A Middleware For Fast And Flexible Sensor Network Deployment*

Karl Aberer, Manfred Hauswirth, Ali Salehi
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
[karl.aberer, manfred.hauswirth, ali.salehi]@epfl.ch

ABSTRACT

A key problem in current sensor network technology is the heterogeneity of the available software and hardware platforms which makes deployment and application development a tedious and time consuming task. To minimize the unnecessary and repetitive implementation of identical functionalities for different platforms, we present our Global Sensor Networks (GSN) middleware which supports the flexible integration and discovery of sensor networks and sensor data, enables fast deployment and addition of new platforms, provides distributed querying, filtering, and combination of sensor data, and supports the dynamic adaption of the system configuration during operation. GSN's central concept is the virtual sensor abstraction which enables the user to declaratively specify XML-based deployment descriptors in combination with the possibility to integrate sensor network data through plain SQL queries over local and remote sensor data sources. In this demonstration, we specifically focus on the deployment aspects and allow users to dynamically reconfigure the running system, to add new sensor networks on the fly, and to monitor the effects of the changes via a graphical interface. The GSN implementation is available from http://globalsn.sourceforge.net/.

1. INTRODUCTION

The growing diversity of available software and hardware platforms in the sensor network domain creates problems when these sensor networks are being deployed and applications being developed on top of them. At the moment developers and administrators have to come up with specialized software and deployment strategies for each of the platforms. This is a time consuming and tedious task which in other, admittedly more "stable" domains has been addressed by abstracting from the physical view of a system to a logical model and implementing this logical model in terms of a middleware. The general purpose of middleware systems is to provide powerful abstractions codifying the essential requirements and concepts of a domain and providing flexible means for extending it to the concrete physical environment. This speeds up deployment and additionally pushes standardized APIs which simplifies application development and applications become "portable" over all physical systems included in the middleware.

In the sensor network context, the current research mainly focuses on routing, aggregation, and energy efficient data management algorithms inside one sensor network. The deployment, application development, and standardization aspects are usually not addressed. However, as the price of wireless sensors diminishes rapidly we can expect to see large numbers of heterogeneous sensor networks being deployed in different locations around the globe and being managed by different organizations. A major challenge in such a "Sensor Internet" environment is minimizing the deployment efforts which is a key cost factor in large systems. The requirements of the software infrastructure for processing, storing, querying and publishing data produced from a sensor network, however, are similar and the main difference between various software platforms is due to different abstractions for the available sensors. Additionally, having a generic middleware for sensor networks not only cuts costs, but also opens the possibility of sharing and integrating data among heterogeneous sensor networks.

Our Global Sensor Networks (GSN) middleware addresses these problems and provides a uniform platform for fast and flexible integration and deployment of heterogeneous sensor networks. The design of GSN follows four main design goals: Simplicity (a minimal set of powerful abstractions which can be easily configured and adopted), adaptivity (adding new types of sensor networks and dynamic (re-) configuration of data sources has to be supported during run-time), scalability (peer-to-peer architecture), and light-weight implementation (small memory foot-print, low hardware and bandwidth requirements, web-based management tools).

For a detailed description and analysis of all aspects of GSN we refer the reader to [1].

2. VIRTUAL SENSORS

As noted above, a small set of powerful, easily combinable abstractions are key to successful middleware design. The key abstraction in GSN is the *virtual sensor*. 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. A virtual sensor can have any number of input streams and produces one output stream. 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

*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.

VLDB '06, September 12-15, 2006, Seoul, Korea.

To support rapid deployment, these properties of virtual sensors are provided in a declarative deployment descriptor. Figure 1 shows a fragment of such a virtual sensor definition which defines a sensor returning an averaged temperature. The <life-cycle> element enables the control of network deployment aspects such as the number of threads available for processing, the <storage> element controls how stream data is persistently stored (among other attributes this controls the temporal processing), and the element <output-structure> defines the structure of the produced output stream. 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 is possible).

```
 ...
<life-cycle pool-size="10" />
<output-structure>
  <field name="TEMPERATURE" type="integer"/>
</output-structure>
<storage  permanent-storage="true" size="10s" />
<input-stream name="dummy" rate="100" >
  <stream-source alias="src1" sampling-rate="1"
     storage-size="1h" disconnect-buffer="10">
     <address wrapper="remote">
       <predicate key="type" val="temperature" />
       <predicate key="location" val="bc143" />
     </address>
     <query>select avg(temperature)
            from WRAPPER</query>
   </stream-source>
  <query>select * from src1</query>
</input-stream>
 ...
```

Figure 1: Virtual sensor definition (fragment)

3. DATA STREAM PROCESSING

In GSN a data stream is a sequence of timestamped 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
3. implicit timestamping of tuples 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. 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.

