A Dynamic and Flexible Sensor Network Platform*

R. Mueller, J. S. Rellermeyer, M. Duller, G. Alonso, D. Kossmann
Department of Computer Science
ETH Zurich
8092 Zurich, Switzerland
{muellren, rellermeyer, michael.duller, alonso, kossmann}@inf.ethz.ch

ABSTRACT

SwissQM is a novel sensor network platform for acquiring data from the real world. Instead of statically hand-crafted programs, SwissQM is a virtual machine capable of executing bytecode programs on the sensor nodes. By using a central and intelligent gateway, it is possible to either push aggregation and other operations into the network, or to execute them on the gateway. Since the gateway is built in an entirely modular style, it can be dynamically extended with new functionality such as user interfaces, user defined functions, or additional query optimizations. The goal of this demonstration is to show the flexibility and the unique features of SwissQM.

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software; H.m [**Information Systems**]: Miscellaneous; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Measurement

Keywords

Sensor Networks, Query Machine, SwissQM, OSGi, R-OSGi

1. INTRODUCTION

Sensor networks are a growing field of research with many emerging commercial applications. Building and deploying sensor networks, however, is still a cumbersome task. Many existing deployments are application specific and the software running on the sensor nodes has been mostly hand-crafted for the particular application.

2. SWISSQM

In [5, 6] we presented SwissQM, a next generation architecture for data acquisition and processing in sensor networks which greatly facilitates building and deploying of such networks. SwissQM is a combination of software running on the sensor nodes and software running on the gateway machine, which provides access to the sensor network. The sensor node part of the software solution is a virtual machine that executes bytecode programs. The software on the gateway provides different interfaces to users of the sensor network and converts their queries into bytecode programs which are then executed on the sensor nodes. The combination of a flexible execution platform on the sensor nodes and a gateway, which is able to translate queries to bytecode, provides a level of abstraction that hides technical details from the users while allowing them to collect data in an easy, declarative way.

2.1 Sensor Nodes

The sensor nodes are organized in a tree where only the root node is connected to a gateway, which connects to the outside world (Figure 1). Each node forwards its data to its parent node and the root node forwards all data from the tree to the gateway. On every node, a stack-based virtual machine that is implemented on top of TinyOS [3] runs QM programs. These programs consist of up to three sections and have certain properties, e.g., the sampling period for the program. The three sections are *init*, *delivery*, and *reception*, each one of them consisting of byte code instructions similar to those of the Java virtual machine. The names of the sections indicate when they are executed. The init section is optional and is executed once when the program starts to initialise the program state. The delivery section is executed once every sampling period and is normally used for sampling the sensors on the node and processing the data. The reception section is executed upon arrival of a message from a child to extract data from the message and merge it with local data.

In addition to those bytecode instructions that can be found on most stack-based virtual machines, the virtual machine of SwissQM features specialized instructions for common data sampling and aggregation tasks (Listing 1). Instructions are provided for sampling sensors and for reading attributes like the node id, its parent's id, or its depth in the tree. A special *merge* instruction is available for efficiently computing aggregates in the network. When running a QM program that performs in-network aggregation, a part of each node's main memory is used as a *synopsis*. In the synopsis, data from the child nodes (added in the reception section) and data from the local sensors (added in the delivery section) are merged. The merge instruction allows for the computation of more than one aggregation and different types of aggregation operations with one single instruction

*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.

SIGMOD'07, June 11–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

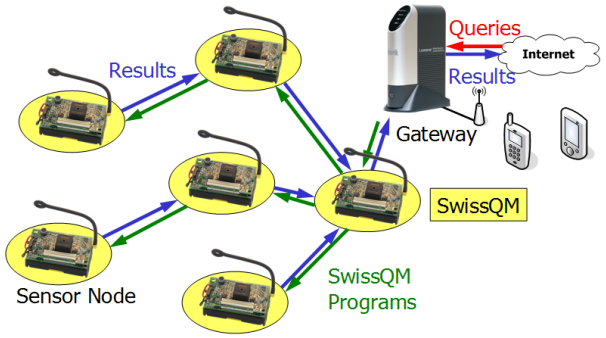

Figure 1: SwissQM Sensor Network

call. Furthermore, it is designed for being easily extensible by additional aggregation operations.

The ability to encode common complex operations like aggregation into one or a few short bytecode instructions results not only in faster execution but also in short programs. This is crucial for efficient program dissemination and dealing with the constrained resources of typical sensor networks.

