ISSN 1726-5479

SENSORS & TRANSDUCERS

vol. 17
Special
1/13

Sensor Devices and Wireless Sensor Networks

International Frequency Sensor Association Publishing

IFSA

Printed in the USA.

Sensors & Transducers

Volume 18, Special Issue
January 2013

www.sensorsportal.com

ISSN 1726-5479

Sensors & Transducers Journal

Contents

Volume 18
Special Issue
January 2013

www.sensorsportal.com

ISSN 1726-5479

Research Articles

Authors are encouraged to submit article in MS Word (doc) and Acrobat (pdf) formats by e-mail: editor@sensorsportal.com
Please visit journal's webpage with preparation instructions: http://www.sensorsportal.com/HTML/DIGEST/Submition.htm

International Frequency Sensor Association (IFSA).

Sensors & Transducers, Vol. 18, Special Issue, January 2013, pp. 14-23

www.sensorsportal.com

Sensors & Transducers

ISSN 1726-5479

http://www.sensorsportal.com

Smart Sensors and Actuators: A Question of Discipline

Hoel IRIS, François PACULL
CEA-LETI MINATEC Campus
17 rue des Martyrs, 38000 Grenoble, France
Tel.: +33 4 56.52.03.58, +33 4 56.52.03.66
E-mail: francois.pacull@cea.fr

Received: 15 November 2012 /Accepted: 14 December 2012 /Published: 22 January 2013

Abstract: Low power consumption and reliability are two important properties in the wireless sensor network area. The approach presented here to improve these aspects is to use a rule-based middleware enforcing a coordination protocol on top of the communication protocols imposed by the different wireless sensor networks. In addition, we move the callee side of this protocol from the gateway to the sensors/actuators in order to make them able to directly respond to this protocol. Then, it is possible to control from the application side the control (sleep/awake) of the sensors and the transactional processing of operations involving a group of sensors/actuators. This has a positive impact both on the consumption and the reliability. Examples illustrating our approach are presented. *Copyright © 2013 IFSA.*

Keywords: Coordination middleware, Smart sensor, Transaction, Power consumption.

1. Introduction

A Wireless sensor network (WSN) is a set of sensors or actuators connected together through a wireless connection. WSNs are nowadays commonly used in various domains because the absence of wire helps the deployment and decreases the installation cost.

The main components of a sensor/actuator are some sensing or acting units, a micro-controller, a transceiver and a power unit.

Basic objectives of sensor networks were accuracy, flexibility, cost effectiveness and ease of deployment. As these properties are now taken for granted, low power consumption and reliability are the next steps to consider. This paper which is an extended version of [1] presents our approach to improve these aspects through a rule-based middleware [2] enforcing a coordination protocol on top of the communication protocols imposed by the different WSNs. Once the high-level coordination protocol understood by the sensors/actuators, it is possible to control from the application side the control (sleep/awake) of the sensors and the transactional processing of operations involving a group of sensors / actuators.

The paper is organized as follows. Section 2 introduces the problem to solve. We describe in Section 3 our approach based on the high level coordination protocol enforced by our middleware and the design of smart sensors able to react smartly to this protocol. Section 4 presents the hardware we considered. In Section 5 and 6 we exemplify our approach both for power consumption and reliability. We conclude in Section 7.

2. Problem to Solve

The usual way actuator/sensor wireless networks are designed does not help to solve the two problems of reliability and power consumption as described here after.

2.1. Reliability

Most of the time the wireless communication policy in WSNs is only best effort. In addition, sensors are battery powered for autonomy reasons and they may stop their activities or decrease the wireless signal strength just because the battery is partially or totally discharged. Thus, there is no guaranty that a sent message is eventually received.

A traditional way to enforce reliability in an asynchronous system in presence of failures is to embed the operations involving the sensors/actuators within transactions [3]. For instance, we consider to open a window (actuator) and to store this information in a database responsible for keeping the state of the system. If this is not embedded in a transaction, it may happen that the change in the database is done while the actuator is not reachable or on the contrary that the window is opened but the database is locked for some external reasons. As a transaction unrolls a two-phase commit (2PC) protocol if an actuator is not reachable in the first phase (i.e. reservation) then the transaction is aborted preventing the database update. Obviously, this makes sense only if the transaction protocol really involves the actuator and not only the gateway otherwise a failure may happen in between the gateway and the actuator after the commit of the transaction.

We propose to implement smart sensors/actuators that directly provide the required transactional capabilities.

2.2. Power Consumption

Power consumption is the sum of the powers consumed for sensing and computing activities plus the power used for the delivery of information through the radio. In general, both of these parts can be slept when they are not used. However, the main issue is precisely to define when they are not used. Indeed, we can distinguish two categories of sensors. The first one, based on an alarm (e.g. presence detector) can be easily slept until something external happens. On the contrary, the second type (e.g. temperature sensor) regularly emits the information without any idea if it is useful or not for the application. Among the improvements proposed, we may find systems that offer configuration facilities allowing the user to define the delay between two emissions. It is also possible to avoid emitting the information if it did not change or if the change is inside a given interval that can be configured [4]. However, even if these efforts are noticeable it is far to be optimal because in general the user has little information to efficiently configure the sensors in advance since most of the knowledge appears during the execution of the application.

Part of the solution is to let the initiative of the application to interrogate the sensors (pull mode) rather than trying to optimize its emission rate (push mode). This is the direction we investigated.

2.3. Our Approach

Our approach is based on the combination of a rule based middleware which coordinates the actions of a group of software components through a high level protocol. This protocol, thanks to a limited yet effective set of primitives allows on the one hand to control the interrogation of the sensors from the application side and on the other hand to embed a set of operations on sensors, actuators and software components within transactions.

