Towards a Common Sensor Network API: Practical Experiences

Filip Suba[1], Christian Prehofer, Jilles van Gurp, Nokia Research Center, Helsinki, Finland
{Christian.Prehofer, Jilles.vanGurp}@nokia.com

Abstract. Wireless sensor networks are becoming more and more popular and there are many vendors of such networks offering solutions with similar functionality, but utilizing different approaches to controlling the network and gathering the data. We evaluated two very different types of networks, Arch Rock and Z-Wave, in three main areas: network configuration, access control, and fault tolerance. Even though some functionality is shared by both types of networks, there are conceptual differences in realizing it that need to be taken into consideration. We propose an API for sensor networks, which considers the above practical issues of existing sensor networks.

1 Introduction

Wireless sensors and sensor networks are an emerging technology, and currently there are many competing approaches with different application targets and features. In this paper, we focus on APIs for developing applications that use sensor networks via some gateway. We envision such a common API not just for working with one sensor networks, but in particular for heterogeneous sensor networks.

Although there are standards like IEEE 802.15.4 and ZigBee [1], which define physical layer, protocol stack, network behavior, vendors use different technologies and the interoperability of products from different vendors is currently very low. Hence there is a need for a common, unifying framework which would enable application developers to create universal, scalable solutions. Sensor networks are a widely studied research topic and there exist several papers on accessing sensor networks and APIs for this [2][3][4][5][6][7][8]. However, these do not consider many operational issues and the incompatibilities between current, commercial sensor networks, and rather address more higher level abstractions for accessing sensor information.

We propose a basic framework that provides an interface to different sensor networks operating on different standards and discuss the issues arising from current, incompatible sensor networks. Our goal was to design an interface that would abstract the various and numerous sensor networks and offer application developers an opportunity to use a general, common solution to gather data and manage the networks. At the same time, we have to support several vendor specific concepts in the API due to the differences mentioned above. We discuss several such issues regarding network configuration, access control and fault tolerance. To validate our concepts, we employed two sensor kits, from Arch Rock and Z-wave, with widely different application programming interfaces to evaluate of our framework.

2 Sensor Network Selection and Gateway Concepts

The goal of this evaluation was to integrate wireless sensor networks from various vendors into a common framework such that these can achieve a common objective. We evaluated six wireless sensor network kits from the following vendors: CrossBow, SensiNet, Arch Rock, SensiCast, DustNetworks, and Zensys. We evaluated these kits based on the available documentation, and selected two of them for further evaluation. The kits we selected are the Arch Rock Primer Pack IP v.1.0.1 [10], and Z-Wave Developer's Kit ver. 5.00 Beta1 from Zensys [9].

[1] Current email address: suba@kth.se

A motivation for this choice was to select two commonly used, but very different kits to understand the different approaches.

2.1 Overview of the Arch Rock and Z-wave Kits

The Arch Rock Primer Pack IP kit is a wireless sensor network that consists of seven sensor nodes and one gateway computer. The sensor nodes are equipped with four internal sensors (temperature, humidity, visible light, and total solar light) and several I/O ports, where either additional sensors or devices controlled by the sensor node can be attached. One of these sensor nodes has to run special software and has to be permanently connected to the gateway device to serve as a bridge between the wireless sensor network and the gateway. The gateway computer provides a web interface to control the network parameters and sensor node configuration. Besides the web interface, the gateway offers two types of web service interfaces as well, a SOAP API and a REST API.

The Z-wave Developer's Kit is intended for development of hardware devices and the embedded software running on these devices rather than high-level application software development. Among other things it contains six wireless nodes, each equipped with one button and several LEDs. Zensys provides several sample embedded applications that can be programmed into the nodes, such as LED dimmer, and a binary sensor (implemented using the button on the nodes). Besides embedded applications, sample PC based applications are provided as well, along with the Z-Wave DLL library that can be used to write software that is capable of controlling the Z-Wave network.

These two kits are very different and represent two diametrical approaches to wireless sensor and actuator networks. While Arch Rock provides very nice, high-level APIs, the system as such is closed and only accessible via the gateway. On the other hand, Z-wave is very open and allows custom software on the sensors, but does not provide gateway APIs. Therefore finding commonalities and abstraction to single common API that could be used to control both types of networks poses a challenge.

