Global Sensor Networks*

Karl Aberer, Manfred Hauswirth, Ali Salehi
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Abstract

The availability of cheap and smart wireless sensing devices provides unprecedented possibilities to monitor the physical world. On the technical side these devices introduce several original research problems, many of them related to the integration of the rampant technology proposals. Global Sensor Network (GSN) is a platform which provides a scalable infrastructure for integrating heterogeneous sensor network technologies using a small set of powerful abstractions. GSN supports the integration and discovery of sensor networks and sensor data, provides distributed querying, filtering, and combination of sensor data, and supports the dynamic adaption of the system configuration during operation through a declarative XML-based language.

Keywords: Sensor networks, sensor middleware, sensor Internet

1 Towards a Sensor Internet

In 1999, *Business Week* named networked micro-sensor technology as one of the 21 most important technologies of the 21st century. Today, cheap and smart devices with multiple on-board sensors, networked with wireless links are available from research groups and companies, for example, MICA Motes from Berkeley or BTNodes from ETH. Currently, sensors run specialized operating systems, mostly TinyOS, and tools exist for accessing and querying sensor data using declarative languages, for example, TinySQL or TAG. To date, the research in the sensor network community has mainly focused on routing protocols and information collection and aggregation in a single sensor network with multiple different sensors connected through wireless links. As the prices of sensors decrease rapidly, there will soon exist huge numbers of sensor networks in different places, managed by different organizations. To fully exploit the potential of this "Sensor Internet," platforms enabling the integration and management

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing.

of the sensor sites and the produced data streams will be required. At the moment sensor networks are being deployed and made accessible in an ad-hoc fashion.

Given the current growth rate, we may soon expect to arrive at a situation comparable to the publication of documents on the Web whose success is mainly based on sharing a few simple logical abstractions (URL, hyperlinks, HTML) and basic communication protocols (HTTP, more recently Web Services) that provide universal access and linking among autonomously published data sources. The Global Sensor Network (GSN) platform takes up these successful ideas and aims at making publication and access to sensor networks and sensor data as simple, powerful, and flexible as accessing Web documents. The design of GSN follows four basic goals:

Simplicity. The goal was to design the system based on a minimal set of powerful abstractions which could be easily configured and adopted to the user's needs. We targeted the possibility to define sensor networks and data streams in a declarative way by using SQL as data manipulation language. As a syntactic framework for system configuration we relied on XML.

Adaptivity. Adding new types of sensor networks and dynamic (re-) configuration of data sources has to be supported during run-time without having to interrupt ongoing system operation (query processing, etc.). To that end we used a container-based implementation allowing dynamic reconfiguration.

Scalability. Targeting a very large number of data producers and consumers with a variety of application requirements, GSN has to consider scalability issues specifically for distributed query processing and distributed discovery of sensor networks. To meet this requirement, the design of GSN is based on a peer-to-peer architecture.

Light-weight implementation. GSN was planned to be easily deployable in standard computing environments (no excessive hardware requirements, standard network connectivity, etc.), portable (Java-based implementation), should require minimal initial configuration, and provide easy-to-use, web-based management tools.

In this paper we provide an overview of the key design decisions of GSN that we made in order to achieve the goals mentioned above. In Section 2 we provide an overview of the conceptual model GSN is based on. In Section 3 we introduce the key abstraction of *virtual sensors*. In Section 4 we discuss our approach to data stream processing. In Section 5 we provide the architecture and implementation of the system. In order to illustrate the potential benefits of GSN we sketch some usage scenarios in Section 6. We conclude in Section 7 by comparing to related efforts in the field.

2 Global Sensor Network Model

The current view on wireless sensor networks is based on a *physical view* as shown in Figure 1. Single sensors or whole sensor networks are connected via wireless connections to an access point. Ad-hoc routing protocols route data to selected sensors (sinks) that communicate with an access point which is connected to the Internet and can process and make available the data received from the sensors. Applications wanting to use sensor data need to first identify the access points, possibly through multiple layers, and then follow the specific access protocols defined.

