Pers Ubiquit Comput (2006) 10: 37–44
DOI 10.1007/s00779-005-0038-3

ORIGINAL ARTICLE

**Eduardo Souto · Germano Guimarães
Glauco Vasconcelos · Mardoqueu Vieira · Nelson Rosa
Carlos Ferraz · Judith Kelner**

Mires: a publish/subscribe middleware for sensor networks

Received: 10 July 2004 / Accepted: 17 November 2004 / Published online: 9 October 2005

Abstract A wireless sensor network (WSN) consists of a large number of small devices with computational power, wireless communication and sensing capability. These networks have been developed for a wide range of applications, such as habitat monitoring, object tracking, precision agriculture, building monitoring and military systems. Meanwhile, middleware systems have also been proposed in to facilitate both the development of these applications and provide common application services. The development of middleware for sensor networks, however, places new challenges on middleware developers due to the low availability of resources and processing capacity of the sensor nodes. In this context, this paper presents the design and implementation of a middleware for WSN named Mires. Mires incorporates characteristics of message-oriented middleware by allowing applications communicate in a publish/subscribe way. In order to illustrate the proposed middleware, we have also developed an environment-monitoring application and a data aggregation service.

Keywords Middleware · Publish/subscribe systems · Wireless sensor networks

1 Introduction

A wireless sensor network (WSN) consists of a large number of small devices with computational power, wireless communication and sensing capability [1]. Sensor nodes are usually scattered in an observation region. Each sensor node in the observation region is responsible for extracting data from the environment (such as temperature, humidity, pressure and luminosity), processing and sending this data through one or more sink nodes, which are in turn responsible for transmitting this data to the final user.

These networks have been developed for a wide range of applications such as habitat monitoring, object tracking, precision agriculture, building monitoring and military systems [1, 2, 3]. Common to these applications is the need for continuous collection and integration of data from a large number of sensor nodes. Hence, a basic issue for middleware is how to satisfy an applications' requirements considering the specific characteristics of sensor networks such as constrained sensor power and network bandwidth. The design of sensor networks applications is therefore highly influenced by resource scarcity (e.g., battery, memory and processor), communication models and application requirements.

In this way, most research in this area has been directed to the development of new protocols that promote efficient resources utilization, mainly with respect to the power consumption [4, 5, 6, 7]. Although these protocols are effective in extending the lifetime of sensor networks, the gap between the protocols and the application inhibits their effective use by application developers [8]. To make these protocols more useful, application designers would benefit from a middleware layer that hides details of communication protocols, while providing an API (Application Programming Interface) that reduces the cost of developing applications.

In this context, the design and implementation of an appropriate middleware layer to fully realise the capabilities of sensor technologies and applications is now an open research issue. Traditional middleware systems such as Java RMI (Remote Method Invocation) [9], EJB (Enterprise JavaBeans) [10] and CORBA (Common Object Request Broker Architecture) [11] are normally heavyweight in terms of memory and computation and therefore not suitable for WSNs [12].

A middleware for WSN should facilitate development, maintenance, deployment and execution of sensing-based

E. Souto (✉) · G. Guimarães · G. Vasconcelos · M. Vieira · N. Rosa · C. Ferraz · J. Kelner
Professor Luis Freire, Cidade Universitria, Federal University of Pernambuco, Informatic Center, 50740-540 Recife, PE, Brazil
E-mail: ejps@cin.ufpe.br
Tel.: +55-81-21268430
E-mail: gfg@cin.ufpe.br
E-mail: gpv@cin.ufpe.br
E-mail: msv@cin.ufpe.br
E-mail: nsr@cin.ufpe.br
E-mail: cagf@cin.ufpe.br
E-mail: jk@cin.ufpe.br

applications. To accomplish this, it is necessary to enable the development of complex and high-level sensing tasks. These tasks must be able to communicate with other tasks in the WSN, coordinate sensors, be distributed amongst individual sensor nodes, support data merging of sensor readings in individual nodes into a higher level results and report the result back to the task issuer. Moreover, one should provide appropriate abstractions and mechanisms for dealing with the heterogeneity of sensor nodes. All mechanisms provided by a middleware system should respect the restrictions involved in WSN systems, which are mostly energy efficiency, robustness and scalability [13].

In addition, the communication between applications in WSNs is essentially based on events (event-driven communication model), which suggests that the traditional request/response approach (synchronous) is not appropriate. In most applications, data transmission is triggered when either an event occurs or the sink node generates a query. This kind of data dissemination is well supported by the publish/subscribe paradigm where a publisher makes information available to subscribers in an asynchronous fashion.

