SENSOR WEB: A SURVEY

Jayampathi Sampath
Department of Computer Science
Georgia State University
jrajapaksage1@student.gsu.edu

Abstract

Availability of microscopic devices and low power wireless communication has led to the introduction of small, low power, motes that could be networked. Over 100 physical, chemical and biological properties can be measured by using sensing technology. Rapid development of cheaper, faster and smart in situ and remote sensors, wireless and mobile network access, autonomous and intelligent geospatial software agents and distributed sensor networks more important than ever. Major challenges remain on how concurrent users program and control such environments. One of the critical components in developing sensor web is to build a backbone that connects heterogeneous in situ sensors and remote sensors over wired or wireless networks. Some research has surveyed Wireless Sensor Networks (WSN)s and middleware for WSNs. However none have investigated the current state of research on sensor web design and development or provided any previous classifications. The survey paper discusses the classification and the current state of the sensor web. The survey paper first introduces sensor network and the revolutionary concept of the Sensor Web. Secondly it describe the requirements, challenges and design principles that sensor web must meet. Then introduce the major classification and discuss the current state of the research and conclude the survey paper by summarizing open problems in Sensor web. (Section 3 – major classifications, section 4-5-6 detail discussion of the classifications and current state of the research)
Abstract

1. Introduction .................................................................1
   1.1 Sensor Network
   1.2 Sensor Web

2. Design Goals.........................................................5

3. Classification of Sensor Web..................................6
   3.1 Hierarchical and Receptor-based Systems
   3.2 Streaming query systems
   3.3 Sensor Middleware/ Querying sensor networks

4. Hierarchical and Receptor-based Systems................8
   4.1 IrisNet
   4.2 Hourglass
   4.3 SensorMap
   4.4 Mobile Web Services Framework
   4.5 GeoSWIFT
   4.6 SenseWeb
   4.7 SWL
   4.8 Composing Semantic Services in OSRE
   4.9 Open Sensor Web Architecture
   4.10 Service Oriented Sensor Web
   4.11 Semantic Sensor Web
   4.12 Sensor Web Enablement
   4.13 HiFi

5. Streaming query systems........................................15
   5.1 Centralized Streaming query systems
      5.1.1 Aurora
   5.2 Distributed Streaming query systems
      5.2.1 Aurora* and Medusa
      5.2.2 PIER
      5.2.3 Borealis
      5.2.5 Fault Tolerance in the Borealis
      5.2.6 Load Distribution in the Borealis
      5.2.7 ISS
   5.3 Distributed and Declarative Streaming query systems
      5.3.1 Sophia

6. Sensor Middleware / Querying sensor networks........20
   6.1. TinyDB
   6.2 Caugar
   6.3 SINA
   6.4 DsWare
   6.5 MiLan

7. Conclusion

References
1. Introduction

Up to now research in the sensor network area has mainly focused on routing, data aggregation, and energy conservation in a single sensor network. However integration of multiple heterogeneous sensor networks has been studied to a limited extent. With the presence of cheaper, faster and smart wireless sensors we can soon expect large number of autonomous sensor networks being deployed. These sensor networks will be managed by different organizations with different technologies. Building an internet infrastructure that connect heterogonous sensor networks and distributed query processing will soon become huge challenge. Developers of wide-area sensing services will deploy their services on this distributed infrastructure. Imagine the following scenario: after an oil spill, an ecologist wishes to know all locations where oil has significantly encroached on the coastal habitat. He queries a coastal-monitoring service, which collects data from video cameras directed at the coastline. In response, He receives both images of these contaminated sites and their geographic locations. The same coastal-monitoring service can store triggered queries, whereby the service notifies the appropriate lifeguards when a strong riptide develops in a beach region[1]. Applications of sensor web inspire and motivate the recent computer science research. For example, medical first responders wish to track patients’ vital signs and treatments wirelessly at the scene of an accident and immediately make this information available to remote doctors. A more pedestrian scenario finds a driver who wishes to navigate the roads of a busy city to find the “best” parking spot to her destination, taking into account cost, weather, current traffic, and preferred walking distance. These applications appear within reach, and yet currently neither of them can be built.

These service and many like them depend on sensor networks, middleware, and distributed query processing. At its lowest level, research in sensor networks examines how to efficiently push sensor data through a wireless infrastructure to one or more base stations. Efficiency is gained by intelligently inferring events from the sensed data using in-network processing. Although there have been much research on-the-fly sensor reprogramming and query languages, most of the proposed solutions essentially end at the base station [14, 15, 19, 23, 30].

Likewise, Internet-based data processing research has taken many forms over the years, including work in classic distributed systems and peer-to-peer scenarios [7, 8]. These systems have focused on other applications, such as federating databases [6], harnessing compute cycles (Seti@home), and Web-based content distribution[1, 3, 4, 11, 12, 18, 20]. Most recently, the Continuous Queries work (CQ) from the database community offers in-network processing of streaming data in networks [3,5,6,7,8,9,10,13,17,37]. All of these groups make different assumptions about their data model and in the connectivity, stability, and consistency, of their networked participants. By considering these facts Sensor Web can be classified into three sections, hierarchical and receptor-based System, streaming query systems, and sensor middleware/ querying sensor networks.

