Demo: A Generic Platform for Sensor Network Applications[*]

Rene Mueller, Jan S. Rellermeyer, Michael Duller, and Gustavo Alonso
Department of Computer Science
ETH Zurich, 8092 Zurich, Switzerland
{rene.mueller,rellermeyer,michael.duller,alonso}@inf.ethz.ch

Abstract

Writing applications for sensor networks often involves low-level programming. In this demo we show a generic sensor network platform (SwissQM/SwissGate) that provides a high level interface for programming sensor networks and also provides a multi-tier architecture for efficiently handling and optimising the operation of the network. The demo is based on a small scale (deployment in a building) where the network is used concurrently by several applications to measure heating, ventilation, and air conditioning control (HVAC) parameters. The network also implements several event detection functions for fire, burglar, and user triggered alarms. In the demo we show how the sensor network can be programmed using queries in several languages (SQL, Java, XQuery), including user-defined functions (in a C-like language) and the results obtained as a stream of data tuples. We also show the ability to efficiently use the network concurrently.

1. Introduction

The main limitations of today's sensor networks are the difficulty in programming them and the typically single use, ad-hoc deployment nature of existing systems. The state of the art today is to set up, with considerable effort, a sensor network for a very concrete and specific purpose (e.g., [1, 3]). Such networks are generally short-lived and not designed for re-use or for extensive reprogramming of their functionality. We envision instead sensor networks as a resource that is available already (in a building, in a city, in a site) and can be used concurrently by different applications for a wide variety of changing purposes.

In such scenarios, however, the current limitations of sensor networks need to be resolved. First, it must be possible for multiple applications to concurrently access the WSN. Second, to facilitate application development, programming must be based on higher levels of abstraction than current languages (e.g., nesC) or operating systems (e.g., TinyOS).

In this demo we present the SwissQM/SwissGate system, a complete platform for programming, deploying, and operating wireless sensor networks. The platform can be programmed using a variety of languages. In the demo we will emphasise programming through a declarative interface using languages like SQL or XQuery. Unlike existing declarative sensor systems like TinyDB [2] or Cougar [10], SwissQM/SwissGate supports user-defined functions and provides a more sophisticated architecture to facilitate optimisation and extensibility. Among other features, we will show the ability of the platform to concurrently run multiple, dynamic queries in an efficient manner.

2. The SwissQM/SwissGate platform

The backbone of the SwissQM/SwissGate system is a novel architecture (Fig. 1) for efficient data acquisition and data processing over sensor networks [4]. In its current form, users interact with the system by submitting queries. These queries are transformed, optimised, merged, and compiled in the SwissGate sub-system. Then they are disseminated to the sensor nodes in the form of short bytecode programs. The nodes run *SwissQM* [5, 8], a virtual machine for sensor networks. Such an approach gives full flexibility and control in making the network more efficient but does not impose any constraints on the language used to program the sensor network. The design of the instruction set in SwissQM also leads to more compact programs, which reduces the error rates during program dissemination. Additionally, the bytecode of SwissQM provides higher expressiveness compared to systems such as TinyDB where a query engine must run on the sensor nodes and only accepts SQL. Queries running on top of SwissQM can contain user-defined functions. As an example, consider the following

[*]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.

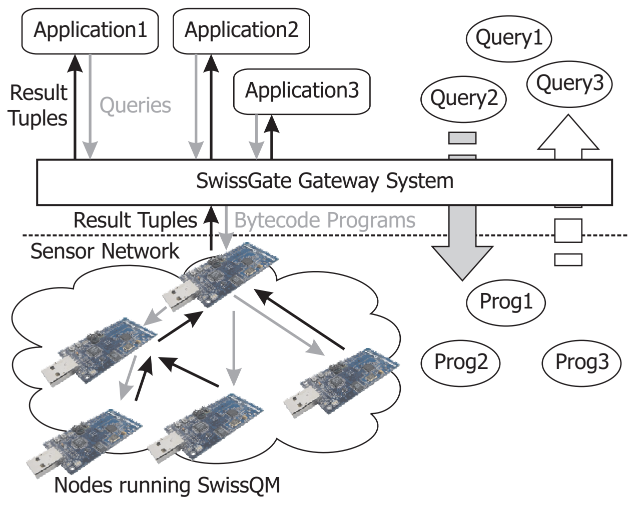

Figure 1. Architecture of SwissQM System

query submitted by a fire detector application:

```
SELECT nodeid,temp FROM sensors
WHERE raise(temp)>10 SAMPLE PERIOD 60s
```

