SensorBus: A Middleware Model for Wireless Sensor Networks

Admilson R. L. Ribeiro
Dept. of Electrical Engineering and Computing
Federal University of Pará
01, Augusto Correa Street
P.O.Box 8619 – Belém PA, Brazil
ZIP 66075-900
+55-91-3183-1302
admilson@ufpa.br

Fabio C.S. Silva
IESAM Computer Engineering
Belem PA, Brazil
1148, Jose Malcher Av.
ZIP 66055-200 – Belem PA, Brazil
+55-91-4005-5480
fcs@prof.iesam-pa.edu.br

Lilian C. Freitas
Dept. of Electrical Engineering and Computing
Federal University of Pará
01, Augusto Correa Street
P.O.Box 8619 – Belém PA, Brazil
ZIP 66075-900
+55-91-3183-1302
liliancf@ufpa.br

João Crisóstomo Costa
Dept. of Electrical Engineering and Computing
Federal University of Pará
01, Augusto Correa Street
P.O.Box 8619 – Belém PA, Brazil
ZIP 66075-900
+55-91-3183-1302
jweyl@ufpa.br

Carlos R. Francês
Dept. of Electrical Engineering and Computing
Federal University of Pará
01, Augusto Correa Street
P.O.Box 8619 – Belém PA, Brazil
ZIP 66075-900
+55-91-31-1302
rfrances@ufpa.br

ABSTRACT

The use of middleware eases the development of distributed applications by abstracting the intricacies (communication and coordination among software components) of the distributed network environment. In wireless sensor networks, this is even trickier because of their specific issues such as addressing, mobility, number of sensors and energy-limited nodes. This paper describes SensorBus, a message-oriented middleware (MOM) model for wireless sensor networks based on the publish-subscribe paradigm and that allows the free exchange of the communication mechanism among sensor nodes allowing as result the capability of using more than one communication mechanism to address the requirements of larger number of applications. We intend to provide a platform which addresses the main characteristics of wireless sensor networks and also allows the development of energy-efficient applications. SensorBus incorporates constraint and query languages which will aid the development of interactive applications. It intends with the utilization of filters reduces data movement minimizing the energy consumption of nodes.

Keywords

Middleware, wireless sensor networks, publish-subscribe paradigm.

1. INTRODUCTION

Recent advances in wireless networking technology, low-power digital circuits, sensing materials and Micro Electro-Mechanical Systems (MEMS) opened up the possibility of building small sensor devices capable of data processing, remote sensing and wireless communication. When several small sensors are scattered and linked over an area we may call this arrangement a “Sensor Network”. These networks can be used for collecting and analyzing data from the physical environment. More specifically, sensor networks are comprised of hundreds or even thousands of heterogeneous sensor nodes exchanging information to perform distributed sensing and collaborative data processing [1].

From a functional perspective sensor networks behave like distributed systems with many different types of sensor nodes. Given the diversity of node functionality and the size of these networks it is important for a user to be able to program and manage the distributed applications that perform the information gathering. A programmer may develop these applications using operating system primitives. This kind of procedure, however, brings another level of complexity to the programmer, in which he not only has to deal with low-level primitives but he will also have to treat issues concerning communication and coordination among software components distributed over the network. A much friendlier approach is the utilization of a middleware in order to provide higher-level primitives to hide issues concerning the distributed environment.

Traditional middleware is not suited to this task because of the characteristics of wireless networks. For example, conventional middleware designed for wired networks raises exceptions when they do not find a specific component, but this situation is much more like the standard than the exception in wireless environments. The lower bandwidth available for wireless networks requires optimizing the transport of data and this is not considered in conventional middleware. The coordination primitives of these middleware products do not take into account the frequent disconnections that happen in wireless networks. Another problem is the size and computing requirements of these middleware products; they are often too large and too heavy to be running in a device with so few resources. Finally, the transparency level provided is not sufficient enough because the

LANC'05, October 10-12, 2005, Cali, Colombia

application running in such devices needs information about the execution context to better adapt itself.

A series of new middleware environments were proposed to deal with the requirements imposed by the wireless environment [2]. Middleware products based on computing reflection are designed to be light (concerning the computing power required to run) and easily configurable. Middleware based on tuple space were proposed to address the problem of frequent disconnections, and present a more natural way to deal with asynchronous communication. Context-aware middleware includes the ability of an application to access its information context (context-awareness). These proposals addressed adequately the issues brought by the mobile networks, but are not well suited to support the specific requirements of the target applications used or to be used in wireless sensor networks because they are designed to support traditional client-server applications used in regular (wired) environments.