Taking into account that the network nodes have well-defined attributes (e.g., temperature, humidity, pressure), a publish/subscribe scheme can be used to query and extract data from the network. In such a scheme, each node announces a set of attributes that describe monitored data types. The user application subscribes to a subset of the attributes offered by the sensor nodes. A subscribe message is broadcasted through the network until it reaches all the sensor nodes. From this moment, the sensor nodes start to monitor, collect, process and transmit the desired information.

This paper presents the design and implementation of a publish/subscribe middleware, named Mires, which encapsulates the network-level protocols (routing and topology control protocols) and provides a high-level API that facilitates the development of applications. In order to validate our approach we have also developed an environment-monitoring application and a data aggregation service.

The remainder of the paper is organized as follows. Section 2 introduces related work in the area. Section 3 provides a detailed description of the proposed publish/subscribe middleware. Section 4 presents an environmental monitoring application used to validate our middleware. Finally, Sect. 5 presents some conclusions and directions for future works.

2 Related work

Several middleware systems have been designed to deal with WSN issues. Maté [14] is an architecture for constructing application-specific virtual machines that executes on top TinyOS [15]. Using this architecture, developers can easily change instruction sets, execution events, and VM subsystems. Maté provides a simple programming interface to sensor nodes. For example, a sense-and-send program can be written with six instructions.

Another middleware, Impala [16], designed for use in the ZebraNet project, supports control in the application itself by exploiting mobile code techniques to change the functionality of the middleware executing at a remote sensor. The key to energy efficiency for Impala is for the sensor node applications to be as modular as possible, enabling small updates that require little power during transmission.

Unlike Impala and Maté, MiLAN (Middleware Linking Applications and Networks) [8] has an architecture that reaches the network protocol. MiLAN is intended to sit on top of multiple physical networks. It acts as a layer that allows network-specific plug-ins to convert MiLAN commands to protocol-specific ones that are passed through the usual network protocol stack. Therefore, MiLAN can continuously adapt to the specific features of whichever network is being used in the communication. Finally, Cougar [17] adopts a database approach where sensor readings are considered to be in "virtual" relational database tables. An SQL-like query language is used to issue tasks to the WSN.

First concrete experiments show that even very simple protocols and algorithms can exhibit surprising complexity at large scale [13].

3 The middleware Mires

WSN applications need to continuously collect and integrate data generated from a large and physically dispersed cohort of sensor nodes. Typically, there are a large number of devices exchanging data, whilst some information sources and sinks may not be present in the network at the same time. Therefore, the request/response communication is not adequate to satisfy this requirement. For example, a client who requests instantaneous updates of information would need to continuously poll the information providers leading to network overload and congestion. Moreover, as energy is a scarce resource, unnecessary information requests should be avoided.

Publish/subscribe communication is adequate for the required information dissemination model of sensor network applications [19, 20]. In this kind of communication, an information supplier publishes messages that are forwarded to one or more subscribers (one-to-many communication). An extension to this basic model allows messages being associated to topics. In this particular case, the subscribers only receive messages associated with the exact topic(s) to which they have subscribed.

The key elements in publish/subscribe communication are the notification service and the buffer where the messages are queued before they are passed to subscribers. The notification service takes responsibility for informing the subscribers when a new message arrives. In this way it allows the asynchronous communication as producers and consumers are fully decoupled. This loose coupling is the prime advantage of this kind of communication for ad hoc and pervasive environments such as WSN [21].

The Mires middleware addresses the implementation of publish/subscribe communication for wireless sensor network applications. The communication between nodes consists of three phases. Initially, the nodes in the network advertise their available topics (e.g., temperature and humidity) collected from local sensors. Next, the advertised messages are routed to the sink node using a multi-hop routing algorithm. A user application (e.g., a graphical user interface) connected to the sink node is able to select (i.e., subscribe) the desired advertised topics to be monitored. Finally, subscribe messages are broadcasted down to the network nodes. After receiving the subscribed topics, nodes are able to publish their collected data to the network. Further details are presented in the following sections.

3.1 MIRES architecture

Figure 1 shows the proposed middleware architecture. From the bottom to the top, the first block corresponds to the sensor node's hardware components. It generally includes a microcontroller unit, one or more sensors and a radio transceiver. These components are directly interfaced and controlled by the operating system (OS). The low-level services provided by the OS can be accessed through standard interfaces.

