WISeMid: Middleware for Integrating Wireless Sensor Networks and the Internet

Jeisa P. O. Domingues, Antonio V. L. Damaso*, and Nelson S. Rosa

Universidade Federal de Pernambuco, Centro de Informática
Caixa Postal 7851 - 50740-540 - Recife - PE - Brazil
{jpo, avld, nsr}@cin.ufpe.br

Abstract. Wireless sensor networks (WSNs) have recently received growing attention as they have great potential for many distributed applications in different scenarios. Whatever the scenario, most WSNs are actually connected to an external network, through which sensed information are passed to the Internet and control messages can reach the WSN. This paper presents WISeMid, a middleware that focuses on integrating Internet and WSN at service level instead of integrating protocol stacks and/or mapping logical addresses. WISeMid allows the integration of WSN and Internet services by providing transparency of access, location and technology. To validate WISeMid, some results of a power consumption evaluation of the middleware are presented.

Key words: Wireless Sensor Networks, Internet, Middleware, Integration, Service

1 Introduction

A Wireless sensor network (WSN) is composed by a large number of sensor nodes which are deployed either inside a phenomenon or very close to it. Those nodes sense a physical aspect (e.g., pressure or temperature), process the sensed data and transmit them to a sink node, which can act as a gateway to other networks, a connection to a powerful data processor or an access point for human interface. Most WSNs are connected to an external network, through which their data can reach the final user and control messages can reach the WSN [1].

WSNs have been receiving growing attention as sensor nodes are becoming smaller, cheaper and intelligent, which enable the development of real and complex applications in scenarios such as military target tracking and surveillance, natural disaster relief, biomedical health monitoring, hazardous environment exploration and seismic sensing. As WSNs become more numerous and their data more valuable, it becomes increasingly important to have common means to share data over the Internet [2].

Since WSNs can be easily deployed to various environments to monitor target objects and various conditions, and to collect information, they are considered

* Antonio V. L. Damaso is supported by CNPq/Brazil

one essential infrastructure for pervasive computing systems [3]. Also, the WSN is one of the many networks that will compose the Ambient Networks [4]. For all those reasons, integrating WSNs with the Internet has become increasingly desirable and necessary.

A number of solutions have been proposed in recent years to provide the integration of WSNs and the Internet. Most of them aim at integrating those networks through mapping protocol stacks and logical address formats used in both networks. Those solutions focus at accessing the network nodes through their logical addresses, which raises several problems.

In this context, this paper proposes a solution that aims at integrating applications instead of networks (that is, protocols stack and/or logical address formats mapping). The idea is to provide an infrastructure, namely WISeMid, that allows integrating applications, which are considered services, in a transparent way. In practice, a service that is offered by a sensor node in a WSN or by a host in the Internet should be accessed in a uniform way irrespective of the client or the service location. Hence, application developers only need to know the service name to access its operations as WISeMid takes responsibility for hiding the heterogeneity of all network low level mechanisms. In order to validate our middleware, a power consumption evaluation is performed and the presented results show that although the components added by WISeMid infrastructure increases the power consumption, the services and features it offers save significant energy, which is a tradeoff worth making.

The remainder of this paper is organized as follows. The related works are presented in Section 2. Section 3 introduces the WISeMid, describing its elements and its implementation. Section 4 presents some WISeMid evaluation results concerning power consumption. Finally, Section 5 discusses some conclusions and presents some future work.

2 Related Work

Some approaches have been proposed to integrate WSNs and the Internet. The simplest one is the gateway-based approach. It may use an application layer gateway, translating query messages from one side (typically Internet) into messages that can be understood on the other side (usually WSN) [2] and/or mapping addresses [5]; or a Delay Tolerant Networks (DTN) gateway, providing interoperability between and among WSNs, which are considered DTN networks [6].

Overlay-based approaches have been proposed, where some sensor nodes use the TCP/IP protocols or some hosts use WSN protocols [7]. Also, mobile agents have been used to dynamically access the WSN from the Internet [1].