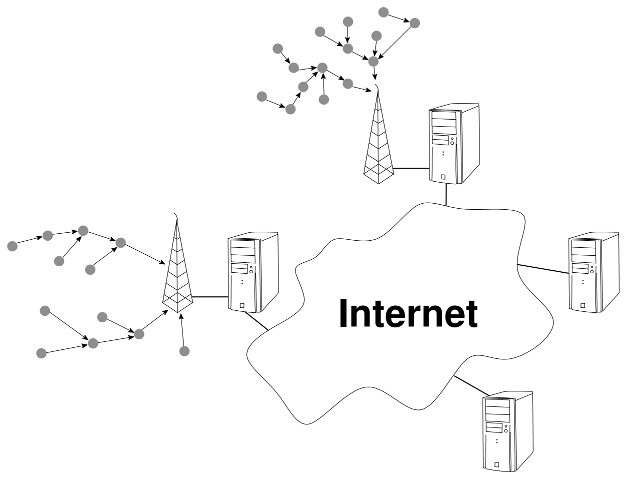

Figure 1: Physical view of sensor networks

A major problem of this model is the dependency of the access to sensor data on the specific implementation. Since sensors can be considered as data sources GSN adopts as a fundamental idea the principle of *data independence*. This abstraction is well-accepted and has proven extremely successful in the design of database systems. It allows the users of GSN to abstract from the highly heterogeneous physical infrastructures coming along with current sensor network platforms, just as in a database system where the user abstracts from the specific implementation of physical data storage and access.

GSN provides a *logical view* on sensor networks based on the *virtual sensor* abstraction. Virtual sensors abstract from the implementation details to access sensor data and model sensor data as temporal streams of relational data. Virtual sensors can also represent derived views on sensor data streams, resulting from post-processing and combination of sensor data from different sources. This is shown informally by the conceptual data flow in Figure 2 which illustrates how data streams from sensor networks can be connected in a peer-to-peer overlay network. Virtual sensors are bound to physical nodes in the network by deploying them to *GSN containers* that host virtual sensors. This approach supports our goal of simplicity, as users with one single abstraction can work with sensor data from heterogeneous sensor sources and any data derived from raw sensor data.

Architecturally GSN adopts a service-oriented view on sensor networks as shown in Figure 3. This architectural paradigm has already successfully proven its applicability in several other domains, for example, Web Services. In this view sensor networks are considered as abstract types of services performing a sensing task and providing a

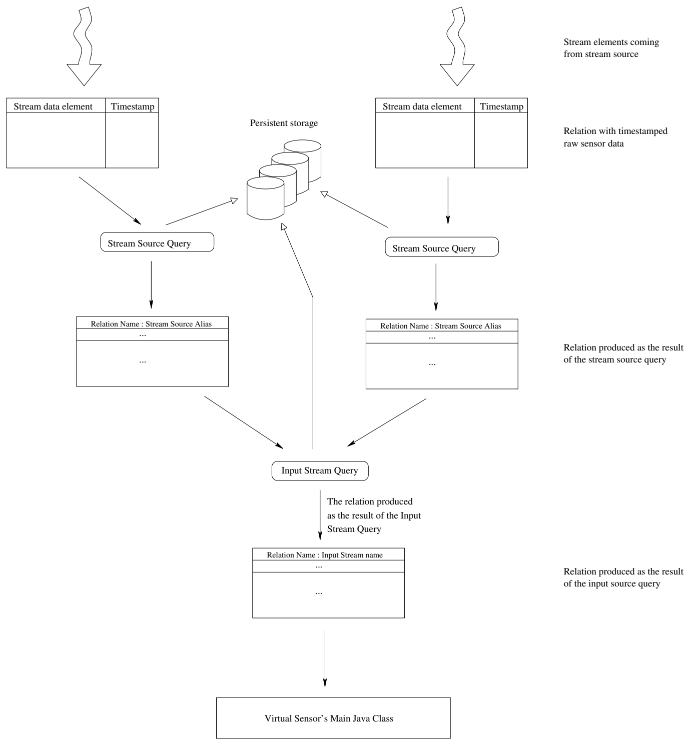
Stream elements coming from stream source
Stream data element
Timestamp
Stream data element
Timestamp
Relation with timestamped raw sensor data
Persistent storage
Stream Source Query
Stream Source Query
Relation Name : Stream Source Alias
Relation Name : Stream Source Alias
Relation produced as the result of the stream source query
Input Stream Query
The relation produced as the result of the Input Stream Query
Relation Name : Input Stream name
Relation produced as the result of the input source query
Virtual Sensor's Main Java Class