Mires is placed on the top of the OS, encapsulating its interfaces and providing higher-level services to the Node Application. Mires internal structure is composed of the publish/subscribe service, a routing component and additional services. The additional services (e.g., data aggregation service) may be easily integrated to the publish/subscribe service if they implement the appropriate interfaces.

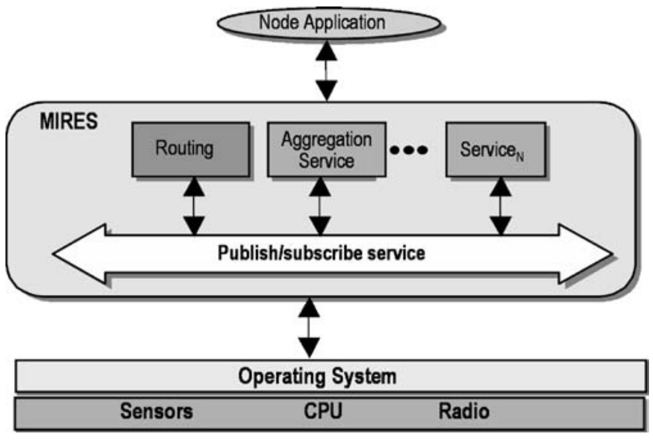


Fig. 1 Mires' architecture

The main component in Mires architecture is the publish/subscribe service. This service mediates the communication between middleware services. It is also responsible for advertising the topics provided by the local application, maintaining the list of topics subscribed by the node application and publishing messages containing data related to the advertised topics. Only messages referring to the subscribed topics are transmitted, thus saving precious power supplies.

In order to transmit either locally generated or forwarded messages received from the network towards the sink node, the publish/subscribe service uses a multi-hop routing algorithm. Since the operating system does not specify a standard routing algorithm, we have opted to incorporate this feature into the middleware. The algorithm must be implemented as an independent component, allowing the utilization of any multi-hop routing algorithm since it implements the required interface by Mires.

As mentioned before, additional services can be easily incorporated to Mires through interfaces that define notification events. Thus, the publish/subscribe service notifies the services (those interested in the message) about message arriving from the network or submitted by the local application. If a service is interested in being notified of a certain event, it needs only to implement a specific handle to that event.

There are three types of notification events: *topicArrival, stateArrival* and *topicSetupArrival*. The *topicArrival* event signals that the node application has submitted data collected from sensors. When this event occurs, the notified service at first decides if the data are locally processed (e.g., aggregated) or transmitted, and then communicates its decision to the publish/subscribe service. The *stateArrival* event is very similar to the previous one except for the fact that the data comes from the network. The network data represents the results from the processing in a node positioned in a lower level of the routing hierarchy. Finally, the *topicSetupArrival* event is the subscribe message broadcasted by the user application. It contains both a list of subscribed topics and configuration information for the services, e.g., the policy that establishes stop criterion (time limit, number of samples).

As soon as the stop criterion for a subscribed topic is satisfied, the service can publish the local processing results to the network. The publish/subscribe service then interacts with the routing component to send a message containing the local results to the node in the next hierarchy level. This process repeats until a publish message reaches the sink node, which is responsible for transmitting the data collected by the sensor network to the user application.

3.2 Publish/Subscribe service

Before presenting details of the proposed publish/subscribe service, it is worth presenting some relevant aspects of the implementation environment upon which the service was built, namely TinyOS. This environment has a component-based programming model provided by the nesC [22] language. NesC is a high-level language for building structured component-based applications. TinyOS programs can be seen as a component graph, where each component provides and uses external interfaces, which are composed by commands and events. Commands are procedures that are implemented in the interface provider. When an interface provider signals an event, this causes the execution of a procedure (event handler) implemented in the interface user. The communication among network nodes is based on the Active Messages [23] paradigm. According to it, each message contains the ID of a handle to be invoked on the target node and data payload to pass in as arguments. This event-based and message-oriented communication paradigm makes TinyOS a good foundation for building a publish/subscribe-based communication infrastructure.

Figure 2 illustrates the connections between the publish/subscribe service component and the other elements of Mires. The *PublishSubscribe* component provides both the *Advertise* and *Publish* interfaces to the node application. It also provides the *PublishState* and the *Notifier* interfaces to *ServiceX* (e.g., aggregation service). These two interfaces allow the addition of services to Mires.