2.2 Z-Wave Gateway

Since the Z-wave Developer's Kit does not include a gateway device or PC software that could allow collecting and storing data and information about the network, we implemented a gateway application using the provided C# DLL in the kit .

Our Z-Wave gateway application is capable of collecting and storing data from sensors, and controlling the Z-Wave network. It requires a Z-Wave node running the standard Z-Wave Static Controller embedded application to be attached to the computer using a serial cable. The gateway application provides a REST API [11] that is similar to the API provided by the Arch Rock gateway computer along with support for some unique features of Z-Wave networks, such as creating associations among devices. We implemented this software in C# programming language. The approximate amount of code that we created for this purpose is 2900 lines.

2.3 Cross Sensor Kit Actuation

One of the main goals of this evaluation was to integrate wireless sensor networks from various vendors so that they can interact to achieve a common goal. We decided to use the concept of associations that can be found in Z-Wave networks for this purpose. In a Z-Wave network, creating an association means linking two Z-Wave devices (e.g. a binary sensor and a light switch) such that the primary device controls the associated device. We built a mechanism into our Z-Wave gateway application that extends this concept and allows the user to create cross sensor network associations, where Z-Wave actuators (e.g. a light switch) can be controlled by Arch Rock sensors (e.g. an Arch Rock temperature sensor can be associated with a Z-Wave device).

2.4 Displaying Sensor Status on Web Page

As part of the evaluation, we created a web-based application that displays the status of all sensors on both networks. Additionally, the application is capable of controlling several features of these networks such as creating associations (between two Z-Wave devices or between an Arch Rock sensor and a Z-Wave device), enabling/disabling sensors, setting thresholds on Arch Rock nodes, (re)naming the nodes, and including/excluding Z-Wave devices in/from the network. To perform these tasks, the application utilizes a service API that will be described in the next section.