In addition, we have designed sensors/actuators aware of this protocol and then able to behave like first class components of our middleware. In other words, we replaced the classical approach where the sensors are accessed via a gateway with no control on what is actually done beyond the gateway to an architecture unrolling our protocol until the physical devices. The transport layer of the sensor network is in this case a vehicle for our coordination protocol.

The usage of this protocol together with sensors aware of this protocol proposes a response to the both mentioned issues: power consumption and reliability.

We describe in the next sections the middleware and its protocol and the global design of our sensors/actuators both at the software and hardware levels.

3. Architecture Approach: Software

3.1. Coordination Middleware

The middleware provides a uniform abstraction layer that eases the integration and coordination of the different components (software and hardware) involved in WSNs.

It relies on the *Associated Memory* paradigm implemented in our case as a distributed set of bags containing resources (tuples). Following Linda [5] approach the bags are accessed through the three following operations:

- `rd()` which takes as parameter a partially instantiated tuple and returns a fully instantiated tuple from the bag whose fields match to the input pattern;
- `put()` which takes as parameter a fully instantiated tuple and insert it in the bag;
- `get()` which takes as parameter a fully instantiated tuple, verifies its presence in the bag and consumes it in an atomic way.

The bag abstraction can encapsulate real tuple spaces but also databases, services, event systems, sensors and actuators.

For the database we can consider each table as one bag and the `read/write/delete` primitives

can be directly mapped onto our `rd(), put() and get()`.

For a remote service (web service, rpc, ...), the partially instantiated tuple passed to the `rd()` operation corresponds to the input parameters of the service. The bag performs internally the required operation by calling the appropriate remote service via the appropriate protocol and the output parameters are concatenated to the input one to constitute the resource to be returned.

A bag can also maps an event-based system since a `rd()` corresponds to a subscribe and a `put()` to a publish.

From the sensor point-of-view, a couple of bags map a set of sensors and contains the resources corresponding to basic information: e.g. tuples `(sensorid, value, timestamp)` or `(sensorid, type)` allows to model all the data and metadata required to manipulate sensors through the `rd()`or the `get()` operations.

For the actuators, the `put()` operation is used to introduce tuples under the form `(actuatorid, function, parameter1, parameter2)`.

Once inserted, the bag can actually trigger the correct action on the physical actuator with the appropriate parameters.

The tree operations `rd(), get()` and `put()`} are used by the *Production rules* [6] to express the way these resources are manipulated in the classical *pre-condition* and *performance* phases.

The rules are enacted by dedicated components called *coordinators*.

Precondition phase: It relies on a sequence of `rd()` operations to find and detect the presence of resources in several bags. This can be sensed values, result of service calls or states stored in tuplespaces or databases. The particularity of this phase is that:

- the result of a `rd()` operation can be used to define some fields of the subsequent `rd()` operations;
- a `rd()` is blocked until a resource corresponding to the pattern is available;
- a `rd()` operation at the right hand side of a blocked `rd()` is not active and will invoke its bags only when the previous `rd()` receives a response.

This mechanism allows to wake up and access the sensors only when it is required by the application.

Performance phase: It combines the three operations `rd(), get()` and `put()` to respectively verify that some resources found in the precondition phase are still present, consume some resources or insert new resources.

In this phase, the operations are embedded in *Distributed Transactions* [3].

This ensures several properties that go beyond traditional production rules. In particular it ensures that:

- we can verify the important conditions responsible for firing the rule (precondition) are still valid in the performance phase;
- the different involved bags are effectively all accessible.

These properties can improve reliability provided that the sensors are aware of the coordination protocol.

3.2. Protocol Aware Sensor

In order to make the sensors/actuators capable to understand the coordination protocol we need to implement at the sensor level the different operations that will be invoked during the enactment of the rules.

In addition, we need to implement at the client side *stubs* that encapsulate the communication layer in order to provide to the coordinator the way to invoke transparently the remote bags.

Stub: We briefly describe how the stub mechanism is used in our middleware in 0.

When the coordinator needs to access a given bag, it asks the name server in order to obtain the stub that will be used to invoke the `rd()`, `get()`} and `put()` operations.

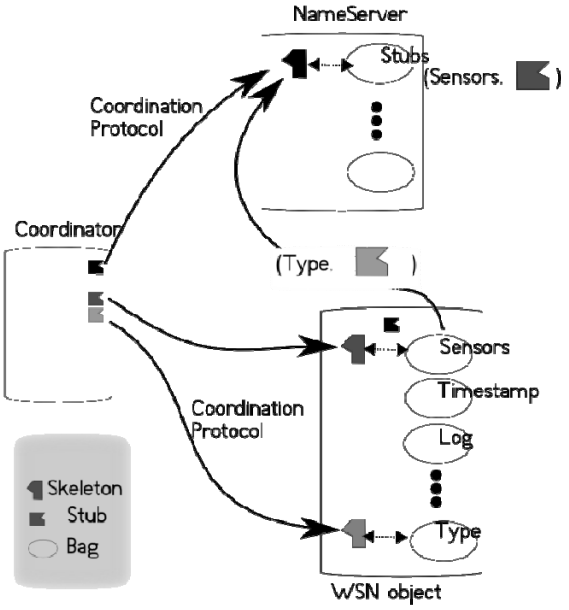


Fig. 1. Stub mechanism.

The name server stores the information related to the stubs as resources of one of its bag called `Stubs`. A stub is inserted by each bag in the name server at starting time. Once obtained by the coordinator the latter just calls the appropriate primitives via the stub and has no idea about the way they are implemented. This means that the transport layer is completely hidden at this level.

A corresponding skeleton able to receive and decode the requests coming through the stub is present at each bag.

A stub allows to invoke two distinct `rd()` operations: one for the precondition phase that may block when no corresponding resource is currently available and one for the performance phase which is decomposed into a `pre_rd()` and `commit_rd()`or `abort_rd()` according to the 2 Phase-Commit used to enforce the transaction.