The query requests the sensor to sample the temperature every minute and report both temperature and the node ID. The query contains a user-defined function (UDF) written in a C-like language (see below). This UDF accumulates the values of two temperature measurements and reports the difference. As embedded in the query, the result is that the predicate evaluates to true if the temperature increased by more than 10 degrees between the last two samples. Thus, the query will make the sensor report the temperature only when it sees an increase of 10 or more degrees between the current and the last measurement. The implementation of this UDF is as follows:

```
int raise(int x) {
  static int oldx = 0;
  int delta = x-oldx;  oldx = x;
  return delta;
}
```

When generating the bytecode program out of the query, SwissGate inlines the user-defined function within the bytecode of the queries where it is used. SwissGate is implemented as a collection of modular OSGi software components. OSGi also permits components to be exchanged at runtime, making the system dynamically extensible. SwissGate is deployed in Concierge [6], our own small-footprint implementation of OSGi.

Besides optimisation (compacting, rewriting) and compilation of the queries, *SwissGate* also processes the result tuple stream generated by the WSN. Operations that cannot be pushed into the network, e.g., large aggregation windows of aggregation queries or complex user-defined functions, are applied at the gateway before the tuple stream is returned to the user/applications. SwissGate also supports a query life-cycle (queries can be started, suspended, resumed, and stopped) and dynamic updates to the queries. Currently, queries can be expressed in two different query languages: SQL and XQuery. The query parsers are implemented as OSGi services. By using R-OSGi [7], our extension to OSGi that provides remoting of services, queries can be submitted by remotely invoking the parser services. Queries can also be submitted through a web front end and a web service. In order to receive the generated result stream applications can subscribe to various *tuple sinks*. Besides the OSGi event mechanism tuples are also delivered to the end applications as RSS streams or a web front end. We also provide a default tuple sink that stores the tuples in a relational database.

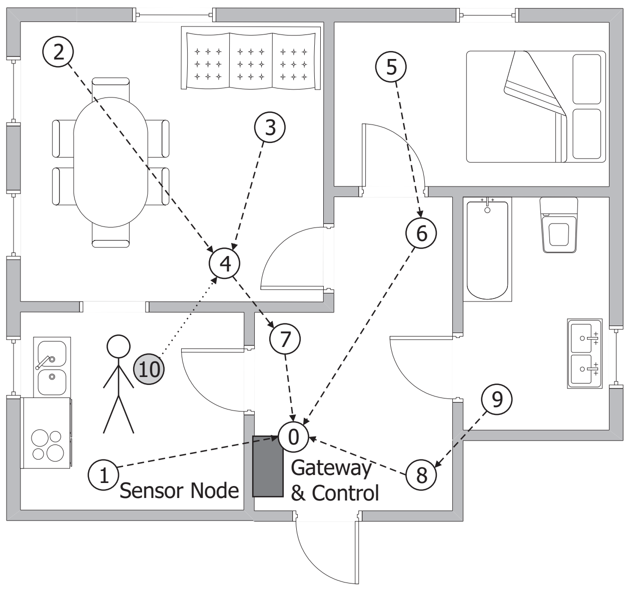

Figure 2. Sensor network deployment in a building

3. The Demo Scenario

The demonstration revolves around a WSN that is deployed in an apartment complex. Each room is equipped with one or more wireless sensor nodes (Fig. 2). We assume that each of these nodes is equipped with a light and temperature sensor and has the SwissQM virtual machine installed. In the demo, we use four different end-applications that make use of the WSN by posing continuous streaming queries to the SwissQM gateway. These queries are:

(1) Heating, ventilation, and air condition control (HVAC): The air-condition application takes the average temperature on the floor as input and controls a fan through an actuator. The following SQL query is issued by the application:

```
SELECT AVG(temp) FROM sensors
SAMPLE PERIOD 5min
```