Figure 2: Conceptual data flow in a GSN node

specific type of data. The sensor services are published through a peer-to-peer directory and searched based on their properties. Applications can discover sensor networks using the registry and subsequently access sensor networks by using a standard data access interface. The use of the service-oriented paradigm facilitates GSN's design goal of adaptivity as it enables on-demand use and combination of sensor networks.

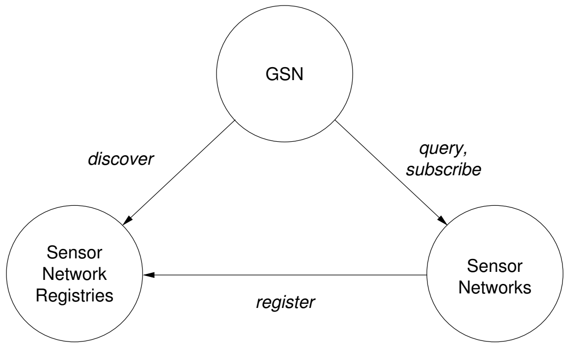

Figure 3: Service-oriented view of sensor networks

3 Virtual Sensor Specification

The key abstraction in GSN is the *virtual sensor* (VS). Virtual sensors abstract from implementation details of access to sensor data and they are the services provided and managed by GSN. A virtual sensor corresponds either to a data stream received directly from sensors or to a data stream derived from other virtual sensors. The specification of a virtual sensor provides all necessary information required for deploying and using it including:

- metadata used for identification and discovery
- the structure of the data streams which the virtual sensor consumes and produces
- a declarative SQL-based specification of the data stream processing performed in a virtual sensor
- functional properties related to persistency, error handling, life-cycle management, and physical deployment

To support rapid deployment, these properties of virtual sensors are provided in a declarative deployment descriptor, as shown in Figure 4.

In order to refer to the data streams produced by a virtual sensor we use logical addressing. Each virtual sensor can be equipped with a set of key-value pairs which can be registered and discovered in GSN. The example specification in Figure 4 shows a virtual sensor with three input streams. The virtual sensors generating the input streams are identified by their metadata (lines 17–18, 25–26, and 33–34) and the metadata for identifying the virtual sensor producing the output stream is given in lines 2–5. This example also demonstrates GSN ability to access multiple stream producers simultaneously.

In GSN data streams are temporal sequences of relational tuples. This is in line with the model used in most stream processing systems. The structure of the data stream a virtual sensor produces is encoded in XML as shown in line 7–11. The structure of the input streams is learned from the respective specifications of their virtual sensor definitions.

Data stream processing is separated into two stages: (1) processing applied to the input streams (lines 20, 28, and 36) and (2) processing for combining data from the different input streams and producing the output stream (lines 38–43). To specify the processing of the input streams we use SQL queries which refer to the input streams by the reserved keyword `WRAPPER`. The attribute `wrapper="remote"` indicates that the data stream is obtained from the Internet through GSN. Thus logical addressing could be used to address the virtual sensor. In the given example the output stream joins the data received from two temperature sensors and returns a camera image if certain conditions on the temperature are satisfied. The specification of the structure of the output stream directly relates to the data stream processing that is performed by the virtual sensor and needs to be consistent with it. In the design of GSN specifications we decided to separate the temporal aspects from the relational data processing using SQL. Thus standard SQL is used for the different processing stages. The temporal processing is controlled by various attributes provided in the input and output stream specifications, e.g., the attribute `storage-size`. Due to its specific importance we will discuss the temporal processing in detail in a later section.

