, doi: 10.1109/access.2022.3200755.

ISSN: 3080-7484