Listing 1: Bytecode program for `SELECT depth, AVG(light) FROM sensors GROUP BY depth SAMPLE PERIOD 1024`

```
.section delivery "@1024ms", "synopsis",
                  "epochclear"
  get_depth
  istore  0 # store grouping expression 'depth'
  get_light
  istore  1
  iconst_1
  istore  2 # initial aggr. state <light,1>
  ipushb  5 # code of AVG aggregation
  iconst_1  # number of aggregation expressions
  iconst_1  # number of grouping expressions
  merge
  send_sy

.section reception
  ipushb  5 # code of AVG aggregation
  iconst_1  # number of aggregation expressions
  iconst_1  # number of grouping expressions
  merge
```

2.2 Gateway

In order to use a SwissQM sensor network, users do not have to deal with the details of the virtual machine described in the preceding section. The gateway part of SwissQM is written in Java and runs on Concierge OSGi [8]. Concierge is an implementation of the OSGi [7] specifications tailored for small devices. Its main purpose is to support the lifecycle of Java components, the so-colled *bundles*. The gateway is entirely designed as a modular unit to allow the user to dynamically add new components such as language parsers or query optimizers at runtime. Whenever a user query or a user defined function is submitted by a client, the content is parsed and transformed into a language-independent intermediate representation (Figure 2). Out of these *user queries*, OSGi bundles are generated to give the user full control over the life cycle of the query or the function. It is, for instance, possible to temporarily disable a query and restart it after some time or even replace an existing query

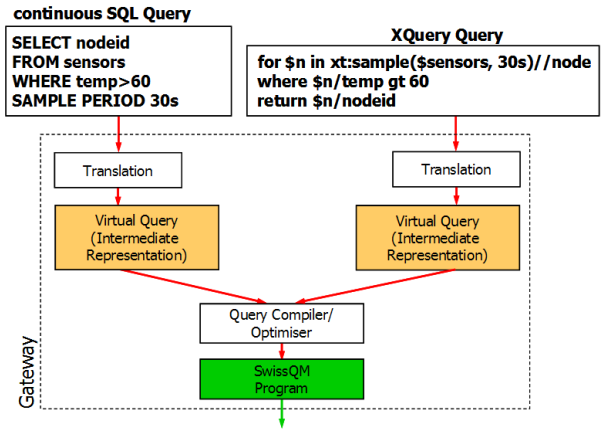

Figure 2: SwissQM Language Handlers

with a new one. As an example, a query can be updated with a more selective one or by one making use of more sophisticated filtering of the sensor data. This extensive query management functionality can be implemented due to strict decoupling of network operations from queries. The gateway maps user queries into *virtual queries*. This process performs both merging and rewriting of queries as described in [4] to reduce the number of queries. Virtual queries are finally mapped into *network queries*, which are then disseminated into network in form of bytecode programs. Virtual queries can be reassigned to different network queries, thus, allowing for online changes in the parameter set of the query. When optimizing and translating queries into QM programs, the gateway takes into considerations the nature of the query and the features and properties of the sensor nodes. For example, by pushing aggregation into the network, the number of messages that have to be transmitted can be reduced. On the other hand, the gateway might decide not to push aggregation into the network but perform it on its own because there is a second query running in parallel which requests the same sensor data but without aggregation. Thus, by not pushing aggregation into the network, transmitting the same data twice (once in raw and once in aggregated form) can be avoided. These are only two examples of choices of the gateway to best leverage the power of the virtual machine running on the sensor nodes.

2.3 Client Interfaces

The gateway part of the SwissQM architecture provides intuitive interfaces to the users through which they can submit their queries. Different categories of end users require different user interfaces. In SwissQM, this is addressed by the modular design. Currently, a range of client interfaces, such as a web access, a webservice client and an integration into the Eclipse IDE is already supported. New interfaces can be added on demand without the need to restart the system and lose running queries.

A web interface is the easiest way of interacting with the system. The user can choose one of the query languages that have support from a parser service. Currently, we have support for SQL. Support for XQuery is under development. Each of the languages can be used to express queries and whenever a new parser is added, the extension of the system is immediately reflected on all client interfaces, in particular

the web interface. When a query is submitted, the result tuples are displayed on a result web page that uses the HTML refresh mechanism to continuously update the set of already received result tuples.

A SOAP interface is provided for custom-tailored client applications. Queries and user defined functions can be submitted via SOAP RPC calls. The results can either be retrieved immediately by registering a Web Service Notification callback or on demand by accessing an RSS feed (Figure 3) that is automatically generated for every query. The latter possibility is especially useful if the client application is not interested in monitoring every newly arriving tuple but instead wants to get all tuples at a later point in time.