(2) Fire detector: In the demo, a fire is detected either when the temperature is sufficiently high or there is a sudden increase in temperature. The fire detector application requires the WSN to send a tuple $(nodeid, temp)$ when a node $nodeid$ satisfies any of the two above conditions. The query used is as follows:

```
SELECT nodeid,temp FROM sensors
WHERE temp>60 OR raise(temp)>10
SAMPLE PERIOD 60s
```

`raise()` is the user-defined function described earlier. Note that in this query a sensor node only generates a tuple when the fire condition is satisfied. The fire detector application can then perform a lookup in the node database in order to determine the room in which the node with $nodeid$ is located and triggers a fire alarm.

(3) Burglar alarm: The light sensors are used for a simple burglar alarm. When activated the light sensors are sampled and trigger an alarm when the difference of the light value between two samples exceeds a given threshold implemented as a first order low-pass filter. An alarm is triggered when that happens. The corresponding query is:

```
SELECT nodeid,light FROM sensors
WHERE raise(temp)>100 SAMPLE PERIOD 10s
```

(4) Emergency signaling: In warden-assisted accommodation (e.g., for the elderly), emergency systems are already in use. A person wearing a radio transmitter can summon an emergency assistance by pressing a button in their wrist watch device [9]. We implement a similar scenario using a WSN as follows. We assume the person wears a mobile sensor that runs SwissQM and connects to other nodes in the building. By using a button the person can trigger the emission of a tuple that contains the $nodeid$ (such that the person that requests assistance can be identified by the end-application). The end-application issues the following query:

```
SELECT nodeid,parent FROM sensors
WHERE oncount(panic)>=5 SAMPLE PERIOD 1s
```

The alarm button of the mobile node is modeled as a boolean sensor `panic`. `oncount(panic)` is a user-defined function that counts the consecutive samples the button is pressed. This assures that an alarm is only triggered if the button is pressed for at least 5s.

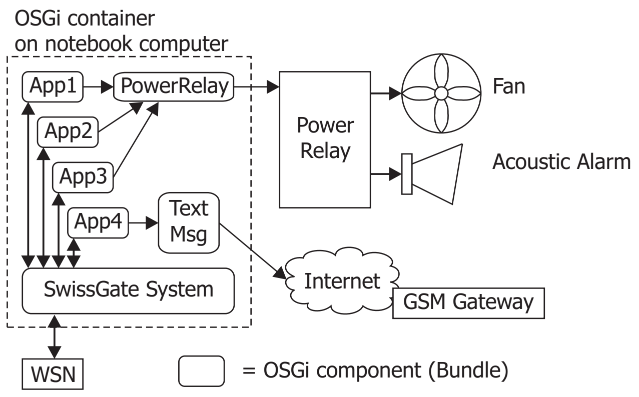

Figure 3. Device setup for demonstration

The demo uses 10 Tmote Sky sensors that run SwissQM. An additional node is used as mobile node for the emergency application described above. A notebook computer runs the SwissGate system and the four end-applications (see Fig. 3) that are implemented as OSGi bundles and thus can be dynamically deployed. For the HVAC application, the application controls a fan; the fire detector and the burglar alarm a signal horn. The emergency signaling application sends alarm calls as emails and text messages.

References

[1] K. G. Langendoen, A. Baggio, and O. W. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *WPDRTS'06*, 2006.

[2] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[3] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA 2002*, pages 88–97, 2002.

[4] R. Mueller, G. Alonso, and D. Kossmann. SwissQM: Next generation data processing in sensor networks. In *CIDR'07*, 2007.

[5] R. Mueller, G. Alonso, and D. Kossmann. A virtual machine for sensor networks. In *EuroSys'07*, pages 145–158, 2007.

[6] J. S. Rellermeyer and G. Alonso. Concierge: a service platform for resource-constrained devices. In *EuroSys'07*, pages 245–258, 2007.

[7] J. S. Rellermeyer and G. Alonso. Services everywhere: OSGi in distributed environments. In *EclipseCon'07*, 2007.

[8] SwissQM project. http://www.swissqm.inf.ethz.ch.

[9] TeleAlarm. Carephones for warden-assisted accomodation. `http://www.telealarm.com`.

[10] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR'03*, 2003.