Although directly employing the TCP/IP suite in the WSN would enable its seamless integration with TCP/IP networks, this approach has several problems [8]: the addressing and routing schemes of IP are host-centric, and does not fit well with the sensor network paradigm, where the main interest is the data generated by the sensors and the individual sensor is of minor importance; the header overhead in TCP/IP is very large for small packets, and its size may constitute nearly 90% of each packet when sending a few bytes of sensor data, which is not acceptable as it wastes valuable energy in radio transmission; TCP does not perform well over wireless links networks where packets frequently are dropped because of bit-errors; and the end-to-end retransmissions used by TCP consumes energy at every hop of the retransmission path. Also, sensor nodes memory and computational resources are limited and not able to run a full instance of the TCP/IP protocol stack. For that problem, some works have proposed simplified versions of TCP/IP protocol stack: [9]-[10].

A reflective, service-oriented middleware for WSN is proposed in [11]. The middleware acts as a broker between applications and the WSN, translating application requirements into WSN configuration parameters. It monitors both network and application execution states, performing a network adaptation whenever it is needed. For an external point of view, applications are service requestors and sink nodes are service providers, releasing the descriptions of the WSN services and offering access to these services. Although it has some similarities with our proposal, it focuses on network adaptation capability. Besides, it assumes the WSN is a service provider, but not a service consumer.

The implementation of tiny web services directly on sensor nodes is presented in [12], including an XML parser, an HTTP server and a simplified TCP/IP protocol stack. As the previous approach, it considers the sensor nodes are only service providers (actually, web service providers), not consumers.

Unlike most of the mentioned works, this paper proposes a solution that focuses on integrating Internet and WSN at service level instead of integrating protocol stacks and/or mapping logical addresses. Also, even though some proposals are service-oriented, they only allow requesting services offered by the WSN, as most integration approaches only focuses in accessing the sensor nodes data from the Internet and not the other way around. Although this is the usual situation, as sensors are typically measured data providers, there are some cases where it is better for the sensor to request a service outside the WSN. That happens when the sensor has no enough resources to perform some computation, or when it becomes more resource consuming to perform such task in the sensor than sending a message to request it. For instance, there are some researches that propose solutions of management applications that run outside the WSN. In those approaches, a powerful computer that is connected to the sink node runs the management application, which collects information from the sensor nodes about their resources condition. Then it makes some computation and sends back to the sensor nodes some configuration changes, like new routes that spend less energy. One example of those solutions can be found in [13].

3 WISeMid

WISeMid (**W**ireless sensor network's and **I**nternet's **S**ervices int**e**gration **Mid**dleware) is a communication infrastructure that supports the integration of WSNs and the Internet at service level (see Fig. 1). In this context, applications running in the Internet/WSN nodes may play the role of service providers or service users. In practice, a service user must be able to communicate with a service provider no matter whether they are running in the same network or not. Hence, WISeMid should provide an infrastructure that allows integrating these services in such a transparent manner that a service should be accessed in the same way irrespective of it being provided by a WSN sensor node or by an Internet host. Additionally, WISeMid should support application services developed in different technologies, such as Web Service, Java RMI, EJB and JMS, although its current implementation supports only WISeMid services.

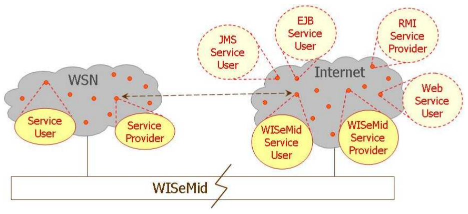

Fig. 1. WISeMid logical view

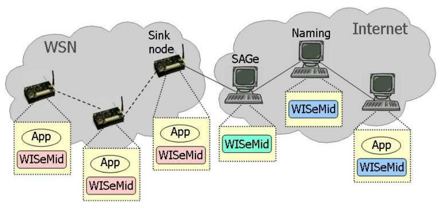

Fig. 2. WISeMid physical view

Figure 2 presents a physical view of how WISeMid spreads out through the Internet and WSN. The WISeMid implementation for WSN and Internet is not the same, as they have different requirements and components (that will be explained in the following sections). The physical communication is performed through an Internet host that is connected to the WSN sink node via a serial port (USB). This host executes a special WISeMid service called SAGe, which acts as a proxy between both networks (see Sect. 3.5).

3.1 Overview

In order to promote the networks integration at service level, our middleware has to address some issues. WISeMid should deal with four different kinds of heterogeneity, namely operating system, network, hardware and programming language. WSN and Internet nodes have different hardware platforms (e.g., PC and MICAz nodes) and network protocol stack (e.g., ZigBee and TCP/IP). In addition, applications running in both networks are developed atop different operating systems (e.g., Windows or Unix-based OS, and TinyOS) and using distinct programming languages (e.g., Java and nesC).

