Halmstad University Post-Print

An Implementation of a 3-tier Hierarchical Wireless Sensor Network

Urban Bilstrup and Per-Arne Wiberg

N.B.: When citing this work, cite the original article.

©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Bilstrup U, Wiberg P. An Implementation of a 3-tier Hierarchical Wireless Sensor Network. In: 2006 IEEE International Conference on industrial Informatics. IEEE; 2006. p. 138-143.

DOI: <http://dx.doi.org/10.1109/INDIN.2006.275749>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-2107>

An Implementation of a 3-tier Hierarchical Wireless Sensor Network

Urban Bilstrup
Centre for Research on Embedded Systems
Halmstad University
Halmstad, Sweden
Urban.Bilstrup@ide.hh.se

Per-Arne Wiberg
Free2move AB
Pilefeldtgatan 77
Halmstad, Sweden
Per-Arne.Wiberg@free2move.se

Abstract – In this paper an implementation of a wireless sensor network is described. The aim with the implementation is to investigate if present design patterns are applicable on wireless sensor networks. A 3-tier model is adopted as a possible candidate for the software as well as for the network architecture. The implemented wireless sensor network consists of a heterogeneous set of hardware devices such as sensors, sensor hubs, beepers, PDAs and connectors. Most of the hardware components are COTS and most of the software is GNU licensed. All these form a 3-tier hierarchical network architecture.

I. INTRODUCTION

Wireless communication has for a long time been foresighted as an upcoming technology in industrial applications [1, 2], e.g., wireless field buses for controlling and monitoring systems, surveillance, supervision and logistic applications. Especially, recent research results [3-6] and standardization [7, 8] in the area of wireless sensor networks are very promising for industrial applications.

The implementation of a wireless sensor network is presented in this paper. The well known 3-tier web application design pattern is applied on the wireless sensor network application. This choice was done as an experiment to see if the holistic similarities that are present between a web application and a wireless sensor network application actually hold when it comes to implementation. Distributed systems for industry is often built out of heterogeneous sets of hardware and software. A wireless sensor network can be seen as a platform used for investigating what kinds of implementation problems that are present in such a heterogeneous distributed systems [6], especially when application domain specific constraints are added.

One constraint that is introduced in this work is the minimization of the energy consumption of the battery powered sensor nodes. The aim is also to evaluate if available hardware and software commercial off-the-shelf (COTS) components can be used for building wireless sensor networks in an efficient way. The requirement on the system is that it should support a large numbers of sensors that measure some physical entity in the environment. The system is intended for a low duty cycle surveillance application.

II. SYSTEM OVERVIEW

The system consists of sensors, sensor hubs, beepers, PDAs and connectors. Each sensor is a capacitive pressure sensor, measuring the physical force. The sensor is event triggered and delivers an alarm message to the sensor hub when a certain threshold is crossed, either from a low to a high pressure value or vice versa. Each sensor is associated

with a sensor hub, see Fig. 1. The hub contains a display showing the status of the sensors. This is one of the human machine interfaces (HMI) of the system. The handheld devices of the system are called the beepers. These contain small displays that tell which sensor hub has triggered the alarm. When a sensor hub receives an alarm from one of its connected sensors, it will trigger an alarm message to a predefined beeper (sensor-beeper pairs are given at system setup) via the connector. When the alarm is triggered, the person holding the beeper has to go to the specific location of the sensor hub and reset the alarm. On the display of the sensor hub the beeper holder can see which sensor has triggered the alarm. This functionality enforces a visual inspection of what has actually happened at the specific area (where the sensor is located). This exemplifies a distributed HMI and especially points out how the HMI and the logical interaction with the system can be used to enforce a certain functionality of an ambient system.

The wireless sensor network is built out of a heterogeneous set of hardware and software components: sensor nodes, sensor hubs, Bluetooth RS-232 dongles, Bluetooth modules, handheld devices, Embedded Linux systems, and the BlueZ Bluetooth protocol stack. Most of these hardware and software components are COTS. There are OEM hardware modules available from Free2move AB [9] and software available under GNU license [10, 11].

