# A Service-Oriented Architecture for QoS Configuration and Management of Wireless Sensor Networks*

Gaetano F. Anastasi, Enrico Bini, Antonio Romano, Giuseppe Lipari
Scuola Superiore Sant'Anna
Pisa, Italy
{g.anastasi, e.bini, a.romano, g.lipari}@sssup.it

## Abstract

*Software infrastructures for networked enterprises may need data coming from low-level pervasive devices, such as Wireless Sensor Networks (WSNs). However, the complex management of such tiny physical devices is not acceptable for high-level enterprise applications. Hence the need for a middleware layer that hides complexity and supports the management of heterogeneous real-time data coming from the environment. In our opinion, the Service Oriented Architecture (SOA) design paradigm is the most suitable for allowing a seamless and effective integration of pervasive technologies into enterprise information systems. In this paper we present a service-oriented, flexible and adaptable middleware that allows applications to configure WSN functionalities and exploit them in the form of Web Services.*

## 1. Introduction

Networked enterprises are heavily based on systems that leverage communication and information technologies to support business processes and strategic decisions. This kind of support originates from data that, not only constitute the building block of the analysis of processes but are also fundamental in driving the execution of enterprise tasks. Data must be collected, filtered, merged, and finally made available to decision-makers (or to trusted users) through the enterprise information system.

Among the various sources of data, the physical environment can be considered as one of the most common. In fact, the production processes strictly interact with the environment, and these interactions can have a significant impact on the quality of the final product, both directly, in case of outdoor production (e.g. agriculture, environmental protection, vehicular traffic monitoring), and indirectly, by affecting the correct functioning of the factory plant (e.g. in factory automation, monitoring and control).

*This work has been supported by the ART DECO research project, funded by the Italian Minister of University, project code n. RBNE05C3AH.

Nowadays, many low-cost technologies exist that permit the collection of data from the physical world, such as Wireless Sensor Network (WSN) or Radio Frequency IDentification (RFID). Considering a recent trend (for example work by Zhang and Wang [15]) that goes towards the integration of RFIDs into WSNs just treating them as peculiar kind of sensors, for the sake of simplicity in this paper we will focus on WSN as the reference technology for the data-gathering level. WSNs are characterized by some distinctive features (like the small size and the scarcity of energy and computational resources) that make them strictly bound to hardware components and/or embedded operating systems. Thus it is difficult to integrate them into enterprise information systems.

One of the main problems is that performance control and management of the *quality of service* (QoS) of the results are obtained by manual ad-hoc programming and configuration. For example, most WSN devices are powered by batteries, and it is therefore important both to minimize their energy consumption, and to monitor and estimate their lifetime. The power consumption depends on the rate at which the data is sampled and sent via radio. Therefore, the final user may want to control and trade off sampling frequency against device lifetime. In addition, for some application it may be important to change the monitored data and area during the system lifetime. However, every time one of the parameters of the monitoring application has to be changed, it is necessary to access the device with its own interface and reprogram it. To simplify integration with higher level software layers, an abstract interface of the WSN is needed, in order to hide the low-level details while maintaining full control over the management of WSN applications.

In our opinion, a general solution to this challenging task passes through the adoption of the Service Oriented Architecture (SOA) design methodology for abstracting the data-gathering level, as it permits to build flexible and interoperable systems in which pervasive technologies can be integrated in a seamless way for assuring both intra-organizational and inter-organizational cooperation and collaboration.

In this paper we propose a service-oriented, flexible and adaptable middleware that allows high-level applica-

tions to easily configure the data-gathering level and exploit provided functionalities in an effortless manner. In the remainder of the paper, related work is briefly analyzed in Section 2, whilst Section 3 describes the architecture of the proposed middleware for WSN. Section 4 details a case study that has been built to show the effectiveness of the proposed solution, and finally Section 5 draws conclusions.

## 2. Related Work

The main goal of the proposed middleware is the effective and seamless integration of pervasive technologies into the information system of networked enterprises. This issue has already been tackled in the literature, for example by Samaras et al. [11] and by Delicato et al. [2]). However, those two proposals are aimed at implementing a service-oriented middleware directly on sensor nodes, by forcing SOA-compatible protocol stacks (like DPWS [9]) in resource constrained devices.