Wireless sensor networks are very similar to conventional wireless networks; including energy-limited nodes, low bandwidth and communication channels more prone to errors. However, communication in wireless sensor nets differs from the end-to-end connections often necessary in usual networks [1]. In other words, the function of the network is to report information considering the phenomenon to the observer who is not necessarily aware that the sensor infrastructure is being used as a means of communication. In addition, energy is much more limited in sensor networks than in other types of wireless nets due to the nature of the sensor devices and the difficulty of reloading batteries in hostile regions. Some works have shown that the execution of 3000 instructions costs the same amount of energy necessary to send 1-bit of data over 100 meters via radio [3]. Those studies indicate that we must prioritize computing over communications.

The communication issues are addressed in the several routing protocols proposed for wireless sensor nets. The communication model allows other ways of addressing the sensor nodes besides single addressing. The sensor nodes can be addressed by their own attributes or by attributes extracted from the physical environment (attribute-based naming). The sharp limitation of energy demands that sensor nodes actively take part in the processing and dissemination of information in order to save as much energy as possible. Although the majority of the protocols reviewed are efficient in saving energy, they differ in addressing capabilities. Some of them utilize single addressing [4] while others utilize attribute-based naming [5]. Thus, each type of application requires an adaptation of the communication mechanism to address specific application issues.

Trying to overcome these problems this paper proposes SensorBus, a message oriented middleware for sensor networks allowing the free exchanging of the communication mechanism among sensor nodes. We propose a platform that provides facilities for the development of energy-efficient applications and that also addresses the key characteristics of sensor networks. This type of middleware should be suited to perform environmental monitoring where single addressing is demanded (small areas) as well as where attribute-based naming is necessary (large areas).

The remainder of this paper is organized as follows: Section 2 describes the type of sensor networks considered in this research; Section 3 presents the target application and explains its requirements; Section 4 broaches the abstractions and mechanisms needed to address the requirements listed on the previous section; Section 5 describes the components of the SensorBus architecture; Section 6 presents the communication architecture and explains the steps needed to develop an application using SensorBus; Section 7 broaches implementing and coding issue; Section 8 presents the related works, and finally Section 9 concludes the paper.

2. ASSUMPTIONS

Most of the algorithms catalogued in the sensor networks literature are hypothetical [1], i.e., they were proposed as an experiment and were not tested in real networks (although many of them were deployed in *testbed* environments). This research is no different. When we speak about wireless sensor networks, we are referring to the projected and experimental designs and deployments discussed in the literature and not to actual instances of wireless sensor networks deployed in the field.

Differently from the real settings, the *testbed* environments are built and organized focusing on the network features one wants to observe and test. This organization involves three main components: infrastructure, network protocols and applications [6]. The infrastructure is formed of sensor nodes and their topology – the way they were scattered over a determined region. The network protocol is responsible for the creation and maintenance of communication links between sensor nodes and the applications. The applications extract information about a determined phenomenon through the sensor nodes. The following topics introduce in more details the assumptions made considering those aspects.

2.1 Applications

The way in which the applications gather data from the sensor nodes depends on the network design. In the literature, we found that there are four data transfer modes between sensor nodes and applications: continuous, event-oriented, query-oriented and hybrid [6]. In the continuous model, sensor nodes transfer their data continuously at a predefined rate. In the event-oriented model, nodes transfer data only when an event of interest occurs. In the query-oriented model the application is responsible for deciding which events are of interest, thus requesting data about a phenomenon. Lastly, in the hybrid model, the three approaches may coexist together. In this research, we adopt a hybrid approach in the way that it utilizes the query and the event-oriented model as will be shown in the target application presented in Section 3.

2.2 Network Protocol

The performance of the network protocol is influenced by the communications model adopted, the packet data transfer mode and the network mobility. In order to evaluate how a network protocol behaves it is important to take into account these aspects.

Communication in sensor networks is classified in two major categories [6]: application and infrastructure. Application communication consists of the transfer of data obtained by the sensor nodes to the observer. This kind of communication is of

two types: cooperative and non-cooperative. In cooperative mode sensor nodes exchange data among themselves before transmitting the data gathered. In non-cooperative mode, however, sensor nodes do not exchange any kind of information; each one is solely responsible for reporting its collected data. The infrastructure data refers to the information needed to set, maintain and optimize the running network. As the network protocol must support both categories, the SensorBus architecture will not address those issues.