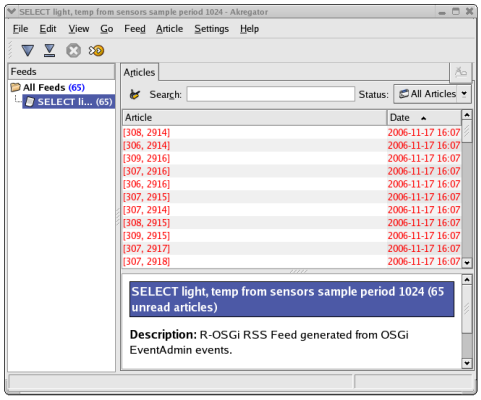

Figure 3: RSS Tuple Feed

The most sophisticated way of accessing the SwissQM gateway is the Eclipse IDE integration (Figure 4). It gives educated users and developers a full tool support for maintaining and controlling both queries and user defined functions in dedicated SwissQM Eclipse projects. Whenever needed, the Eclipse Plugin can connect to a running instance of the SwissQM gateway and submit new information. Since Eclipse is also built on top of an OSGi framework (Equinox) [1], the key technology for accessing remote OSGi services used for this communication is the R-OSGi system [9]. This system allows every OSGi framework to transparently use remote services of other remote frameworks, such as the SwissQM gateway. Results can be displayed in tables or in graphical statistics that use the Test and Performance Tool Platform (TPTP) [2] capabilities of Eclipse.

3. SYSTEM SETUP

3.1 Sensor Network

Our sensor deployment deployment for testing purposes consists of 38 Tmote sky sensor nodes by Moteiv. For the demonstration, we will use a subset of them. Each node is equipped with a TI MSP430 micro controller running at 8MHz and providing 10kB RAM and 48kByte Flash memory. The nodes feature sensors for humidity, temperature, and light. The communication among the nodes uses IEEE 802.15.4 radio operating in the 2.4GHz band. All motes are running the SwissQM bytecode interpreter on top of TinyOS.

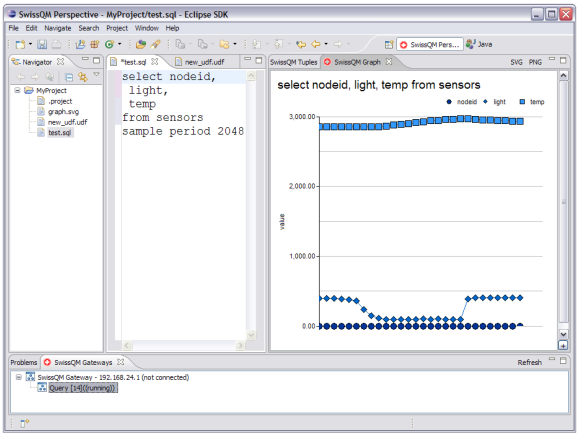

Figure 4: Eclipse Integration

3.2 Gateway Device

The gateway is a Linksys NSLU-2 (Slug) embedded Linux device based on a 266 MHz IXP420 ARM processor. We use JamVM as Java virtual machine, Concierge OSGi as framework, and R-OSGi for remotely accessing the OSGi services. The web interface runs as a servlet on the OSGi R3 HttpService. The web service SOAP interface and the RSS feeds are automatically generated by R-OSGi to SOAP and EventAdmin to RSS bridges.

4. DEMONSTRATION

In the demonstration we will show several experiments with sensor networks. Starting from an arbitrary query, it is possible to observe the full workflow of the system. Persons attending the demo will be able to see and analyze the generated bytecode, let the sensor boards process the program, and observe the post processing of tuples at the gateway. Each step of the system architecture is fully visible and can be tested and evaluated. During the demonstration, we will plug in different extensions of the system, e.g., a user defined function or additional interfaces.

5. REFERENCES

[1] Eclipse Equinox. `http://www.eclipse.org/equinox/`.

[2] Eclipse Test and Performance Tool Platform. `http://www.eclipse.org/tptp/`.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS 2000*, November 2000.

[4] R. Müller and G. Alonso. Efficient Sharing of Sensor Networks. In *MASS*, 2006.

[5] R. Müller and G. Alonso. A Virtual Machine for Sensor Networks. In *EuroSys*, 2007.

[6] R. Müller, G. Alonso, and D. Kossmann. SwissQM: Next generation data processing in sensor networks. In *CIDR*, 2007.

[7] Open Service Gateway Initiative. `http://www.osgi.org`.

[8] J. S. Rellermeyer and G. Alonso. Concierge: A Service Platform for Resource-Constrained Devices. In *EuroSys*, 2007.

[9] J. S. Rellermeyer and G. Alonso. Services everywhere: OSGi in Distributed Environments. In *EclipseCon*, 2007.