In our opinion, this approach has the major drawback of imposing too much complexity in devices that are not enough powerful to transmit and elaborate XML messages. To overcome such constraints, authors propose to adopt a-priori knowledge in XML message definition, thus losing middleware flexibility. Moreover, the usage of web services in resource constrained devices imposes a certain energy and latency overhead (as an example, cost for such implementations has been quantified in the work by Priyantha et al. [10]) that could be unacceptable in some cases.

By contrast, our approach concentrates the logic that abstracts the WSN on a powerful gateway, to which the sink node is connected. Such solution is not new, for example it has been used by Gil-Martinez-Abarca et al. [3] for enabling management of remote bootstrap of network nodes through the Internet; and by Kansal et al. [4] for building a peer-to-peer infrastructure for sharing sensors through the Internet. However, such works address a wide area of pertinence and they do not explicitly address typical WSN issues, like the energy management of nodes and the QoS support for applications.

A gateway-based solution has been also proposed by Moeller and Sleman [8], aiming at integrating WSNs into other existing IP-based networks. However, as their work is addressed to the ambient intelligence at home, they only abstract functionalities of single sensors, i.e. applications are aware of the network deployment and request services directly to a node; our approach instead abstracts functionalities of the whole network, i.e. applications request services for a geographic area without the need to know how many nodes are deployed there or how they communicate each other.

Moreover, a common criticism of previous mentioned works is that they do not allow applications to reconfigure WSNs according to their needs and are not flexible with respect to existing network protocols. In supporting these features, our approach has some similarities with MiLAN [5], a middleware that allows applications to specify their QoS requirements and configure the network to maximize the application lifetime while providing the required QoS level. However, in MiLAN applications specify their requirements by means of graphs that have to be specialized for each particular sensing scenario. Moreover, data directly flow from each single node to applications, and thus a-posteriori treatments of data cannot be exploited for transparently addressing different application requirements related to same nodes. For these reasons, this approach is less suitable for ensuring integration and interoperability.

We instead allow applications to specify their requirements in a standardized way, by means of Service Level Agreements (SLAs) that each application can independently negotiate at run-time, in such a way that an application does not need to know the QoS requirements of other applications. In addition, our architecture allows applications to exploit gathered data by means of Web Services technologies, both for ensuring flexibility in data delivery and guaranteeing integration and interoperability.

It is worth to note that our approach completely differs from that of querying systems like TinyDB [6]. In fact, such systems permits to extract data from a WSN but they do not generally provide high-level interfaces for QoS configuration and management. Moreover, such systems usually exploit low-level techniques for gathering data and can thus be considered as tight extensions of a particular WSN technology. For this reason, they could in turn be used for developing a WSN whose configuration and management are provided by our architecture, that is, as explained in the next section, independent by design of the underlying WSN technology.

## 3. The Middleware

This section describes SensorsMW, a service-oriented, flexible and adaptable middleware for WSNs, which has been designed and developed in the context of the ARTDECO research project (`http://artdeco.elet.polimi.it/`). SensorsMW is **service-oriented** in abstracting WSNs as a collection of services, permitting a fruitfully exploitation of pervasive technologies in enterprise contexts; it is **flexible** as it can be used in many contexts or domains, even when specific network issues have to be addressed; it is **adaptable** in permitting the reuse of well-known low-level techniques and supporting legacy deployments that can be already in-place.

The proposed middleware has been designed keeping in mind the main issues of this domain [14, 7], as highlighted by its key features, that can be summarized as follows:

- it supports **QoS specification and management** by using a contract negotiation scheme based on Service Level Agreements (SLAs). As an example, it permits an easy access to network-provided data with different time and space granularity; it supports time

and space recognition of network events; it provides both periodic data sampling and event-driven notifications.

- it allows applications to **reconfigure and maintain** the network during its lifetime. As an example, the middleware supports fault detection management by signaling when the number of active devices in a certain area goes below a certain threshold specified in a special contract. Other contracts permits the energy monitoring, by treating it as a special case of generic data monitoring. Moreover, it is possible to implement energy-aware data collection and data fusion in the network itself to spare energy depending on the user requirements.