The *PublishState* interface defines the command used by *ServiceX* to publish their processing results. The *Notifier* interface defines three events to which new services must provide handler implementations and then to be notified by the publish/subscribe service of occurrences like local or remote data arrival.

The *PublishSubscribe* service uses three interfaces (*Send*,*Receive* and *Intercept*) that are implemented by the *Bcast* and *MultiHopRouter* communication components. The *MultiHopRouter* component is responsible for establishing the routing hierarchy towards the sink node. The *Bcast* component is responsible to broadcast set up information (e.g., subscribed topics by the user application) across the network.

The following sequence diagrams show the interactions between the publish/subscribe component and the other Mires' components. In Fig. 3, the node application advertises to the *PublishSubscribe* service its capability of sensing data related to a certain topic (*advertise phase*). The *PublishSubscribe* service encapsulates this information in an *advertiseMsg* and sends it to the network via the *MultiHopRouter* component.

The interaction in the bottom of Fig. 3 refers to the occurrence of an *advertiseMsg* message arrival in the node. All messages are addressed to the sink node, but in the intermediate nodes the *MultiHopRouter* component signals an *intercept* event whenever it receives an *advertiseMsg* message. Next, the *PublishSubscribe* service extracts the advertised topic information from the message, updates its internal control structure and returns an indication to the *MultiHopRouter* that the message should be forwarded to the upper nodes in the network hierarchy.

Figure 4 illustrates the topic subscription interaction that is initiated after the advertise phase. The user application invokes the sink node's send command in order to broadcast the subscribed topics to the network. In each node that receives the *subscribeMsg*, the *Bcast* component signals a *receive* event. Then, the *PublishSubscribe* component extracts the set up information from the message and signals the *topicSetupArrival* event to notify the *ServiceX* components attached to it.

Figure 5 shows how the monitored data are both published and processed by the network. The *NodeApplication* periodically collects readings from the node's sensors and invokes the publish command of the *Pub-*

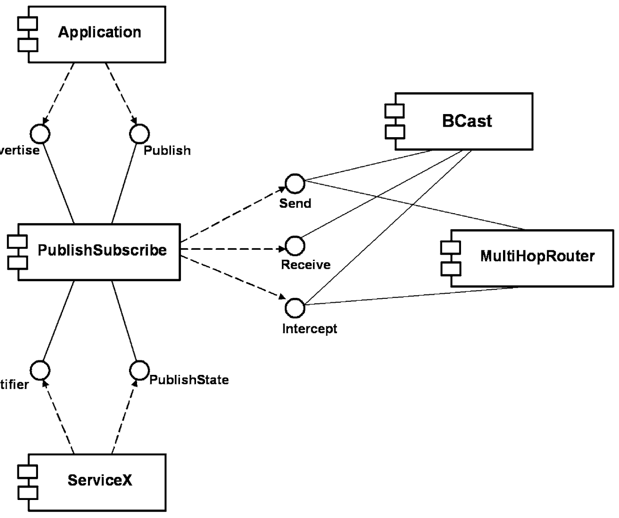


Fig. 2 Publish/subscribe component diagram

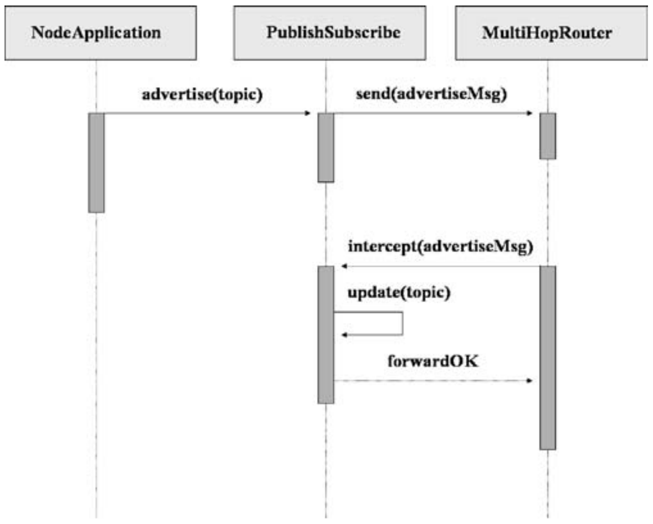


Fig. 3 Topic advertisement sequence diagram

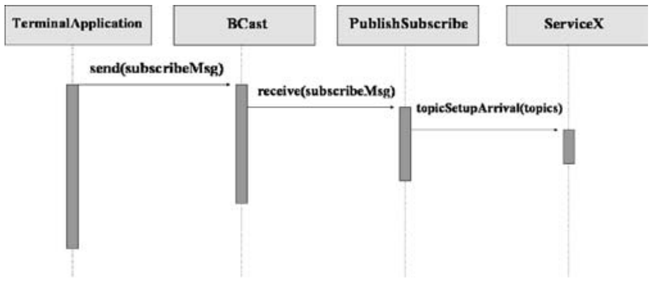


Fig. 4 Topic subscription sequence diagram

lishSubscribe component. If other services are attached to it, the publish/subscribe component notifies them through the *topicArrival* event.