The `get()`} and `put()` operations are also decomposed in two parts for the same reasons.

Another point concerns the management of the sensor wake-up since we consider that the sensor is in sleep mode by default, and wakes up before calling a primitives of the protocol. Then a signal, whose role is to wake-up the sensor, is sent to it before any request.

Moreover, we consider that the sensor returns in sleep more right after the termination of the `rd()` and the `commit_*()` or `abort_*()` (where * can be `rd, get` or `put`). This signals can be based on different technologies, this is discussed further.

Skeleton: at the sensor side. We consider now, the treatment to be done at the micro-controller side for operation of the precondition and performance phases.

The only operation required in the precondition phase is the `rd()` operation. It has the following behaviours:

- If a resource is available (i.e. a value may be read and returned immediately) then the read is non-blocking. This corresponds to sensors like temperature or humidity measuring most of the time a physical value. The resource based on the sensed value is directly returned to the stub call and nothing else has to be done.
- If a resource is not available then the read needs to be blocked until a resource becomes available. This corresponds to sensors like presence sensor or threshold detector that are bind to an alarm.

As it would not be efficient on a power consumption point of view to let the sensor alive just to block the `rd()` operation, we implement it in a different way.

The value returned to the `rd()` warns the calling stub that nothing is currently available and that the sensor will take the initiative to send the resource when it becomes available. At the stub level, we just block.

When the sensor receives the alarm (e.g. detection of a presence), which wakes it up, the resource is returned to the stub along with an identifier which allows the stub to retrieve to which `rd()` it corresponds. The resource remains stored locally in RAM at the sensor level for the performance phase. The sensor can return in sleep mode and the stub can end the process corresponding to this `rd()`.

For the performance phase, during the `pre_*()` operations, the sensor/actuator has to store intermediate information that will be used during the `commit_*()` or `abort_*()` operations: i.e. the considered resource passed as parameter.

For the `pre_rd()` and `pre_get()` the concerned resource, if available, is locked to prevent any other `pre_*()` operation coming from other transactions to be initiated. If the resource can be locked, `ok` is returned and the lock status is locally stored. Otherwise `notok` is returned.

For the `pre_put()` we verify that the operation is possible: for instance that the actuator can be manipulated or that it remains enough energy to accomplish the required action. Accordingly, `ok` or `notok` is returned.

When the `commit_*()` is invoked the considered operation is effectively done: for a `commit_rd()` no particular action on the resource need to be done. On the contrary for a `commit_get()` the resource is removed (consumed). Finally for a `commit_put()` the action is done on the actuator, generally parameterized according to the resource.

If an `abort_*()` is received then nothing need to be done since this indicate that the overall transaction has failed.

In all cases (`commit_*()` or `abort_*()`), the intermediate information (locks, ...) need to be cleaned since it is no longer useful.

Model of a sensor/actuator network object. To define a prototype object that encapsulates a sensor/actuator network we can consider the following set of bags required for its management.

For the sensors part we have the following bags.

- `Sensors`: stores the sensor id and the value read;
- `TimeStamp`: stores the id and last reading time;
- `Log:` stores the id and reading time;
- `Type:` stores the id and its type (i.e.: temperature, ...).

For the actuators part we have specific bags for the different types of actuators. These bags implement for the `put()`the actual action to be realized to act on the physical actuator.

With a classical approach, this object would be located at the gateway level as shown on 0 with the first object.

With a smart sensor approach, we have decomposed the set of bags in order to let some of the bags at the gateway level and to move the others at the micro-controller side.

`Sensors` and `TimeStamp` bags which contain respectively the value associated to a sensor and the timestamp corresponding to the last sensed value are implemented on the micro-controller side. All the other bags are implemented at the gateway side. The main reason is these bags contain information that

either records the sensed value or store some status updated very sparsely. Thus, in order to save memory space and power-on time on smart sensor we keep these bags on the gateway level and we implement the bags which makes sense at the micro-controller side. For instance, if we need to access the log of the successive values read by a temperature sensor, it is not required to ask the micro-controller itself, this can be done at the gateway level. On the contrary, the current temperature needs to be asked to the micro-controller.

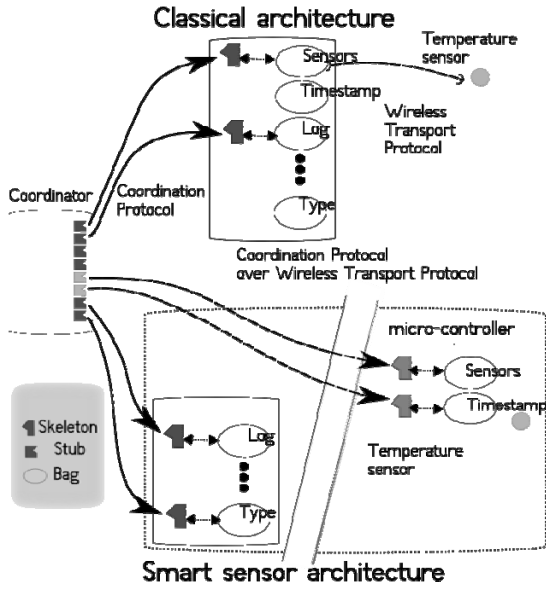


Fig. 2. Sensor network object: the smart sensor approach.

For an actuator, we implement the bag which is responsible for the real action on the environment on the micro-controller side since we want to ensure the transactional property. Some verifications are done during the `pre_put()` operation. For instance we can verify that the rotation we would like to do on a motor is possible without damage. We could also verify that the command is sensible: to ask to close a window that is already close is typically something that is not normal. We can also verify that the actuator is physically able to do the requested command or that the remaining energy is enough to perform the action. In case of trouble, a `notok` is returned.