In addition to the specification of the data-related properties a virtual sensor also provides high-level specifications of functional properties: The `<storage>` element allows the user to control how stream data is persistently stored, the `<life-cycle>` element enables the control of network deployment aspects such as the number of threads available for processing, the `<priority>` element controls the processing priority of this virtual sensor when competing for resources with other virtual sensors, and the `disconnect-buffer-size` attribute specifies the amount of storage provided to deal with temporary disconnections.

```
<virtual-sensor name="room-monitor" priority="11">
   <addressing>
      <predicate key="geographical">BC143</predicate>
      <predicate key="usage">room monitoring</predicate>
   </addressing>
   <life-cycle pool-size="10" />
   <output-structure>
      <field name="image" type="binary:jpeg" />
      <field name="temp"  type="int" />
   </output-structure>
   <storage permanent-storage="true" />
   <input-streams>
      <input-stream name="cam">
         <stream-source alias="cam"  storage-size="1"
                        disconnect-buffer-size="10">
            <address wrapper="remote">
               <predicate key="geographical">BC143</predicate>
               <predicate key="type">Camera</predicate>
            </address>
            <query>select * from WRAPPER</query>
         </stream-source>
         <stream-source alias="temperature1" storage-size="1m"
                        disconnect-buffer-size="10">
            <address wrapper="remote">
               <predicate key="type">temperature</predicate>
               <predicate key="geographical">BC143-N</predicate>
            </address>
            <query>select AVG(temp1) as T1 from WRAPPER</query>
         </stream-source>
         <stream-source alias="temperature2"  storage-size="1m"
                        disconnect-buffer-size="10">
            <address wrapper="remote">
               <predicate key="type">temperature</predicate>
               <predicate key="geographical">BC143-S</predicate>
            </address>
            <query>select AVG(temp2) as T2 from WRAPPER</query>
         </stream-source>
         <query>
            select cam.picture as image, temperature.T1 as temp
            from   cam, temperature1
            where  temperature1.T1 > 30 AND
                   temperature1.T1 = temperature2.T2
         </query>
      </input-stream>
   </input-streams>
</virtual-sensor>
```

Figure 4: A complex virtual sensor definition using other virtual sensors as input

In contrast to Figure 4, which shows the specification of a virtual sensor for processing data streams received from other virtual sensors, Figure 5 shows a virtual sensor producing a data stream generated directly by a sensor, in this case a TinyOS-based sensor.

```
1   <virtual-sensor name="Light-sensor1" priority="11">
2      <class>gsn.vsensor.BridgeVirtualSensor</class>
3      <author>Ali Salehi</author>
4      <email>ali.salehi@epfl.ch</email>
5      <description>A TinyOS temperature vsensor</description>
6      <life-cycle pool-size="10" />
7      <output-specification rate="500" />
8      <addressing>
9         <predicate key="geographical">BC143-N</predicate>
10        <predicate key="type">temperature</predicate>
11     </addressing>
12     <output-structure>
13        <field name="temperature" type="int" />
14     </output-structure>
15     <storage history-size="10s" permanent-storage="true" />
16     <input-streams>
17        <input-stream name="temperature" >
18           <stream-source alias="tsensor" storage-size="1">
19              <address wrapper="tinyos">
20                 <predicate key="host">lsirpc24.epfl.ch</predicate>
21                 <predicate key="port">9001</predicate>
22              </address>
23              <query>
24                 select WRAPPER.TEMPERATURE as temperature,
25                        WRAPPER.TIMED as timestamp from WRAPPER
26              </query>
27           </stream-source>
28           <query>
29              select temperature from tsensor
30           </query>
31        </input-stream>
32     </input-streams>
33  </virtual-sensor>
```