The heterogeneity of programming language raises another issue: data type mapping. For example, consider that a service user is written in nesC (it runs in the WSN) and a service provider is written in Java (it runs in the Internet). When the service user invokes an operation in the service provider, it is necessary to translate nesC data types into Java ones in a transparent way to application developers.

Along with handling the considered heterogeneities, it is also necessary to define the basic abstraction adopted for building applications (e.g., objects, services), the communicating entities (e.g., client/servers, peers), the way these entities communicate (e.g., synchronously, asynchronously), and distributed services provided by the middleware (e.g., naming service).

3.2 IDL

As service is the key concept in the proposed approach, an initial step consists of defining how a service is described. For this particular purpose, we have defined the WISeMid IDL (Interface Definition Language) that enables us to define service interfaces in a uniform way, i.e., wherever the service runs (WSN or Internet), its interface is described using the proposed IDL. The general structure of an interface defined in WISeMid IDL is shown as follows:

```
1: module PACKAGE_NAME{
2:   interface INTERFACE_NAME{
3:     [OPER_TYPE] OUTCOME_TYPE  OPER_NAME([TYPE ARG1,...]) [raises(EXCEPTION_NAME1,...)]
4:   } }
```

The module (package) that contains the service should be initially specified (1). Then, the service interface includes its name (2) and provided operations (3). Each operation has a name, input/output typed parameters and may raise exceptions. Additionally, an operation is by default a request-response operation, but it may be defined as a one-way operation, which means that no response is expected when the operation is invoked.

3.3 Requirements

Considering the issues introduced in Sect. 3.1, the following WISeMid's requirements have been defined: (R01) service providers should register their services in a Naming Service; (R02) service users should ask the Naming Service for the service it wants to use; (R03) WISeMid should provide location transparency, so the service users are not aware of the location of the service being used; (R04) WISeMid should provide access transparency, i.e., service users access local and remote services in a similar way; (R05) service data should have the same interpretation whatever the programming languages used to implement the service users and service providers; (R06) service providers/users communicate among themselves using Request/Reply communication pattern; (R07) service communication is synchronous; and (R08) services should be stateless and untyped.

In addition to these requirements, which concern both networks, there are some specific requirements regarding the limited resources of sensor nodes: (R09) the messages transmitted to the WSN should be kept as short as possible (for instance, if an argument type uses four bytes to represent a value but the argument value fits in one byte, and there is a compatible type that uses only one byte to represent that value, our middleware should convert this argument to the smaller type before sending it to a sensor node); and (R10) unnecessary messages should not be forwarded to the WSN (for example, when an Internet application requests a sensed data like temperature, whose variability considering second/minute time scale is not so significative, WISeMid may decide not to forward this request to the WSN, returning to the application the last value obtained from the WSN).

3.4 Architecture

The WISeMid architecture is depicted in Fig. 3 and consists of three layers: Infrastructure, Distribution and Common Services.

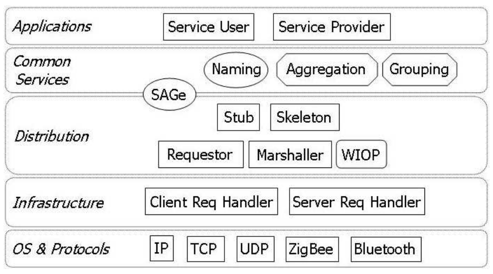

Fig. 3. WISeMid Architecture

The Common Services layer includes services that are not particular to a specific application domain: Aggregation, which performs sensor data aggregation and runs in the WSN; Grouping, which defines clusters inside the WSN; Naming, that stores information needed to access a service and runs in the Internet; and SAGe, that is in charge of forwarding messages from/to WSN and runs in the Internet. Additionally, SAGe also provides location transparency acting as a service proxy between both networks and performs some tasks concerning the aforementioned WSN specific requirements in order to avoid waste of sensors restricted resources (see Sect. 3.5).

The Distribution layer includes the following elements: the Stub, which represents a local instance of the service within the client process and offers the same interface as the remote service; the Requestor, which constructs a remote invocation on the client side from parameters such as remote service location, service name and arguments; the Skeleton, which dispatches remote invocations to the remote service using the invocation information sent by the Requestor; and the Marshaller, which serializes and deserializes the parameters passed between client and server using the WIOP Messages. WIOP is our middleware interoperability protocol, which is described in the next section.