4. Architecture Approach: Hardware

This section presents the hardware details of the different smart sensors we implemented.

4.1. The Platforms

We have considered three embedded platforms to experiment different levels of performance at the microcontroler level and different communication standards: *OpenPicus Flyport (Wifi-802.11)* [7], *Atmel SAM3N-EK (802.15.4)* [9] and *Arduino Pro 3.3V (802.15.4)* [11].

These three architectures are widely used for their low power and low cost characteristics. They all three offer easy-to-use development environments avoiding most of cross-compilation problems and micro-controller driver development part.

OpenPicus Flyport (Wi-Fi): This platform is based on a Microchip MRF24WB0MA/RM Wi-Fi chip and a Microchip PIC 24F 16bits processor. The board embeds a small web-server, running on top of FreeRTOS operating system. This web-server allows to encapsulate remote function call through simple URLs over HTTP protocol with a mechanism closed to cgi-scripts. This is very similar to what is done in our middleware for the default transport layer within the stub. This allowed us a straight forward implementation of the smart sensor. This is one of the reasons behind the choice of this board.

For the integration of physical sensors/actuators, the board offers eighteen digital I/Os and four analog inputs. Digital pins can be mapped between one I2C, one SPI, four UARTs and nine PWMs. Three different external interrupts can wake up the Flyport from standby mode.

The associated communication link is a classical Wi-Fi which allow a very easy integration in existing installation with a range that may cover the entire floor of a building.

Atmel SAM3 (802.15.4): We have used a SAM3N-EK board that was available in our laboratory with a Cortex-M3 32bits based micro-controller. This board offered a convenient experimentation framework and as the Atmel environment allows, with a same API, to target every ARM and AVR micro-controller belonging the Atmel family the solution is quite generic. This is the reason why we used this platform.

The ARM Cortex-M3 core is used by many constructors, including ST Microelectronic, Atmel and Texas Instrument, to build low power but still powerful chips.

The board offers 64 I/Os including PWMs, I2Cs, SPIs, UARTs and ADCs. Some I/Os are connected at on-board LCD, buzzer, SD card host, user switch, RS232 connector or touch sensors. Sixteen different external interrupts can wake up SAM3N-EK from standby mode.

We used for the communication Digi-XBee series one modules[Xbee], which provides IEEE 802.15.4 wireless networking protocol. A module is connected to the board via one UART and another is plugged on the gateway via an FTDI breakout board.

We use them in peer-to-peer mode. The data rate is *250 Kbit/*s on the *2.4 GHz* RF band. The maximum frame length is one hundred bytes. This is less than the Wi-Fi solution but with an adhoc encoding it is

efficient enough to allow serialized procedure calls to hold into a single frame.

Arduino (802.15.4): The Arduino board we used is based on Atmel AVR 8bits architectures. It is open source and supported by a large community. This is the reason that conducted us to use this board.

We used the Pro 3.3 V version due to the reduced on-board electronics and a working voltage that corresponds directly to XBee modules we used for the communication. The XBee module is connected to the micro-controller via the UART interface like for the Atmel board.

The board offers fourteen digital I/Os and six analog inputs. Digital pins support one UART, six PWMs and one SPI. Analogue pin could also be used for adding one I2C interface. Two different external interrupts can wake-up Arduino from standby mode.

4.2. Power Consideration

Boards: Power consumption of considered micro-controllers is under *10 µA* at *3.3 V* in standby-mode, while it is at least *5 mA* in run mode.

However, in the case of the SAM3-EK Atmel board, the complete board is a development board for which at least one electronic device was not designed for low power usage. Thus, we did not considered this board for the power consumption experiment.

On the contrary, openPicus Flyport and Arduino Pro 3.3 V boards are designed for low power, in particular in standby-mode. The respective consumption measured with a full charge lithium-ion battery (*4.1 V*) with wireless unit in the same state than the micro-controller are summarized in 0.

Table 1. Consumption of the full board.

Microcontroler/wireless	Standby	Run mode
Arduino / Xbee	206 µA	51.1 mA
OpenPicus Flyport/WiFi	97 µA	127.5 mA

Wireless communication unit: XBee (802.15.4) [13] and WiFi (802.11) have been considered. They both offer power down mode to control their power state from the micro-controller. Respective consumptions are in 0.

Table 2. Consumption of the wireless communication unit.

Wireless module	Standby	Communication
Xbee	10 µA	50 mA
WiFi	0.1 µA	120 mA

The first is more expensive in standby mode but can be integrated to a larger number of micro-controllers.

The second is more expensive in communication mode but it handles a bigger amount of data with a higher communication range.

Radio control: Primitives `radio wake up` and `radio go sleep` are called during the board `power down` and `power up` procedures. In this way, when the micro-controllers are in run mode, the wireless communication link is available. We could have tested smarter radio power control but since our goal was to have the board in sleeping mode most of the time we did not go further in this direction.

Sleep mode and wake up events: As in sleeping mode, the consumption is divided by at least one hundred, our approach is to force the sensor to be in sleeping mode almost all the time. It is wake-up on demand either because the coordinator sends a signal before invoking the appropriate primitive or because an alarm is triggered by a physical sensor linked to the board. Both events raise a different interrupt received by the micro-controller and thus they are treated accordingly in order to execute the appropriate code (See Section 0).

Wake up signal: For the wake up signal sent by the coordinator (gateway side) it is an open problem out the scope of this paper that would require further investigation. Then, we only propose here some solutions that could be used.

The first solution is based on a simple modulated IR signal that can be used if the gateway is in the same location than the sensor.

The second is based on a less expensive wireless signal [15] (e.g. *433 MHz*) that can be used to this purpose.

Finally, the third may consider the new passive RFID technology working in a range of 15 meters.

In our first experiment we only used the IR solution.