The packet data transfer is a routing issue concerning the network protocol. This routing is divided into three types: flooding, unicast and multicast [6]. In the flooding approach, the sensor node broadcasts its information to neighboring nodes that, in turn, broadcast this information to their neighboring nodes until the information reaches the destination node. Alternatively, the sensor node may transmit its data directly to the observer using unicast multi-hop routing and also might use a cluster-head through one-to-one unicast. Lastly, the multicast approach casts information to predefined groups of nodes. The routing protocol is responsible for treating packet data transfer relieving SensorBus of these issues.

Regarding mobility, sensor networks are divided into static and dynamic [6]. In static nets there is no movement by sensor nodes, the observers or the phenomenon to be studied. Conversely, in dynamic networks the nodes, observers and the phenomenon might well change their locations. This kind of network is further classified by the mobility of its components in dynamic nets with mobile observer, dynamic nets with mobile sensors and dynamic nets with mobile phenomena respectively. In the first, the observer is mobile in relation to the sensors and phenomena; in the second; the sensors are moving with respect to each other and the observer; and in the later; the phenomenon itself is in motion. The routing protocol is also responsible for treating mobility issues, relieving SensorBus of these concerns.

2.3 Infrastructure

As for the infrastructure, the issues to take into consideration are location, access point and sensor node's computing power. The nodes have well-know locations and are to be scattered over a well-defined area. We will assume that all information is transmitted and received by means of a unique access point called the sink node. Despite the fact that, for this model, we will consider all nodes as being the same, there is nothing to prevent one node from having more memory, more energy or more computing power available.

3. ENVIRONMENTAL MONITORING APPLICATIONS

Environmental Monitoring Applications are used to evaluate qualitatively and quantitatively the natural resources of a determined area. These applications collect data, analyze and follow continuously and systematically environmental variables in order to identify current patterns and predict future trends [7]. Environmental monitoring provides information about the factors influencing conservation, preservation, degradation and environmental recovery. One might consider it a tool of evaluation and control.

Wireless sensor networks can be used for performing environmental monitoring in indoor locations such as a building or a house or outdoors locations such as forests, lakes, deserts, rivers, etc. Internal monitoring might be described as tracking the variables in an indoor location. For example, one might deploy an infrared camera to track motion in a room that is supposed to be secure; if motion is detected an internal device might trigger an alarm. Sometimes in order to detect and identify an event, information from more than one sensor might be required. These results are processed and compared with the signature of the event of interest. In outdoor monitoring, there may be thousands of sensors scattered over an area and when an event of interest occurs such as temperature change, moisture change or CO2 increase the sensor might trigger the management events module which in turn sends the observer a signal to notify him or her of the event. Wireless sensors might be useful in a way that can save money in deploying a sensor infrastructure such as described in [8] where the authors were able to decrease the number of sensors needed to monitor forest fires in comparison with a wired model.

In summary, the value of a wireless sensor network relies in its ability to provide information over a large area in reply to the questions put to users. The query mode is the most common approach used. Another approach is the mode in which sensors may remain waiting for some event to happen. By observing these aspects we draw the first requirement (R1) of our middleware model: The system must be able to function in two modes: query-driven and event-oriented.

Depending on the application would be more convenient to access a specific node or a specific property. For example, in internal environmental monitoring, if one wants to know the temperature of a determined room you will have to access the information collected by a specific sensor, thus requiring unique node addressing and identification. On the other hand, in external environmental monitoring, sensor nodes do not need to be uniquely identified, as in this kind of application the purpose is to collect the value of a certain variable in a given area. From that observation we extract the second requirement (R2): The system must be able to address uniquely the sensor nodes and also by attribute (property to be observed).

In some applications the mobility of sensor nodes must be taken into account. For example, sensors scattered over a forest for collecting dampness and temperature data are to be static, i.e. they must not change its geographical location, while that placing sensors in a river's surface for collecting data about its contamination levels characterizes a mobile environment. Thus, the third requirement (R3) of our middleware model is taking into consideration mobility issues.

The sensing coverage area of a given wireless node is smaller than its radio coverage. Besides, sensors operate in noisy environments. To achieve a trustworthy sensory resolution a high density of sensors is required. In some applications the size of the coverage area leads to a great number of sensor nodes. A simple application in the field of environmental monitoring such as surveillance of oceans and forests requires from hundreds to thousands of nodes. In other applications, like internal environmental monitoring, the amount of nodes is limited by the size of the area. Therefore, the fourth requirement (R4) is to take into account the size of the network.

In external environmental monitoring, the nodes are spread in a hostile region, where it is not possible to access them for maintenance. The lifetime of each sensor node depends exclusively on the little available energy for the node. To conserve energy, the speed of the CPU and the bandwidth of the RF channel (Radio Frequency) must be limited. This requirement adds some restrictions in CPU performance, memory size, RF bandwidth and in battery size. In applications where the sensors are not spread in a hostile region it is possible to access them for maintenance and the battery lifetime of each sensor does not become a critical aspect. Finally, the fifth requirement (R5) is to take into consideration the limited energy resources of each sensor node.