The network topology is depicted in Fig. 1. The system is configured offline, meaning that the topology is known in advance before the start up. The only mobile devices are the beepers, which are associated and re-associated with different connectors (one at a time) as the beeper holder is moving around. No new devices can be configured dynamically during system run. The connectors are the most powerful devices in the system. These are wirelessly point-to-point interconnected and provide wireless connections to the sensor hubs and beepers.

Fig 1. Hierarchical communication architecture

The beepers, sensor hubs, and the PDA provide the HMI to the system. The PDA is used only for configuration of the system. The sensor hubs are wirelessly connected to the sensors using a proprietary communication protocol in the 2.45 GHz ISM band. The point-to-point communication between the connectors and the communication between the connectors and hubs/beepers is done using Bluetooth.

Fig 2. Hierarchical functionality architecture.

The implementation is inspired by a multi-tier web design pattern [12]. The system is a 3-tier architecture, depicted in Fig. 2, consisting of a sensor tier, an interaction tier, and a backbone tier. One can directly identify the similarity between a general 3-tier application and the inherent structure in a wireless sensor network. The general 3-tier application usually consists of a data tier (e.g. databases), a logical tier and a presentation tier. In our architecture the sensor tier collects the data. This tier is the

same as the data tier/database in the web design pattern. The backbone tier in Fig.2 is the most powerful part of the network and makes the logical decisions. This is the same as the logical tier in the web design pattern. The interaction tier is the HMI of the wireless sensor network and is the same as the presentation tier of the web design pattern. In our architecture the network and the software are merged.

One of the intentions with the general 3-tier software architecture is to modularize the software. The difference between the general 3-tier software architecture and our wireless sensor network is that the latter is often more fine grained distributed over several physical devices. In the general 3-tier software architecture, the database, the application, and the client station usually are spatially distributed. In the wireless sensor network each tier is represented by multiple physical units performing the same thing, i.e., it is not a strict client-server model. The functionality of the individual tier is distributed over several devices.

In Fig. 2 one can see that the functionality of the interaction tier is horizontally distributed over several devices. The beeper and the PDA are part of both the backbone tier and interaction tier, whereas the sensor hub is part of all three tiers.

Another thing that differentiates the wireless sensor network from the general 3-tier software architectures is that the former introduces strict resource restrictions. The sensor hub and the connectors have continuous power supply while the sensor nodes and the beepers are powered by battery. The emphasis is on putting as much resource consuming functionality as possible in the connector. Even if the sensor hub has a continuous power supply it is resource limited when it comes to memory and computational capacity. Memory and computation demanding functionality is to as large an extent as possible placed in the backbone tier. There is of course a trade off between where to place the computational capacity and the increase in the communication when data is moved around.

III. BACKBONE TIER

The backbone tier is responsible for system configuration, logging, keeping track of the network topology, message forwarding, message filtering, and error detection.

A. Connector

The function of the connector in the backbone tier is to receive and forward messages to other connectors or to the interaction tier. All connections in the backbone tier are based on the Bluetooth technology. The messages are broadcast between the connectors. Every connector has two standard serial ports, which are equipped with standard Bluetooth-RS-232 dongles [14]. Each dongle is running the serial port profile (SPP) [15]. The number of connectors that can be interconnected is limited by the capacity of the point-to-point link and the overall system load.

The connector node is based on the F2M05 Embedded Linux System [13], and the BlueZ Bluetooth protocol stack [11] is included with the Linux core. An F2MC1 Bluetooth

module [17] is also integrated directly in the system. Since the system has two serial ports equipped with dongles and one Bluetooth module directly in the system the connector supports up to three piconets, see Fig. 3. Bluetooth is using a master-slave scheme and every piconet in Bluetooth supports up to seven slaves. This is a restriction that Bluetooth is putting on the system.