Figure 5: A virtual sensor definition using a physical temperature sensor using TinyOS

In contrast to the complex virtual sensor, this virtual sensor obtains its data stream from a specific physically deployed sensor, rather than through GSN. This is indicated by the attribute `wrapper="tinyos"` in line 19. This specification relies on the availability of the implementation of such a wrapper that connects the GSN to a specific type of sensor or sensor network. The implementation of the wrapper is referred to in line 2. The wrapper is responsible for providing implementations of the access functions used in the specification of the output stream of this virtual sensor (`WRAPPER.TEMPERATURE` and `WRAPPER.TIMED` in lines 23–26). In this variant of a virtual sensor specification, a binding of the virtual sensor to its physical implementation is required. Thus the sensor is addressed physically (lines 19–22) rather than logically. Despite this necessary difference in addressing, locally and remotely produced data streams are otherwise treated logically the same way.

4 Data Stream Processing in GSN

Data stream processing has received substantial attention in the recent years in other application domains, such as network monitoring or telecommunications. As a result, a rich set of solutions for query languages and query processing for data streams exist on which we can build. Currently, most stream processing systems use a global reference time as basis for their temporal semantics because they were designed for centralized architectures in the first place.

As GSN is targeted at a distributed "Sensor Internet" imposing a specific temporal semantics might be inadequate and maintaining it might come at unacceptable cost. GSN provides essential building blocks for dealing with time but leaves temporal semantics largely to applications. In our opinion, this pragmatic approach is viable as it reflects the requirements and capabilities of sensor network processing.

In GSN a data stream is a set of timestamped relations, i.e., each element of the data stream consists of a set of tuples. The order of the data stream is derived from the ordering of the timestamps and the GSN container provides basic support to manage and manipulate the timestamps. These services essentially consist of the following components:

1. a local clock at each GSN container
2. implicit management of a timestamp attribute (TIMEID)
3. implicit timestamping of stream elements upon arrival at the GSN container (reception time)
4. a windowing mechanism which allows the user to define count- or time-based windows on data streams.

In this way it is always possible to trace the temporal history of data stream elements throughout the processing history in GSN. Multiple time attributes can be associated with data streams and can be manipulated through SQL queries. In this way sensor networks can be used as observation tools for the physical world, in which network and processing delays are inherent properties of the observation process which cannot be made transparent by abstraction.

Let us illustrate this by a simple example: Assume a bank is being robbed and images of the crime scene taken by the security cameras are transmitted to the police. For the insurance company the time the images are taken at the bank will be relevant when processing a claim, whereas for the police report the time the images arrived at the police station will be relevant to justify the time of intervention. Depending on the context the robbery is thus taking place at different times.

We describe now the services provided by GSN for supporting temporal processing. The production of a new output stream element of a virtual sensor is always triggered by the arrival of a data stream element from one of its input streams. Thus processing is event-driven. Informally, the processing steps are then as follows:

1. By default the new data stream element is timestamped using the local clock of the virtual sensor provided that the stream element had no timestamp.

2. Based on the timestamps for each input stream the stream elements are selected according to the definition of the time window and the resulting sets of relations are unnested into flat relations.

3. The input stream queries are evaluated and stored into temporary relations.

4. The output query for producing the output stream element is executed based on the temporary relations.

5. The result is permanently stored if required and all consumers of the virtual sensor are notified of the new stream element.

Additionally, GSN provides a number of possibilities to control the temporal processing of data streams, for example:

- Bounding the rate of a data stream in order to avoid overloads of the system that might cause undesirable delays.

- Sampling of data streams in order to reduce the data rate.