In the next interaction, the service invokes the *publishState* command to the *PublishSubscribe* component passing its processing results. Then, the *PublishSubscribe* service encapsulates the local state inside a publishMsg and sends it to the network by using the MultiHopRouter component.

Finally, the third interaction occurs when a *publishMsg* arrives at a node. The MultiHopRouter signals an *intercept* event containing the publishMsg. The publish/subscribe component extracts the remote state from the message and notifies the other services by signalling the *stateArrival* event. Thus, a notified service is supposed to merge the remote state with its local one.

In the next section, we discuss how Mires can be used to easily develop an environment monitoring application, a very common application of wireless sensor networks. We also exemplify how a service can be attached to the Mires framework in a straightforward manner.

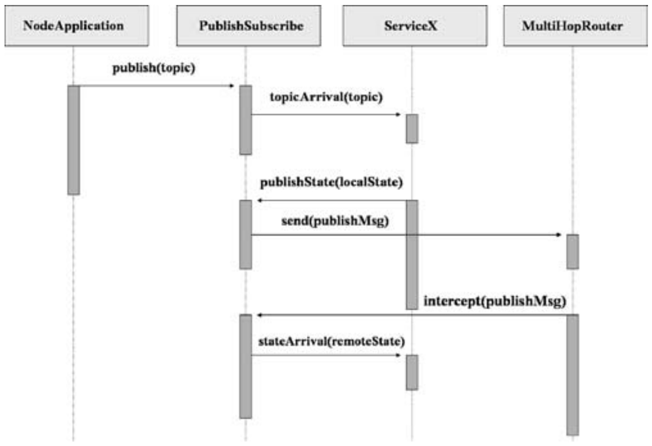


Fig. 5 Data publishing sequence diagram

4 Case study: an environment-monitoring application

This section presents an environment-monitoring application built to validate the implementation of our publish/subscribe middleware. To this end, an aggregation service was implemented and connected to Mires.

4.1 Scenario

Imagine a cubical environment-monitoring scenario with the following characteristics: each environment is formed by a group of sensor nodes that can monitor variables such as temperature, humidity, sound and luminosity. Figure 6 represents this scenario.

The sensors are grouped in rooms forming clusters. Each room has a node responsible for the communication with the sink node, called a cluster head. Each node inside a cluster integrates the information of nodes under it in the routing hierarchy by means of some aggregation technique and reports the results up in the hierarchy until the cluster head. A sink node receives the sensing tasks from the external application and spreads them to the network. Thus, the flow of tasks goes from the sink node to the sensor nodes, while the data flow goes in the opposite direction. The external application will collect this data and deal with it in the desired form.

4.2 Aggregation service

Environment-monitoring applications usually require that collected data from sensor nodes be aggregated in order to reduce the number of transmissions in the network. Data aggregation is the combination of data from different sources by functions such as *suppression, min, max* and *average* [24]. A naive implementation of