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. Informally, the processing steps then are 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 which 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, etc.

GSN's query processing approach is related to TelegraphCQ 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 SQL. At the moment GSN supports SQL queries with the full range of operations allowed by the standard 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.

4. GSN ARCHITECTURE

GSN follows a container-based architecture and each container can host and manage one or more virtual sensors concurrently. The container manages every aspect of the virtual sensors at runtime including remote access, interaction with the sensor network, security, persistence, data filtering, concurrency, and access to and pooling of resources. This paradigm enables on-demand use and combination of sensor networks. Virtual sensor descriptions are identified by user-definable key-value pairs which are published in a peer-to-peer directory so that virtual sensors can be discovered and accessed based on any combination of their properties, for example, geographical location and sensor type. GSN nodes communicate among each other in a peer-to-peer fashion. Figure 2 depicts the internal architecture of a GSN node.

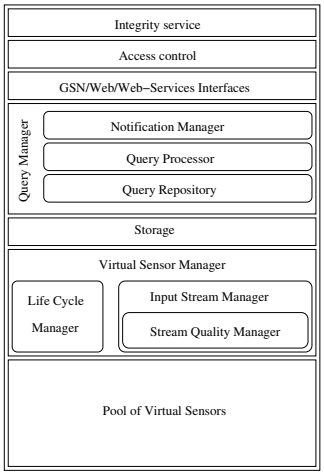

Figure 2: GSN container architecture

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. Its lifecycle manager (LCM) subcomponent provides and manages the resources provided to a virtual sensor and manages the interactions with a virtual sensor (sensor readings, etc.) while the input stream manager (ISM) manages the input streams and ensures stream quality (disconnections, unexpected delays, missing values, etc.). 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, and 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 it to any required notification channel. The top three layers deal with access to the GSN container. The interface layer provides access functions for other GSN containers and via the Web (through a browser or via web services), the access control layer ensures that access is provided only to entitled parties, and the data integrity layer guarantees data integrity and confidentiality through electronic signatures and encryption (this can be defined at different levels, for example, for the whole GSN container or for an individual virtual sensor).

5. IMPLEMENTATION

The GSN implementation consists of the GSN-CORE, implemented in Java, and the platform-specific GSN-WRAPPERS, implemented in Java, C, and C++, depending on the available toolkits for accessing sensors. The implementation currently has approximately 20,000 lines of code and is available from SourceForge (http://globalsn.sourceforge.net/). GSN is implemented to be highly modular in order to be deployable on various hardware platforms from workstations to small programmable PDAs, i.e., depending on the specific platforms only a subset of modules may be used. GSN also includes visualization systems for plotting data and visualizing the network structure.

For deploying a virtual sensor the user only has to specify an XML deployment descriptor as briefly outlined in Section 2 if GSN already includes software support for the concerned hardware and software. Adding a new type of sensor or sensor network can be done by supplying a Java wrapper conforming to the GSN API and interfacing the system to be included. The effort to implement wrappers is quite low, i.e., typically around 100-200 lines of Java code. For example, the TinyOS wrapper required 150 lines of code. Our experience shows that new wrappers can be included usually in less than 1 day. Currently GSN includes already wrappers for the TinyOS family of motes (Mica, Mica2, Mica2Dot, TinyNodes, etc.), USB and wireless (HTTP-based) cameras (e.g., AXIS 206W camera), and several RFID readers (e.g., Texas Instruments).