5. Example: Consumption Aspect

We first illustrate our approach with a very simple example. Secondly, we describe a practical scenario where our approach could bring a significant improvement.

5.1. Simple Example

We consider a simple application where we collect external temperature in order to control the heating system according to the variation of the temperature and the speed of this variation. Then, we have an algorithm that defines according to a window of sensed temperatures the most appropriate time to make the next set of measures. Basically, if the temperature does not change a lot we increase the

delay between two measures and we decrease it if the variation grows.
It is impossible to compute in advance the time when the measurements need to be done.

The classical way is to ask the sensors to send the information each δt and to sleep the rest of the time. With δt *=5 min*s, we have *24*12* measures sent by each sensor to the gateway per day. Most of them will not be used at all and for some of them we have inaccuracy due to the acquisition rate that does not match exactly the algorithm needs.

If we use a rule triggering the interrogation of the sensors upon the insertion of a resource by the algorithm, we first remove the inaccuracy and second we can decrease the activity of the sensors to the exact required number of measures.

For our purpose, we consider that the number of really useful measures is only *50* per day (*5* times less).

The average power consumption *PC* is given by

$$RC = R \times C_{RunMode} + (1-R) \times C_{S\tan dbyMode} \quad (1)$$

with *R* the run time ratio

$$R = \frac{Run\ \ Time}{Total\ \ Time}$$

We have measured that a run period (wake-up, send, and sleep) takes 0.04 seconds for the Arduino and 1 second for the Flyport.

For the Flyport, in the classical implementation (C), in one day we have *24 * 12* sequences of *1* second in run time mode. The sensor is sleeping the remaining part of the time. This gives the following ratio:

$$R_c = \frac{RunTime}{TotalTime} = \frac{24 \times 12 \times 1}{24 \times 60 \times 60} = 0.333\% = 333 \cdot 10^{-5}$$

In the smart sensor *(S)* implementations the time in run time mode is only *50* sequences of *1* second.

$$R_S = \frac{50 \times 1}{24 \times 60 \times 60} = 0.0578\% = 57.8 \cdot 10^{-5}$$

Reported to equation (1) we obtain with the figures of 0 the following consumption for the two:

$$C_C = 333 \cdot 10^{-5} \times 127.5mA + \\ + (1 - 333 \cdot 10^{-5}) \times 0.091mA = 0.516mA$$

$$C_S = 57.8 \cdot 10^{-5} \times 127{,}5mA + \\ + (1 - 57.8 \cdot 10^{-5}) \times 0.091mA = 0.165mA$$

For the Arduino, the run time last only 0.04 seconds.

$$R_C = \frac{24 \times 12 \times 0.04}{24 \times 60 \times 60} = 0.0133\% = 133 \cdot 10^{-6}$$

$$R_S = \frac{50 \times 0.04}{24 \times 60 \times 60} = 0.00231\% = 23.1 \cdot 10^{-6}$$

This gives the following results for the consumption with the two implementations.

$$C_C = 133 \cdot 10^{-6} \times 57.1mA + \\ + (1 - 133 \cdot 10^{-6}) \times 0.206mA = 0.02135mA$$

$$C_S = 23.1 \cdot 10^{-6} \times 57.1mA + \\ + (1 - 23.1 \cdot 10^{-6}) \times 0.206mA = 0.2073mA$$

If we consider a battery capacity *Cap*, the following formulae gives the saved time in hour.

$$H = \frac{C_{ap}}{C_S} - \frac{C_{ap}}{C_C}$$

With a capacity C_{ap}*=1300 mAh* for the battery we save *H=7892-2521 = 5371 hours* i.e. 244 days for the Flyport and *H = 6270 - 6087 = 183 hours* i.e. 8 days for the Arduino.

The battery, with smart sensor implementation, allows respectively 328 days for the Flyport and 261 days for the Arduino as summarized in the 0.

Table 3. Battery autonomy for both approaches.

Micro-controler/wireless	Classical	Smart
Arduino / Xbee	253 days	261 days
OpenPicus Flyport/WiFi	105 days	328 days

With the classical approach the Arduino is far better thanks to its very low consumption in running mode. However, with the smart sensor approach the Flyport is far better than the Arduino thanks to its very low consumption in standby.

As a conclusion, the Flyport board in the smart sensors implementation offers 20 % more autonomy to the application. The interesting result is that solving the power consumption issue only acting on the consumption of the wireless communication unit is probably not the unique research direction. This means also that using a simpler to deploy communication protocol (i.e. WiFi) is affordable if the sensors are used in a smarter way.

5.2. Practical Scenario

We now consider a flood monitoring systems based on sensors that are positioned along the banks of a river. With such sensors, the consumption is a crucial point and this scenario is a perfect candidate for our approach. We consider the installation depicted in the 0.

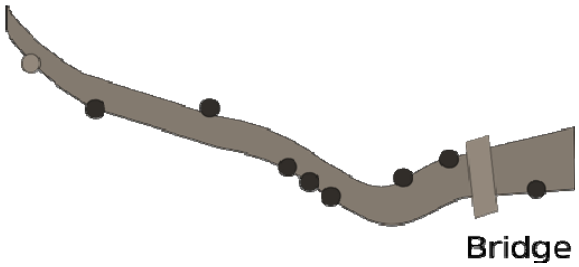


Fig. 3. Location and type of sensors.

The first sensor (in grey in the figure) is located upstream and embeds a mechanical mechanism able to generate a signal when the latter is touched by the water. The mechanism consists of a compressed spring that is capable when released to generate enough energy to deliver a signal. Once compressed the spring is trapped in a water-soluble matrix. When the matrix is altered by the water, a signal is emitted. This sensor can be used only once but it is perfect for our scenario.

The other sensors (in black in the figure) are sleeping waiting for the wake up signal discussed in section 0 they are distributed along the bank at key positions.