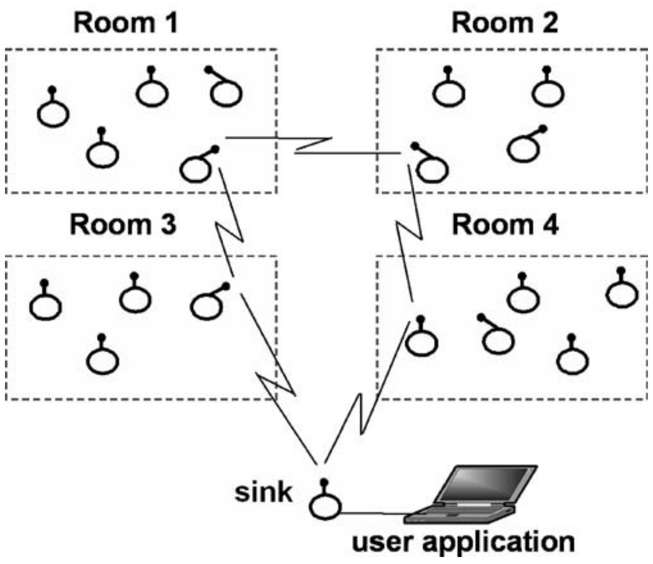


Fig. 6 Sensor nodes distributed in rooms

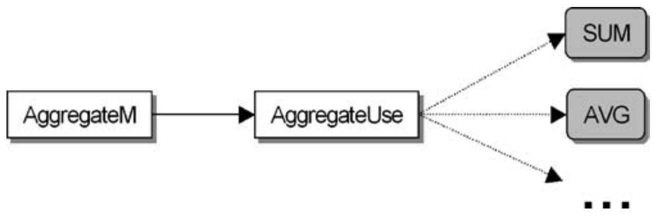


Fig. 7 Aggregation service structure

sensor network aggregation would be to use a centralized, *server-based* approach where all sensor readings are sent to a base station, which computes the aggregates [25]. In our case study, however, the aggregation has been performed at each sensor node, by allowing sensors to conduct in-network data reduction. This technique has been used to reduce the number of message transmissions, latency and power consumption [5, 18, 26]. To accomplish this requirement, we have implemented an aggregation service, which can be configured during the subscribe process. The user can set out how data will be aggregated (*aggregation function*) along with the stop criteria (*aggregation policy*). When the stop criterion is satisfied in a node, it publishes its local result to the next node according to the routing algorithm. The aggregation service implements three basic actions:

- Aggregate data originating from the local sensor(s) with data coming from the network
- Control the association between the topics and its aggregation functions. That is necessary because nodes can receive data of several topics and each topic can be associated to a different aggregation function.
- Verify the satisfaction of the aggregation policy's criteria and request publication of the associated state[1] with a topic. For example, the time criterion defines an interval where the aggregation function is applied on values obtained locally or generated from its sub-nodes. When the time is over, the local state is transmitted and restarted.

An aggregation function defines how the data are combined. Figure 7 shows the structure of the proposed aggregation service. The *AggregationM* module contains the implementation of the aggregation service, handling the notifications sent by the publication service. The aggregation functions are implemented in separate modules such as AVG and SUM. The *AggregateUse* module carries out an activity of de-multiplexing, passing requests for the correct aggregation module in accordance to its identifier. This way, flexibility to add new aggregation functions is guaranteed, just requiring the creation of a module for the new function and the association between the function and an identifier to a configuration file.

As this aggregation service is implemented in a distributed manner it decreases the communication required to compute an aggregate versus a centralized aggregation approach.

[1]A state is the data structure that represents the local result for the aggregation of a certain topic.

4.3 Application example

In order to illustrate the applicability of the proposed architecture, we present an application example in which the user subscribes its topics of interest and visualizes the response from the network through a graphical interface of the incoming data. Figure 8 shows a user application and its interaction with the middleware architecture. The MIRES execution can be divided in the following phases: network establishment, announcement, subscription and publication.

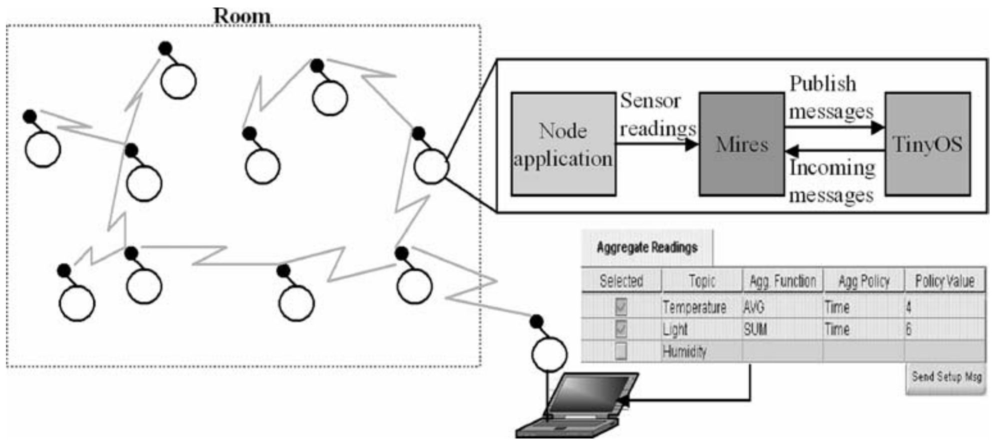
After the deployment of sensor nodes, configuration messages start to be exchanged with the objective to establish routes towards the sink node. This process is controlled by the routing component of Mires, which signals the conclusion of the network establishment to the sensor node application. In this case study, the routing component implements a multi-hop algorithm.