- it is **independent** of the underlying WSN technology. In fact it does not depend on the network size and topology and porting from one technology to a different one implies the porting of just two subcomponents (see the WSNGateway component). It is possible to transparently perform services within the network (e.g. data fusion and filtering) or in the gateway, depending on the services available in the low level WSN technology.

In this section, the proposed architecture is overviewed by describing its components in details. Moreover, the specifications of contracts that regulate provided services are presented, in order to illustrate the different capabilities of SensorsMW.

### 3.1. Architecture

SensorsMW abstracts a WSN and encapsulates network protocols by providing applications with a high-level interface. In order to provide the possibility to configure a WSN according to the QoS requirements of client applications, SensorsMW leverages WS-Agreement [1], a framework that allows service providers to easily create, manage and monitor Service Level Agreements (SLAs) that service providers and service consumers establish for defining characteristics of service provisioning.

The interaction scheme is depicted in Figure 1: SensorsMW acts as a service provider, as it provides services to client applications, that have to negotiate SLAs (also called agreements or contracts in this paper) before consuming services. Services are provided through a Web Services [12] interface, because it permits an effective integration and interoperability in enterprise systems by using XML as standard format of data exchange.

In Figure 2 we show the four main components of SensorsMW, which are now described in details.

#### ContractsCreator component

This component is responsible for interacting with client applications in all the operations that regard the creation and management of contracts. It comprises the following subcomponents.

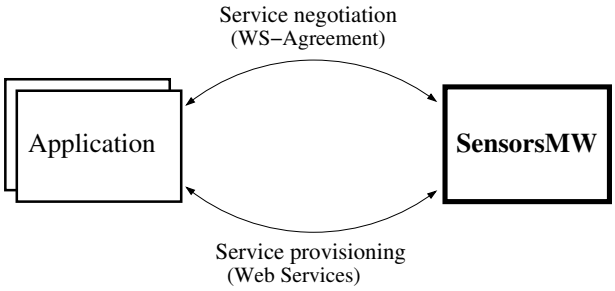


**Figure 1. SensorsMW interactions with client applications**

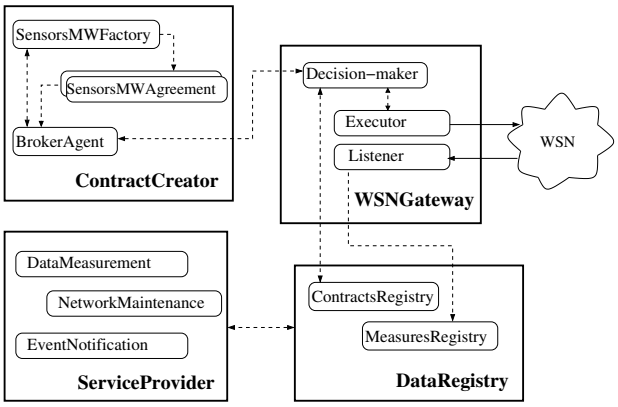


**Figure 2. SensorsMW architecture**

***SensorsMWFactory.*** It interacts with the clients in the agreement creation process and is responsible for publishing the agreement templates related to services provided by the system (see Section 3.2 for details about template specifications). The templates are filled by clients according to their requirements and then evaluated by the component. In case the client proposal can be satisfied, SensorsMWFactory interacts with SensorsMWAgreement for creating the agreement.

***SensorsMWAgreement.*** It implements all the operations related to an agreement, including agreement establishment and deletion, and status information provisioning. The agreement can be established only after an agreement creation process has been successfull completed by the SensorsMWFactory component. There will be an instance of SensorsMWAgreement for each contract that is actually in-place.

***BrokerAgent.*** It is responsible for forwarding requests of SensorsMWFactory and SensorsMWAgreement to the lower levels of the architecture. In particular, it forwards: *(a)* admission requests coming from the SensorsMWFactory when a new contract has to be admitted; *(b)* deletion requests coming from the SensorsMWAgreement when a contract has to be deleted. The introduction of this subcomponent improves the responsiveness of the SensorsMWFactory and SensorsMWAgreement subcomponents, that have to interact with clients, and also decouples

the ContractsCreator from the WSNGateway, that could be deployed in two different physical hosts.