- Bounding the lifetime of a data stream in order to reserve resources only when they are needed.

In order to specify data stream processing as described a suitable language is needed. A number of proposals exist already, so we compare the language approach of GSN to the major proposals from the literature. In the Aurora project (http://www.cs.brown.edu/research/aurora/) users can compose stream relationships and construct queries in a graphical representation which is then used as input for the query planner. The Continuous Query Language (CQL) suggested by the STREAM project (http://www-db.stanford.edu/stream/) extends standard SQL syntax with new constructs for temporal semantics and defines a mapping between streams and relations. Similarly, in Cougar (http://www.cs.cornell.edu/database/cougar/) an extended version of SQL is used, modeling temporal characteristics in the language itself.

The StreaQuel language suggested by the TelegraphCQ project (http://telegraph.cs.berkeley.edu/) follows a different path and tries to isolate temporal semantics from the query language through external definitions in a C-like syntax. For example, for specifying a sliding window for a query a *for*-loop is used. The actual query is then formulated in an SQL-like syntax. GSN's approach is related to TelegraphCQ's as it separates the time-related constructs from the actual query. Temporal specifications, e.g., the window size, are provided in XML in the virtual sensor specification, while data processing is specified in a subset of SQL. At the moment GSN supports SELECT queries with the full range of operations allowed by the standard SQL syntax, i.e., joins, subqueries, ordering, grouping, unions, intersections, etc. The advantage of using SQL is that it is well-known and SQL query optimization and planning techniques can be directly applied.

5 System Architecture and Implementation

Having discussed the basic abstractions and the query processing features of GSN, we now take a closer look at the overall system architecture. GSN provides a container implementation for hosting virtual sensors. Similar to Web application servers, GSN provides an environment in which sensor network services can easily and flexibly be specified and deployed by hiding most of the system complexity in the GSN container. Using the declarative specifications, virtual sensors can be deployed and reconfigured in GSN containers at runtime. Communication and processing among different GSN containers is performed in a peer-to-peer style through standard Internet and Web protocols. By viewing GSN containers as cooperating peers in a decentralized system, we tried avoid some of the intrinsic scalability problems of many other systems which rely on a centralized or hierarchical architecture. Additionally, we benefit from a high degree of fault tolerance. Targeting a "Sensor Internet" as the long-term goal we also need to take into account that such a system will consist of "Autonomous Sensor Systems" with a large degree of freedom and only limited possibilities of control, similarly as in the Internet.

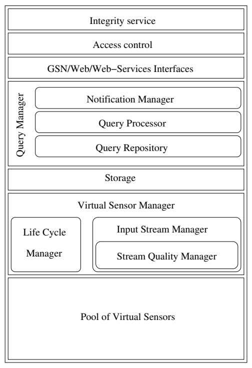

Figure 6: GSN container architecture

Figure 6 shows the basic architecture of a single GSN container. Each GSN container hosts a number of virtual sensor definitions for the set of virtual sensors it is responsible for. The virtual sensor manager (VSM) is responsible for providing access to the virtual sensors, managing the delivery of sensor data, and providing the necessary administrative infrastructure. The VSM has two main subcomponents: The life-cycle manager (LCM) provides and manages the resources provided to a virtual sensor and manages the interactions with a virtual sensor (sensor readings, etc.). The input stream manager (ISM) is responsible for handling sensor disconnections, missing values, unexpected delays, etc., thus ensuring stream quality of service via the included stream quality manager (SQM). The data from/to the VSM passes through the storage layer which is in charge of providing and managing persistent storage for data streams. Query processing is done by the query manager (QM) which includes the query processor being in charge of SQL parsing, query planning, execution of queries (using an adaptive query execution plan). The query repository manages all registered queries (subscriptions) and defines and maintains the set of currently active queries for the query processor. The notification manager deals with the delivery of events and query results to the registered clients. The notification manager has an extensible architecture which allows the user to customize its functionality, for example, having results mailed or being notified via SMS.