Message parsing, message switching and message filtering are included in the application in the backbone tier. The necessary logical control information is provided by a set of tables and registers stored in every connector:

- The *backbone tier architecture table* includes the architecture of the backbone (how the connectors are interconnected).
- The *architecture and message filtering table* contains the information about the connections between connector-PDA, connector-sensor hub and connector-beeper.
- The *beeper location register* supplies the connector with information about where a specific beeper is situated (which connector a specific beeper is associated with).

Fig. 3. Bluetooth piconets.

All Bluetooth devices that are members of the network are identified by their unique Bluetooth addresses (48-bit IEEE 802 standard addresses). The connector is identified by the Bluetooth address of the embedded Bluetooth module, not the Bluetooth addresses of the Bluetooth RS-232 dongles.

The application protocol (APP) is a command control protocol running on top of the SPP Bluetooth connections. In order to have a consensus about the total system state at all connectors, the control command packets are broadcast to all connectors. Messages towards the sensor hubs and beepers are filtered. Only control command messages directly addressed to a beeper/sensor hub are forwarded. The intention with this message filtering is that the sensor hubs and the beepers should handle an as small amount of information as possible.

B. Beeper, sensor hub and PDA

The parts of the beeper, the sensor hub and the PDA that

belong to the backbone tier are responsible for the communication with the connector by using Bluetooth. A more thorough technical description of the beeper, the sensor and the PDA, is given in Section 4.

IV. INTERACTION TIER

The interaction tier provides the human machine interface (HMI). It contains the beeper, the display and the keypad on the sensor hub and the PDA for configuration. The logical functionality of the HMI is horizontally distributed over several physical devices. This distribution of the HMI enforces a specific behavior by the user (beeper holder) of the system since the user has to pay a visit to the sensor hub that triggered an alarm.

A. Beeper

The beeper contains a simple microprocessor, a display presenting messages and a buzzer that informs the beeper holder about incoming messages. The beeper is also equipped with the Bluetooth module F2MC1 which is running the serial port profile (SPP) configured as an endpoint device. The text presented on the display is simply a text string that comes along with the incoming control command message. The microprocessor runs the application protocol and the message parser, which extracts text messages. When a new message has arrived it is shown on the beeper's display and the beeper starts to buzz.

The beeper is the only mobile device in the system. It can migrate from one connector to another. In order to keep track of which connector a beeper is associated with a specific register (*beeper locations register*) in every connector is tracking the beeper-connector pairs. This is the only network topology register that changes dynamically when the system is running. All connectors have the same knowledge about the topology.

B. Sensor hub

The sensor hub is based on a PIC microprocessor, a display, a simple keypad for human interaction, an RF transceiver and the integrated Bluetooth module F2MC1 configured as an end point running the SPP. From the interaction tier perspective the sensor hub provides an HMI where local sensor status is displayed and human input can be performed, e.g., reset of present alarms. Messages (e.g. alarm messages) generated in the sensor hub are directly forwarded by the Bluetooth connection, to the backbone tier, towards the connectors.

C. PDA

A PDA, a laptop or any general handheld device that is Bluetooth enabled with a web client can be used for configuration of the system. The configuration of the system is done through a web client, where the HMI is actually located at the connector and is available by using an embedded web server.

V. SENSOR TIER

The sensor tier is the information source in the system. The sensor tier is divided between the sensor devices and

the sensor hubs. The radio transceivers used for wireless communication between the sensor devices and the sensor hubs are based on commercially available transceivers from Nordic VLSI [18] and a proprietary wireless communication protocol developed during this work.

A. Sensor

The sensor is a capacitive pressure sensor (a 15 x 15 cm soft plate capacitor) measuring the physical force. The sensor is made out of a soft plastic material. It contains a Nordic VLSI transceiver [18], a microchip microprocessor [19] and some glue electronics for the sensor. The sensor can be in one of three different modes:

- In *non-associated mode*, the sensor only wakes up once every minute and transmits a hello message.
- In *sensing mode*, the sensor is associated with a sensor hub. In this mode the sensor device is sampling its sensor periodically (once every minute) and sends hello messages every twenty seconds.
- When the sensor has crossed a predetermined threshold it will change to *sensing alarm mode*. Then the sensor will send an alarm message to the sensor hub.

B. Sensor hub

As can be seen in Fig. 2 the sensor hub is the communication interface between the backbone tier and the sensor tier and it also contains parts of the interaction tier. Upstream, towards the connector, the Bluetooth link provides the wireless connectivity. Downstream towards the sensors, the communication is performed via a 2.45 GHz radio interface with a proprietary communication protocol for low power consumption.

The sensor hub contains some network topology information, for example the IDs of the sensors that are connected to it. It further keeps track of all configuration parameters of each individual sensor device.

The sensor hub is literally the master of the sensors. A sensor obeys the sensor hub blindly as soon as a sensor device is activated. Each hello message sent from a sensor contains all configuration information that defines the functionality of the sensor. This information is the sensor ID, the sampling rate (wake up periodicity), the threshold for triggering an alarm transmission, the mode (sensing, not sensing), pending message acknowledgements, timeout before a sensor is assumed to have lost its connection, the battery condition etc., all contained in a 25-byte message. When the sensor hub sends an acknowledgement in response to a hello message it answers with the same parameters. The sensor hub can immediately change any parameter in the acknowledgement message and a change in any parameter is directly adopted by the sensor device.

VI. ERROR DETECTION AND RELIABILITY

Since wireless links are used for communication, error detection and reliability must be provided by the system. A broken wireless link or an empty battery at a sensor/beeper can disable the functionality of the system. Therefore there

must be some form of warning when the system is malfunctioning. This is achieved by using periodic *hello messages*, which are sent by all members of the network. The sensors send a *hello* message to the sensor hub and the hub answers with an acknowledgment, see Fig. 4. The sensor hubs will also send *hello* messages but to the connectors and not to the sensors. The periodicity of the fault detecting procedure is decided during set up and configuration of the system. For each wireless connection there is a timer in the calling end, which periodically triggers the *hello message*. The receiver of the *hello message* generates the response (sends an acknowledgment) to notify the calling end that the two ends are still connected to each other. If the calling end has not received an acknowledgment for the last three *hello messages* it will send a system error alarm. Further all information transportation is carried out with call-response procedures.

If the connection between a beeper and all connectors is broken, only a local alarm in the beeper is triggered to inform the holder of the beeper that he/she has lost connection to the system. The sensors that are paired with a lost beeper are automatically re-associated with any other present beeper in the system.

Fig. 4. Shows the *hello* procedure of the system.

If a sensor hub, a sensor or a connector breaks down the system will trigger a system error alarm and stop functioning. The breakdown is revealed by the absence of *hello/ack* messages. If a breakdown occurs the whole system must be restarted.

Bluetooth's error detection is used on the Bluetooth links and in the case with the proprietary wireless communication protocol (the communication between the sensors and the sensor hub) a CRC is used. If a faulty message is detected the message is just thrown away without notifying the sender. The actual packet loss is detected by the request-response scheme.

VII. CONFIGURATION

The system is set up using a Bluetooth enabled device (e.g., PDA) with a web client. A configuration file is uploaded to the connector that has been assigned to be the configuration master. This can be any of the two end

connectors in the linear backbone network. The end connectors have only one point-to-point Bluetooth connection to a neighbor connector, see Fig. 1. The first thing that happens when the configuration file has been received by the configuration master is that all point-to-point connections in the backbone tier are set up. The configuration file contains all the information about the network topology, e.g., which sensors that belong to a specific sensor hub, or which sensor hubs that are connected to a specific connector.

After the backbone configuration each connector establishes contact with the sensor hubs that were given in the configuration file. The connectors also start to search for beepers that are associated to the whole system. Each connector reports on the backbone, when it has connected all the predetermined sensor hubs successfully and which beepers it has found. When all devices are found the connectors tell the sensor hubs to start to connect to all sensors that are associated to them. If all the sensors of a specific sensor hub are successfully connected, the sensor hub reports that back to the connector. When all sensor hubs connected to one connector have reported that they are successful the connector reports that on the backbone network. When all devices have been successfully connected and configured, the configuration master sends a success response to the web client used for configuration.

VIII. DISCUSSION

In this implementation study of a wireless sensor network we have deliberately used a heterogeneous set of standards combined with heterogeneous hardware and software. The reason for this is that most distributed systems in the industry evolve to these kinds of heterogeneous systems after a couple of years, when more and more devices and functionality are added and systems are merged with other systems and so forth. In our case the heterogeneous system was created already from the beginning as an experimental platform. This was done in order to test new solutions and new design patterns.

The software of the wireless sensor network requires a finer grained modularization of the software than the general 3-tier software architecture. Communication and computation tasks must be seen from a common restricted resource perspective. Resource restrictions often enforce sidesteps from the original design, where original modules are shared and merged with others. The spatial position of certain functionalities in the system architecture is the most powerful tool available to control resource utilization. The rough modularization in the general 3-tier software architecture lacks this capability; specific functionalities are traditionally placed at a certain place in the system architecture. Breaking up these modules in the 3-tier architecture is often hard and requires a great effort. Some conclusions from the design pattern point of view are that the communication and computation must be seen from a common restricted resource perspective. Modules should be fine grained and provide general interfaces. Software frameworks that support this kind of distributed resource constrained systems have to be developed. Tested ready-to-use modules are necessary to speed up the development

and secure the correctness of the system

As much GNU software as possible is used, e.g., Linux, a serial port proxy, BlueZ protocol stack, a web server etc. This is done in order to save money and apply already tested software. Unfortunately most open source software has not been designed for resource limited platforms such as wireless sensor networks. One can conclude that available GNU software is not very well designed for embedded systems and especially not when it comes to meeting constraints concerning the power consumption, memory and computation resources. The low power tier (the sensor tier) clearly disqualifies most present open source software.

The complex communication architecture (Bluetooth) is a result of creating a flexible architecture where part of the system could be used standalone. The sensor hub is possible to use together with sensors, without any connector and most applications would probably not need to interconnect several connectors. However as an experiment and as a platform for evaluating problems when using heterogeneous devices in a system it was a good thing to do.

One Bluetooth piconet can support up to seven slaves. This introduces a restriction to the implementation and will be a problem when dealing with the mobile units (beepers) of the system. One connector can easily get overpopulated if all beepers (>7) try to associate with one specific connector.

Another major problem with Bluetooth is that there exists no commercial implementation of the scatternet functionality. A scatternet consists of several interconnected piconets where they share devices among each other. One device can only be active in one piconet at a time. If this functionality had been present we would not have been forced to use three Bluetooth devices on one connector.

An interesting future work is to try to apply an n-tier architecture on the platform, where the information is processed in a totally distributed and context dependent way.

The sensor hub's functionality is implemented without any micro kernel which makes the handling of one serial port, a keypad, a display and a RF interface a nightmare. It would have been wise to use any available kernel from the beginning. It would also be interesting to introduce a small kernel in the sensors. A micro kernel would introduce a computational overhead in the sensors but provide a more structured programming interface. A future candidate for the implementation of sensor tier is the Tiny OS concept from Berkeley [22]. So far we have not been able to test the Tiny OS on the sensor nodes and sensor hubs but it is scheduled as future work.

A problem necessary to deal with in order for the system to be easily deployed is to develop a graphical configuration tool where the user simply adds devices by picking and placing devices on a worksheet and connecting them by drawing lines between them a similar tool as the icon based programming interfaces used by Lego Mindstorm [20] or LabView [21]. Another possibility is to implement some form of self configuration where only the

logical rules are configured by the operator.

General issues concerning the design of wireless sensor networks are of course the problem of proving the correctness of a distributed system. It is very hard to actually pinpoint errors, especially when it comes to transient errors. The complexity of a distributed system, such as a wireless sensor network, is high even if the functionality may seem to be very limited. Non-coherent system state is a general problem and has a tendency to generate system errors if not properly implemented. One example is that a device thinks that a wireless communication link is lost caused by retransmission timeout even if the link is not lost, e.g., problem with local clocks' drift. As a concluding remark one can say that the 3-tier architecture reflects the structure of a wireless sensor network on a holistic level but it lacks the inherent structure to distribute functionality over several devices. This is necessary in order to fulfil resource restrictions in a heterogeneous wireless sensor network.

IX. REFERENCES

- [1] Theodore S. Rappaport, "Indoor Radio Communications for Factories in the Future", *IEEE Commun. Mag.*, Vol. 27. Issue 5, May 1989, pp. 15-24.
- [2] P-A. Wiberg and U. Bilstrup, "Wireless technology in industry – Applications and user scenarios", in *Proceeding of the 8th IEEE Conference on Emerging Technologies and Factory Automation*, Antibes-Juan les Pins, France October 2001, pp. 123-133.
- [3] A. Willig K. Matheus and A. Wolisz, "Wireless Technology in Industrial Networks," in *Proceedings of the IEEE*, Vol. 93, No. 6, June 2005 pp. 1130-1151.
- [4] R. Zurawski ed., *The Industrial Information Handbook*, CRC press, U.S., 2005.
- [5] K. Koumpis, L. Hanna, M. Andersson and M. Johansson, "A review and roadmap of wireless industrial control," available at: <http://wireless.industrial-networking.com/articles/articledisplay.asp?id=913> (060215).
- [6] D. Estrin et. al., *Embedded, Everywhere – A research Agenda for Networked Systems of Embedded Computers*, Committee on Networked Systems of Embedded computers, Computer Science and Telecommunication Board and National Research Council, National Academy Press, Washington DC, 2001.
- [7] IEEE Computer Society LAN MAN standards Committee. *IEEE Std 802.15.4-2003 Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. The institute of Electrical and Electronics Engineers, New York, NY, May 2003.
- [8] ZigBee Alliance, *ZigBee standard v 1.0*, 2005.
- [9] Free2move AB, www.Free2move.se, (060215).
- [10] Linux kernel 2.6.11 for ETRAX available at: <http://developer.axis.com/download/software.html> (060215).
- [11] Sourceforge, BlueZ - Official Linux Bluetooth protocol stack, available at: www.bluez.org, (060215).
- [12] E. Morrogh, *Information Architecture: An Emerging 21st Century Profession*, Prentice Hall, U.S., November 2002.
- [13] Free2move Embedded Linux System - F2M05, information available at www.free2move.se/prod_emb.shtml (060215).
- [14] Bluetooth serial port plug – F2M01, information available at www.free2move.se/prod_spp.shtml, (060515).
- [15] Bluetooth SIG, *Specification of the Bluetooth system – Profiles v. 1.1*, February, 2001.
- [16] Free2move AB, *Wireless UART protocol v. xx*, information available from www.free2move.se (060515).
- [17] Bluetooth Class 1 module – F2M03C1, information available at www.free2move.se/prod_mod.shtml, (060515).
- [18] Nordic VLSI, "Single chip 2.4GHz Transceiver nRF2401", available at: <http://www.nvlsi.no> (060215).
- [19] MICROCHIP, "PIC16F87/88 Data sheet", available at <http://www.microchip.com> (050215).
- [20] D. Baum, M. Gasperi, R. Hempel and L. Villa, *Extreme Mindstorms – An advanced guide to LEGO Mindstorms*, Apress, U.S., 2000.
- [21] LabView, information available at <http://www.ni.com/labview/> (060515)
- [22] Tiny OS, available at: <http://www.tinyos.net/> (060215)