This survey paper will give you a clearer insight of platforms which enable the dynamic integration and management of sensor networks and the produced data streams.
1.1 Sensor Network

Recent advances in wireless communication and electronics have enabled the development of small, low cost, low power and multi-functional sensors (e.g., temperature, humidity, light, and etc.) that are communicate in short distance (up to 150 m). These tiny nodes consist of CPU, memory, commutation and sensing components. A Sensor Network composed of large number of sensor nodes that are densely deployed. The position of sensor node need not be engineered or predetermined. The “sensor network” can be defined as a computer accessible network of many spatially distributed devices using sensors to monitor conditions at different locations.

1.2 Sensor Web

Currently there is no universal understanding about what comprises a sensor web. Different research communities use different terminologies such as pervasive computing, wireless sensor networks, embedded sensing, distributed sensor networks and scalable information networks and partially touch the sensor web concept. The term sensor web was first introduced by NASA Sensor Web Applied Research Planning Group in 2001. According to there definition A Sensor Web is a system of intra-communicating spatially distributed sensor pods that can be deployed to monitor and explore new environments. Some definition focuses on tactical battle field operations. These definitions had limited the sensor web to small scope. As defined at the February 2007 NASA ESTO/AIST workshop on sensor web technologies, a sensor web is:

A coordinated observation infrastructure composed of a distributed collection of resources that can collectively behave as a single, autonomous, taskable, dynamically adaptive and reconfigurable observing system that provides raw and processed data, along with associated meta-data, via a set of standards-based service-oriented interfaces.

Sensor web consists of both in situ sensors and remote sensors. Sensor can be mobile or stationary such as webcams, traffic cameras, flood gauges, weather towers, air pollution monitors, stress gauges on bridges, mobile bio-sensors, webcams, and satellite-borne earth imaging devices. Various sensing resources connect to web that act like a central computer. Sensor web can be thought as a global sensor.

1.2.1 Sensor Web Applications

World Wide Sensor Web enables wide verity of useful services:
- Where is the nearest available parking space? What is the traffic like on the bridge? How long is the queue at the gas station? Where is the bus?
- What is the temperature at my favorite beach? What is the water algae level?
- Waiting-time monitors for reporting on queuing delays at post offices, food courts, and so on
- Lost-and-found services for locating lost objects or pets
- Early warning services in public health
- Homeland defense services
- Computer network monitoring services
- Monitoring and control of supply chains, logistics, factories, pipelines or energy usage
- Internet scale sensing observatories (Volcano, underwater, rainforest).

1.2.2 Sensor Web has several key demands:

- **Interoperability**
  
  By connecting distributed and heterogeneous in situ and remote sensors to information center can form a sensor web. Information centers can store, disseminates exchange, manages, displays and analyzes the sensing intonations. Interoperability allows interconnecting various components to form the sensor web.

- **Intelligence**
  
  Small sensors consist of a processors and able process data and communicate with each other. Connectivity of these sensors through sensors web make another level of intelligence.

- **Low cost, scalability, and reliability**
  
  New sensors can be added in to existing sensor web easily without changing its existing design. Because of advancement of semiconductor technology sensor’s processor is becoming faster and cheaper than ever. This allows deploy sensors as dense as desired. Sensors may fail because of spent batteries or damaged transducers. Sensor Web tolerant to failures of individual sensors and provides reliability.

- **High resolution**
  
  Sensor web consists of both in situ sensors and remote sensors. Remote sensing has larger spatial coverage and in situ sensing has higher temporal resolution

1.2.2 **Layers of the Sensor Web**

Sensor web is an integration of following three layers.
- Sensor Layer
- Communication Layer
- Information Layer
1.2.2.1 Sensor Layer

Over 100 physical, chemical, and biological properties can now be measured by sensors. Sensors have become smaller, cheaper, more reliable, more power efficient, more widely available, and more intelligent. Trillions of sophisticated sensors will be embedded into our daily lives, thereby providing extensive monitoring in the near future. (Estrin et al., 2001). Sensors can be classified into two in situ and remote, based on medium or object that the sensors are sensing. In situ sensors have higher accuracy and better resolution and less expensive. Remote sensors have better spatial resolution than in situ sensors.

1.2.2.2 Communication layer

Communication layer controls the data/command transmission within and between sensor layer and information layer. Communication layer can be internet, satellite, cellphone or radio based network. Therefore communication layers of the sensor web will be hybrid systems. Finally transferred information is routed through the internet to the information layer.

1.2.2.3 Information layer

Information layer is the place where the sensing resources can be stored, disseminated, exchanged, managed, displayed and analyzed. Sensor resources may be sensors, sensor’s locations, sensor’s real time, near real time or achieved measurements, command and control to sensors, models that need to sensor measurements as inputs, and other related information for user’s applications. Users have seamless access to ready use sensing resources of the sensor web. Interoperability is the key to a successful information layer for the Sensor Web.
Fig. 1. Sensor web and three layers of sensor web

The rest of the paper is organized as follows, section 2 describes the requirements, challenges and design principles that sensor web must meet, section 3 introduce the major classification and then section 4,5,6 discuss detail classification and the current state of the research.

2. Design Goals

The major component of the information layer is to build a geospatial information structure which spatially enables the Sensor Web. Location of a sensor makes sensor’s observation meaningful. The design of the spatial Sensor Web infrastructure needs to satisfy following goals.

2.1 A mechanism for sensor discovery

The Sensor Web is a collection of large number of heterogeneous sensors which is similar to World Wide Web concept. How to find a sensor that meet user’s requirement is a difficult problem. The user view sensor web as a global sensor network and use high level queries. The worldwide sensor web should support rich queries, which could include arithmetic, aggregation, and other database operators.