4. MECHANISMS AND ABSTRACTIONS

This middleware model is comprised of three mechanisms and one abstraction. The publish-subscribe paradigm is employed as well as constraints and query languages and application filters to meet R1, R2 and R5 requirements. The design patterns abstraction is used to meet R2, R3 and R4 requirements.

4.1 Publish-Subscribe Paradigm

The SensorBus is a Message Oriented Middleware (MOM) that employs the publish-subscribe paradigm. In this approach, a component that generates events (producer) publishes the types of events that will be available to other components (consumers) [9]. The consumer interested in a determined event "subscribes" to this event, receiving from this moment on notifications about the event "subscribed" to. These notifications are sent asynchronously from producers to all interested consumers. The MOM performs the functions of collecting producer's messages, filtering and transforming such messages (when necessary) and routing them to the appropriate consumers.

The publish-subscribe communication is anonymous, asynchronous and multicast. Data are sent and received by asynchronous broadcast messages, based in subject, independent from identity and location of producers and consumers. This kind of communication enlists desirable properties for sensor networks; for example, this model saves energy while a given node does not need to be waiting for a synchronous response to proceed as it is in networks that implements end-to-end connections, increasing the lifetime of the network. Furthermore, as it also implements multicast, a group of sensor might be formed regarding a specific application.

As a consequence, the adoption of the publish-subscribe paradigm meets the R1 requirement, concerning the need for events and the R2 requirement pertaining to attribute addressing. In addition it also meets the R5 requirement related to energy saving.

4.2 Constraint and Query Languages

Constraint and Query languages are used to filter collecting data by specifying restrictions in the values and preferences of the attributes. A statement in these languages is a string that represents an expression.

The constraint language only includes constants (values) and operations over values. Values and operations with integer, float, boolean and *strings* are allowed. The language admits several types of expressions.

- The expressions can be comparative: == (equality), != (inequality), >, >=, <, <=. For instance, Temperature < 36.6 means to consider data where the attribute Temperature is less than 36.6 degrees Celsius.
- The expressions can be boolean: AND, OR, NOT. For example, Temperature >= 26.6 AND Temperature < = 36.6 implies to consider data where the value of the attribute Temperature is between 26.6 and 36.6 degrees Celsius.
- The expressions can be numerical with the mathematical operators + (addition), - (subtraction), * (multiplication) and / (division).

The query language has its syntax based on a subgroup of the conditional expression syntax SQL92 [10]. It is an extension of the constraints language with new functions. This new language embodies identifiers that can hold a constant value. A mapping between identifiers and values is required. In the evaluation of an expression, the occurrence of an identifier is replaced by its associated value. The addition of new operators (between, like, in, is, escape) allows submitting queries similar to those used in databases compliant with SQL92. For example, queries of the type -- Temperature between 26.6 and 36.6 -- are possible.

The constraint and query languages are intended to ease the work programming of online applications. This type of application access the information sent in real-time by the sensor nodes. Thus, it completes the attendance of the R1 requirement on the way of operation for query.

4.3 Application Filters

Filters are application specific software modules that deal with diffusion and data processing [11]. Filters are provided before deploying a sensor network. Each filter is specified using a list of attributes to make possible the matching with the incoming data.

Filters are used to make internal aggregation of data, collaborative signals processing, caching and tasks that control the data flow in the sensor network [11]. In SensorBus, filters will be used to limit the data flow in the network. A filter can be designed to restrict the range of values of a determined attribute, for example the application requires that the attribute Temperature has values ranging between 20 and 30 degrees Celsius, the values outside this particular range are of no interest. The filtering process discards the unnecessary data reducing the flow between the nodes. This decrease reduces the consumption of energy in sensor nodes. Thus, it completes the attendance of the R5 requirement about the energy saving of the sensor nodes.

4.4 Design Patterns

Design patterns are descriptions of objects and communicating classes that are customized to solve a general design problem within a particular context [12]. It describes commonly recurring design architectures extracted from the experience of one or more domain specialists. A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design [12]. We make use of design patterns in SensorBus project and the types we have utilized are as follows:

The *Observer pattern:* Defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically. We utilize this pattern to implement the publish-subscribe mechanism.

The *Interpreter pattern:* Defines a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language. We make use of this pattern to implement the constraint and query language.