**ServiceProvider component**

This component is responsible for providing services to client applications, in accordance with established contracts. In SensorsMW, three main services have been individuated as essentials: they exploit the database provided by the DataRegistry component for providing their functionalities, as described in the following.

***DataMeasurement.*** This service allows client applications to obtain currently gathered data, by just presenting the identifier of the established contract. As discussed in Section 3.2, the contract contains all the configuration parameters used by the WSN for gathering measurement data related to a certain physical quantity.

***EventNotification.*** This service allows client applications to receive notifications about events of interest, related to the measurement of a certain physical quantity. Applications can configure events they are interested in, and subscribe to them by means of specific contracts, that will be described in Section 3.2.

***NetworkMaintenance.*** This service allows client applications to perform network maintenance by measuring and monitoring quantities that are necessary for a proper WSN functioning, like the battery level or the number of active sensors in a certain region. Applications can exploit this SensorsMW features by establishing proper contracts, that will be described in Section 3.2.

**DataRegistry component**

This component is responsible for managing all those data that have to be persistently stored for the proper functioning of SensorsMW. It comprises the following subcomponents.

***ContractsRegistry.*** This subcomponent maintains the registry of all contracts currently established with client applications. Each contract is represented by an unique identifier plus the featuring parameters, that depend on the type of contract (see Section 3.2 for a detailed description of such parameters). The knowledge contained in this registry can be used for admitting new contracts and for providing applications with information regarding established contracts.

***MeasuresRegistry.*** This subcomponents maintains the registry of measures gathered by the WSN, in accordance with the presently established contracts. A measure is represented in the registry by the following parameters:

- the identifier of the measured physical quantity,
- the measure value,
- the datum aggregation type,
- the location in which the datum has been gathered,
- the time and date at which the datum has been gathered.

In order to provide applications with requested data, the knowledge contained in the MeasuresRegistry is leveraged by the ServiceProvider component, that contains the logic for binding data with contracts and for correlating data in order to respect established contracts: as an example, the ServiceProvider component may aggregate data *a-posteriori* if such in-network processing feature is not available in the WSN.

**WSNGateway component**

This component is responsible for acting as a gateway with respect to the WSN, in the sense that all the communications to and from the WSN pass through this component. It comprises the following subcomponents.

***Decision-maker.*** This subcomponent makes the scene when a new contract has to be admitted and it decides if a service requested by a client with a certain parameter configuration can be provided by the system. This includes both an analysis of existing contracts and of the current status of the WSN. When the component takes decision about a high-level request, it communicates the response to the BrokerAgent, that in turn forwards it to the SensorsMWFactory. If the response is negative, the Decision-maker does not take any further action; if positive, it triggers the creation of a new contract in the DataRegistry and interacts with the Executor for triggering tasks for the WSN, in order to fulfill new requirements of applications.

***Executor.*** This subcomponent receives commands from the Decision-maker and translates them into a language understandable by the sensor nodes. This level of indirection permits the independence of the admission control logic contained in the Decision-maker from the low-level technology used for the WSN programming, from whom the Executor is strictly dependent. However, it is worth to note that, to port SensorsMW to another WSN technology, the Executor and the Listener, described later, are the only subcomponents that need to be customized.

***Listener.*** This subcomponent receives data gathered from sensor nodes and stores them in the DataRegistry. It can be subdivided in two main modules (not highlighted in Figure 2): one is responsible for listening data communications from sensors and it is strictly dependent of the low-level WSN technology, the other one is responsible for binding data coming from nodes with respective locations, formatting measures as specified by the MeasuresRegistry and triggering storage.

### 3.2. Contract specification

The SensorsMW layer allows applications to configure the WSN according to their needs before service provisioning. In particular, for each kind of service, SensorsMW provides an agreement template that has to be

fulfilled by applications in order to create agreement proposals. If an agreement proposal is accepted, a contract is established with the client application, and both parties are obliged to honour it.

The agreement templates provided by the SensorsMW layer have been designed by keeping in mind the following principles: *(a)* being well-structured and easily usable by clients; *(b)* being compatible with limited hardware resources of sensor nodes (e.g. battery power).