2.2 Integration with existing sensor networks

There are many sensor networks already exiting and operating with it own standards, protocols and data formats. By integrating sensors with existing networks will save money and resources.
2.3 Accommodating a variety of observables and observed values

Different types of sensors sense different type of measurements and collect different observations. Sensor Web should support these various types of sensors.

2.4 Georeferencing the sensor observations

Sensor Web should allow users to provide not only the observed value but also spatial information.

2.5 Providing open access

Design of Sensor Web infrastructure needs to open. It will not depend on specific programming language, vendor platform or any specific data format. Openness makes Sensor Web more powerful. It needs to support interconnection and collaborations between sensor and sensor networks.

2.6 Planet-wide local data collection and storage

World wide sensors collect huge amount of data and want to store most recent observations and historical records. The system should store observations near their sources and transmit them across the Internet only as needed.

2.7 Real-time adaptation of collection and processing

The system should be able to reconfigure data collection and filtering processes in reaction to sense data and may have to control actuators.

3. Classification of Sensor Web

Sensor Web can be broadly classified into three sections.

3.1 Hierarchical and Receptor-based Systems

There are variety of systems aimed at managing and querying the data produced by sensors, both physical and virtual. These systems have assumed topologies similar to the high fan-in systems but there are significant differences. Theses are the system in which large number of receptor exist at the edge of the network that collect raw data readings. These heterogeneous edge devices produce data that will be aggregated locally with data from other nearby devices. That data will be further aggregated within a larger area, and so on. This kind of arrangement is called high-fan-in systems.
This hierarchical bowtie shape arises because of two reasons. First, data cleaning, filtering and aggregation close to the edges will save the bandwidth and processing costs. Second, many of target applications naturally have a hierarchical structure.

IrisNet[1] uses a two level hierarchy consisting of receptors feeding into a core composed of a set of nodes running a distributed database. Several systems have produced frameworks for simplifying the development of scalable, robust internet services concentrating on problems that are generic across all internet services, such as load balancing, resource allocation, and network placement. IrisNet addresses issues that are unique to services that need to collect vast amounts of data and process queries on the data. IrisNet organized in a hierarchy but it does not address hierarchical aggregation or successive processing of queries.

The HiFi[3] system consists of a three-level hierarchy: receptors, initial processing, and core processing. The receptors consist of sensor networks and query processor like TinyDB[14]. The receptors feed their streams of partially aggregated data to the initial processing level. In this level aggregates data further by small computing devices. The data processing in this level is done by stream query processor like scale down version of TeleGrapghCQ[37]. Processed and aggregated streams in the second level feed to the final level which is fully-flagged server running TeleGrapghCQ.

3.2 Streaming query systems

These systems are applications that monitor continuous streams of data. For example military applications that monitor readings from sensors worn by soldiers, financial analysis applications that monitor streams of stock data reported from various stock exchanges, and audio-visual departments that monitor the location of borrowed equipment. These applications substantially differ from conventional business data processing.

Monitoring applications are very difficult to implement in traditional DBMSs because basic computational model is wrong. Traditional DBMSs have a human active
database passive model (HADP) while monitoring applications often require a database active human passive model (DAHP). HADP means DBMS is a passive warehouse storing a large collection of data elements and that humans initiate queries and transactions on this warehouse. In DAHP model the DBMS get their data from external sources rather than from humans issuing transactions and alert humans when abnormal activity is detected. These applications require storing some history of values reported in a stream, they are trigger oriented, have to answer with incomplete information, they have real time requirements.

These systems handles large numbers of continuous queries over high-volume, highly variable data streams. Systems in area are TelegraphCQ, Aurora, Aurora*, Medusa and PIER. The Aurora Project has branched into two separate efforts to extend stream processing to a distributed environment. Aurora* is designed for a single administrative domain and addresses QoS and dynamic operator repartitioning and movement to achieve load balancing and fault-tolerance. The Medusa System arranges single site Aurora data stream processors in a loosely federated network mediated by agoric principles to enable spanning of organizational boundaries and load balancing.

3.3 Sensor Middleware/ Querying sensor networks

In sensor web there must be a method to change the behavior of a sensor network on the fly. This will leads to save the power consumption of the network. In network-level abstractions, a sensor network is treated as a whole and is regarded as a single abstract machine. Whole network consider as a virtual database system. Sensor networks are often for collecting sensing data, the database approach is one solution. Database systems allow users to issue queries in a declarative SQL-like language. Database abstraction provides a simple and easy-to-use interface. However, it is suitable only for describing query operations to a sensor network. Although Cougar and TinyDB extend SQL so that users can express continuous sensing tasks, they are still not expressive enough to cover all sorts of sensor network applications.

4. Hierarchical and Receptor-based Systems

4.1 IrisNet

IrisNet[1] introduce two tire architecture called sensing agents (SA) and organizing agents (OA).
SAs implement a generic data acquisition interface which allow to access different sensors. SAs collect raw sensor data from several sensors. A sensor host receives one or more raw sensor feeds from directly attached sensors and stores them in circular shared memory buffers. Multiple services can share an SA and access these buffers.

OAs provide services that store service-specific data the SAs produce in a distributed database. OA participates in only one sensing service. The group of OAs for a single service collects and organizes sensor data to answer the particular class of queries relevant to the service.