These sensors are managed by coordination rules in order to help people to take the right decision in case of sinister. A simplified example of the rules is given in the 0.

```
   [¨TriggerSensors¨].rd(“sensor_area_33”,
”alert”) &
   [¨Clock¨].rd(ticket,timestamp) &
   [¨Sensors¨].rd(“id_1”,val1) &
   [¨Sensors¨].rd(“id_2”,val2) &
   [¨Sensors¨].rd(“id_3”,val3) &
   COMPUTE sid = algo.extra_sensor(“area_33”,
val1, val2, val3) &
   [¨Sensors¨].rd(sid,val4) &
   COMPUTE criticalRanking =
algo.get_ranking(“area_33”, val2, val4)
   ::
   {
   [¨Clock¨].get(ticket,timestamp) &
   [¨Database¨].put(timestamp,”id_1”,val1) ;
   [¨Database¨].put(timestamp,”id_2”,val2) ;
   [¨Database¨].put(timestamp,”id_3”,val3) ;
   [¨Database¨].put(timestamp,”id_4”,val4) ;
   [¨Supervisor¨].put(timestamp,criticalRanki
ng) &
   }.
```

Fig. 4. Example of coordination rule.

In normal situation the bag `TriggerSensors` is empty and the rule is blocked waiting for something to happen. This means that the different `rd()` operations in the bag `Sensors` are not yet done and the sensors are all in sleeping mode with a minimal consumption. Thus without sinister, the system can stay in this configuration and last several years.

As soon as the first sensor sent its signal a resource is present in the bag `TriggerSensors` with the identification of the area in order to use the appropriate corresponding settings to feed the situation evaluation algorithm. This resource will stay in the bag because it is never consumed in the performance phase. Thus, it will be available to all the instances of the rules generated all along the sinister management. The presence of a resource in the bag `Clock` allows on the one end to have temporal information (`timestamp`) and to pace the execution of the rule since we have an evaluation of the rule each time such a resource appears in the bag. This can be done for instance by the algorithm that can insert in the bag `Clock` resources at a rate depending on the current situation.

The lines 3-5 correspond to the `rd()` of some sensors located to the river bank. These reads include the wake-up signal to put the sensors normally sleeping in running mode for the duration of the measure.

The line 6 corresponds to a call to our algorithm that in the given example finds the most pertinent additional sensor to interrogate according to the current situation: e.g. past values stored in the historical database, the current values for our 3 sensors or the area.

The line 7 corresponds to the actual interrogation of this additional sensor.

Finally, in line 8 we call our algorithm with the values read (here from the sensor 2 and the additional sensor) to compute the evaluation of the situation.

In the performance phase lines 10-17, we consume the `ticket` resource stored in the `Clock` bag preventing the evaluation of other rules with the same ticket and thus controlling the execution pace. In addition in the lines 12-15 the values read from the sensors are stored in the historical database and the global `criticalRanking` information evaluating the situation is sent to the supervisor team along with the `timestamp`.

With this simple rule we show the flexibility of our rule based language and we just sketch what it is possible to do. In a real scenario we would consider several rules and a more complex usage or the values read by the sensors but the principles would remain the same.

On an energy saving point of view, we can see that the system uses minimal energy when no sinister is detected.

6. Example: Transactional Aspect

As for the previous section we first illustrate our approach with a very simple example, and then we provide a more elaborated scenario based on the same principle.

6.1. Simple Example

We consider 2 servo-motors controlled respectively by the Arduino and the Flyport. Each of them can operate on 180 degrees. (see 0). They are configured to accept orders only in their respective functioning range: 0-180 for the Flyport, and 180-360 for the Arduino. So only their combination may offer a full 360 angle

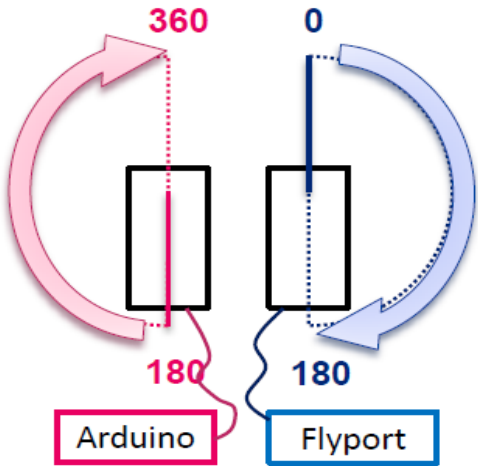


Fig. 5. Combination of 2 servo-motors.

0 presents the rule controlling the system.

```
   [“Application”, “Angle”].rd(angle) &
   ::
   {
   [“Application”, “Angle”].get(angle) ;
   [“Flyport”,    “Actuator”].put(“position”,
angle) ;
   [“Arduino”,    “Actuator”].put(“position”,
“180”) ;
   }
   {
   [“Application”, “Angle”].get(angle) ;
   [“Flyport”,    “Actuator”].put(”position”,
“180”) ;
   [“Arduino”,    “Actuator”].put(“position”,
angle) ;
   }.
```

Fig. 6. Coordination rule controlling the 2 smart actuators.

The precondition phase is reduced to a single `rd()`whose role is to obtain the angle to rotate. The performance phase is composed of 2 transactions that are triggered sequentially. The first one (lines 3-7) fails if the requested `angle` passed to the Flyport (line 5) is not in the interval [0:180], the second one (lines 8-12) fails if the requested `angle` passed to the Arduino (line 11) is not in the interval [180:360].

Then, according to the angle, only one of the two transactions is committed solving directly the problem. In addition, even if it was redundant here, if the first transaction commits, the resource `angle` is consumed and the second transaction that is competing for this resource would anyway fail. Inserting a new angle in the bag `[“Application”, “Angle”]` triggers another time the rule that acts on the servo-motors. Obviously, this is a toy example but the principle may be used in the following practical scenario.