For these reasons, templates are specified by using the WS-Agreement [1] framework and are characterized by a certain time span of validity $\Delta t$, that is based on an estimation of the remaining lifetime of nodes as a function of the current battery level and the requested sampling time. By assuming an exponential discharge of the battery, the voltage over time obeys to the law

$$V(t) = c(s)e^{\alpha(s)t} \tag{1}$$

with $c(s)$ and $\alpha(s)$ being coefficients depending on the sampling period $s$, whose values will be estimated in Section 4.1. Notice that $\alpha(s) < 0$ since the battery discharges over time. From Eq. (1), the validity interval of the contract can be computed as:

$$\Delta t = \frac{1}{\alpha(s)} \log \frac{V_{\min}}{V_0} \tag{2}$$

where $V_0$ is the current measure of the voltage and $V_{\min}$ is the minimum operative threshold for the node.

In SensorsMW, three different types of services have been individuated (see Section 3.1), whose execution parameters can be negotiated by means of templates. For this reason, three different kind of contracts have been specified, that can be summarized as follows:

1. *periodic measurement* contract, to periodically measure a certain physical quantity;
2. *event monitoring* contract, to monitor specific events related to quantity measurement;
3. *network management* contract, to control and maintain particular situations related to WSN functioning.

Each kind of contract is characterized by key parameters, that will be described in the following.

### *Periodic measurement* contract

This contract allows applications to periodically measure a certain physical quantity by specifying some parameters that characterize the WSN functioning during service provisioning. It is characterized by the following parameters:

- the physical quantity to be measured
- the time span for the measurement
- the sampling time of measure
- the data aggregation mode
- the region of interest
- the QoS level

The SensorsMW layer allows applications to negotiate such parameters by formally describing them through XML Schema [13] elements, that are inserted in a `ServiceDescriptionTerms` section of an Agreement Template. A single SDT can refer to only one physical quantity, as the various quantities measured by a WSN can have very different features from one each other. A possible SDT for an agreement template related to a periodic measurement service can be the following.

```
<wsag:ServiceDescriptionTerm
       wsag:Name="temperature_measurement"
       wsag:ServiceName="data_measurement">
  <smw:DataMeasurement>
    <smw:Measure>Temperature</smw:Measure>
    <smw:AggregationPeriod>
      PT1H10M
    </smw:AggregationPeriod>
    <smw:SamplingTime>PT10S</smw:SamplingTime>
    <smw:Aggregation>avg</smw:Aggregation>
    <smw:Region>
      <smw:Location>NorthArea</smw:Location>
      <smw:Location>SouthArea</smw:Location>
    </smw:Region>
    <smw:QoSLevel>100</smw:QoSLevel>
  </smw:DataMeasurement>
</wsag:ServiceDescriptionTerm>
```

Values are specified by using proper XML data types that are described as follows.

***Measure.*** It expresses the physical quantity to be measured as enumerate.

***AggregationPeriod.*** It expresses the time span for data measurement by using the *duration* XML data type.The example specifies a time span of 1h and 10min.

***SamplingTime.*** It expresses the sampling time of sensing by using the *duration* XML data type. In the example, data are sampled by sensors each 10 seconds.

***Aggregation.*** It expresses the aggregation mode of data collected in the same location, by using an enumerate data type (possible values could be `avg`, `max`, `min`).

***Region.*** The list of locations we are interested to monitor.

***Location.*** It expresses the location of interest by using an unique identifier.

***QoSLevel.*** It expresses the QoS level to be provided, by using values belonging to the set $\{x \in N\colon 0 \leq x \leq 100\}$. A QoS level equal to 100 is equivalent to the maximum quality of service.

Depending on the particular service configuration, some parameters could not be negotiated during the agreement phase. Other parameters are instead negotiable and their default values can be modified by applications when presenting an agreement proposal. In particular, in order to be adherent to WS-Agreement specification, the `CreationConstraints` template section must contain an `Item` element for each SDT parameter that can

be modified. By using the `Item` element, possible values for changeable parameters can also be specified, following the *restriction* model suggested by XML Schema. As an example, the `Aggregation` item can be limited to assume only the `min`, `max` and `avg` values.