4.1.1 Choice of database.
IrisNet uses XML to represent sensed data and claim that XML is well suited to represent hierarchical data and self describing tags to organize data. The schema describes the static data and dynamic data.

4.1.2 Distributing the database.
IrisNet executes a distributed algorithm with several statistics to dynamically partition the sensor database among the collection of OA,s. This adaptive data placement algorithm reduces the average query response time and network traffic.
4.1.3 Query routing.
They have used XPATH query language to query the xml documents. In IrisNet, the query is routed directly to the lowest common ancestor of the nodes that the query described. As node can be replicated IrisNet selects one of lowest common ancestor as OA and route query to that OA. On receiving the query OA evaluates the result. If it has not enough data OA send sub queries to other OAs. They perform similar gathering task. Finally the LAC OA collects the different responses and sends combine result back to the user.

4.1.4 Programmable SAs
A single SA has one or more senselet which are used to upload and control the code that filters sensor readings dynamically in a service-specific fashion. A senselet instruct to the SA to take raw sensor reading and process them and send to the nearest OA.

4.2 Hourglass
The Hourglass[36] project is developing a data collection network (DCN) for accessing sensor-based data. Their infrastructure consists of an overlay network of wired nodes collecting data from various sensor networks. They generalize system components into producers, consumers, and services and focus on how best to establish and maintain circuits in the overlay network.

Hourglass is an internet based infrastructure that support naming, discovery, schema management, routing, and aggregating data from potentially many geographically diverse sensor networks.

The essential data model in Hourglass is a circuit, which can be thought of as a set of network links that connect one or more sensor network sources to one or more recipients of that data[36]. Data flowing through the circuit can be filtered, aggregated, compressed, or temporarily buffered by a set of services.
An hourglass system is shown in figure 5. In an Hourglass data flow is based on a circuit. A circuit is a data path through the system that ensures that an application receives the data in which it is interested. A circuit consists of intermediate services that perform operations on the data. Each service provider consists of circuit manager. Circuit manager responsible for the set-up and management of circuits, and a registry, which helps to discover the service. A new entity can join the hourglass system by implementing core functionality required by these interfaces. Sensor network and applications may connect to the Hourglass system via proxy services in order to minimize cost of running an Hourglass service.

4.2.1 Circuit
A circuit is formed by connecting a set of data producers, a data consumer and in-network services into a tree. Circuits are created by circuit manager according to request from applications and needs to be refreshed periodically. A circuit also has a globally unique circuit identifier that is used to refer to it throughout the system.
4.3 SensorMap

SensorMap[18] portal consist of four components GeoDB, DataHub, Aggregator and SensorMap GUI. The GeoDB indexes static metadata about sensors so that it can be queried efficiently. The metadata includes information such as publisher name, sensor location, sensor name, sensor type, data type, unit, sensor data access methods, and free text descriptions. GeoDB indexes the metadata by using hierarchical triangular mesh indexing scheme which is particularly suitable for geographic queries. The indexing is implemented as table-valued functions in a SQL server. The DataHub web service provides an interface for registration of new sensor and archiving of real-time sensor data. For sensors that provide public web interfaces, they can register their URL directly to GeoDB. For sensors with Internet connection but no URL, the DataHub web service provides a simple interface to cache sensor data. The Aggregator or the SensorMap GUI directly retrieves these cached data from DataHub rather than trying to contact the sensors. The Aggregator clusters are geographically near-by sensors and summarize data from sensor clusters in useful ways. The SensorMap GUI lets users query data sources and view results on the map.

4.4 Mobile Web Services Framework

This paper [20] presents technology for search web of variable types, sensor, sensor data storage and way to access and control through World Wide Web. The paper designs Ubiquitous Sensor Network based mobile sensor web service framework for environment monitoring in a standard protocol of mobile web service based mobile environment. The framework handles the information from sensor nodes based on Service Oriented Architecture, and transfer data to data storage through wireless connection by arrangement of field sensor node and server with sensor network. And it uses standard SOAP and XML for different platform call service. The frame work provides sensor module that saves data from distributed sensor for measurement of physical space. Gathering module which consists of Zigbee RF module combining after removed from saving sensor module inter-communicates when selectively identifies ID connecting with some gathering modules which exists in regular distance, and transmit sensing information to PC or server. It is characterized that it consists of PC or server which saves and manages each sensing information from several gathering devices. Each sensing information which is saved in database provides service according to mobile user’s demand.

4.5 GeoSWIFT

The aim of the GeoSWIFT[21] is to develop an open distributed geospatial infrastructure for Sensor Web. The core of GeoSWIFT is the open geospatial sensing services, which serve as a single queryable “global sensor” for Sensor Web users. The role of GeoSWIFT Sensing Server is to provide a Web-enabled interface for sensor systems and its geospatial information as well. The Server acts as a wrapper which hides the different communication protocols, data formats and standards of sensor
systems behind the server and provides a standard interface for clients to collect and access sensor observations and manipulate them in different ways.

### 4.6 SenseWeb

SenseWeb[22], system can initiate and access sensor data streams from shared sensors across the Internet. It ensures optimal sensor selection and efficient sharing among multiple applications. The designed SenseWeb architecture let multiple concurrent applications share sensing resources contributed by several entities in a flexible but uniform manner. The architecture’s key components are the coordinator; sensors, sensor gateways, and mobile proxy; data transformers; and applications. The coordinator is the central point of access into the system for all applications and sensor contributors. The functions of the coordinator are internally divided between two components: the tasking module and streaming sensor database. The tasking module accepts applications’ sensing requirements and tries to satisfy these from available sensing resources. Streaming sensor database attempts to minimize the load on the sensors or the respective sensor gateways by combining the requests for common data and using a cache for recently accessed data. A transformer converts data semantics through Processing. Applications are all users of sensor data.