When the establishment of the routing tree is finished, the announcement phase is initiated. In this phase, the sensor node application informs the middleware that it will publish data of a specific topic (e.g., temperature). It is Mires' responsibility to inform the other sensor nodes in the routing tree. Thus, the application does not need to worry about the procedure of topic diffusion in the network.

Once the announced topics arrive at the sink node, they are passed to the user application. In this case study, a graphical interface was built to allow the user to select one or more topics of interest, specifying desired policies and aggregation functions. A screenshot of this application is shown in Fig. 8, which demonstrates a configuration where the user is interested in receiving data about temperature every 4 min and luminosity every 6 min. The user application sends a subscribe message to the sink node, and then Mires is responsible for the dissemination of the configurations of the user to the other nodes in the network. This procedure corresponds to the subscription phase.

After the subscription phase, the application starts to publish the data of interest, as subscribed to by the user application. In the aggregation service, the data are combined in accordance with the function and policies established by the user application in the subscription phase. Mires is responsible for transmitting the data processed by the aggregation service to other nodes.

In summary, the user application needs only inform Mires which topics (temperature, luminosity or humidity) interest it. The middleware receives this configuration message (topics, function and aggregation policies) and sends the requested data back to the user application. In this way, Mires undertakes the control and communication of the sensors' readings, thus reducing the semantic gap of the application development.



Selected	Topic	Agg. Function	Agg Policy	Policy Value
✓	Temperature	AVG	Time	4
✓	Light	SUM	Time	6
☐	Humidity			

Fig. 8 Application example

5 Conclusions and future work

Wireless sensor network applications place new challenges on middleware developers due to the low availability of resources and processing capacity of the sensor nodes. This work's main contribution is to demonstrate that the publish/subscribe paradigm can be successfully applied in WSNs, providing an asynchronous communication model that is better suited than the traditional request/response model to cope with the event-driven nature of the sensor networks applications. Mires has been designed to facilitate the development of applications over WSNs. This middleware is implemented on the top of TinyOS, an event-based operating system explicitly designed for network sensors.

In Mires, each node advertises topics available in its sensor hardware. A user application receives these topics and selects the desired topics to be monitored. After this, nodes are able to publish the collected data of interest. The Mires' core component is the publish/subscribe service. This service acts as an intermediate between the application running locally and the communication components of the operating system. It is responsible for advertising the topics provided by the local application, maintaining a list of the topics subscribed by the user application and publishing messages containing data related to the advertised topics. Only messages referring to the subscribed topics are transmitted, thus saving precious energy supplies.

In our case study, we implemented a service that can be attached to the Mires framework in a straightforward manner. The aggregation service is responsible for combining the data from different sources, eliminating the redundancy of the transmitted data, minimizing the number of transmissions and, thus, increasing the network lifetime. This fact occurs because the parent nodes in the routing hierarchy process the messages sent by child nodes and the results are then published upwards in the network. This processing consists of the computation of a mathematical function that summarizes the results obtained by the cluster participants.

The next steps in Mires development are related to the evaluation of its impact on the overall network performance and power consumption. Improvements will be made to make it more robust to sudden topology changes and individual node crashes. Tests using real sensor nodes (motes) are also been planned. Finally, additional services such as security, trading and resource management will be integrated into Mires.

Acknowledgements This work is partially sponsored by CNPq and CAPES, Brazilian government research agency.