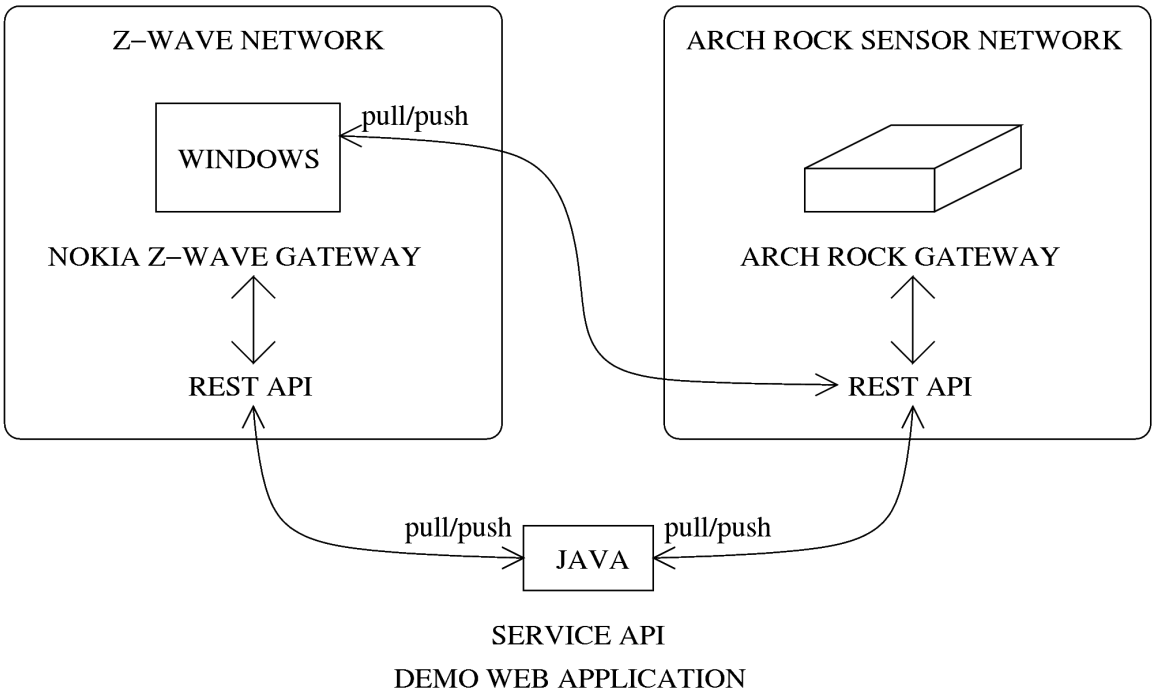

Figure 1: Architecture of the Integrated Sensor Network

3 A Common Sensor Network Gateway API

The primary goal of the evaluation was to create a common framework and API that enables us to control both types of sensor networks, as illustrated in Figure 1. The framework is implemented on top of CDC Java and the OSGi framework. This allows us to run the framework on both PCs and embedded devices such as the Nokia N800 (we have tested the software on this device). It utilizes the REST APIs provided by the Arch Rock Gateway server and our own Gateway application for the Z-Wave kit. The approximate amount of code that we produced to achieve this goal is 2800 lines.

First, we created an API that enabled us to gather data from, and control an Arch Rock sensor network. A subset of this API was already working with Z-Wave networks as well since our gateway application was designed to be similar to the Arch Rock API. However, there are some unique features and functionality that are not shared between the kits or that work differently. Consequently, some of the Arch Rock specific features are not supported in our Z-Wave gateway. Our intention was not to create an extra abstraction layer that supported all features of both kits. For example, including new concepts such as naming and management would have resulted in a layer of indirection which hampers performance, transparency and complicates vendor specific extensions. While such a more abstract vendor independent API is desirable, we considered this as not practical for current heterogeneous sensor networks.

Our proposed API is based on REST [11] and uses standard HTTP requests to provide for a simple and efficient way to access sensor information. For lack of space, we discuss here the general functionality, as shown below. A more detailed example is shown later.

Network management:

- Register nodes with the gateway (i.e. add/remove nodes to/from a particular network)
- Identify the nodes (i.e. users should know which node they are working with, e.g. sound beep/led light signaling)
- Get/set gateway's reactions on detected event (e.g. select from a list a predefined functions, or run user-specified application)
- Get/set associations between sensors and devices that can be controlled (e.g. Z-Wave devices)
- Get routing paths

Node configuration:

- Get status information about the nodes (e.g. battery level, radio signal strength, etc)
- Get/set sampling periods of the sensors
- Naming the nodes
- Upload/delete simple scripts to/from the sensor nodes

Data gathering:

- Get measurement data from the gateway (latest available, or everything within user-specified time interval)
- Export and delete measurement data from the gateway
- Get immediate values from the sensors

Access control and user management

- Get/set access control for different measurement archives
- Manage user accounts

Neither Arch Rock nor Zensys provide support for all of these features in their APIs. While Arch Rock tries to offer a simple general purpose solution, Zensys focuses mainly on controlling other network devices by the sensor nodes and real time reacting on events. The subset of our API that is actually provided by Arch Rock is:

Network management:

- Identify the nodes by blinking a blue LED on the node
- Get routing paths

Node configuration:

- Get status information about the nodes (e.g. battery level, radio signal strength, etc)
- Get/set sampling periods of the sensors
- Naming the nodes (this also signals the gateway to start collecting data from the node)

Data gathering:

- Get measurement data from the gateway (latest available, or everything within user-specified time interval)
- Get immediate values from the sensors

We implemented our Z-Wave gateway, based on the Z-Wave libraries, to provide features comparable to Arch Rock's API. We implemented some unique features of Z-Wave networks as well, such as creating/removing associations. Our Z-Wave gateway provides the following functionality below.

Network management:

- Register nodes with the gateway (i.e. add/remove nodes to/from a particular network)
- Get/set associations between sensors and other Z-Wave devices

Node configuration:

- Get status information about the nodes (also including supported command classes)
- Get/set sampling periods of the sensors
- Naming the nodes

Data gathering:

- Get latest measurement data from the gateway
- Get immediate values from the sensors

From the above it can be seen that each kit has unique features that are not supported by the other one. Some features in our proposed API are not supported in either kit (e.g. security features). Our solution provides a union of these APIs, so that application developers can use both, common and unique features of both types of networks.