### 4.7 SWL

This is a language (SWL: Sensor Web Language)[25] for communication with wireless sensor networks. It is extension of SensorML which provides XML schema. SWL provides a foundation for web-based real time querying. Four versions of the SWL grammar are used to generate the four different versions of the SWL interpreter. SWL messages are exchanged between all four components of the sensor web. Each version of the SWL message interpreter is generated by a compiler. This approach lets us base all the interpreters on SWL grammar, and generate different versions using variants of the visitor classes. To change a sensor web configuration one edits the corresponding SWL grammar defining the sensor web, and regenerates the interpreters for each component. This allows for fast deployment and update of sensor webs, and provides a method for automating the distribution of the four software components.

### 4.8 Composing Semantic Services in OSRE

This paper[28] presents architecture and techniques for declarative application in sensor rich environments. User only needs to specify the end results rather than how these results are computed. The environment provides set of semantic services that can be discovered by mobile user. A user issues declarative queries that express the goal for the environment to report back or react to. A query planner generates a workflow of semantics services that is then assigned to a set of physical nodes in the environment for execution. The assignment process takes into account the service workflow, its resource requirements, and available resources in the network. The assignment output is a task graph that assigns each service to a node in the network. A node takes a subset
of the workflow, instantiates corresponding services, and executes them. The execution results are then returned to the user or cached for future queries.

4.9 Open Sensor Web Architecture

The paper presents Open Sensor Web Architecture (NOSA)[29] is built upon the Sensor Web Enablement (SWE) standard defined by the Open Geospatial Consortium (OGC), which is composed of a set of specifications, including SensorML, Observation & Measurement, Sensor Collection Service, Sensor Planning Service and Web Notification Service. NOSA is an implementation of the OGC SWE standard, which standardizes the vision of Sensor Web. SensorML, O&M, SCS, SPS and WNS are coupled together to create an integrated platform for registering, discovering and accessing heterogeneous distributed sensors using Web Services. NOSA presents a reusable, scalable, extensible, and interoperable service oriented Sensor Web architecture that conforms to the SWE standard, integrates Sensor Web with Grid Computing and provides middleware support for Sensor Webs.

4.10 Service Oriented Sensor Web

Service Oriented Sensor Web[33] presents sensor web as a combination of SOA, grid computing and sensor networks. Various sensors and sensor nodes form a web view and are treated as available services to all the users who access the Web. This brings the heterogeneous sensors into an integrated and uniform platform supporting dynamic discovery and access. A client can query the entire sensor web and get the response either from real-time sensors that have been registered in the web or existing data from a remote database. It provides the middleware infrastructure and the programming environment for creating, accessing, and utilizing sensor services through the Web. This architecture follows SWE standards, integrates sensor web with grid computing and provide middleware supports for sensor webs.

4.11 Semantic Sensor Web

The SSW[34] is a framework for providing enhanced meaning for sensor observations to enable situation awareness. It enhances meaning by adding semantic annotations to existing standard sensor languages of the SWE. This enhanced access to sensor data than SWE alone and act as a linking mechanism to bridge the gap between the primarily syntactic XML-based metadata standards of the SWE and the RDF/OWL-based metadata standards of the Semantic Web. The paper provides an environment for enhanced query and reasoning within the sensor domain by incorporating OGC and W3C standardization efforts into a SSW.

4.12 Sensor Web Enablement

SWE[35] consists of a set of standard services to build a unique framework for discovering and interacting with web-connected sensors and for assembling and utilizing sensor networks on the web. SWE is focused on developing standards to
enable the discovery, exchange, and processing of sensor observations, as well as the
tasking of sensor systems. The vision is to define and approve the standards foundation
for "plug-and-play" Web-based sensor networks. SWE built and prototyped seven
standards O&M, SensorML, TML, SOS, SPS, SAS and WNS.

4.13 HiFi

The HiFi[3] system has built using TelegraphCQ stream query processor and the
TinyDB, sensor database system. It consists of a three-level hierarchy: receptors,
initial processing, and core processing. The receptors consist of sensor networks and
query processor like TinyDB. The receptors feed their streams of partially aggregated
data to the initial processing level. In this level aggregates data further by small
computing devices. The data processing in this level is done by stream query processor
like scale down version of TeleGrapghCQ. Processed and aggregated streams in the
second level feed to the final level which is fully-flagged server running
TeleGrapghCQ. This node runs queries that correlate streams across all devices.

5. Streaming query systems

5.1 Centralized Streaming query systems

In the recent years in other application domains such as network monitoring or
telecommunications data stream processing has received huge attention. Because of
that a rich set of query languages and query processing approaches for data streams
exist. They were designed for centralized architectures in the first place.