A key factor determining the usability of GSN is its scalability in the number of queries and clients. To evaluate GSN's scalability we performed experiments with 22 motes and 15 cameras arranged in 4 sensor networks connected via GSN. The devices produced data items every 10, 25, 50, 100, 250, 500, and 1000 milliseconds and we measure the internal processing times of a GSN node for various sizes of produced data items as shown in Figure 3. High data rates put some stress on the system but the absolute delays are still quite tolerable. The delays drop sharply if the interval is increased and then converge to a nearly constant time at a rate of approximately 4 readings/second or less. This result shows that GSN can tolerate high rates and incurs low overhead for realistic rates.

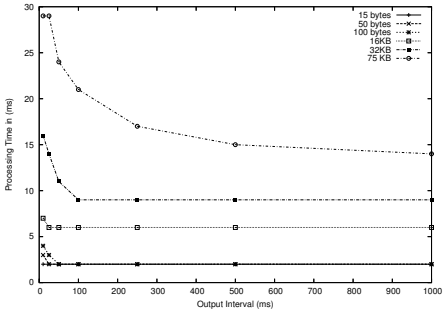

Figure 3: GSN node under time-triggered load

Figure 4 shows an experimental result for query processing latencies in a single GSN node for a stream element size (SES) of 32KB. We used random queries with 3 filtering predicates in the `where` clause on average, using random history sizes from 1 second up to 30 minutes and uniformly distributed random sampling rates in the interval $[0.01, 1]$ (seconds). Additionally, bursts were produced with a probability of 0.25.

The spikes in the figure correspond to the bursts. The query processing latency heavily depends on the database used by the GSN node (we used MySQL in the experiments) and as expected the database's performance is directly related to the number of clients as with higher numbers more queries are sent to the database and also the cost of query compiling increases. Nevertheless, the query processing time is reasonably low, as the graph shows that the total time required to process 500 queries is around 40 milliseconds, i.e., the processing time per client while handling 500 clients is less than 1 millisecond.

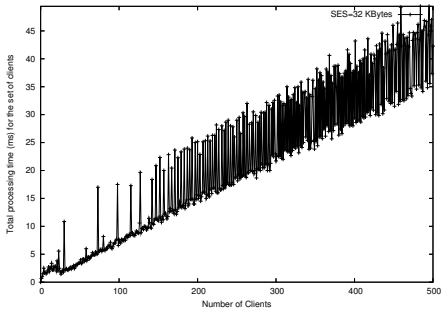

Figure 4: Query processing latency in a GSN node

Due to space constraints we can only give some key evaluation results. A detailed evaluation is provided in [1].

6. DEMONSTRATION

The key advantages of GSN are its modularity and extensibility, the low effort required for integrating new sensors, and the flexibility in supporting fast and simple deployment. In the demonstration we specifically focus on the last item, as deployment and maintenance are known to be the major cost factors by far in the software industry. In the demo we will set up four sensor networks as shown in Figure 5, i.e., one sensor network with RFID readers and tags, a wireless camera sensor network, and two wireless sensor networks using MICA2 motes equipped with light, temperature, and 2D acceleration sensors. The RFID and the motes network share one GSN node, whereas the other two networks have a dedicated GSN node (this is an arbitrary setup we chose for the demo and it can be changed easily).

In the demonstration the audience are first invited to query the pre-configured system setup via a Web interface to exemplify the functionalities described in the previous sections. The setup is done such that the audience can query the individual networks, but also complex configurations that integrate the data of several of the networks are included. This gives some initial hands-on experience of the small "Sensor Internet" we have set up with a set of predefined queries. Typical examples here would be to query for the average light intensity and temperature in the last 10 minutes (active query), or when the RFID reader recognizes an RFID tag, a picture of the person/item it is attached to would be returned from the camera network together with the current light intensity and temperature taken from the other networks (notification).

After this initial familiarization with GSN, we invite the audience to interactively change the setup of the system on-the-fly while the system is running. In detail we will demonstrate the following:

- We show how to rapidly deploy a sensor network without any programming effort just by providing a simple XML configuration file.
- The audience are invited to add, remove, and reconfigure virtual sensors while the system is running and processing queries. This will show the support for on-the-fly configuration changes and demonstrate the plug-and-play capabilities of GSN for dynamically adding and removing sensors and networks.