The Infrastructure layer consists of the Client Request Handler and the Server Request Handler, which handle network communication using the communication facilities provided by the operating systems, e.g., sockets (Windows) and GennericCom (TinyOS).

As sensor nodes have limited resources, some elements of the WISeMid architecture are not present in the WSN: the Requestor is not implemented by the WSN service users (its functions are deployed by the Stub), and WIOP messages are treated as byte sequences, which means that the Marshaller is not necessary.

3.5 Implementation

The WISeMid implementation is divided into two parts, one for the WSN nodes, developed in nesC, and another for Internet hosts, developed in Java. Implementation details of the main middleware elements are described as follows.

WIOP. The WISeMid Inter-ORB Protocol (WIOP) is a GIOP-based protocol that defines the Request/Reply messages between clients and servers. A WIOP message is divided into header and body. The WIOP message header is composed by the following fields: endianness (e.g., big endian or little endian); msgType (e.g., request or reply message); and msgSize, which stores the message size in bytes. The WIOP message body may contain a Request or a Reply message. These messages have also a header and a body.

The Request message header comprises the fields: requestId, which stores the Request message ID; responseExpected, which signals whether the request has a Reply message or not; serviceId, which is the ID of the requested service; and operation, which represents the name of the operation being invoked. The Request body consists of the number of arguments (numArgs) followed by a sequence of type and value of each argument.

The Reply message header contains the fields: requestId, which stores the related Request message ID; replyStatus, which signals whether there was any exception while executing the request, and its possible values are NO_EXCEPTION (0), USER_EXCEPTION (1), SYSTEM_EXCEPTION (2) and LOCATION_FORWARD (3). The Reply body is composed by the result type and its value.

Although containing the same fields, the WIOP field sizes are smaller in the version running in the WSN (called $\mathrm{WIOP_s}$) than in the Internet version of WIOP (called $\mathrm{WIOP_i}$). In order to avoid energy waste during radio transmission, $\mathrm{WIOP_s}$ messages are kept as minimal as possible. Also, the way arguments are stored in the Request body is different for the WSN version. In the $\mathrm{WIOP_i}$, the arguments are stored one by one, being each argument composed by a type and a value. For example, three arguments would be stored in the following sequence: type1, value1, type2, value2, type3, value3. In the $\mathrm{WIOP_s}$, only the first argument is individually stored (with its type and value in a row). From the second argument on, the arguments are grouped into couples where the types of both arguments come first followed by their respective values. That happens because types are represented by integer numbers between 0 and 11 (e.g., the `float` type is represented by the number 7), and therefore each type can be stored in only 4 bits. Hence two types can be grouped into one byte, being followed by their related argument values. In this case, three arguments would be stored in the following sequence: type1, value1, type2, type3, value2, value3.

Besides saving energy by its reduced size, the sensor message format also concerns about sensor limited processing as it is already deployed as a byte array, avoiding the need of a Marshaller implementation.

The WISeMid Naming Service and Internet services use the Internet format, while the sensor services use the WSN format. Only SAGe handles both formats.

Naming Service. The WISeMid Naming Service stores the references of services executing in the Internet and WSN in such way that a service may only be accessed/used after being registered in the Naming Service. The Naming's service interface includes five operations: Bind, to register a service by its name, associating it with its reference; Lookup, to return the reference associated to a service name; Rebind, to change the reference that is associated with a service name; Unbind, to unregister a service name; and List, to list all registered services. The service reference includes the service ID, endianness, the IP address and the port number. In the case the service is running in the WSN (i.e., the sensor node has not an IP address), the stored IP address is the SAGe address.