References

1. Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E (2002) A survey on sensor networks. IEEE Communications Magazine, pp. 102–114, August
2. Cerpa et al (2002) Habitat monitoring: application driver for wireless communications technology. ACM SIGCOMM workshop on data communications in Latin America and the Caribbean, Costa Rica, April
3. Pottie GJ, Kaiser WJ (2000) Wireless integrated network sensors. Communications of the ACM 43(5):51–58
4. Heinzelman W, Chandrakasan A, Balakrishnan H (2000) Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the IEEE Hawaii international conference on system sciences, Hawaii, USA, January
5. Lindsey S, Raghavendra CS (2002) PEGASIS: Power efficient gathering in sensor information systems. In: Proceedings of the IEEE aerospace conference, Montana, USA, March
6. Younis M, Youssef M, Arisha K (2002) Energy-aware routing in cluster-based sensor networks. In: Proceedings of the 10th IEEE/ACM international symposium on modelling, analysis and simulation of computer and telecommunication systems, Fort Worth, Texas, USA, October
7. Manjeshwar, Agrawal DP (2001) TEEN: a protocol for enhanced efficiency in wireless sensor networks. In: Proceedings of the 1st International workshop on parallel and distributed computing issues in wireless networks and mobile computing, San Francisco, CA, USA, April
8. Heinzelman W, Murphy A, Carvalho H, Perillo M (2004) Middleware to support sensor network applications. IEEE Network Magazine Special Issue, pp 6–14, January
9. Wollrath, Riggs R, Waldo J (1996) A distributed object model for the Java system. Usenix conference on object oriented technologies and systems, May

10. Thomas, Seybold P (1998) Enterprise JavaBeans Technology. available in http://java.sun.com/products/ejb/whitepaper.html, December
11. Object Management Group (1999) The common object request broker: architecture and specification. Published by the Object Management Group (OMG), Revision 2.3, June
12. Yu Y, Krishnamachari B, Prasanna VK (2004) Issues in designing middleware for wireless sensor networks. IEEE Network Magazine Special Issue 18(1):15–21
13. Rmer K, Kasten O, Mattern F (2002) Middleware challenges for wireless sensor networks. ACM SIGMOBILE Mobile Communication and Communications Review 6(2)
14. Levis P, Culler D (2002) Maté: a tiny virtual machine for sensor networks", In: Proceedings of the 10th international conference on achitectural support for programming languages and operating systems, San Jose, CA, USA, October
15. Hill J, Szewczyk R, Woo A, Hollar S, Culler D, Pister K (2000) System architecture directions for networked sensors. In: ACM SIGOPS operating systems review 34(5):93–104, December
16. Liu T, Martonosi M (2003) Impala: a middleware system for managing autonomic, parallel sensor systems. In: Proceedings of the ninth ACM SIGPLAN symposium on principles and practice of parallel programming, San Diego, CA, USA, June
17. Bonnet P, Gehrke JE, Seshadri P (2000) Querying the physical world. IEEE Personal Communications 7(5):10–15
18. Krishnamachari, Estrin D, Wicker SB (2002) The impact of data aggregation in wireless sensor networks. In: Proceedings of the 22nd international conference on distributed computing systems, pp 575–578, Vienna, Austria, July
19. Yoneki E (2003) Mobile applications with a middleware system in publish-subscribe paradigm. In the 3rd Workshop on applications and services in wireless networks, Bern, Switzerland, July
20. Cugola GH, Jacobsen A (2002) Using publish/subscribe middleware for mobile systems. In the ACM SIGMOBILE mobile computing and communications review. ACM Press, New York, 6(4):25–33 USA, October
21. Cilia M, Fiege L, Haul C, Zeidler A, Buchmann AP (2003) Looking into the past: enhancing mobile publish/subscribe middleware. In: Proceedings of the 2nd international workshop on distributed event-based systems, San Diego, CA, USA, June
22. Levis GP, Culler D, Brewer E (2003) nesC 1.1 Language reference manual", In: TinyOS documentation site, available in http://today.cs.berkeley.edu/tos/tinyos-1.x/doc/nesc/ref.pdf, May
23. Buonadonna P, Hill J, Culler D (2001) Active message communication for tiny networked sensors. In: Proceedings of the 20th annual joint conference of the IEEE computer and communications societies, Anchorage, Alaska, USA, April
24. Krishnamachari B, Estrin D, Wicker S (2002) Modelling data centric routing in wireless sensor networks. In: Proceedings of the 21th annual joint conference of the IEEE computer and communications societies, New York, USA, June
25. Madden SR, Franklin MJ, Hellerstein JM, Hong W (2002) TAG: a tiny aggregation service for ad-hoc sensor networks. In: Proceedings of the symposium on operating systems design and implementation, Boston, MA, USA, December
26. Yao Y, Gehrke J (2002) The Cougar approach to in-network query processing in sensor networks. In: Proceedings of the ACM SIGMOD international conference on management of data, Madison, Wisconsin, USA, September
27. Levis P (2003) Ad-hoc routing component architecture", in the TinyOS documentation site, available in http://today.cs.berkeley.edu/tos/tinyos-1.x/doc/ad-hoc.pdf, February