The top three layers in Figure 6 deal with access to the GSN container. The interface layer provides access functions for other GSN container and via the Web (through a browser or via web services). These functionalities are protected and shielded by the access control layer providing access only to entitled parties and the data integrity layer which provides data integrity and confidentiality through electronic signatures and encryption. Data access and data integrity can be defined at different levels, for example, for the whole GSN container or for individual virtual sensor.

We implemented the GSN container in Java. The core implementation of the container is around 15,000 lines of Java code which encompasses the query processor, query manager, network manager, input stream manager, storage manager and life cycle manager in addition to the web interface. For deploying a virtual sensor a user has to specify the XML deployment descriptors as illustrated in Section 3. For enabling a new type of sensor or sensor network a Java-based wrapper implementation is required. At the current time we implemented wrappers for the TinyOS family of wireless sensor networks (e.g., Mica2, Mica2Dot,...), wired and wireless (HTTP-based) cameras (e.g., AXIS 206W camera) and several RFID readers provided by Texas Instruments. Since most of the main services are implemented by the GSN container, the implementation of a new wrapper requires the implementation of a standard wrapper interface which requires typically around 100-200 lines of Java (e.g., 150 lines for the TinyOS wrapper).

6 A Simple Case Study

To illustrate the potential applications and flexibility of GSN we provide a simple applications scenario in this section. Figure 7 shows a small-scale example configuration in university buildings similar to the ones we are experimenting with.

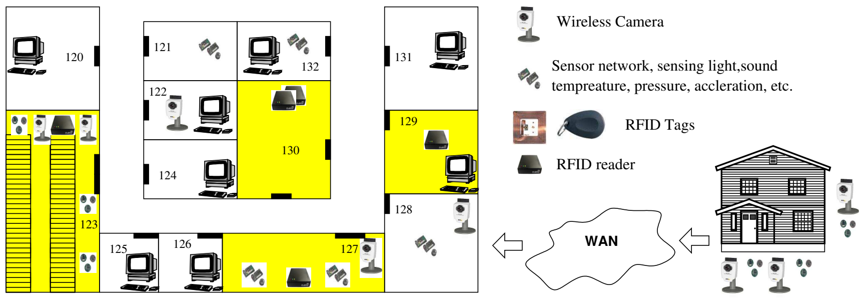

Figure 7: A simple scenario

We assume the following, fairly typical hardware setup:

- wireless cameras with built-in HTTP access;
- wireless sensors (motes) equipped with light, sound, temperature, pressure, GPS, etc. sensors; we assume that all motes in a single room form a sensor network;
- RFID tags which are attached to the key rings of people, and to books, mobile phones and laptops in the buildings; and
- several RFID readers whose coverage ranges are shown in yellow in Figure 7.

Further we assume that:

- each computer runs a GSN container;
- the webcams are accessed directly via HTTP for which some GSN container holds the corresponding virtual sensor definitions;
- and the other sensors (motes) or a complete sensor network can either be accessed via the local area network or are physically connected to one of the computers close to them; in the first case any GSN container can host the virtual sensor definitions, in the second case, we assume that the computer which has the physical connection also hosts the virtual sensor definitions.

Given this setup, GSN allows the user to accomplish a large variety of tasks. For example, the library manager can register a query to be notified when there are more than 15 books (equipped with RFIDs) in one room in addition to the monthly report on the most popular books of the month (e.g., to buy more of them). Individual users can post one-shot queries to the library (room 123) to get the status of certain books or, in case a book of interest is currently not available, can register a continuous query to be notified when the book is returned to the library.

In another example, a person in room 128 may be interested to receive a stream of camera images whenever a movement in the house is detected or the sound sensor attached to a mote observes some noise above a certain threshold. Assuming the sysadmin's office is 130 and he has an RFID tag on his key ring and the user in office 132 wants to meet him, this user can post a passive query to be notified (e.g., via SMS) whenever the overall GSN system observes the sysadmin in a certain radius around his office.