The *Facade pattern:* Defines a unified (higher-level) interface to a set of interfaces in a subsystem that makes the subsystem easier to use. We use this pattern to implement the middleware high-level primitives which will be available to developers.

The *Mediator* pattern: Defines an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

The *Adapter pattern:* Converts the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

The *Router pattern:* Decouples multiple sources of input from multiple sources of output to route data correctly without blocking.

The design patterns *Mediator*, *Adapter* e *Router* are utilized to implement the middleware message bus. The exchangeable communication mechanism was written using these patterns. This mechanism allows the utilization of any routing protocol designed for sensor networks meeting as a result the requirements R2, R3 and R4.

5. SENSORBUS ARCHITECTURE

SensorBus is comprised of the following elements: an application service, a message service and a context service as shown in Figure 1.

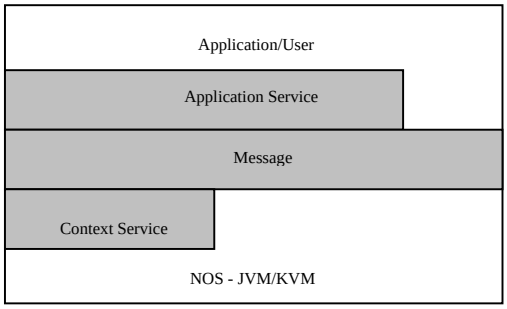

Figure 1. Middleware architecture.

The following sections present each one of the services mentioned by the means of UML (*Unified Modeling Language*) component diagrams [13].

5.1 Application Service

The application service provides Application Programming Interface (API) which simplifies application development. This service is comprised of three components as shown in Figure 2:

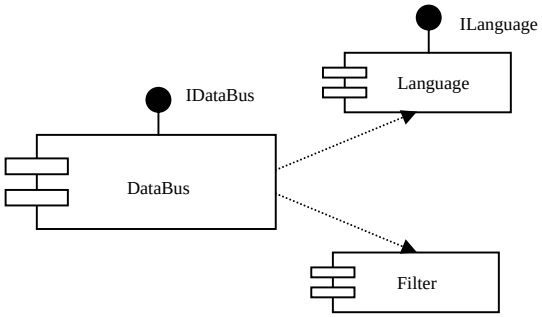

Figure 2. Application Service Architecture.

DataBus: component providing a set of operations relating to bus communication for consumers and producers. These operations include: Announcement of data item (producer); to find a data item (consumer); Announcement of data change (consumer); Exclude data item (producer).

Filter: component providing a set of operations relating to data filtering.

Language: component that implements the commands and the constraint and query language interpreter.

5.2 Message Service

Message service is responsible for providing communication and coordination for the distributed components, abstracting the developer from these issues. This service also comprises three components as is shown in figure 3:

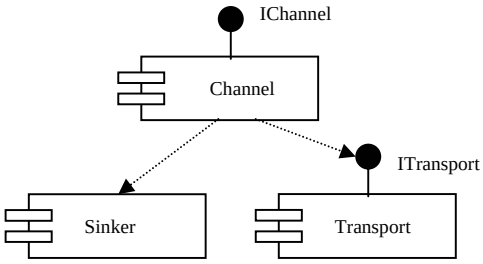

Figure 3. Message Service Architecture.

Channel: Component designed to deal with the specific transport implementations. Each instance of Channel represents a simple system channel. The component Channel maintains the global state information about the availability of channels and is also responsible for exchanging channel's messages to the transport implementation and vice versa.

Transport: The communication among the nodes is made through a specific transport implementation such as *sockets*. Each transport implementation communicates through a channel with a message exchange server called *Sinker*. All transport implementations have a common interface which is called *ITransport*.

Sinker: Component responsible for routing messages among instances of transport implementation, each instance corresponding to an instance of Channel.

5.3 Context Service

Inherently, an application running on a wireless sensor network needs to capture information from the execution context, for example, battery level, memory availability, bandwidth, location, application specific information such as temperature and pressure, etc. The middleware gets this information by interacting with several heterogeneous sensors; for example, the level of energy remaining on batteries can be obtained by executing an operating system primitive, location can be acquired from various communications technology such as GPS, infrared and RF. This work does not take into consideration how the context sensing is executed; it is assumed that each sensor provides an interface so the middleware can use it to get the value of the resource of interest.

The context service manages the heterogeneous sensors that collect information from the environment. For each resource the middleware manages, there is an adapter that interacts with the physical sensor, processes its information thus obtaining the information demanded by the application. Only resource adapters that are necessary to the running application will be loaded to avoid unnecessary spending of the node's scarce computing power. Figure 4 shows an energy adapter interacting with an energy sensor (an operating system primitive, in this example).