During the design, we faced several problems that come from conceptual differences between the two kits. Although there are commonalities in the features provided by the two types of networks, these features are often realized very differently and we had to invent various mechanisms to overcome these differences in order to create an API that would work with both types of networks. An example of such problem is node addressing, where we had to modify the node identifiers of Z-Wave devices to match the format of Arch Rock node identifiers (see Section 4.1).

A small example to illustrate some sample code is the following issue: There can be several sensors on each Arch Rock node and only one sensor on every Z-Wave sensor device. We solved this problem by having one method that works with both networks, such as the following:

```
public String getMultilevelSensorValue (String nodeID,  String sensor,
            String gateway, String username, String password) {
    return parseXML (doRest ("http://"
            + gateway + ':' + PORT + "/gw/rest/V1/?method=events.readLast&name="
            + sensor + "ReadEvent&addr=" + nodeID, username, password), "Value")[0];
}
```

The application developer needs to specify the name of the sensor on the node. This is redundant for the Z-Wave network where there is only one sensor on each sensor device. So in that case, the developer can (but does not need to) always use the Z-Wave specific version of the same method:

```
public String getMultilevelSensorValue (
            String zWaveNodeID, String gateway,   String username, String password) {
    String result = "";
    int sensor = getSensorType (zWaveNodeID, gateway, username, password);

    String tmpstr = "";
    switch (sensor) {
            case Sensor.TEMPERATURE: tmpstr = "TemperatureReadEvent";
            break;
            case Sensor.LIGHTPAR: tmpstr = "LightPARReadEvent";
            break;
            default: tmpstr = "GeneralPurposeReadEvent";
    }
    result = parseXML (
            doRest ("http://" + gateway + ':' + PORT
            + "/gw/rest/V1/?method=events.readLast&name=" + tmpstr + "&addr="
            + zWaveNodeID, username, password), "Value")[0];
    return result;
}
```

The name of the sensor to be used in the REST request is determined automatically. The key issues addressed in the framework are discussed in the next section in more detail.

4 Analysis of the Framework

There are many different scenarios and settings in which wireless sensor networks could be deployed and used. The very purpose of sensing the environment can vary from case to case, influencing the demands on behavior, configuration, security and fault tolerance of a wireless sensor network. Examples of scenarios can be indoor monitoring (collecting data from indoor environment over longer period of time, e.g. temperature in workshop), outdoor monitoring (collecting data over long period of time, e.g. forest temperature/humidity), intrusion detection (using sensors to determine whether physical security has been breached, e.g. motion sensors, open doors/windows), alarms (sensing critical variables of the environment and triggering appropriate actions, e.g. temperature in the engine room).

In the above usage scenarios, there are well known trade offs between energy efficiency and responsiveness that need to be reflected in APIs. For instance, in long term monitoring, the user is generally interested in collecting data in regular time intervals. These time intervals usually tend to be longer (e.g. several minutes) and the gathered data are used mainly for statistical purposes. This is a case where no real time reactivity is required and the wireless sensor nodes can be in a sleep mode for most of the time and wake up only for a short amount of time in regular intervals to send (push) sensor readings to the gateway, thus saving battery power. On the other hand, there are scenarios where the user wishes (needs) to know the immediate value of a sensor. In these cases, real time reactivity might be essential and the wireless sensor nodes should listen for requests coming from the gateway (pull). Consequently, these sensors consume a lot more battery power compared to sensors that sleep most of the time.

Both, Arch Rock and Z-Wave sensor networks support data pushing (sensor nodes push data to the gateway in either periodic time intervals, or when an event occurs on the sensor node). However, only Arch Rock sensor networks support real-time data pulling (the gateway can retrieve immediate sensor values from a sensor node). Both types of sensor networks provide web services capable of data pulling (requesting data samples from the gateway). However, neither offers any mechanism for event driven decision making on their gateways. Our API implements a unified way of fetching the latest data samples that are stored on the gateway for both types of sensor networks. It also supports requesting immediate data samples for Arch Rock networks only.

4.1 Network Configuration

There are several configuration parameters that can be set on wireless sensor networks. These can vary from configuring core network attributes, such as radio power strength, or operation frequency to simple metadata that are assigned either to the network in general, or to particular sensor nodes and that are usually stored in the gateway.