SAGe. As stated in Sect. 3.4, SAGe (**S**ensor **A**dvanced **G**at**e**way) [14] is an important element in WISeMid architecture. Running in the Internet host connected to the WSN sink node via a serial port (see Fig. 2), SAGe's main function is to act as a service proxy between both networks by enabling the communication between services running on Internet hosts and WSN nodes in a transparent way. Furthermore, SAGe also performs some tasks concerning the previously defined WSN specific requirements (R09 and R10). This section describes how SAGe provides the location transparency and implements those requirements.

a. Binding of a WSN service: Once a WSN service starts, it sends a message invoking the Bind operation of the Naming Service. When SAGe receives that message, it creates a `ServiceReference` to the WSN service including the SAGe's IP address and port, then sends a binding request to the Naming Service, registering the WSN service as a SAGe service. It also keeps the created reference cached as a `SageServiceReference`, which assigns the `ServiceReference` to the node ID of the sensor providing the service. In such manner, SAGe knows which sensor node a Request message to a WSN service must be forwarded to.

b. Invocation of a WSN service: When SAGe receives a Request message from the Internet invoking a WSN service, it converts the message to a WIOP_s Request and sends it to the WSN using the `SageServiceReference` that was cached when the WSN service was bound. Once the Reply message from the sensor service provider is received, SAGe converts it into a WIOP_i Reply message and forwards it to the Internet service user. If no `SageServiceReference` is found, SAGe does not forward the request to the WSN. Instead, it sends a Reply message reporting an error to the Internet host that requested the service.

c. Invocation of an Internet service: When a sensor node service user performs a lookup for an Internet service, SAGe checks if this service is already known, i.e., if its reference is cached. If the service is unknown, SAGe converts and forwards the lookup request to the WISeMid Naming Service. When it receives the WIOP$_i$ Reply, it stores the returned `ServiceReference` (as a `SageServiceReference`) and sends the service ID to the sensor node service. Using the received service ID, the WSN service invokes the Internet service operation. When SAGe receives the sensor Request message, it uses the cached `ServiceReference` to invoke the requested operation and, once the Reply message arrives, SAGe converts and forwards it to the sensor node service user.

d. WSN requirements implementation: SAGe implements the WSN specific requirements defined in Sect. 3.3. To meet requirement R09, SAGe performs an additional step when converting an Internet Request message into a sensor Request message. For each argument in the Request body, it tries to fit the argument value in a smaller type (that is, a compatible type that uses less bytes). For instance, if the argument is a `long` (an integer of 8 bytes) but its value is '525', it can be stored into a `short` (an integer of 2 bytes). Thus SAGe converts the argument from a `long` into a `short` and adds only 2 bytes to the WIOP$_s$ instead of the original 8 bytes, avoiding the transmission of unnecessary 6 bytes. The same step is performed for the result value of WIOP$_i$ Reply messages when converting them into WIOP$_s$ Reply messages.

To deploy the other WSN specific requirement (R10), SAGe performs three more procedures. The first one is not forwarding to WSN any Internet Request which asks for a sensor service that has not been registered. It would be useless and energy wasting since no sensor announced that service. The second procedure occurs when a sensor requests an Internet service. It consists in not giving up at the first unsuccessful attempt to connect to the Internet service provider. Considering that it may be a sporadic problem, SAGe tries again to connect to the server a configurable number of times before returning an error to the sensor node. This measure aims to refrain the sensor from sending another Request in case the answer is fundamental for its application. The last procedure consists in avoiding sending equivalent Request messages (that is, messages asking for the same service with the same parameters) during a short period of time. Assuming that some sensed values do not change very quickly, sending the same Request for the same service in a short time period will likely return the same value, resulting in unnecessary processing and energy consumption. Hence, SAGe groups equivalent Request messages and, for a configurable period of time, only one Request is sent to the sensor service provider, and the received Reply message is stored and forwarded as an answer to all the equivalent Request messages. For the cases where the sensed value changes very often, this procedure may be turned off by setting to null (i.e., 0 seconds) the Reply message storage timeout.

4 Evaluation

As energy is a critical resource in wireless sensor networks, this section presents results about some experiments that analyze how WISeMid affects power consumption in the sensor node. For all scenarios, we use two MICAz motes: one connected to a MTS400 basic environment sensor board, running the application that constitutes the scenario; and another connected to a MIB520 USB programming board, working as a base station (BS), i.e., the sink node. The BS is connected to an Internet host that runs the WISeMid SAGe service or TinyOS SerialForwarder application, which acts as a proxy between the WSN and the Internet. Also, two other services run on Internet hosts: the Naming service and the service/application under evaluation.

In order to estimate the power consumption of the sensor node, an oscilloscope (Agilent DSO03202A) has been used. A PC is connected to the oscilloscope that captures the code snippet execution start and end times by monitoring a led of the sensor, which is turned on/off to signalize the execution start/end. The PC runs a tool named AMALGHMA [15], which is responsible for calculating the power consumption.