If someone loses a mobile phone (with an RFID tag attached to its battery slot), one can check for its last location in the building simply by posting a query on the previous observations provided by all sensor networks deployed in the building. One can also register a continuous query to the system in order to be notified (e.g., via email) whenever the mobile phone is observed by any of the sensor networks.

7 Related Systems and Conclusions

So far only few architecture architectures to support interconnected sensor networks exist. Probably the closest approach to GSN is the work by Sgroi et.al. [4] who suggest basic abstractions, a standard set of services, and an API to free application developers from the details of the underlying sensor networks. However, the focus is on systematic definition and classification of abstractions and services, while GSN takes a more general view and provides not only APIs but a complete query processing and management infrastructure with a declarative language interface.

Hourglass [5] provides an Internet-based infrastructure for connecting sensor networks to applications and offers topic-based discovery and data-processing services. Similar to GSN it tries to hide internals of sensors from the user but focuses on maintaining quality of service of data streams in the presence of disconnections while GSN is more targeted at flexible configurations, general abstractions and distributed query support.

HiFi [1] provides efficient, hierarchical data stream query processing to acquire, filter, and aggregate data from multiple devices in a static environment while GSN takes a peer-to-peer perspective assuming a dynamic environment and allowing any node to be a data source, data sink, or data aggregator.

IrisNet [2] proposes a two-tier architecture consisting of sensing agents (SA) which collect and pre-process sensor data and organizing agents (OA) which store sensor data in a hierarchical, distributed XML database. This database is modeled after the design of the Internet DNS and supports XPath queries. In contrast to that, GSN follows a symmetric peer-to-peer approach as already mentioned and supports publish/subscribe besides active queries.

Besides these architectures, a large number of systems for query processing in sensor networks exist. Aurora (Brandeis University, Braun University, MIT), STREAM (Stanford), TelegraphCQ (UC Berkeley), and Cougar (Cornell) have already been briefly related to GSN. Other related systems are Borealis (MIT, Brown University, and Brandeis University) and TinyDB (MIT). Additionally, several systems providing publish / subscribe-style query processing comparable to GSN exist, for example, [3].

The availability of cheap sensing devices will soon create a huge number of sensor networks whose full power will be unleashed as soon as they are interconnected in a "Sensor Internet" to offer powerful, large-scale data processing and integration. To enable this vision, we believe that it is important to follow the example of other successful, global information technologies such as the Web, whose success was mainly based on sharing a few simple logical abstractions and basic communication protocols that provide universal access and linking among autonomously published data sources. The Global Sensor Networks (GSN) platform presented in this article follows this model by making publication of sensor data and access to sensor networks and sensor data as simple, powerful, and flexible as accessing Web documents. GSN hides arbitrary data sources behind its virtual sensor abstraction and provides simple and uniform access to the host of heterogeneous technologies available through powerful declarative specification and query tools which support on-the-fly configuration and adaptation of the running system.

References

[1] M. Franklin, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design Considerations for High Fan-in Systems: The HiFi Approach. In *CIDR*, 2005.

[2] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), 2003.

[3] A. J. G. Gray and W. Nutt. A Data Stream Publish/Subscribe Architecture with Self-adapting Queries. In *International Conference on Cooperative Information Systems (CoopIS)*, 2005.

[4] M. Sgroi, A. Wolisz, A. Sangiovanni-Vincentelli, and J. M. Rabaey. A service-based universal application interface for ad hoc wireless sensor and actuator networks. In W. Weber (Infineon), J. Rabaey (UC Berkeley), and E. Aarts (Philips), editors, *Ambient Intelligence*. Springer Verlag, 2005.

[5] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Technical Report TR-21-04, Harvard University, Electrical Engineering and Computer Science, 2004. http://www.eecs.harvard.edu/~syrah/hourglass/papers/tr2104.pdf.