Arch Rock sensor networks provide several ways of setting network parameters. Some of them need to be configured only manually via the web interface of the gateway and by attaching a sensor node to the gateway by a USB cable; others can be configured programmatically via the web service API provided by the gateway. The configuration that can be done only via the gateway's web interface includes actions such as setting gateway's time and date, updating software of the nodes, setting size of gateway's databases, or changing frequency channel of the sensor nodes, so that they can communicate with the gateway and among each other. The network configuration actions that can be performed via the web service API include for example setting metadata to the sensor nodes (e.g. names, coordinates on a virtual map, etc.). Assuming that all sensor nodes of an Arch Rock sensor network operate on the same frequency channel, adding nodes to the network is done by assigning a name to the new node (this serves as a signal for the

gateway to start collecting information from the node). Removing nodes happens automatically (i.e. when the node is not found in the network, it will not be included in the node list returned by the gateway). However, the node's metadata is saved on the gateway. When the note reappears on the network, data collecting is automatically resumed. Sensor nodes are uniquely identified by their MAC addresses assigned by the manufacturer. In the web service API, the MAC address is used as a 64-bit number in hexadecimal format. Each node must have a name, which is stored as metadata on the gateway. If a node does not have a name assigned to it, the gateway does not store data from this node, or information about this node.

Z-Wave networks consisting of sensor devices provide several similar, but also several unique configuration options. In the case of Z-wave, the network can be configured only by the web service interface provided by our gateway application. This interface enables users to set node metadata (e.g. name) and to create or delete associations between a sensor node and another Z-Wave device (such as a light switch). Adding and removing nodes does not happen automatically, and is supported by the web service API that is provided by our Z-Wave gateway application. All devices on a Z-Wave network are identified by a unique 8-bit number, which is assigned to the devices when they are added to the network. In the web service API, this number is represented in hexadecimal format and extended to 64-bit (for compatibility with Arch Rock MAC address format). Each node can, but does not necessarily have to have a name, which is stored as metadata on the gateway. However, this feature was implemented in the gateway for compatibility purposes only. Naming of nodes is not defined in the Z-Wave protocol, and therefore, names cannot be used to address the nodes and have no effect on functionality of the network.

Our common API provides an intersection of the network configuration features of these two vendors, which is mainly about setting and getting metadata of a node (including naming the node) and setting and getting the sampling period of a node. However, even though the API for performing these tasks is shared between these two types of networks, the usage might differ, as for example, it is necessary to name a new Arch Rock node in order to give a signal to the gateway to start collecting data from, and information about the node, whereas in a Z-Wave network this is not required.

4.2 Access Control and Network Identification

Different usage scenarios of wireless sensor networks usually require different approaches to security, and thus several security tools should be inseparable part of each sensor network, such as user management, or mechanism that ensures that a particular node is registered with a particular gateway and that the node sends information only to that gateway. Furthermore, nodes are associated with one sensor network and must be clearly separated from other networks (using same technology).

One of the security features of Arch Rock sensor networks is simple HTTP authentication. However, there is no user management tool available for the administrators to create different user accounts with different privileges. There are only two user accounts available in an Arch Rock network, one can be used to authorize users to use the gateway's web interface, and the other one can be used to authorize users to use the web service API. Arch Rock uses 26 frequency channels as a mechanism to separate different sensor networks, where all nodes (including the bridging node connected to the gateway) belonging to the same network have to be configured to use the same frequency channel.

In a Z-Wave network, all devices that belong to the same network use a common network identifier called HomeID that is assigned to each node when the node is added to the network and is the same as HomeID of the primary controller. The HomeID of the primary controller is provided by the manufacturer of the primary controller.

As there is neither support for frequency changing in the Arch Rock's REST API, nor changing the HomeID of Z-Wave devices, our common API does not provide support for this

functionality. However, using our API, developers can create applications capable of adding/removing nodes to the network. Since the Arch Rock network can be configured to use simple HTTP authentication, our API supports it as well. To keep our Z-Wave gateway simple, we did not implement any HTTP authentication there. Neither Arch Rock's API, nor the API of our gateway supports managing user accounts. The only way to specify the username and password for the HTTP authentication that can be configured on an Arch Rock network is to use the web interface of the Arch Rock gateway.