Five scenarios have been analyzed so far and they can be divided into two groups: one that studies the impact of WISeMid infrastructure on the power consumption of a WSN node, and one that evaluates the efficiency of some WISeMid features specially designed for saving power. In order to make the results more reliable, all values presented here are actually a mean value of 1000 executions of the code in study.

4.1 WISeMid infrastructure impact

These scenarios compare the power consumption of a service that uses the WISeMid infrastructure to a similar application that uses only TinyOS. It is worth noting that TinyOS does not have the notion of service, therefore an application with similar functionality to the service actually runs on the TinyOS.

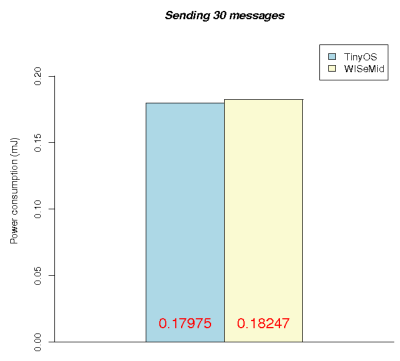

Fig. 4. Power consumption for sending 30 packets - TinyOS versus WISemid

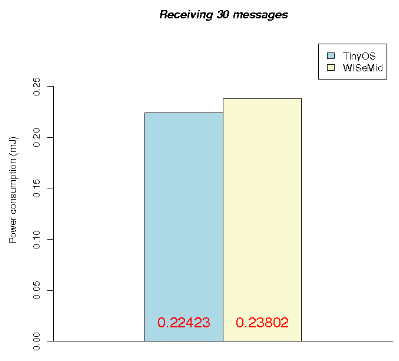

Fig. 5. Power consumption for receiving 30 packets - TinyOS versus WISemid

The first scenario measures the power consumption of a service that sends 30 messages to the Internet with an interval of 100ms between them. The reason for using 30 messages is to make the difference between the power consumption for both cases more evident without being costly. Sending only one would result in an almost unnoticeable difference whereas more than thirty would make the measurements slower and difficult to synchronize with the oscilloscope window. The interval of 100ms between consecutive messages is necessary to assure the next message will only be sent after the previous one has been completely transmitted. Smaller values have been tried but messages were still being lost.

When using the WISeMid infrastructure, a one-way WIOP message is created and sent through the WISeMid components. When the TinyOS is used, a message is normally created (i.e., by defining a `struct`). Both messages contain only one byte that carries the maximum value: 127 (11111111). Also, while the WISeMid service uses SAGe, the TinyOS application uses the SerialForwarder. Figure 4 shows the results of this scenario. Using the WISeMid spent 1.5% more energy than using only TinyOS. It was already expected that WISeMid was more power demanding as it adds more layers (components) to the sensor node. However, it was a small increase rate, which is acceptable considering the benefits it brings.

In the second scenario, the power consumption of a service that receives 30 messages from the Internet is calculated. The interval between each message is 100ms. Similarly to the previous scenario, WIOP messages are used to the WISeMid service whereas typical messages are handled by the TinyOS application, both carrying a byte with the value 127. The results for this scenario are presented in Fig. 5. As expected, the WISeMid service consumed more power than the TinyOS application (6.15%). Similarly to the previous experiment, the benefits of adopting WISeMid has a small cost in terms of power consumption.

The last scenario of this group gathers the two previous scenarios and adds the message processing and temperature reading, which are the necessary steps to answer a requisition to a Temperature service provided by the sensor node. This service implements the `TEMP` interface and thus provides the `getTemp()` operation, which returns the sensed temperature in Celsius degrees. The service interface is described in WISeMid IDL as shown:

```
module example{
  interface TEMP{
    long getTemp(); } }
```

This interface definition was compiled by the developed WISeMid IDL compiler, namely ProxiesGen, to generate the temperature service's stub (Java) and skeleton (nesC). The Java stub is used by the Internet application to access the service, whilst the nesC skeleton enables the access to the temperature service on the server side.