5.1.1 Aurora

Aurora [5] is a centralized stream processing system. The basic job of Aurora is to
process incoming streams in the way defined by an application administrator. It is a
basically data flow system and uses box and arrow paradigm. Hence tuples flow
through loop free directed graph of processing operations. Finally output streams
presented to applications which must be programmed to deal with the asynchronous
tuples in an output stream. It also maintains historical storage to support ad-hoc
queries. Aurora has eight primitive operations for expressing its stream processing
requirements. Every Aurora application must be associated with a query that defines
its processing requirements. It also supports real time processing, view and ad-hoc
queries. All of these operations use the same conceptual building blocks. Fig 6.
shows the high level system model of Aurora.
5.1.1.1 Operators
Some of operators used in Aurora as follows. *Windowed* operators that operate on sets of consecutive tuples from a stream ("windows") at a time. Every windowed operator applies an input user-defined function to a window. *Slide* advances a window by "sliding" it downstream by some number of tuples. Tumble resembles Slide except that consecutive windows have no tuples in common. Latch resembles Tumble but can maintain internal state between window calculations.

5.1.1.2 Query Model
Basic building blocks of queries are standard set of well-defined operators (boxes). Arrows represent input streams of an operator. Operator transforms input streams into one or more output streams (out arrows). These queries are constructed using box-and-arrow based GUI.

5.1.1.3 Run-time Operation
Fig. 7. shows the single node Aurora run time architecture. The main component of the system is the scheduler that determines which box to run and how many tuples that might be waiting in front of a given box to process and how far to push them toward the output. Aurora also has a Storage Manager that is used to buffer queues when main memory runs out. Aurora continuously monitors the QoS of output tuples and support to drives the Scheduler in its decision making. It also informs the Load Shedder when and where it is appropriate to discard tuples in order to shed load. Load shedding improves the QoS delivered to the applications. When load shedding is fail Aurora will try to reoptimize the network using standard query optimization techniques. It is also retune the scheduler by collecting new statistics or switching scheduler disciplines.

5.2 Distributed Streaming query systems

Stream processing systems operate in distributed fashion because stream oriented systems are inherently geographically distributed and distribution support scalable load management and higher availability. They will run across the internet on computers.

5.2.1 Aurora* and Medusa

This paper presents two distributed stream processing systems Aurora* and Medusa [6]. Aurora* is a relatively small scale distribution all within a one administrative domain. Medusa is a large scale distribution across administrative boundaries. Aurora* consists of multiple single node Aurora belongs to same administrative domain and cooperate to run the Aurora query network on the input stream. Each Aurora node supporting the running system will continuously monitor its local operation, its workload, and available resources. All dynamic reconfiguration will take place in a decentralized fashion, involving only local, pair-wise interactions between Aurora nodes. A Medusa participant is a collection of computing devices administered by a single entity. Those participants range form collections of stream processing nodes capable of running Aurora and providing part of the global service, to PCs or PDAs that allow user access to the system, to networks of sensors and their proxies that provide input streams.

The distributed architecture divides into intra-participant distribution (a relatively small-scale distribution all within one administrative domain, handled by Aurora*) and inter-participant distribution (a large-scale distribution across administrative boundaries, handled by Medusa).

Aurora* is one administrative domain consists of collection of single-node Aurora servers that can cooperate to run the Aurora query network on the input streams. First Aurora* query network create a crude partitioning of boxes within entire network. Each Aurora node continuously monitor its local operation, its workload, and available resources. It will coordinate with each other in order to balance the resources.
Medusa consist of a collection of computing devices administrate by a single entry. Medusa is an agoric system using economic principles to manage and share the load.

Both Aurora* and Medusa require a scalable communication infrastructure. This infrastructure must include a naming scheme for participants and query operators and a method for discovering where any portion of a query plan is currently running and what operators are currently in place, route messages between participants and nodes, multiplex messages on to transport-layer streams between participants and nodes, and enable stream processing to be distributed and moved across nodes. The communications infrastructure is an overlay network, layered on top of the underlying Internet substrate.

5.2.2 PIER

PIER[7,8] presents technology for massively distributed query processing at a significantly larger scale and method for querying of Internet-based data \textit{in situ}, without the need for database design, maintenance or integration. This paper introduce peer to peer information exchange and retrieval query engine that scales up to thousands of participating nodes which is built on top of a Distributed Hash Table. The paper presents new architecture combining traditional query processing with recent peer to peer networking technologies.

5.2.3 Borealis

Borealis[9] is a distributed stream processing engine. It derives core stream processing functionality from Aurora and distributed functionality from Medusa. The collection of continuous queries submitted to Borealis can be seen as one giant network of operators whose processing is distributed to multiple sites. In sensor networks, sensor proxy interface can act as a Borealis site. In each site, Borealis server is running whose main component is a query processor. Streams are fed in to query processor and results are pulled through I/O queues which route tuples to and from remote Borealis nodes and clients. Query processor also contains storage manager which is responsible for storage and retrieval of data.

5.2.4 Fault Tolerance in the Borealis

This paper[10] present a replication-based approach to fault-tolerant distributed stream processing in the face of node failures, network failures, and network partitions. The replication scheme and algorithm implemented in Borealis. Each node implements three stage state machine. As long as all upstream neighbors of a node are producing stable tuples, the node is in the stable state. In this state, it processes tuples as they arrive and passes stable results to downstream neighbors. If one input stream becomes unavailable or starts carrying tentative tuples, a node goes into the up failure state, where it tries to find another stable source for the input stream. Once a node receives the stable versions of all previously missing or tentative input tuples, it transitions into the stabilization stat
5.2.5 Load Distribution in the Borealis

The paper[13] propose new load distribution algorithm. It balances the average load among the processing nodes and the load variance on each node. The algorithm consist of two parts, first the global algorithm to make the initial operator distribution second the dynamic load redistribution algorithm which moves operators between nodes in a pair-wise fashion. In the global algorithm care about quality of the resulting mapping plan without considering how much load is moved. In the pairwise algorithm, try to find a good tradeoff between the amount of load moved and the quality of the resulting mapping plan. Both algorithms are based on the basic load-balancing scheme.