The same mechanism can be leveraged to present the list of location to users, in such a way that different identifiers can be used from time to time to best depict the monitored area. The binding between sensor nodes and locations can be always modified by means of configuration files that associate a NodeID with a LocationID.

#### Event monitoring contract

The event monitoring contract allows applications to show interest in certain events and, in particular, to monitor specific events related to quantity measurement.

An event monitoring contract has same similarities with the periodic measurement contract, as highlighted by its key parameters, listed in the following:

- the physical quantity of interest
- the event triggering condition
- the event notification maximum delay
- the data aggregation mode
- the region of interest
- the QoS level

For specifying such parameters, this type of contract uses some of the XML data types already defined for the *periodic measurement* contract: they are `Measure`, `Aggregation`, `Region` and `QoSLevel`. In addition, two XML data types are defined for specifying triggering conditions and notification delays.

***MeasurementInterval.*** It expresses, through the `LowerBound` and `UpperBound` elements, the closed interval of the real line, whose values are interpreted according to the International System of Units (for temperature we consider Celsius temperature). As an example, the attainment of 20°C can be set as an event triggering condition by just specifying `20.0` in the `LowerBound` element and `INF` in the `UpperBound` element.

***NotificationDelay.*** It expresses the granted delay from an event occurrence to its signaling, by using the *duration* XML data type.

#### Network maintenance contract

The proper functioning of a WSN can be influenced by many events, like sensor node failures, battery discharges, node displacements or additions. For these reasons, this kind of contract allows applications to configure services for controlling and maintaining a WSN, in order to prevent dangerous events or take proper actions in case they happen.

SensorsMW gives the possibility to negotiate both measurement services and event-based services on critical quantities for the WSN maintenance, like energy consumption and the number of sensors in a certain region. In particular, the following key parameters has been individuated for measurement services: the quantity of interest, the time span for which the measurement has to be done, the region of interest. Instead, the parameters for event-based services are: the quantity of interest, the event triggering condition, the event notification delay, the region of interest.

The negotiation of network maintenance services is very similar to that described for periodic measurement services and event monitoring ones, as the number of sensors or the battery level can be treated as quantities to be measured in the network. As an example, it is possible to establish a contract for monitoring the number of active sensors on a region and triggering an event when it is behind a certain threshold by simply specifying `SensorNumber` in the field `Measure`.

## 4. Case study

In this section a case study is presented to show the effectiveness of the proposed architecture and demonstrate its adaptability and flexibility for building WSN middlewares that are QoS-aware and power-aware.

As a candidate application, we consider the temperature monitoring in a certain region of a wineyard and we study the behavior of a concrete implementation of SensorsMW, tailored to TinyOS 2.x and featured by a simple QoS admission control module. Moreover, a WSN testbed has been designed and deployed to be used in this case study. In the following, we first present such testbed, and then we detail our case study.

### 4.1. WSN testbed

For the sake of simplicity, the WSN testbed is characterized by a network star topology, in which a node acts as a coordinator and the other ones act as end-devices. The nodes are deployed all around the monitored area and have different tasks according to their category:

- The *Coordinator* node, connected to a resource-unconstrained machine, is responsible for interfacing the WSN with the higher level architecture through the WSNGateway. The coordinator receives data coming from the end-devices and forwards them to the WSNGateway. Also, it receives commands from the WSNGateway and forwards them to proper end-devices. In the current implementation, commands concern activation and deactivation of sensors on a node, and setting/changing of the sampling periods.

- The *End-Device* node is responsible for gathering data from active sensors and sending them to the coordinator node.

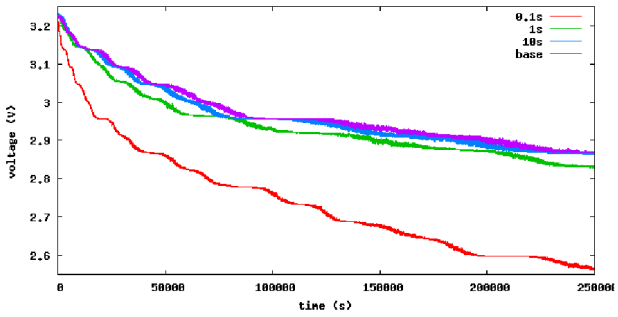


**Figure 3. Voltage as a function of time**

Every end-device node is a Crossbow MicaZ, featuring the ATMEL ATmega128 8-bit microcontroller; the Chipcon SmartRF CC2420 2.4 GHz IEEE 802.15.4 compliant RF transceiver; and the Crossbow Sensor Board MDA100CB, a sensor board that provides a precision thermistor, a light sensor/photocell and general prototyping area.