Note that in this case the service user is located in the Internet and the service provider is in the WSN node, but they do not know where each other is located. That happens due to the location transparency provided by the WISeMid Infrastructure (meeting requirement R03). In the equivalent code that uses only TinyOS, the Internet application must know the SerialForwarder IP address and port number and handle all network communications using, for instance, socket connections. The service abstraction provided by WISeMid explains the power

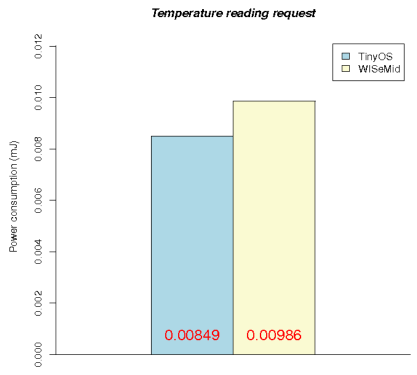

Fig. 6. Power consumption for requesting a sensor service 30 times - TinyOS versus WISemid

consumption increase of 16.11% comparing to the TinyOS application version, as presented in Fig. 6. Although it is not a negligible increase, the facilities offered by the WISeMid service abstraction as well as the energy saving brought by some WISeMid services compensates that, as the next results show.

4.2 WISeMid services

This section presents the results of scenarios that analyze the power that some WISeMid services save.

The first scenario studies the energy saving offered by the Aggregation service provided by the WISeMid. For that purpose, a sensor service sends 30 WIOP messages in a row, with an interval of 100ms between the messages. When the Aggregation service is used, instead of 30 messages, the sensor node only sends one message carrying the mean value of the 30 messages. As Fig. 8 shows, the Aggregation service saves 11.18% of energy.

In the last scenario, a feature offered by SAGe as implementation of the WSN specific requirement R10 is evaluated. When this feature is on, SAGe stores every Reply message for a given period of time and forwards it to all equivalent Request messages that arrive during this period, avoiding to send Requests that will return the same result (see Sect. 3.5). To evaluate that, an Internet service user requests the Temperature service 50 consecutive times. We have increased the number of requisitions from 30 to 50 in order to make the difference between the power consumption for both cases more noticeable. As the initial experiments considered an interval of 100ms between consecutive requisitions, the Reply message storage timeout was set to 300ms to allow sending three requisitions during this time and check if SAGe would "block" two of them. After confirming that, we kept the timeout value, but decided to make this scenario more realistic by using random intervals between consecutive requisitions. Thus those intervals are now randomly generated following a Uniform distribution with parameters 10 and 280, which allows the reception of 1 to 30 requisitions during the 300ms a

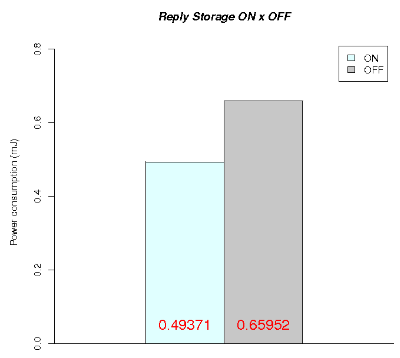

Fig. 7. Energy saving by using the SAGe's Reply Storage feature

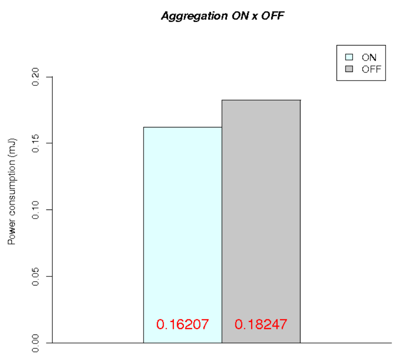

Fig. 8. Energy saving by using the WISeMid's Aggregation service

Reply message is stored, depending on the generated values. In order to compare the power consumption with this feature "on" and "off", the same seed was used for both cases. Figure 7 shows that this SAGe feature saves 25.14% of energy as it avoids sending unnecessary requests to the WSN.

The results presented in this section show that on one hand WISeMid infrastructure increases power consumption with its additional components for sensor code, but on the other hand the services and features it offers save significant energy, which is a tradeoff worth making.

5 Conclusion and Future Work

In this paper, we presented the WISeMid middleware as an approach to address the WSN and Internet integration issue. The proposed approach concentrates on solving this problem by integrating services instead of layers. WISeMid provides an infrastructure that allows integrating WSN and Internet services with transparency of access, location and technology. Hence, a service that is offered by a sensor node in a WSN or by a host in the Internet can be accessed in the same way irrespective of the client or the service localization.