5.2.6 ISS

Internet-scale sensing systems (ISS)[11] consist of a large number of geographically distributed data sources tied into a framework for collecting, filtering, and processing potentially large volumes of real-time data. ISS systems differ from conventional distributed systems. This paper proposes design principles for ISS to achieve scalable, robust ISS platform. These techniques avoid complexity of traditional replication and consistency mechanisms, in recognition of differing data needs of ISS applications. ISS systems differ from conventional distributed systems, including the number of data sources, differing data quality requirements, and necessity to continue operating despite link and node failures. The approach is dependability based on a set of metrics that reflect on the quality of the answers returned by the system.

5.3 Distributed and Declarative Streaming query systems

In this approach it can be viewed as a multi-user distributed expression evaluator in which sensors and actuators form the edge level devices. This approach has several advantages in managing and controlling a complex, federated, and evolving network. (1) a declarative logic language provides a natural way to express the kinds of statements that are common to this application domain, through temporal and positional logic rules, facts and expressions, and (2) distributed evaluation of such logic expressions provides many opportunities for performance optimization yielding an efficient system.[17]

5.3.1 Sophia

In Sophia[17] distributed set of sensors report data about aspects of systems local state and the local perspective of the rest of the network. A declarative programming environment, which is distributed, evaluates logic statements about the system. Then set of actuators in distributed network perform local actions. First Sophia uses a single, flat logic terms database for storing all of its terms. The stored terms include all predicate rules and facts, as well as loadable modules, which are just a set of
terms. Second it includes a local unification engine that is based on standard logic unification, with some differences. Third, Sophia interfaces with sensors and actuators of the host system, which effectively serve as I/O for Sophia. Fourth, a remote evaluator is responsible for delegating evaluation of an expression to a particular remote Sophia node. Finally, an expression scheduling mechanism is responsible for maintaining the calendar of evaluations scheduled for the future.

6. Sensor Middleware / Querying sensor networks

6.1. TinyDB

TinyDB [14] is a query processing system which is generated for extracting information from a TinyOS sensor network. It provides a simple, SQL-like user interface, it does not require writing embedded C codes to specify the information want to extract. It collects the data, specified by a query, from motes, filters it according to the query parameters, aggregates it again according to the aggregation function specified in the query, and routes the data to a PC. TinyDB uses power-efficient in-network algorithm. The TinyDB system has subsystems, Sensor Network Software and java based client interface. The Sensor Network Software runs on all motes in the network. The client interface consists of a set of Java classes and applications which is used to access the motes from a PC. These java classes include a network interface class that allows applications to inject queries and listen for results, classes to build and transmit queries, a class to receive and parse query results and a class to extract information about the attributes and capabilities of devices. Fig. 8 shows the high level model of TinyDB.

![Fig. 8. high level system model of TinyDB](image-url)
In TinyDB aggregation is a core service. It processes aggregates in the network by computing over the data as it flows through the sensors, discarding irrelevant data and combining relevant readings into more compact records when possible.

6.1.1 Ad-Hoc Routing Algorithm
TinyDB maintains a routing tree (spanding tree) rooted at the base station. It executes following algorithm to build the routing tree.

- one mote is appointed to be the root (it is the point where the user interfaces to the network).
- The root broadcasts a message asking motes to organize into a routing tree.
- In that message it specifies its own id and its level, or distance from the root.
- Any mote without an assigned level that hears this message assigns its own level to be the level in the message plus one.
- chooses the sender of the message as its parent.
- Each of these motes then rebroadcasts the routing message, inserting their own ids and levels.
- These routing messages are periodically broadcast from the root, so that the process of topology discovery goes on continuously.
- Parents are retained unless a child does not hear from them for some long period of time, at which point it selects a new parent using this same process.

Then every sensor has its own query processor that process and aggregates sensor data and maintains all routing information. TinyDB intelligently distributes and executes aggregation queries in the sensor network in a time and power-efficient manner. TinyDB focuses on acquisitional issues: where, when and how often to sample and deliver the data. TinyDB supports SQL-style queries (without joins) over a single table called sensors. Treat entire sensor network as a “table”. Columns present the sensor attributes like node id, value, and locations etc. Example query:

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

Difference between TinyDB queries and SQL queries is that the output of a TAG query is a stream of values, rather than a single aggregate value.

There is an extension of TinyDB, sensornet query engine to support more sophisticated data analysis which introduce some of unique features and constrains of embedding complex sensing algorithms in an extensible, declarative query framework. TinyDB provides a foundation for many kinds of simple monitoring queries. Those include two extensions to existing query language. Events provide a mechanism for initiating data collection in response to some external stimulus. Events are generated explicitly either by another query, by software in the operating system or by specialized hardware on
the node that triggers the operating system. Storage points accumulate a small buffer of data that may be referenced in other queries.

6.2 Caugar

The approach of the Caugar [19] is tasking the sensor networks using declarative queries. This approach hides the node level programming from the user. It implements a query-based database interface and uses SQL-like language for gaining information of wireless sensor networks. To achieve energy-efficiency, Cougar pushes selection operations to the sensor nodes so that they can reduce the amount of data to be collected.