On every node, the TinyOS 2.0 embedded operating system runs along with the application program written in the NesC language. In particular, the Coordinator node runs the *BaseStation* TinyOS application to manage data packets coming from the network and send command packets to the network. Every end-device runs an application program composed by several TinyOS interfaces linked together to provide the following features:

1. dynamic adjustment of sensor sampling rates;
2. power saving through the asynchronous low power listening (LPL) strategy;
3. dynamic sensor activation/deactivation.

The first two features have been exploited for measuring the decrease of the battery level as a function of time. According to the LPL strategy the radio is switched off for a sleep interval to save energy. Then the radio is turned on to listen for possible incoming packets. If no packet is received the radio is switched off again. In our testbed we set a sleep interval of $500ms$.

In Figure 3 the top plot (labelled *base*) reports the voltage over time when no measurement is required to the node. The other plots, from top to bottom, report the voltage for increasing value of sampling rates ($10s$, $1s$, $0.1s$). All the experiments have been conducted for a duration of 250,000 seconds (almost three days).

For the purpose of this experiment, each node sends a packet for every sample and, as expected, the battery discharges faster with higher sampling rate. Starting from these results, in accordance to Eq. 1 we have derived a linear relationship of the sampling period $s$ and the coefficients $c$ and $\alpha$ as follows:

$$c(s) = 1.9288 + 0.0121\, s$$
$$\alpha(s) = -0.3505 + 0.0013 \cdot 10^{-6} s$$

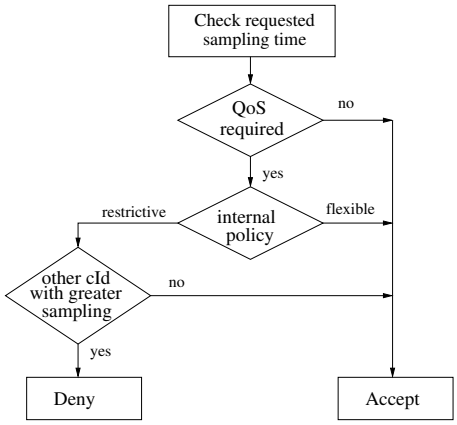


**Figure 4. Access control module**

### 4.2. Middleware testbed

For this case study, the proposed middleware architecture has been implemented for abstracting a TinyOS-based WSN, such as the one described in the previous subsection. For tailoring SensorsMW onto TinyOS, only the Listener and the Executor subcomponent (see Section 3) has been modified, by properly adapting the *Listen* and *Send* TinyOS applications.

Moreover, a simple access control module has been designed to be plugged into the SensorsMW architecture through the Decision-maker subcomponent. Such access control module behaves as follows (Figure 4 shows its flowchart).

When applications require QoS-enabled services, SensorsMW sets the sampling time of nodes in a certain location to the minimum value necessary to gather data at the exact instants of time, whilst in case of QoS-disabled services, provided data are interpolated basing on existing measures gathered at instants different than required. Two different policies (*flexible* and *restrictive*) are introduced, to differentiate the system behavior in accordance to the status of internal variables. In our implementation we use the battery level of nodes as the reference internal variable, since the battery level decreases faster when setting a shorter sampling time. Following this reasoning, the *restrictive* policy accepts a new contract only if the required sampling time is larger than the one actually set for selected locations, to prevent the new contract affects existing ones. The *flexible* policy instead accepts a new contract even when the required sampling period is smaller than actual ones and the sampling time in the related locations is set to the greatest common divisor of admitted sampling times.