To validate WISeMid, a power consumption evaluation was presented, showing that although the components added by WISeMid infrastructure increases the power consumption, the services and features it offers save significant energy, which is a tradeoff worth making.

In terms of future work, other power consumption evaluation are now being conducted. Also, we are improving the proposed middleware by including typed services, allowing a client to ask for a service by only specifying its type, and a life cycle manager for the remote services, which will enable stateful services. Some features will also be added to SAGe, such as turning it into a distributed service, to refrain it from becoming a bottleneck in large-scale WSN, and deploying conversion between WIOP and others interoperability protocols (e.g., IIOP, JRMP), to enable WISeMid to support application services developed in different technologies, like Web Service, Java RMI, EJB and JMS.

References

1. Bai, J., Zang, C., Wang, T., Yu, H.: A Mobile Agents-Based Real-time Mechanism for Wireless Sensor Network Access on the Internet. In: 2006 IEEE International Conference on Information Acquisition, pp. 311–315 (2006)
2. Reddy, S., Chen, G., Fulkerson, B., Kim, S.-J., Park, U., Yau, N., Cho, J., Hansen, M., Heidemann, J.: Sensor-Internet Share and Search: Enabling Collaboration of Citizen Scientists. In: Workshop for Data Sharing and Interoperability (IPSN 07), pp. 11–16. (2007)
3. Zheng, Y., Cao, J., Chan, A.T.S., Chan, K.C.C.: Sensors and Wireless Sensor Networks for Pervasive Computing Applications, Subsequences. J. Ubiquitous Computing and Intelligence 1 (1), 17–34 (2007)
4. Niebert, N., Prehofer, C., Hancock, R., Norp, T., Nielsen, J.: Ambient Networks - A New Concept for Mobile Networking. Technical report, Wireless World Research Forum (2004)
5. Kim, J.-H., Kim, D.-H., Kwak, H.-Y., Byun, Y.-C.: Address Internetworking between WSNs and Internet supporting Web Services. In: 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007), pp. 232–240 (2007).
6. Ho, M., Fall, K.: Poster: Delay Tolerant Networking for Sensor Networks. In: 1st IEEE Conf. Sensor and Ad Hoc Communications and Networks (2004)
7. Dai, H., Han, R.: Unifying Micro Sensor Networks with the Internet via Overlay Networking. In: 29th Annual IEEE International Conference on Local Computer Networks, pp. 571–572 (2004)
8. Dunkels, A., Voigt, T., Alonso, J., Ritter, H., Schiller, J.: Connecting Wireless Sensornets with TCP/IP Networks. In: 2nd International Conference on Wired/ Wireless Internet Communications, pp. 143–152 (2004)
9. Dunkels, A.: Full TCP/IP for 8-Bit Architectures. In: 1st International Conference on Mobile Systems, Applications and Services, pp. 85–98 (2003)
10. Durvy, M.: Poster Abstract: Making Sensor Networks IPv6 Ready. In: 6th ACM Conference on Networked Embedded Sensor Systems (2008)
11. Delicato, F.C., Pires, P.F., Rust, L., Pirmez, L., Rezende, J.F.: Reflective Middleware for Wireless Sensor Networks. In: 2005 ACM Symposium on Applied Computing, pp. 1155–1159 (2005)
12. Priyantha, N.B., Kansal, A., Goraczko, M., Zhao, F.:Tiny Web Services: Design and Implementation of Interoperable and EvolvableSensor Networks. In: 6th ACM Conference on Embedded Network Sensor Systems, pp. 253–266 (2008)
13. Ozturgut, H., Scholz, C., Wieland, T. and Niedermeier, C.: SCOPE - Sensor Mote Configuration and Operation Enhancement. In: 22nd International Conference on Architecture of Computing Systems, pp. 84–95 (2009)
14. Damaso, A., Domingues, J. and Rosa, N.: SAGe: Sensor Advanced Gateway for Integrating Wireless Sensor Networks and Internet. In: 3rd Workshop on Applications of Ad hoc and Sensor Networks (AASNET), (2010) To appear.
15. Tavares, E.: Software Synthesis for Energy-Constrained Hard Real-Time Embedded Systems. PhD Thesis, Center of informatics, Federal University of Pernambuco. November (2009)