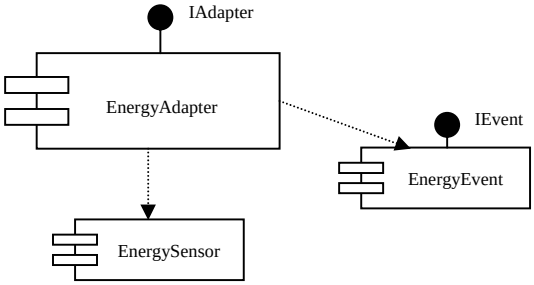

Figure 4. Context Service Architecture.

6. MIDDLEWARE ARCHITECTURE EXAMPLE

Figure 5 shows the sensor network communication architecture. Each node in the field has the ability to collect data and send it to the next sink node. The sink node can be a mobile node acting as a data source or a fixed host computer (a PC). The user node connects with the sink node through a conventional wireless LAN (e.g. IEEE 802.11x).

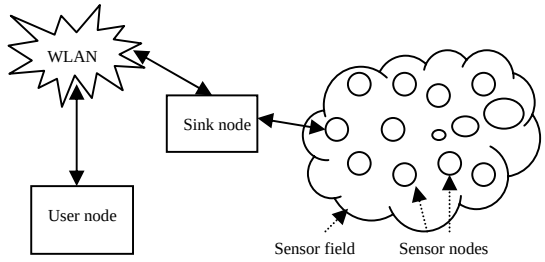

Figure 5. Communication Architecture.

The services and components of SensorBus are distributed in three distinct types of sensor nodes. The components DataBus, Language, Channel and Transport are in the user node. The Sinker component is in the sink node. The sensor nodes contains the Channel and Transport components while filter component and context service will only be loaded if the application requires energy management and other resources such as memory and bandwidth.

The development of an application using SensorBus consists in coding the parts for the producer and consumer. The consumer code runs in the user machine while the producer code runs in sensor nodes. The minimum steps required for the use of SensorBus are as follows:

1. Create a new DataBus instance. A new transport implementation is created by identifying a specific Sinker;
2. Instantiate a producer or a consumer;
3. Instantiate a "Channel" entity;
4. Register the just created producer or consumer for the channel; and
5. The producer generates data items and places them into Channel while the consumer finds and "crunches" those data.

SensorBus offers other functions that might be implemented, such as listing the available channels, adding new channels and stop receiving new channels.

The producer sensor code has to be implemented before setup of the network. If it is not possible to retrieve the sensor for maintenance, the attributes of the data sent will always be the same. To overcome this obstacle the constraint and query languages are used to add new queries that had not been initially foreseen. These queries are sent by the interested consumer (client) in the form of messages.

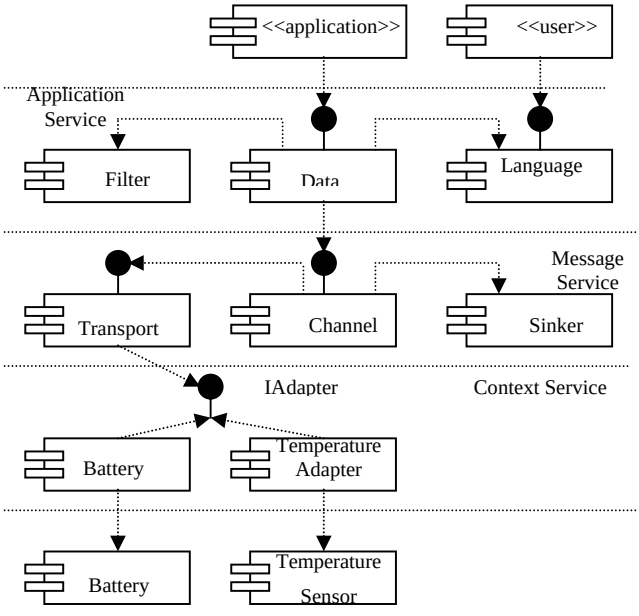

Figure 6. Middleware architecture example.

Filters are as well implemented in nodes. Soon after a producer is instantiated, a filter is also instantiated and registered for a new channel.

Figure 6 shows the components that may be active in a given moment. Although most of the components are the same for a given application, different settings may occur on the context service. The figure shows only two adapters running at the same time and interacting with its associated sensors (temperature and battery). Distinct sensors can be used depending on the physical measurement to be taken and the type of computing resource to be managed.