6.2. Practical Scenario

We consider now a mobile robot equipped with a pan-tilt camera and we want to follow a moving object. The robot is managed by a smart board containing actuators for the wheel motors and sensors for detecting obstacles. The camera is also controlled by a smart board that is configured to respect the rotation physically allowed by the pan-tilt mechanism.

We have an external software component interrogated in the precondition phase that computes from a captured image the next positions of the robot and the camera in order to have the object centered. The result is then a list of possibilities involving only the robot, only the camera or some combinations of the both. This is used in the performance phase to define the sequence of transactions to be tried. For the robot, a transaction may fail if the new position is incompatible with an obstacle detected by one of the sensors. For the camera a transaction may fail because it would require an impossible rotation on one of the two axes (pan or tilt). In the same way as for our toy example we can control the described behavior with a single rule.

Another solution could be to have different rules competing for the `instruction` given by our software component. Each rule would have a single transaction considering one of the alternatives. This solution would be more efficient since all the possible transactions are tried in parallel and the first one to commit prevents all the others to commit by consuming the resource `instruction`. However, this solution does not allow giving a preference when several possibilities are acceptable.

Finally, a combination of several rules with several transactions allows tuning the two parameters: response time and preference.

Obviously, nothing prevent to consider a more complex scenario taking other parameters into account such as the robot speed, the distance to the object, etc.

7. Conclusions

We have presented an approach to improve power consumption and reliability in the wireless sensor network area. This approach is based on a high level rule-based middleware that coordinates the operations involving the sensors and actuators. This offers the possibility to activate the sensors with a signal (external to the communication) when they are really needed and thus let them most of the time in standby mode to reduce power consumption.

In addition the coordination protocol allows embedding actions on sensors and actuators within transactions to improve reliability and to detect directly at the actuator level that a requested action is physically impossible. This simplifies drastically the definition of complex scenario needing alternatives to reach the final goal according to an unknown or fast evolving context.

By imposing discipline to the sensors and actuators behaviors we improve the global smartness of the system.

We have implemented three different smart sensor boards able to understand the coordination protocol of our rule based middleware and we have tested them in different toy applications which put emphasis on the particular advantages of our approach.

Finally, we have described two more ambitious scenarios which use the same principles than the one illustrated in our toy examples to show the usefulness of our approach.

Acknowledgements

This work has been partially funded by the FP7 SCUBA project under grant nb 288079.

References

[1]. H. Iris and F. Pacull, Protocol Awareness: A Step Towards Smarter Sensors, in *Proceeding of the 3rd International Conference on Sensor Device Technologies and Applications (SENSORDEVICES 2012)*, 19 - 24 August 2012, Rome, Italy, pp.148-153.

[2]. L.-F. Ducreux, C. Guyon-Gardeux, S. Lesecq, F. Pacull, and S. R. Thior, Resource-based middleware in the context of heterogeneous building automation systems, in *Proceeding of the 38th IEEE Annual Conference of the Industrial Electronics Society (IECON' 2012)*, 25-28 October 2012, pp. 4847 - 4852.

[3]. P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency control and recovery in database systems, *BostonAddison-Wesley Longman Publishing*, MA, USA, 1987.

[4]. S. Feizi, G. Angelopoulos, V. Goyal, and M. Medard, Energy-efficient time-stampless adaptive nonuniform sampling, in *Sensors, 2011 IEEE*, October 2011, pp. 912–915.

[5]. N. Carriero and D. Gelernter, Linda in context, *Communication of ACM*, Vol. 32, April 1989, pp. 444–458.

[6]. T. A. Cooper and N. Wogrin, Rule Based Programming with OPS5, *Morgan Kaufmann*, July 1988.

[7]. OpenPicus Flyport Documentation Repository.

[8]. http://http://www.openpicus.com/site/downloads/downloads, 2012.

[9]. SAM3N-EK Documentation Repository.

[10]. http://www.atmel.com/tools/SAM3N-EK.aspx?tab=overview, 2012.

[11]. Arduino board Documentation Repository.

[12]. http://arduino.cc/en/Main/ArduinoBoardPro, 2012.

[13]. XBee-PRO 802.15.4 OEM RF Modules.

[14]. http://http://www.digi.com/xbee, 2012.

[15]. S. J. Marinkovic and E. M. Popovici, Nano-power wireless wake-up receiver with serial peripheral interface., *IEEE Journal on Selected Areas in Communications*, Vol. 29, No. 8, 2011, pp. 1641–1647.

Conference Announcement

Topic E2: Transportation & Mobility

The Euromat conference series, organised by the Federation of European Materials Societies (FEMS), is one of the largest events of its kind in Europe, covering the full width of materials science and technology. We would like to direct your attention to the following Symposia which are focussing specifically on transport applications:

E2.I: **Modeling, simulation, optimization of materials and structures in transportation**
Prof. Kambiz Kayvantash, Société CADLM, Massy (F)

E2.II: **Intelligent and adaptive materials and structures**
Dr.-Ing. Dirk Lehmhus, ISIS Sensorial Materials Scientific Centre, Bremen (D)

E2.III: **Energy absorbing and protective materials and structures**
Prof. Massimiliano Avalle, Politecnico di Torino, Torino (I)

E2.IV: **Production, properties and applications of hybrid materials and structures**
Dr.-Ing. Kai Schimanski, Foundation Institut für Werkstofftechnik (IWT), Bremen (D)

DEADLINE CALL FOR PAPERS END OF JANUARY – WATCH OUT FOR DETAILS AT www.euromat2013.fems.eu OR CONTACT

ISIS Sensorial Materials Scientific Centre, University of Bremen

Board of Directors
Prof. Dr.-Ing M. Busse
Prof. Dr. W. Lang
Prof. Dr.-Ing. H.-W. Zoch