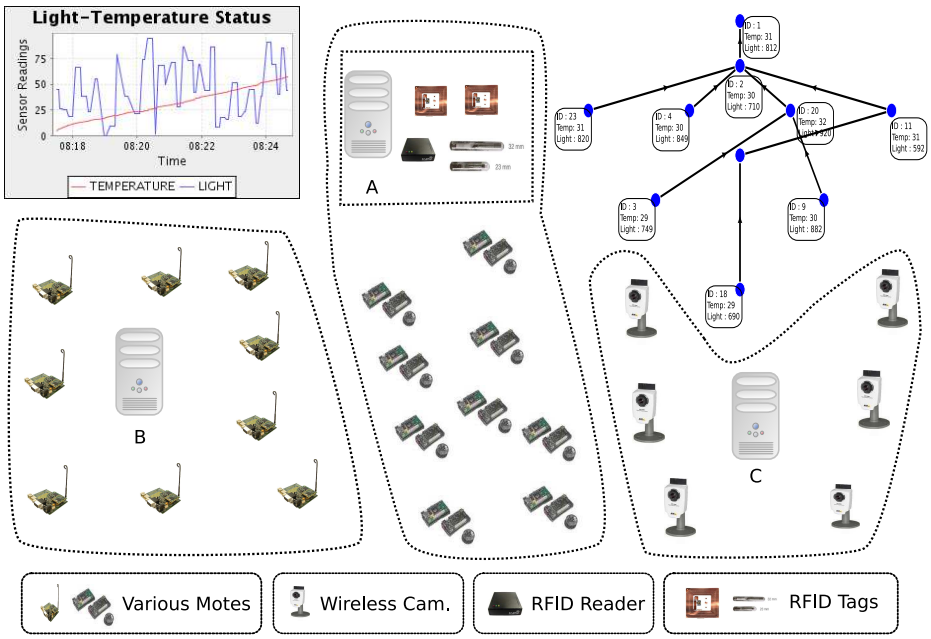

Figure 5: Physical deployment at demonstration

- The audience are invited to define new virtual sensors performing joins and/or filtering on one or more of the sensor networks deployed in the demonstration. This part of the demo shows how a new sensor network which is based on the data produced by other (heterogeneous) sensor networks can be created by just providing some declarative configurations and without any software programming efforts.
- The audience are invited to define the events in the sensor networks they are interested in and another audience can trigger the event by performing an action such as passing a RFID tag in front of the RFID reader or hiding the light sensor on the motes (or a combination of such actions). This part of the demo exemplifies how simple events can be specified and how various notifications channels can be used.

During the whole demonstration, the audience are able to monitor the effective status of all parts of the system and how it reacts to changes in the configuration through a web interface and various plots and network connection figures.

7. RELATED WORK

Probably the closest approach to GSN is [2] which suggests 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 middleware. Hourglass [3] provides an infrastructure for connecting sensor networks to applications and offers topic-based discovery and data-processing services. Like GSN it tries to hide internals of sensors from the user but focuses on maintaining quality of service of data streams. HiFi [4] provides 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 [5] 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 modeled after the Internet DNS and supporting XPath queries. In contrast to that, GSN follows a symmetric peer-to-peer approach as already mentioned and supports publish/subscribe besides active queries.

8. CONCLUSIONS

GSN provides a flexible middleware for fast deployment of sensor networks meeting the challenges that arise in real-world environments. The key concept in GSN are virtual sensors which abstract from implementation details of access to sensor data and correspond either to data streams received directly from sensors or to data streams derived from other virtual sensors. Thus 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. Due to space constraints we could only provide an overview of GSN. A detailed description and analysis of all aspects of GSN is given in [1]. The GSN implementation is available from http://globalsn.sourceforge.net/.

9. REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi, "The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks," Tech. Rep. LSIR-REPORT-2006-006, Ecole Polytechnique Fédérale de Lausanne, 2006.

[2] 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 *Ambient Intelligence*, Springer Verlag, 2005.

[3] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, "Hourglass: An Infrastructure for Connecting Sensor Networks and Applications," Tech. Rep. TR-21-04, Harvard University, EECS, 2004. http://www.eecs.harvard.edu/~syrah/hourglass/papers/tr2104.pdf.

[4] 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.

[5] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "IrisNet: An Architecture for a World-Wide Sensor Web," *IEEE Pervasive Computing*, vol. 2, no. 4, 2003.