7. IMPLEMENTATION ISSUES

The *testbed* setup for SensorBus evaluation consists of Intel-based equipment equipped with 802.11b cards. The sink node is a centrino-based Dell Lattitude notebook, the sensor nodes are deployed in handheld computers HP iPAQ running Linux operating system on Intel XScale processor. The sensor nodes are placed at various locations in the Electrical Engineering Department Building (about 40m×60m) at Federal University of Pará. Linksys Wireless LAN cards are used working in the DCF mode with a channel bandwidth of 11Mbps. In the building, there is interference from IEEE 802.11 access points (AP) and other electronic devices.

7.1 Working Prototype

For our working prototype, we have chosen the Java platform as our implementation technology because of its broad installed base and to ensure compatibility with most hardware platforms. The KVM (*Kilobyte Virtual Machine*) [14] is being used due to its freely available source-code and its designed targeting towards small limited-resources devices similar to the sensor nodes of this work. The SensorBus API and its constraint and query language are being coded as Java classes. Due to issues regarding efficiency, the code that will run in sensor nodes is being implemented as native code.

An object *serialization* mechanism was implemented because KVM does not support this facility. *Serialization* mechanism converts an object and its state into a byte stream allowing this object to be moved over a network or persisted in a local file system. Object recovery is performed through another mechanism called *deserialization*. Other Java technologies as J2SE – *Java 2 Standard Edition* utilize this kind of facility to support the encoding of objects into a stream of bytes while protecting private and transient data. Serialization is used in distributed programming through sockets or *Remote Method Invocation* (RMI). We have coded a semiautomatic *serialization* in order to store the state of the objects. To achieve this, we had to define a series of new interfaces and classes.

One of the most critical problems with *serialization* is security because when an object is converted in a byte stream any attacker equipped with properly sniffer software can intercept and access it; in this case even private attributes can be accessed without any special technique. To tackle this issue, secure protocols as HTTPS (*Secure HyperText Transport Protocol*) or serialization encryption can be used.

7.2 Simulation

Having implemented this working prototype, this research now intends to do performance evaluation by using the very well known tool NS – *Network Simulator* [15]. We plan to integrate the SensorBus middleware with NS in a way that the sensor nodes will plug into NS in order to provide real data for feeding the simulator model. To do so, an execution environment will be added to the simulator. This environment will run as a sole UNIX process and will be plugged to the NS protocol stack through a sensor agent.

The sensor agent is actually a NS agent responsible for connecting an execution environment instance to the NS protocol stack. The communication takes place through a pair of UDP (*User Datagram Protocol*) sockets. Incoming packets are encapsulated in NS packets and transmitted through the simulated sensor network. Parameters that need to be known to the protocol stack are placed in the header of the NS packet while the rest of the information is added to the payload of the NS packet. Similarly, outgoing packets are retrieved from the NS packets and sent to the execution environment to be processed.

It will be necessary to provide a mechanism to synchronize the execution and simulation environment since they run in distinct times. Simulations will be performed using a NS *Directed Diffusion* transport implementation [5] for wireless sensor networks.

8. RELATED WORKS

In [16] an overall description of the challenges involving middleware for wireless sensor networks is presented focusing on the restraint aspects of these systems.

Cougar [17] and SINA [18], *Sensor Information Networking Architecture*, provide a distributed database interface for wireless sensor networks that use a query language to allow applications to run monitoring functions. Cougar manages the power by distributing the query among the sensor nodes to minimize energy required in data gathering. SINA adds low-level mechanisms to build hierarchical clustering of sensors aiming at efficient data aggregation and also provides protocols which limit the rebroadcast of similar information to neighbor's nodes.

AutoSec [19], *Automatic Service Composition*, manages resources of the sensor networks by providing access control for applications to ensure quality of service. This approach is very similar to conventional middleware technology but the techniques to collect resource information are suitable for wireless sensor networks.

DSWare [20] provides service abstraction similar to AutoSec, but instead of having a service provided by only one sensor node, the service is supplied by a group of neighbor's nodes.

S*mart Messages* Project [21] proposes a distributed computing model based on migration of executing units. *Smart messages* are migratory units containing data and code. The goal of *Smart Messages Project* is to develop a computing model and systems architecture to Networks Embedded Systems (NES).

EnviroTrack [22] is a middleware for object-based distributed systems that lifts the abstraction level for programming

environmental monitoring applications. It contains mechanisms that abstract groups of sensors into logical objects.

Impala [23] exploits mobile code techniques to alter the middleware's functionality running on a sensor node. The key to energy-efficient management in Impala is that applications are as modular and concise as possible so little changes demands fewer energy resources.

MiLAN [24] was developed to allow dynamic network setup to meet the applications performance requirements. The applications represent its requests by the means of specialized graphics which incorporates changes due to applications needs.