4.3 Fault Tolerance

Fault tolerance is a very important feature of every system, not only WS&AN. In the world of wireless sensor networks, fault tolerance means ability to respond to node failures (due to battery drainage, node damage, or loss), cases when the gateway fails to function the way it was designed to, or increase of message latencies (due to routing via many other nodes, or node failure). In our case study, we were trying to determine the response of the two evaluated systems to these issues.

First, we evaluated whether it is possible to get to know in a timely manner if a node is missing from the sensor network (e.g. due to battery drainage). In case of Arch Rock sensor network, we placed three sensor nodes in the network. The nodes were discovered by the gateway and we assigned new names to them. Afterwards, we made the gateway request a data sample from all enabled sensors on all nodes in the network, so that the gateway databases contain at least one data sample from the three new nodes. When issuing the command to achieve this, the gateway returned node addresses of all nodes that it succeeded to get data samples from (the new three nodes were among them). Then we physically removed one of the three sensor nodes from the network, and issued the same command. We found out, that the address of the removed node was not among the node addresses returned by the gateway. Therefore, we concluded that it is possible to determine, if a node is missing in an Arch Rock sensor network, or not. As for Z-Wave network of sensor devices, we placed two new nodes to the network using the PCController application that was included with the kit. We used our own application based on the Z-Wave DLL library that was provided with the kit to issue a SET command from the ASSOCIATION command class to one of these nodes. This command succeeded (ZWaveSendData method of IApplicationLayer interface as defined in the Z-Wave DLL library returned with TXStatus.CompleteOK) and the response from the node (REPORT command from the ASSOCIATION command class) arrived within 43 milliseconds. Then we physically removed this node from the network. We launched the same application again, trying to send a SET command from the ASSOCIATION command class to the missing node. Surprisingly, the command succeeded (a result of TXStatus.CompleteOK was returned), although, the response never came. Therefore, we concluded that the current DLL library does not recognize if a node is missing in the network. For this purpose, an additional mechanism could be introduced, that would start a timer countdown (e.g. 5 seconds), and if a response (a REPORT command of the command class corresponding to the initial command) does not come within this time, we can assume that the node has failed.

Second, we wanted to know, if a node that runs out of battery power, and therefore disappears from the network, needs to be reconfigured and reincluded to the network after the batteries has been changed. Both, Arch Rock and Z-Wave networks have proved to tolerate this fault, as in both networks, the node gains its original functionality, settings and network competence after the power supply has been restored.

When deploying a wireless sensor network, next very important factor is network reconfiguration delay. If a node within the network fails, and the node happens to route traffic of other nodes, many questions arise, such as how long it takes until an alternative route is established within the network, or if the messages that were supposed to be routed by the failed node are lost, or just delayed. During our evaluation of Arch Rock sensor network, we found, that if a node is unable to transmit measured data samples to the gateway, the messages are buffered

within the node and sent to the gateway when the route is reestablished. This means that in case of Arch Rock sensor network, message loss does not occur (until buffer overflow). The network reconfiguration delay (time that is needed to establish an alternative route within the network) that we measured was approximately ten seconds. For the Z-Wave network, our gateway application is designed in such way, that if a direct connection between the gateway and the Z-Wave node does not exist, the route for the message is determined and created upon every message transmission, which means no delay in network reconfiguration. If the route cannot be determined, the message is lost and will not be retransmitted.

Using our common API, application developers can test whether there is any Arch Rock node missing in the network by listing all nodes on the network, and if a node is missing (temporarily, or permanently), it will not be included among the other nodes. However, currently there is no support for such functionality for the Z-Wave network, therefore using the common method for listing all nodes can be used only for determining presence of Arch Rock nodes. Z-Wave nodes are included in the resulting list even if they are not present on the network any more, but have not been excluded from the network explicitly.

5 Related Work

The work in [2] presents a generic API for sensor networks, focusing on flexibility, extendibility and expressiveness. For instance, retrieving sensor readings depending on location, not just sensor ID is supported. Similarly, [5] [7] present a very generic view on sensor networks and focus on expressiveness and abstraction. Another option for one sensor network is to use a data base oriented interface based on systems like TinyDB [8], which is however aiming on a single sensor network. For these approaches, practical experiences with different existing, off-the-shelf sensor networks are missing. Also, the paper presented here focuses more on issues regarding configuration and failure handling, which are often ignored in other works.