Managing Director
Dr.-Ing. Dirk Lehmhus

Wiener Straße 12
28357 Bremen

Fon +49 (0)421 5665 408
Fax +49 (0)421 5665 499

dirk.lehmhus@uni-bremen.de
www.isis.uni-bremen.de

Aims and Scope

Sensors & Transducers is a peer reviewed international, interdisciplinary journal that provides an advanced forum for the science and technology of physical, chemical sensors and biosensors. It publishes original research articles, timely state-of-the-art reviews and application specific articles with the following devices areas:

- Physical, chemical and biosensors;
- Digital, frequency, period, duty-cycle, time interval, PWM, pulse number output sensors and transducers;
- Theory, principles, effects, design, standardization and modeling;
- Smart sensors and systems;
- Sensor instrumentation;
- Virtual instruments;
- Sensors interfaces, buses and networks;
- Signal processing and interfacing;
- Frequency (period, duty-cycle)-to-code converters, ADC;
- Technologies and materials;
- Nanosensors;
- Microsystems;
- Applications.

Further information on this journal is available from the Publisher's web site:
http://www.sensorsportal.com/HTML/DIGEST/Submission.htm

Subscriptions

An annual subscription includes 12 regular issues and some special issues. Annual subscription rates for 2013 are the following:

Electronic version (in printable pdf format): 400.00 EUR
Printed with b/w illustrations: 640.00 EUR
Printed full color version: 760.00 EUR

40 % discount is available for IFSA Members.

Prices include shipping costs by mail. Further information about subscription is available through IFSA Publishing's web site: http://www.sensorsportal.com/HTML/DIGEST/Journal_Subscription.htm

Advertising Information

If you are interested in advertising or other commercial opportunities please e-mail sales@sensorsportal.com and your enquiry will be passed to the correct person who will respond to you within 24 hours. Please download also our Media Planner 2013: http://www.sensorsportal.com/DOWNLOADS/Media_Kit_2012.pdf

Books for Review

Publications should be sent to the IFSA Publishing Office: Ronda de Ramon Otero Pedrayo, 42C, 1-5, 08860, Castelldefels, Barcelona, Spain.

Abstracting Services

This journal is cited, indexed and abstracted by Chemical Abstracts, EBSCO Publishing, IndexCopernicus Journals Master List, ProQuest Science Journals, CAS Source Index (CASSI), Ulrich's Periodicals Directory, Scirus, Google Scholar, etc. Since 2011 *Sensors & Transducers* journal is covered and indexed (including a Scopus, Embase, Engineering Village and Reaxys) in *Elsevier* products.

Instructions for Authors

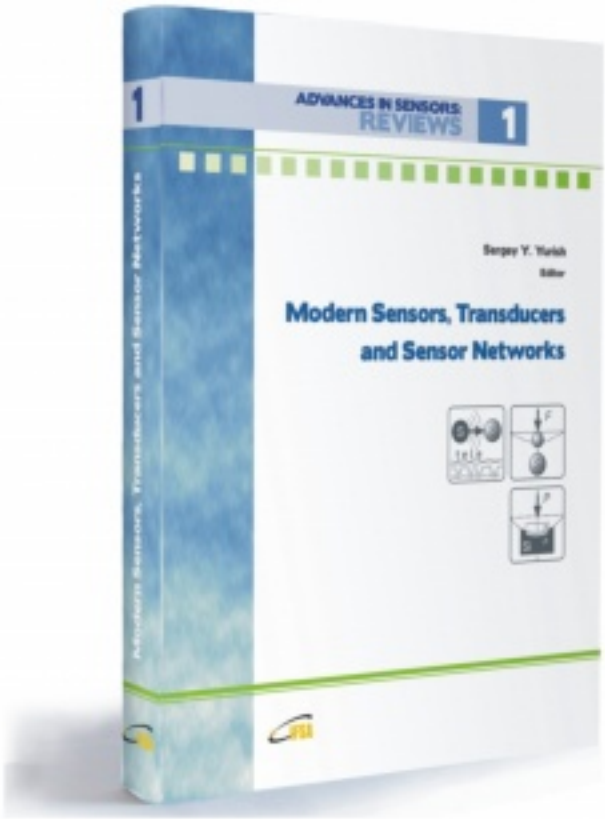
Please visit the journal web page http://www.sensorsportal.com/HTML/DIGEST/Submission.htm Authors must follow the instructions very carefully when submitting their manuscripts. Manuscript must be send electronically in both: MS Word 2003 for Windows (doc) and Acrobat (pdf) formats by e-mail: editor@sensorsportal.com

International Frequency Sensor Association Publishing

ADVANCES IN SENSORS: REVIEWS 1

Sergey Y. Yurish
Editor

Modern Sensors, Transducers and Sensor Networks



Modern Sensors, Transducers and Sensor Networks is the first book from the Advances in Sensors: Reviews book Series contains dozen collected sensor related state-of-the-art reviews written by 31 internationally recognized experts from academia and industry.

Built upon the series Advances in Sensors: Reviews - a premier sensor review source, the *Modern Sensors, Transducers and Sensor Networks* presents an overview of highlights in the field. Coverage includes current developments in sensing nanomaterials, technologies, MEMS sensor design, synthesis, modeling and applications of sensors, transducers and wireless sensor networks, signal detection and advanced signal processing, as well as new sensing principles and methods of measurements.

Modern Sensors, Transducers and Sensor Networks is intended for anyone who wants to cover a comprehensive range of topics in the field of sensors paradigms and developments. It provides guidance for technology solution developers from academia, research institutions, and industry, providing them with a broader perspective of sensor science and industry.

Order online:
http://sensorsportal.com/HTML/BOOKSTORE/Advance_in_Sensors.htm

www.sensorsportal.com