In [25], an adaptative middleware is proposed to explore the commitment between resource spending and quality during information collecting. The main goal is to decrease the transmissions among sensor nodes without compromising the overall result.

Every one of those middleware proposals is designed to make efficient use of wireless sensor networks; they do not support free exchange of the transport mechanism. More specifically, most of those approaches are not capable of altering the routing protocol to meet different application requirements.

9. CONCLUDING REMARKS

As was demonstrated, application development is closely related to wireless sensor network design. Each communication mechanism provided by a determined routing protocol is application specific, e.g. it is designed to meet some application specific requirement. We suggest that the utility of the middleware for wireless sensor networks is supported by decoupling the communication mechanism from the programming interfaces and also by capability of using more than one communication mechanism to address the requirements of larger number of applications. We have shown that SensorBus, a sensor network middleware that we are developing to meet these goals, can aid the development of different types of sensor network applications.

10. REFERENCES

[1] P. Rentala, R. Musunuri, S. Gandham and U. Saxena, *Survey on Sensor Networks*, Technical Report, University of Texas, Dept. of Computer Science, 2002.

[2] G. -C. Roman, A. L. Murphy, and G. P. Picco, Software Engineering for Mobility: A Roadmap. In The Future of Software Engineering – *22nd Int. Conf. On Software Engineering (ICSE2000)*, pages 243-258. ACM Press, May 2000.

[3] J. Pottie and W. J. Kaiser, Embedding the internet wireless integrated network sensors, *Communications of the ACM*, vol. 43, no. 5, pp. 51-58, May 2000.

[4] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, Energy-efficient communication protocol for wireless micro sensor networks. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, Pages 3005-3014, 2000.

[5] C. Intanagonwiwat, R. Govindan, and D. Estrin, Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 56-67, Boston, MA, USA, Aug. 2000.

[6] T. Sameer, N. B. Abu-Ghazaleh and Heinzelman W. A Taxonomy of Wireless Micro-Sensor Network Models. *Mobile Computing and Communications Review*, Volume 1, Number 2, 2003.

[7] Guia de Chefe – Brazilian Institute of Environment (IBAMA) *http://www2.ibama.gov.br/unidades/guiadechefe/guia/t-1corpo.htm.* December, 2004.

[8] B. C. Arrue, A. Ollero e J. R. M. de DIOS, An intelligent system for false alarm reduction in infrared forest-fire detection, *IEEE Intelligent Systems*, vol. 15, pp. 64-73, 2000.

[9] G. Couloris, J. Dollimore, e T. Kindberg *Distributed Systems: Concepts and Design*. Third edition. Addison-Wesley, 2001.

[10] SQL92 – Database Language SQL – July 30, 1992. *http://www.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/sql/sql1992.txt*

[11] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146-159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[12] E. Gamma, R. Helm, R. Johnson e J. Vlissides, *Design Patterns*. Addison-Wesley, 1995.

[13] J. Rumbaugh, I. Jacobson and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1998.

[14] KVM – The K Virtual Machine Specification. *http://java.sun.com/products/kvm/*, August 2004.

[15] UCB/LBNL/VINT Network Simulator – NS (Version 2). *http//www.isi.edu./nsnam/ns/*, August 2004.

[16] K. Römer, O. Kasten and F. Mattern. Middleware Challenges for Wireless Sensor Networks. *Mobile Computing and Communications Review*, volume 6, number 2, 2002.

[17] P. Bonnet, J. Gehrke and P. Seshadri. Querying the Physycal World. *IEEE Personal Communication*, 7:10-15, October 2000.

[18] C. Srisathapornphat, C. Jaikaeo and C. Shen. Sensor Information Networking Architecture, *International Workshop on Pervasive Computing (IWPC00)*, Toronto Canada, August 2000.

[19] Q. Han and N. Venkatasubramanian. AutoSec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2(7), 2001.

[20] S. Li, S. Son, and J. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, April 2003.

[21] Smart Messages project. March, 2003. *http://discolab.rutgers.edu/sm.*

[22] T. Abdelzaher, B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic and A. Wood. *EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks.* Technical report, Department of Computer Science, University of Virginia, 2003.

[23] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP´03)*, June 2003

[24] A. Murphy and W. Heinzelman, *MiLan: Middleware linking applications and networks*, Technical Report TR-795, University of Rochester, 2002.

[25] X. Yu, K. Niyogi, S. Mehrotra and N. Venkatasubramanian, Adaptive middleware for distributed sensor networks, *IEEE Distributed Systems Online*, May 2003.