As a possible scenario, assume that no contract has been stipulated and three client applications require a QoS-enabled DataMeasurement service related to the temperature monitoring, when the policy is *flexible*. Applications configure the service by creating, in different instants of time, contracts that differ for the parameters described in Table 4.2. As the applications require QoS-enabled services and the policy is set to flexible, the access

| Application | Agreement | Sampling (s) | Location |
|---|---|---|---|
| **1** | agr1 | 30 | loc1, loc3 |
| **2** | agr2 | 20 | loc2 |
| **3** | agr3 | 10 | loc1 |

**Table 1. Application requested parameters**

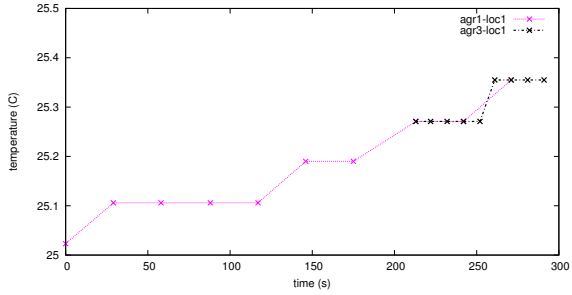


**Figure 5. Varying of temperature**

control module admits all the three applications. In particular, when application 3 requires a sampling of $10s$ for location `loc1`, the sampling time of nodes in that location is set to $10s$, in a manner that both application 1 and 3 can obtain data gathered at the required time instants. If at this time the system changes its internal policy to *restrictive*, the sampling time for nodes of location `loc1` will never be set to a lower value than $10s$.

The ServiceProvider component is responsible for providing data following the requirements of applications, as can be seen by the plot of figure 5, in which the temperature obtain by application 1 and 3 for location `loc1` is plotted as a function of time. This scenario highlights as SensorsMW can transparent support different requirements of applications, even when consuming services related to the same WSN areas.

## 5. Conclusion

In this paper, we presented a service-oriented, flexible and adaptable middleware for QoS configuration and management of Wireless Sensor Networks. A case study has been also built and presented to show the effectiveness of the proposed solution.

In particular, our architecture supports QoS specification and management by using a contract negotiation scheme based on Service Level Agreements; it allows applications to reconfigure and maintain the network during its lifetime and it is independent of the underlying WSN technology. Moreover, it is characterized by an accurate design that permits to both abstract WSNs for a seamless integration into enterprise information systems and address specific low-level features that must be taken into consideration for guaranteeing certain QoS levels.

## References


[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Service Agreement Specification (WS-Agreement), March 2007. http://www.ogf.org/documents/GFD.107.pdf.

[2] F. Delicato, P. Pires, L. Pinnez, L. Fernando, and L. da Costa. A flexible web service based architecture for wireless sensor networks. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 730–735, May 2003.

[3] J. Gil-Martinez-Abarca, F. Macia-Perez, D. Marcos-Jorquera, and V. Gilart-Iglesias. Wake on lan over internet as web service. In *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, pages 1261 –1268, sept. 2006.

[4] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8 –13, oct.-dec. 2007.

[5] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6–14, Jan/Feb 2004.

[6] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[7] W. Masri and Z. Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 349–356, 2007.

[8] R. Moeller and A. Sleman. Wireless networking services for implementation of ambient intelligence at home. In *Devices, Circuits and Systems, 2008. ICCDCS 2008. 7th International Caribbean Conference on*, pages 1 –5, 2008.

[9] a. OASIS Standard. Devices profile for web services (dpws) version 1.1, July 2009.

[10] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 253–266, New York, NY, USA, 2008. ACM.

[11] I. K. Samaras, J. V. Gialelis, and G. D. Hassapis. Integrating wireless sensor networks into enterprise information systems by using web services. In *SENSORCOMM '09: Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications*, pages 580–587, Washington, DC, USA, 2009. IEEE Computer Society.

[12] W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, 2003.

[13] P. Walmsley and D. C. Fallside. XML schema part 0: Primer second edition. W3C recommendation, W3C, Oct. 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[14] Y. Yu, B. Krishnamachari, and V. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, Jan/Feb 2004.

[15] L. Zhang and Z. Wang. Integration of rfid into wireless sensor networks: Architectures, opportunities and challenging problems. *Grid and Cooperative Computing Workshops, International Conference on*, 0:463–469, 2006.