JSR 256 [3] is a API for controlling sensors in mobile devices. It is primarily intended to control sensors inside a device, and does not address network specific and failure handling as we discuss here. As a further approach, the SIP protocol has been proposed [6] as a suitable means to access sensor networks. SIP sessions and the proxy concept are in general suitable for accessing sensor networks, as they support sessions, messaging and redirection based on proxies. However, we consider web-based interfaces as a more primitive and basic interface which should be provided first. Furthermore, the paper does not address the above issues of heterogeneous sensor networks.

The work which is closest to the research here is TinyREST [4], which also used REST style interfaces to access heterogeneous networks such as Mica notes and UPnP. The paper does however not show experiences with heterogeneous sensor networks which address the above issues of management and failures.

6 Conclusions

In this work, we analyzed two diametrically different approaches to wireless sensor networks. The main goal of the analysis was to create a common framework, an abstraction layer and a common API that may be used to control the underlying, widely different sensor networks. Arch Rock's solution already provided a REST API that enabled us to perform various tasks on the Arch Rock sensor network. Zensys's solution, on the other hand, was aimed primarily at developers of Z-Wave supported devices, rather than developers of network applications. Hence we built a gateway to the Z-Wave network and a REST API with unique features of the Z-Wave network.

In our analysis of the two wireless sensor network kits and their APIs, we focused on three main areas: network configuration, security, and fault tolerance. In the field of network

configuration, we found that even though some functionality is included in both types of networks, there are conceptual differences in realizing it (e.g. adding/removing nodes to/from the network). And even if a task is performed in similar manner, there might be different semantics in different types of networks (e.g. giving a name to an Arch Rock node gives a signal to the gateway to start collecting data from that node). Regarding access control and setup, we discovered again that the two networks are quite different and have distinct approaches to problems such as separating sensor networks from the same vendor (Arch Rock uses 26 frequency channels, Z-Wave uses HomeID). Regarding fault tolerance, we found several differences between the two types of networks (e.g. Arch Rock provides means to determine whether a node is lost, Z-Wave does not).

While there has been considerable work on APIs for sensor networks, this work has mostly focused on expressiveness and abstraction, while leaving out many issues which are essential when operating sensor networks efficiently, both in terms of energy consumption and in management overhead. Although we have made progress on several of the above topics we think that further research is needed to find sensor network APIs which cover all aspects of operating a sensor network in a vendor-independent way.

7 References

[1] J Hill, M Horton, R Kling, L Krishnamurthy, The platforms enabling wireless sensor networks, Communications of the ACM, 2004

[2] Jari K. Juntunen, Mauri Kuorilehto, Mikko Kohvakka, Ville A. Kaseva, Marko Hännikäinen, Timo D. Hämäläinen, WSN API: Application Programming Interface for Wireless Sensor Networks. The 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)

[3] JSR 256: Mobile Sensor API. JSRs: Java Specification. jcp.org/en/jsr/detail?id=256, 2006

[4] T Luckenbach, P Gober, S Arbanowski, A, TinyREST: A Protocol for Integrating Sensor Networks into the Internet, REALWSN, 2005

[5] Å Östmark, J Eliasson, P Lindgren, A van Halteren, An Infrastructure for Service Oriented Sensor Networks, Journal of Computers, 2006

[6] S Krishnamurthy, TinySIP: Providing Seamless Access to Sensor-based Services, Mobile and Ubiquitous Systems: Networking & Services, 2006

[7] Marco Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli and Jan M. Rabaey, A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor NetworksIn Ambient Intelligence, W. Weber , J.M. Rabaey, E. Aarts (Editors), Springer 2005

[8] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong. TinyDB: An Acqusitional Query Processing System for Sensor Networks. ACM TODS, 2005

[9] Zensys Z-Wave Developer's Kit, http://www.zen-sys.com/modules/Products&Techonology/, 2007

[10] Arch Rock Primer Pack/IP, http://www.archrock.com/product/, 2007

[11] Fielding, Roy Thomas (2000), Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine