Available online at www.sciencedirect.com
ScienceDirect
Sensors and Actuators A 142 (2008) 352–360
SENSORS AND ACTUATORS A PHYSICAL
www.elsevier.com/locate/sna

Smart sensor architecture for mobile-terminal-centric ambient intelligence

Iiro Jantunen [a,*], Hannu Laine [a], Pertti Huuskonen [b], Dirk Trossen [a], Vladimir Ermolov [a]

[a] *Nokia Research Center, Helsinki, Finland*
[b] *Nokia Research Center, Tampere, Finland*

Received 30 September 2006; received in revised form 2 February 2007; accepted 10 April 2007
Available online 19 April 2007

Abstract

This study is about developing an open architecture platform for implementing mobile-phone-centric ambient intelligence. In the proposed sensor network architecture, a mobile phone acts as a central node hosting applications and connecting a local sensor network to back-end servers in the internet. The architecture includes a context awareness layer that abstracts sensor measurements into context atoms through rule-based reasoning and notifies changes in atoms to local and remote applications. The technologies used in the architecture include Simple Sensor Interface (SSI) protocol, nanoIP and low-power short-range radios. The architecture has been implemented and successfully demonstrated using several applications with a commercially available mobile phone with add-on electronics.

Keywords: Smart sensor architecture; Wireless sensors; Remote sensing; Context awareness

1. Introduction

Various architectures have been developed for wireless sensor networks. These architectures usually are ad-hoc peer-to-peer networks requiring computational and networking capacity on sensor nodes [1]. To simplify development of ambient intelligence services, we use a star architecture, in which a mobile phone acts as the trusted user interface device, providing both local network and internet connections and capability to run application software to provide functionality, e.g., context awareness (see Fig. 1) [2].

Compared to other candidates for a user-carried interface device (laptop computers or PDAs), mobile phones have several advantages: highest penetration and acceptance amongst users, relatively low cost and small size, both local and long range wireless connections from everywhere to everywhere, access to a wide range of services via internet, data storage possibility and local computational capacity, and that no additional user interfaces need to be carried by the user.

In our architecture, the mobile phone searches and reads sensors, stores and analyzes sensor values to extract context information, hosts sensor network applications and forwards the appropriate data to servers in the internet to provide extra services and functionality.

The architecture requirements are openness, modularity, scalability and energy efficiency. Openness and modularity are needed to enable creation of novel ambient intelligence applications and services by different industry players. Scalability is needed to enable development of vertical applications of different scale on different application areas (e.g., well-being, health, home automation).

As wireless sensors should be small and not require changing or charging batteries often, low energy use is a key parameter in developing sensor networks. Networking, both protocols and radio interface, should be as low consumption as possible. Also low price is of prime importance to allow widespread use of the architecture.

This architecture has been developed in the EU 6th framework integrated project MIMOSA. The architecture is referred to as MIMOSA architecture in this paper.

This paper is organized so that in Chapter 2, we introduce the key components of the architecture along with software layers. Chapter 3 describes how the architecture has been imple-

* Corresponding author. Tel.: +358 504868699; fax: +358 718063214.
E-mail address: iiro.jantunen@nokia.com (I. Jantunen).

0924-4247/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sna.2007.04.014

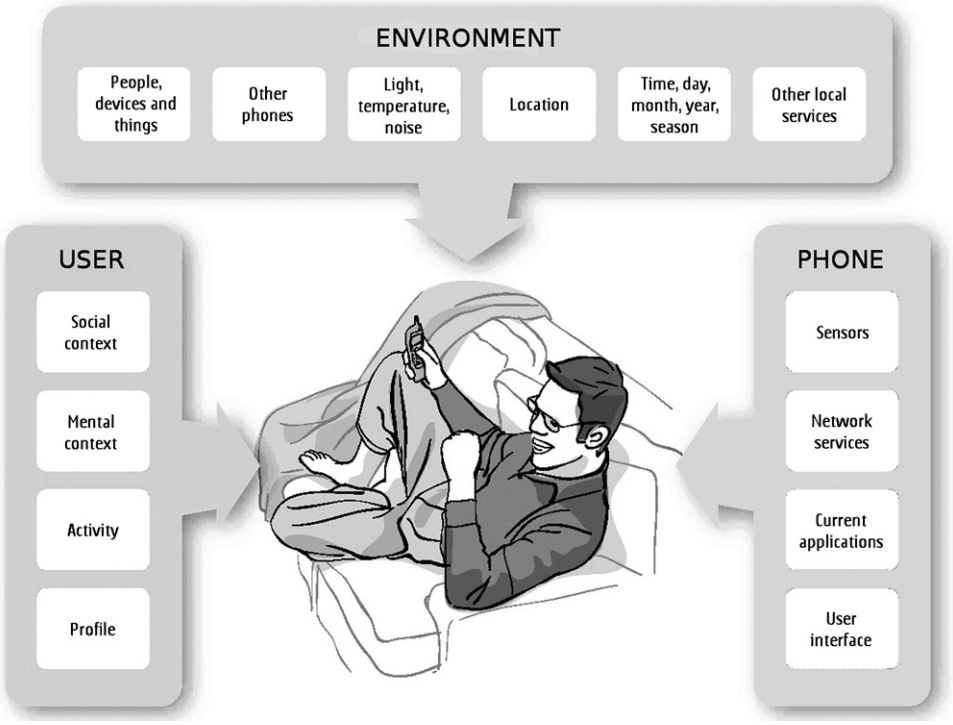

Fig. 1. Mobile phone as the central device of sensor network.

mented to demonstrate its functionality. In Chapter 4, results of the demonstrations are presented and Chapter 5 concludes the paper.

2. MIMOSA architecture

The MIMOSA architecture is designed to be modular and freely scalable. The software architecture is based on three layers: Context Layer, Sensor Layer, and Local Connectivity Layer. The application programming interfaces (API) of the layers are open for 3rd parties. These layers, and remote sensing middleware, will be presented in the following sections. Open interfaces for 3rd party sensor hardware are also provided.

The MIMOSA architecture (see Fig. 2) defines four types of entities: terminal devices (mobile phones) with built-in sensors, sensor radio nodes, wireless remote-powered sensors, and back-end servers.

The terminal device provides capacity to run applications based on sensor data, and in addition to a cellular network connection, has Bluetooth, Wibree and RFID radio interfaces. Back-end servers are computers providing data storage, data processing and extra services.

To provide for different power usage and price requirements in various smart sensor applications, we propose two classes of sensor devices: sensor radio nodes that are wireless battery-powered smart sensors running sensor server software, and wireless remote-powered sensors that are passive RFID tags with a sensor.

Bluetooth is an interesting radio technology for mobile-phone-based sensor solutions, as most phones provide a Bluetooth radio. As low-power (e.g., button cell powered) sensor devices can not bear the cost and power consumption associated with Bluetooth, we propose here the use of Wibree (formerly known as Bluetooth Low End Extension or BT LEE), which is a radio technology specially designed for sensor networks [3,4]. Wibree solves the cost and power use problem by introducing minor power-saving additions to the Bluetooth chip. It is relatively cheap to provide stand-alone Wibree chips for wireless sensors or add the Wibree functionality to Bluetooth chips for mobile phones—in contrast to adding another low-power radio technology designed for sensors, e.g., ZigBee [5].

To simplify the terminal by removing the need of yet another radio front-end, the ISO 18000-4 RFID was chosen as it works

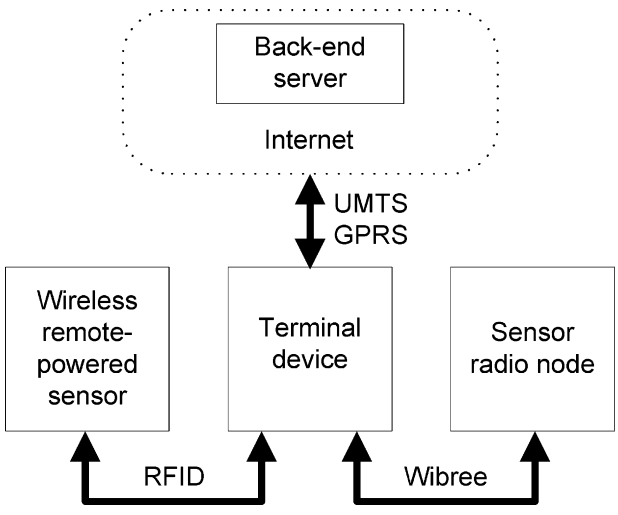

Fig. 2. Architecture overview.

Fig. 3. NanoUDP message format. Prtcl is the protocol and flag byte. Length is the total length of payload and possible CRC. Src and Dst are source and destination port numbers, both 0×28 for SSI. CRC checksum is used if requested by the application.

on the 2.4 GHz frequency band already used by Bluetooth and Wibree.

Plug-in type implementation of sensors is the key to modularity. The Sensor API detects what sensors are available regardless of their location: directly connected on the terminal, in a sensor radio node, or in an RFID tag. The Sensor API on the host device keeps a list of available sensors and provides functions for accessing them.

2.1. Local Connectivity Layer

The Local Connectivity Layer provides a Local Connectivity API to send and receive messages to and from locally connected sensors and remote devices over Wibree radio and RFID tags. To provide low overhead networking, commercial solutions are available. For lightweight sensor applications, loss of an individual sensor value is often not so important, so traffic control or message tracking is not needed. To make possible wide use of the proposed architecture, we have chosen an open source networking solution, nanoIP [6].

NanoIP was selected as a networking solution with less overhead than TCP/IP has, while also being freely usable, in contrast to a proprietary protocol, such as ANT by Dynastream [7]. The small overhead of 5 bytes is achieved, for instance, by relying on the MAC (medium access control) layer for addressing. We use the nanoUDP message type (see Fig. 3), that includes a protocol byte, 2-byte length of message, source and destination port numbers and the SSI message as payload. NanoUDP is designed for situations where an individual message is not critical, and so does not provide proof of message receive nor retransmission. If getting the messages through is considered critical, nanoTCP should be used, as it provides flow control and retransmission if needed. The price for this, however, is heavier message overhead (9 bytes) and increased network traffic.

The Wibree API provides methods for device discovery, connection setup and data delivery over Wibree radio. The RFID API provides methods to identify tags and read or write tag memory content using an RFID interrogator.

For locally connected sensors (add-on hardware), point-to-point wired connections (UART, SPI, I2C) are used instead of nanoIP networking. For such situations, the SSI/UART message format (see Fig. 4) is used.

Fig. 4. SSI/UART message format. Start byte indicating beginning of message is 0×FE. Length and ~Length are length and bitwise not of length of message payload with optional CRC.

Table 1
SSI v1.2 command set

Cmd	Dir	Description
Q/q	C→	Query for sensor devices
A/a	←S	Query reply
C/c	C→	Discover sensors on device
N/n	←S	Discovery reply
Z/z	C→	Reset SSI device
G/g	C→	Get configuration data of a sensor
X/x	←S	Configuration data response
S/s	C→	Set configuration data for a sensor
R/r	C→	Request sensor data
V/v	←S	Sensor data response
D/d	←S	Sensor data response with one byte status field
M/m	←S	Sensor data response with many data points
O/o	C→	Create sensor observer
Y/y	←S	Sensor observer created
K/k	C→	Delete sensor observer
L/l	←S	Request sensor listener
J/j	C→	Sensor listener created
E/e	↔	Error messages
F/f	↔	Free data for custom purposes

All SSI commands are one byte ASCII. If the command is in lower case, a CRC checksum (see Figs. 3 and 4) is calculated and attached to the end of the message. On direction (Dir) column, C and S are client and server.

2.2. Sensor Layer

The Sensor Layer provides a sensor client, an RFID sensor tag reading interface and a Sensor API for upper level or 3rd party software. The layer finds and reads both locally connected and wireless sensors.

Simple Sensor Interface (SSI) protocol [8] defines a method for reading sensors regardless their type, location or connection between the sensor and the reader. SSI is a client-server architecture, where sensor devices act as SSI servers and terminal devices as SSI clients. A single sensor device can have multiple sensors. Both polling sensors by client terminals and streaming data from sensor servers are supported (see Table 1). SSI is an application level protocol that can be used over any network environment.

The command-answer pairs Q-A and C-N handle finding sensor devices and information about the sensors available on them. Sensor reading (polling) is done with the command-answer pair R-V (see Fig. 5). In streaming, the sensor device is called "sensor observer" and the reading device is "sensor listener". The SSI client can request a sensor observer and the SSI server can request a sensor listener. To speed up data transfer in streaming, the SSI server can buffer data points to send them in messages of type M. The number of data points that can be sent in a single M message depends on the communications buffer length, which in our implementation is 128 bytes, resulting in a limit of 29 data points, each 4 bytes.

RFID sensor tag is a Mode 1 (passive backscatter RFID system) tag as defined by the ISO 18000-4 standard. SSI protocol specification defines the memory layout of RFID sensor tags (see Table 2). RFID sensor tag reader toggles the sensor activation bit (byte address 0×3A), waits until the sensor status is "ready" (0×18) and then reads the memory area, where the sen-

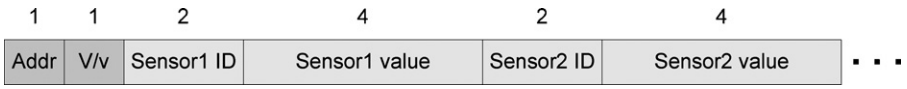

Fig. 5. SSI sensor data response 'V' message. All the sensor values requested are sent. Addr is the device address, one byte, followed by the message type 'V' or 'v'. After that, the sensors' values requested with 'R' or 'r' command follow.

sor value is stored (0×12–0×15). After reading the sensor value, the sensor status bit is cleared by the reader.

2.3. Context Layer

The Context Layer abstracts sensor-level data into higher level units, so-called "context atoms". Each atom stores the latest measurement data, together with semantic information. For instance, the temperature atom may list the latest value obtained from the sensor, but also the measurement unit (e.g., "Kelvin"), scale, documentary text about the source (e.g., "Body temperature"), and a reference to the ontology that defines the meaning of measurement among thousands of other possible interpretations (e.g., "Mimosa/Physiological/Body-temperature").

Fig. 6 shows an example of a context atom. This "Location" atom has been defined by a server whose address is given in the "source" field, for a person known as "username". Such parameters will become useful when tracking changes in the atom, and when looking for an ontology that defines the semantics of this atom. The atom details the time of the latest change. Further

```
<atom name="Location"  source="mupe.no-ip.org"
      userId="username"
      timestamp="2006-05-28 09:57:58,325 +0300">
      <struct name="Airport">
            <string name="Name">Helsinki-Vantaa</string>
            <struct name="Hotspot">
                  <string name="Name">Entrance</string>
                  <array name="NearbyDevices">
                        <string>00119FB001DF</string>
                        <string>00119FB1EB08</string>
                        <string>000EED1E2244</string>
                        <string>00119FB001EF</string>
                  </array>
            </struct>
      </struct>
</atom>
```

Fig. 6. An example of a context atom.

work might include separate timestamps for creation, modification and access. The "struct" keyword allows for nesting of the atom's attributes. In this case, this "Location" atom contains and "Airport" structure, that has one defined "Hotspot" with four devices recognized by their Bluetooth addresses.

Applications can subscribe to the Context Engine to follow context atoms. The Context Layer monitors changes to the value of those atoms and sends notifications to the applications, both local and remote. Any rules that reference the atom in their conditions are triggered upon changes. Rules may then trigger other rules, create or change context atoms, launch applications or send messages to users. The Context Layer is implemented as a Symbian application with mechanisms for atom maintenance, rule processing, simple UIs, and Sensor Layer interfaces.

Fig. 7 depicts the key functional blocks of the Context Engine. The Context Silo is the central data structure that keeps track of the individual context atoms. It has been modeled after the blackboard paradigm; that is, the data on the blackboard is visible to all interested parties and all those parties may modify the data and contribute new data.

The Communication Manager monitors changes to the blackboard, keeps track of parties that have subscribed to a particular piece of data, and notifies those parties when the data gets changed. In practice, such parties are usually applications residing on the same phone, or remote applications that get notified over CEP (Context Exchange Protocol).

Various reasoning engines can observe the context data on the blackboard and contribute new data. The Script Engine has been implemented in Mimosa to allow rule-based reasoning. The rules ("scripts") are modeled after the usual IF-THEN production system pattern: when atoms on the blackboard changed, the rules with matching IF parts are triggered and their THEN parts are carried out. Typically, the rules will trigger new rules. The

Table 2
SSI compatible RFID sensor tag memory map

B	Byte address	Description
8	0x00 – 0x07	Tag ID
4	0x08 – 0x09	Tag manufacturer
	0x0A – 0x0B	Tag hardware type
6	0x0C	Tag memory layout: Embedded Application Code. 0x53 for SSI compliant tags.
	0x0D – 0x11	Tag memory layout: Tag Memory Map Allocation
8	0x12 – 0x15	Sensor value (variable)
	0x16	Sensor value type (0x00 = floating point, 0x01 = signed integer…)
	0x17	Floating point multiplier
	0x18	Sensor status flags. Bit 0 indicates if the sensor value is valid data (bit 0 = 1) or not yet valid (bit 0 = 0).
	0x19	For future use. Could be, e.g., number of sensors on RFID tag.
16	0x1A – 0x29	Sensor description in ASCII.
8	0x2A – 0x31	Unit description in ASCII.
8	0x32 – 0x35	Minimum value the sensor can provide.
	0x36 – 0x39	Maximum value the sensor can provide.
4	0x3A	Activate sensor. Bit 0 written by the reader to request the tag to write the sensor data. Bits 1 – 7 free for future use.
	0x3B – 0x3D	Free for future use.

Memory area 0x00 – 0x11 is defined by ISO 18000-4 standard. 0x12 – is defined by SSI specification. Memory area 0x3E – can be used, for instance, for having more than one sensor on the tag.

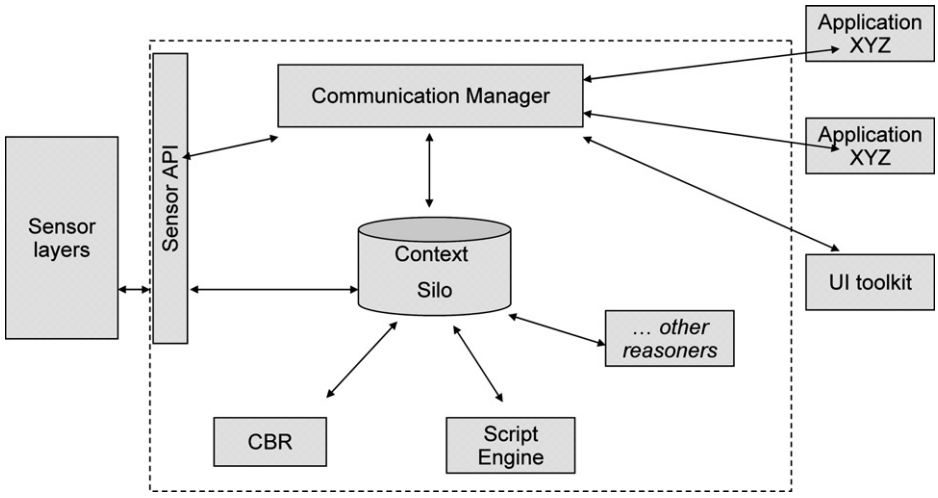

Fig. 7. The functional blocks of Context Engine and their relations to neighboring layers.

operation follows a forward chaining principle, which is typical of sensor data driven systems, including Mimosa.

The rules have been defined with XML (see example in Fig. 8). The XML tree syntax allows for arbitrary nesting of the IF and THEN parts, so the scripts are in fact little applets triggered by the blackboard.

Applications can upload and remove inference scripts using from Script Engine using its API. The upload, as notifications, can happen both with local and remote applications.

As the script language allows input, output, conditional execution, and user notifications, it can be used to implement simple applications. For instance, in Mimosa we built simple scripts that monitor the temperature and humidity of a wine cellar, and upon abnormal conditions sent a text message to the phones of maintenance staff.

2.4. Remote sensing middleware

The Mimosa Remote Sensing Architecture (M-RSA) middleware provides sensor data selection, acquisition and delivery to back-end servers. The middleware [9] resides at both application server and gateway(s), consisting of components for event delivery, acquisition, query resolution, aggregation, access control, storage, and registration and availability discovery (see Fig. 9). In the following, we will outline the functionality of each of these components.

The event delivery component provides functions for all remote communication between an application server and a gateway. At the application server and gateway, there exist subscriber/publisher as well as event server functionality, i.e., both sides subscribe for and provide information to each other. In order to be independent from a particular transport protocol, the event delivery component provides the functionality to upper layer components through transport-specific mapping onto SMS, SIP, HTTP or TCP/IP.

The acquisition component implements local and remote acquisition of sensor data. On the application server side, the acquisition component is one of the two components to be accessed by the application(s).

The application server pre-processes acquisition requests by checking access rights with respect to requested data, verifying the availability of particular sensors that are required to fulfill the request and verifying the availability of a particular aggregation functionality, if required. If these verifications return positive,

```
<script>
 <if>
  <equal>
   <cep:atomRef
    name="location.semantic"
    ref="current" userId="sally" />
   <cep:string>home</cep:string>
  </equal>
 <actions>
   <notify receiver="phone:applications/userNotifier"
    <cep:string name="message">
     Sally is at home
    </cep:string>
   </notify>
  </actions>
 </if>
</script>
```

Fig. 8. An example context script.

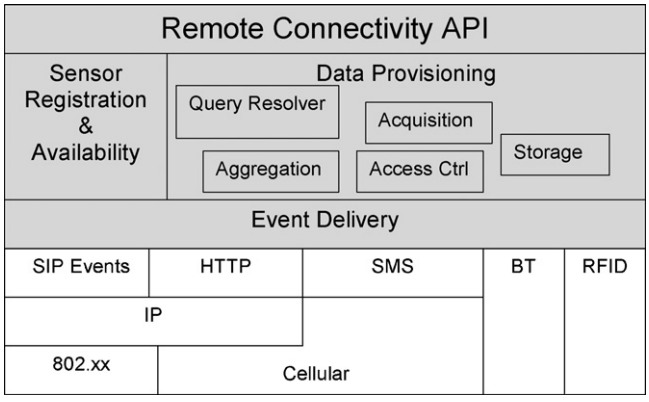

Fig. 9. Remote sensing middleware components.

the acquisition request is sent to the appropriate gateway (the identifier given by the application). In this way, the application server acts as a subscriber for an event at the intermediary gateway, the gateway's acquisition component acting as an event server.

Incoming acquisition requests contain queries for sensor data (or aggregations of sensor data). These queries are based on a defined abstraction model for the sensor data and its aggregations. The acquisition query is forwarded to the query resolver component. It is the task of the query resolver component to parse the request, based on the abstraction model and the query language syntax, determine the sensor data to be acquired by the acquisition component and determine the aggregation functionality that is required when aggregated data is required. The results are delivered back to the acquisition component, which in turn sets up the local acquisition and aggregation according to the query.

The aggregation component provides functionality for the acquisition component to support aggregated sensor data. The required aggregation functionality is also determined within the query resolver component. The acquisition component then requests the determined aggregation functionality from the aggregation component. If the query is fulfilled, appropriate notifications are sent to the application server, triggering callbacks to notify the application.

During the local acquisition process, the acquisition component provides the aggregation component with appropriate sensor data to perform the desired aggregation. If the obtained sensor data is required for future use in order to perform the aggregation, it can be stored with the local storage component until the aggregation can be performed. Future extensions foresee downloading appropriate aggregation code for implementing requested functionality in cases where the required functionality does not exist locally.

Enabling controlled access to the sensor data and its aggregations is of key importance in order to preserve the privacy of the data. The access control component implements this functionality in the application server. This component is consulted for an incoming acquisition request in order to verify appropriate access rights for the particular application.

Apart from data acquisition, discovery constitutes another big role of the middleware, enabling applications to query not only for the availability of certain sensors but also aggregation functionality. The registration and availability (R&A) component implements the necessary functionality for such discovery.

For this, the application server's R&A component serves as a central registry for all registered gateways. Gateways register with the application server when starting up and publish available sensors as well as any available aggregation functionality with the R&A component at the application server. For the former, the R&A component in the gateway subscribes to the local sensor capabilities using the Sensor API (see Section 2.2). After storing the obtained sensor information locally (for use with gateway-local applications), the information is published at the application server's R&A component.

The available information in the R&A component in the application server can either be queried by the application directly or by the acquisition component before sending an

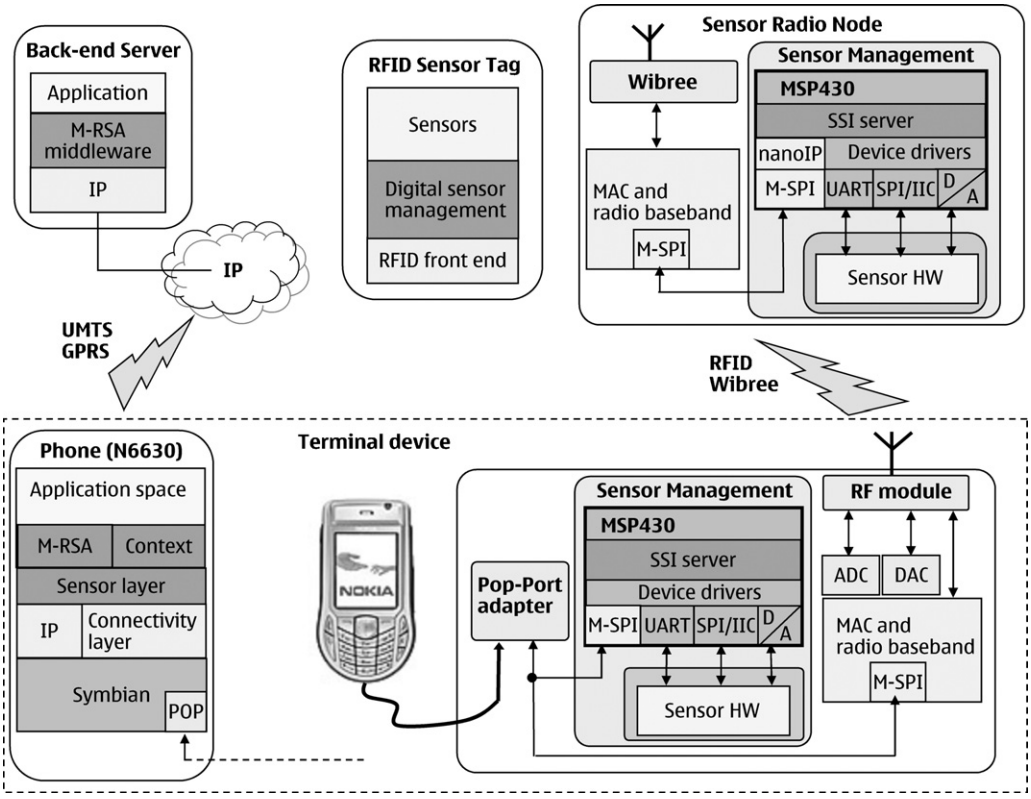

Fig. 10. Architecture implementation.

acquisition request to the intermediary gateway. Apart from a simple discovery request, the component also provides functionality to subscribe to the availability of sensors or aggregation functionality. Hence, applications are notified when such information will become available in the future.

3. Architecture implementation

The proposed architecture (see Fig. 10) is demonstrated with four different hardware entities: terminal device, sensor radio node, RFID sensor tag, and back-end server.

The Terminal Device is a 3G mobile phone, a Nokia 6630 (Series 60), with add-on electronics and software layers to provide MIMOSA functionality. The mobile phone runs local sensor applications, and acts as a gateway between the local sensor environment and internet, where remote servers can provide extra functionality.

The MIMOSA terminal provides Bluetooth, Wibree and RFID radios for reading sensors. Both Wibree and RFID technologies use the same analog RF module. Medium access control protocols of the systems are implemented on an FPGA included in the MIMOSA hardware. The Wibree and RFID protocols are independent of each other but the RF resource arbitrator knows which one of the systems is using the RF resources and prevents the other system using them if already allocated.

The connection from phone to MIMOSA hardware is over an USB to SPI converter (Pop-Port adapter). The MIMOSA SPI (M-SPI) is an SPI bus with extra interrupt lines for local sensor management board, Wibree and RFID systems.

The add-on sensors are interfaced and managed by a sensor management board of the same kind as in the Sensor Radio Node (see below). The only difference is in networking, which is done by point-to-point SSI/UART protocol over the M-SPI bus.

The Sensor Radio Node is a wireless battery-powered smart sensor with Wibree radio, nanoIP networking, and SSI server software. Sensor management (see Fig. 11) is done with a MSP430 microcontroller which runs the SSI server, nanoIP networking, and drivers for sensor and communications hardware. There is also a real time clock, which can be used as a time "sensor". The same sensor management board is used in both sensor radio nodes and terminal devices, the difference being in communications software.

Sensor management board provides a standard connector for sensor hardware, providing a +3.3 V power supply, and including standard UART, SPI and I2C digital interfaces, 8 analog input lines and 15 configurable general digital input/output lines. The latter can be used, for example, to implement a non-standard digital interface to the sensor(s), or provide sensor enable or clock signals.

RFID Sensor Tag is a wireless remote-powered sensor designed for low-cost sensing. The RFID interrogator powers the tag and writes the number 1 to the corresponding bit of the activation byte address (see Table 2). The sensor control hardware then writes the sensor value to its memory and sets the sensor status bit to ready (1). The sensor value is then read by the Sensor API of the terminal device.

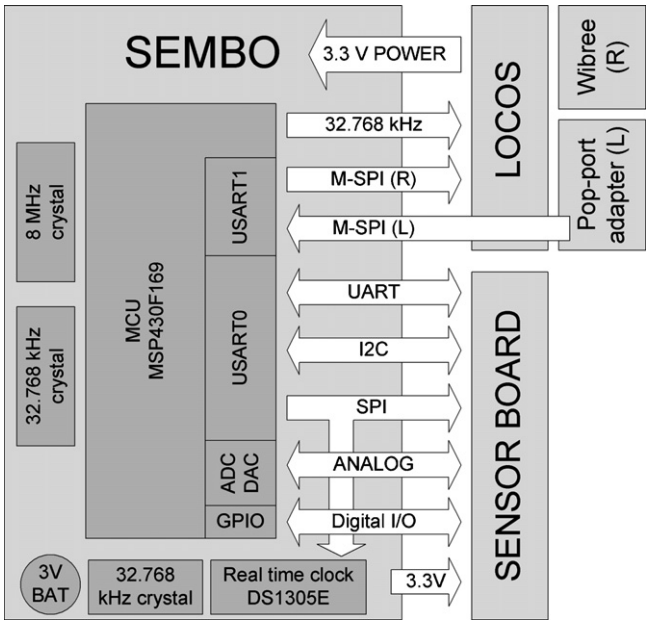

Fig. 11. Sensor management and interfaces. SEMBO is sensor management board, LOCOS is local connectivity board. On a Sensor Radio Node (R), Wibree board is used. On a Terminal Device (L), a pop-port adapter is used.

The analog part includes the tag front end (rectifier, voltage limitation, and backscattering), RFID analog circuitry (demodulator, clock, regulation, current and voltage references, and power-on reset) and analog sensor interface (sigma delta based). The digital part includes RFID protocol management, state machine, sensor interface, and sensor filters. A standalone capacitive sensor is used.

As a back-end server, a laptop computer with MIMOSA Remote Sensing Architecture (M-RSA) middleware in another location was used to read sensor data over the network. The terminal device acts here as the link that forwards sensor readings from the sensors to the computer.

4. Results and discussion

MIMOSA architecture has been successfully demonstrated in a laboratory environment with a wireless weather station (see Fig. 12) as an application. The demonstration consists of a MIMOSA terminal with a user interface application using the

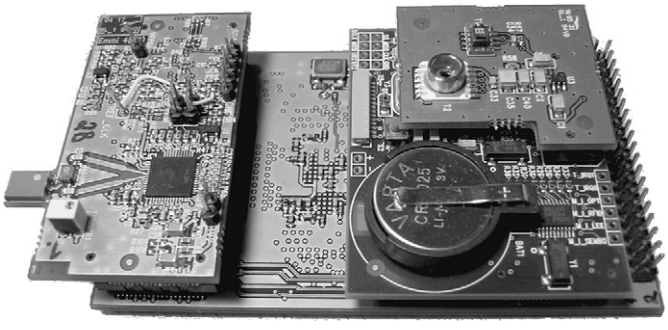

Fig. 12. Wireless weather station. Wibree board on the left, connectivity board on the bottom, sensor management board on right with sensor board on top.

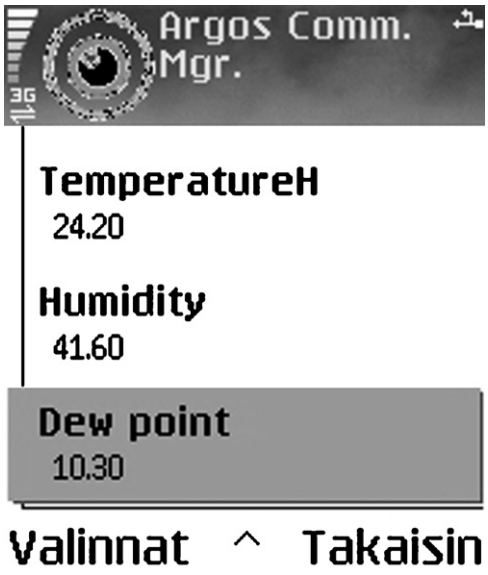

Fig. 13. Screen capture from MIMOSA terminal reading the wireless weather station.

Sensor API to read the environmental sensor values (temperature, pressure and humidity) over Wibree radio. The MIMOSA Remote Sensing Architecture (M-RSA) application on the terminal forwards selected values over the 3G cellular network and internet to a laptop computer acting as a back-end server.

The wireless weather station consists of a Wibree board, a connectivity board with an FPGA running the medium access control protocols, a sensor management board with a microcontroller running device drivers and an SSI server, and a sensor board with Intersema MS5534 pressure/temperature and Sensirion SHT11 humidity/dew point/temperature sensors. The sensors use custom digital interfaces to conserve energy, thus the general digital input/output pins of the sensor management board are used.

The demonstration included searching for sensor devices, discovery of sensors in a found device and reading the sensors (polling and streaming) with a mobile phone (see Fig. 13), along with forwarding the selected sensor values to the back-up server

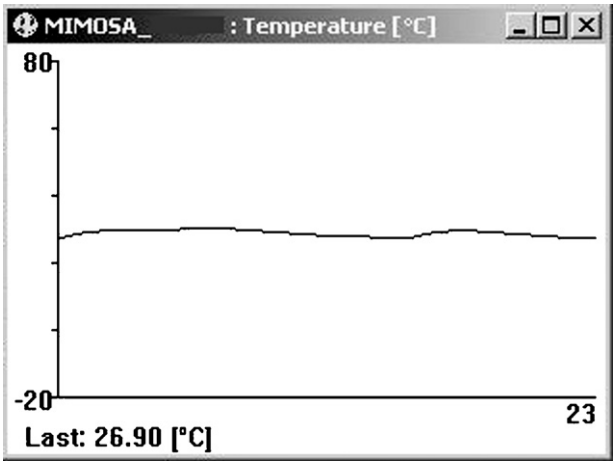

Fig. 14. Screen capture from back-end server reading temperature values from the wireless weather station via a MIMOSA terminal.

(see Fig. 14). The RFID sensor tag was demonstrated successfully by reading the sensor value from the tag memory using the proposed architecture.

To demonstrate streaming sensor data with the reference implementation the speed of sending and receiving data with SSI messages over nanoIP and Wibree could be raised to about 125 samples/s without losing data packages. The sample rate was achieved by streaming 'M' type SSI messages with 28 buffered samples each (one message every 215 ms).

If successful transmission and reception of individual messages is considered to be critical for a given solution, nanoUDP messaging can be replaced by nanoTCP, which provides flow control and retransmission.

5. Conclusions

MIMOSA architecture is a smart sensor architecture optimized for flexibility and low-power use. The key entities included in the architecture are terminal devices which are mobile phones with add-on electronics to provide extra functionality, active sensor radio nodes with their own power sources, passive RFID sensor tags which are powered by the reading signal, and back-end servers. Terminal device provides, along with the capability to run sensor applications, internet connection and acts as a link between the sensors in the vicinity to back-end servers via cellular network and internet. The sensors can be either directly attached to the terminal device, or linked via an RFID or Wibree connection.

The proposed architecture offers an open architecture platform for implementing mobile-phone-centric ambient intelligence in various application areas.

Acknowledgements

This study is part of the MIMOSA project under EU FP6 contract IST-2002-507045. The French CEA-LETI has provided the RFID Sensor Tag and the RFID interrogator logic blocks running in the FPGA of the local connectivity board. VTT Technical Research Centre of Finland has provided the RF block for the RF module.

References

[1] C.-Y. Chong, S.P. Kumar, Sensor networks: evolution, opportunities, and challenges, Proc. IEEE 91 (2003) 1247–1256.
[2] A.K. Dey, Understanding and using context, Pers. Ubiquitous Comput. 5 (2001) 4–7.
[3] Wibree, 2006, http://www.wibree.com.
[4] M. Honkanen, A. Lappetelainen, K. Kivekäs, Low end extension for bluetooth, in: IEEE Radio and Wireless Conference 2004, Atlanta, GA, USA, September 19–22, 2004, pp. 199–202.
[5] ZigBee, 2006, http://www.zigbee.org.
[6] Z. Shelby, P. Mahonen, J. Riihijärvi, O. Raivio, P. Huuskonen, NanoIP: the zen of embedded networking, in: IEEE International Conference on Communications 2003, May 11–15, 2003, vol. 2, pp. 1218–1222.
[7] This is ANT, a Wireless Personal Area Network solution, 2006, http://www.thisisant.com.
[8] SSI protocol, 2006, http://ssi-protocol.net.

[9] D. Trossen, D. Pavel, Building a ubiquitous platform for remote sensing using smartphones, in: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services 2005, San Diego, CA, USA, July 17–21, 2005, pp. 485–489.

Biographies

Iiro Jantunen was born in 1973 in Helsinki, Finland. He received his MSc degree in engineering physics from Helsinki University of Technology in 2001. He joined Nokia Research Center in 2004 and focuses on research topics in the area of sensor networks.

Hannu Laine was born in 1973 in Järvenpää, Finland. He received his MSc degree in Systems and operations research from Helsinki University of Technology in 1999, and thereafter he has been working with the Nokia Research Center focusing on research topics on the area of wireless short-range communications networks.

Pertti Huuskonen is principal scientist with Nokia Research Center, Tampere, Finland. His research interests include context awareness, ubiquitous computing, and mobile interaction. He has been applying artificial intelligence in his previous positions at VTT and CERN on such diverse fields as industrial control, high-energy physics and telecommunication. He holds a doctorate in computer science from University of Oulu, Finland, 1997. He lives on cheese.

Dirk Trossen has been with Nokia Research since 2000, working in areas of context-aware and adaptive service architectures, wireless sensing and long-term evolution of Internet architecture. Dirk holds a PhD from the University of Technology in Aachen, Germany from 2000 and a MSc from the same university, obtained in 1996.

Vladimir Ermolov graduated with honors in 1981 and received the PhD in 1986 from the Moscow Engineering Physics University (MEPhI). Since 1981, he had been as senior research associate with the laboratory of Dielectric Devices (MEPhI). He worked many times as visiting researcher in the Department of Physics, Helsinki University and Fraunhofer Institute of Nondestructive testing, Germany. During his professional career he has worked in areas of sensors for measurements of ocean parameters, acoustic devices for signal processing, SAW devices, variable acoustic devices, acoustic effects in magnetic materials and their technical applications, nonlinear effects in solid state and acoustical nondestructive evaluation. He joined Nokia research center at 1998 where he has been leading several projects in areas of MEMS, Ambient Intelligence and mass memory technologies. He is the author and co-author of 56 scientific publications and the holder and co-holder of 24 patents.