In Caugar each sensor has a query proxy layer which is interacting with both routing layer and application layer. Query optimizer in the gateway creates the distributed query processing plan according to the user query with support of catalog information and query specification. Then the plan is disseminated to all relevant sensor nodes. Control structures synchronize sensor behavior and start the query. Finally record sends back to the gateway. This computation happens on the fly.

6.2.1 An Example

Let Q be long running query to monitor the average temperature of an office every t seconds if notifies to an administrator the average temperature is greater than user defined threshold. First the query optimizer optimize the query Q taking existing query workload into account and merge the query with similar existing query P. Then new query plan is QP. The query plan QP determines the leader of this query and this is the location node where the computation of the average temperature takes place. The leader is fixed sensor with more power remaining or a randomly selected node using distributed leader selection algorithm. None leader nodes reads the values periodically and send the values to the leader. Fig. 9. shows the query plan of a non leader node. Their plan may contain aggregate data from other sensors.

![Query plan of a non-leader node](image)

Fig. 9. Query plan of a non-leader node.
The leader calculates the average and check with the threshold value. Fig. 10 shows the query plan of the leader.

![Query Plan of the Leader Node](image)

The query plans are disseminated to the query proxies of all relevant sensor nodes at the start time. Then query proxies will register the query, create the local operator tree, active relevant sensors and then return the result.

### 6.3 SINA

SINA [39] allows sensor applications to issue queries, command tasks, collect replies and monitor changes within the network. SINA modules are running on each sensor node, provide adaptive organization of sensor information, and facilitate query, event monitoring and tasking. In SINA sensor nodes are automatically clustered in order to support energy efficiency and operation scalability. It also has attribute based naming scheme based on an associative broadcast to manage the spreadsheets. The SINA kernel is based on a spreadsheet database or querying and monitoring. Each logical datasheet consist of cells and each cell represents a sensor node attribute. Each sensor node maintains a whole datasheet and each cell is unique. So one can view the sensor network is a collection of datasheets. The cells are The cells are initiated in a node by requests from other nodes. The nodes make requests in a SQL-like statement.

### 6.4 DsWare

DsWare[40] is a data service middleware exists between the application layer and the network layer. It provides data service abstraction to applications. Fig. 11. shows this architecture. It supports group based decision making and reliable data centric storage. It also provides data storage, data caching, group management, event detection, data subscription and scheduling along with architecture modules. This middleware
includes data centric services that process and aggregates the data and then sends the result to the base station. It is not a good solution for heterogeneity and mobility.

![Diagram of the DsWare software architecture](image)

Fig. 11. DsWare software architecture

### 6.5 MiLAN

MiLAN[23] provides a data service that features QoS support. In MiLAN, an application submits a query with a QoS requirement. QoS is defined by the level of certainty about an attribute, based on the assumption that each sensor can measure some basic attributes with predefined reliability. In response to a query, MiLAN creates an execution plan, which specifies the source nodes and the routing tree, such that it satisfies the QoS requirement while maximizing energy efficiency.

### 7. Conclusion

Sensor Web infrastructure has become an interesting topic in various research communities. Sensor Web related research topics span multiple domains include distributed systems, wireless sensor networks, remote sensing, artificial intelligence, and sensor web services. Today sensor networks are not just passive instruments. We can introduce both processing and “intelligence” into the network. Processing can happen at many levels in a Sensor Web environment including on individual sensor nodes, at aggregation points within the network, at the base station or gateway. Sensor web fundamentally change the scientific observation from a passive process to an active one. This will be a deep impact on many aspects of science.

Sensor web technology can be used to successfully address a variety of practical problems. However, these problems are hard due to several reasons. First, is the great deal of variability of application domains. Application requirements varies tremendously from data collection, in network compression, data rate, to hardware requirement from application to application. Second, System dynamism and unreliability is high because nodes/links join and leave the network dynamically and parts of system may be deployed
over time by different vendors, using different technologies. Third, there is unreliability in data due to sensor noise, packet loss and incomplete information due to partial observability of the world.

The Sensor Web research is still in its early stages and the above mentioned challenges still remains as open problems for researchers to address. One such important open problem is to build standardize interfaces between applications, databases, networks, data service providers, and hardware. Models should be developed that treat uncertainty in both data and system as first-class entity for reasoning and system management. There is also a need to build tools for system configurations/management and for data collection and visualization. Another open problem is how we expose sensor data to the internet, using HTTP, SOAP, RDF or Raw byte-streams. The sensor web technologies should allow sensor network querying and reprogramming. The user tasks often specified in high-level queries. Thus, discovering and harnessing data from multiple sensor networks is another problem yet to be addressed. In sensor web, a vast numbers of data sources and simultaneous queries need to be handled. Therefore there should also be mechanisms distribute query processing across the Internet.
Reference:


http://www.cs.brown.edu/research/db/publications/sigmod05.demo.pdf


http://www.planet-lab.org/files/pdn/PDN-03-014/pdn-03-014.pdf

WSW’06 at SenSys’06,


http://www.futurehealth.rochester.edu/milan/IEEENetwork03.pdf

portal.acm.org/citation.cfm?id=1446545


[62]. Quan Le-Trung et al, “Internetworking Wireless Sensor Networks with the Internet”, SWISNET project,University of Oslo, heim.ifi.uio.no/~quanle/master/Internetworking WSNs Internet.pdf


