μDDS: A MIDDLEWARE FOR REAL-TIME WIRELESS EMBEDDED SYSTEMS

Apolinar González, Walter Mata, Luis Villaseñor, Raúl Aquino, Magaly Chávez, Alfons Crespo

Abstract— **A Wireless Sensor Network (WSN) is formed by a large quantity of small devices with certain computing power, wireless communication and sensing capabilities. These types of networks have become popular as they have been developed for applications which can carry out a vast quantity of tasks, including home and building monitoring, object tracking, precision agriculture, military applications, disaster recovery, among others. For this type of applications a middleware is used in software systems to bridge the gap between the application and the underlying operating system and networks. As a result, a middleware system can facilitate the development of applications and is designed to provide common services to the applications. The development of a middleware for sensor networks presents several challenges due to the limited computational resources and energy of the different nodes. This work is related with the design, implementation and test of a micro middleware for WSN with real-time (i.e. temporal) restrictions; the proposal incorporates characteristics of a message oriented middleware thus allowing the applications to communicate by employing the publish/subscribe model. Experimental evaluation shows that the proposed middleware provides a stable and timely service for providing different QoS levels.**

I. Introduction

A Wireless Sensor Network (WSN) is formed by a large quantity of small devices with certain computational power, wireless communication and sensing capabilities [1]. The sensor nodes are generally disseminated on the region of study, where each sensor node is responsible for extracting data of the phenomenon of interest, such as, humidity, temperature, pressure, brightness, etc. The sensor nodes are capable of processing and sending the collected data to one or more sinks, which are in charge of transmitting the data to the end user application.

The WSN have become popular as they have been developed for applications which can carry out a vast quantity of tasks, including home and building monitoring, object tracking, precision agriculture, military applications, and disasters recovery [1]-[5]. The paradigm of these nets differs from the habitual ones which are based on the information management in that WSNs have knowledge over what is happening in the environment where they are deployed, thus the decision making process depends on the analysis of the sensed variable along the area of interest.

With respect to the development of applications for WNSs, a middleware can be used to bridge the gap between the application and the underlying operating systems and networks [6],[7]. One of the basic purposes of any middleware is to satisfy the application requirements; in this work the middleware takes into account the specific features of a WSN like the restrictions in energy, communication and computing power [8]; consequently the design and development of the sensor networks is highly related to specific resources like battery, memory and processor capabilities, as well as, the communication models and the application requirements.

II. Related Works

There are several middleware proposals in the literature, some of the most representative are described in this section. Cougar [9] is focused on a model based on consults, where the sensed data is considered to be in a virtual relational database. Mate [10] is a small virtual machine communication centered approach executed on TinyOS [11] and the developers, by means of the use of this architecture, can change in a simple way the set of instructions, the execution of events and the subsystems of the virtual machines. Impala [12] is a middleware for the ZebraNet project, it is composed of 2 layers: the upper layer that contains all the application protocols as well as the programs for ZebraNet, while the lower layer contains three middleware agents: the Application Adapter, the Application Updater, and the Event Filter. Garnet [13] presents an architectural framework that provides a data stream centric abstraction. In a fixed network the data is gathered by the applications which use the typical mechanisms of advertising, discovery, registration, authentication and publish/subscribe to identify, subscribe to and receive the data streams of interest. It has some components which are receivers, sensors/actuator, filtering and dispatching services, consumer processes and services, the Super coordinator, etc. MiLAN [14] sits on top of multiple physical networks, and has an abstraction layer that allows network-specific plugins to convert the MiLAN commands

Manuscript received August, 2009. This work was developed as a part of the D2ARS Project supported by CYTED. UNESCO code 120325;330417;120314;120305.

W. A., A. G., R. A., M. C., are with the Universidad de Colima, Av. Universidad, Col. Las Víboras, Colima, Col., México. C.P. 28040; (e-mail: {wmata, apogon, aquinor, maggilyly}@ucol.mx).

L. V. is with the CICESE Research Center. Ensenada, B.C. Mexico (e-mail: luisvi@cicese.mx)

A. C. is with the Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Camino de Vera S/N, Valencia, España. (email: alfons@disca.upv.es)

to protocols-specific commands which are transmitted through the usual network protocol stack, this plugins are important as they help to determine which sets of nodes satisfy the QoS requirements of the application. Finally Mires [15] presents a publish/subscribe model, and incorporates two additional services: routing component and additional services. The communication between the nodes is given in three phases. First, the nodes in the network announce their available topics, such as, temperature or humidity which are collected from local sensors. Second, the advertised messages are routed to the sink node using a multi-hop routing algorithm and the user application can connect to these nodes to monitor the desired topics. Finally, subscribe messages are broadcast down to the network nodes. Mires is located on top of the OS, encapsulating its interfaces and providing higher-level services to the node application.

III. An Overview Of Data Distribution Service For Real-Time Systems

The OMG Data Distribution Service (DDS) specification [16] standardizes the software application programming interface (API), where a distributed application can use the publish/subscribe communication mechanism which is centered in the data. It is based on Model Driven Architecture (MDA) [17],[18], which defines a Platform Independent Model (PIM) that is a view of a system from the platform independent viewpoint, as a result the middleware developers can derive any Platform Specific Model (PSM) which can be adjusted to the application requirements; thus allowing the construction of different DDS implementations dedicated to very specific needs [16].

A. DDS Conceptual Model

DDS introduces two levels of interfaces, which are:

- DCPS (Data-Centric Publish/Subscribe), a low-level mandatory API that provides the functionality required for an application to publish and subscribe the values of data objects. This layer provides support for 21 QoS policies as we will see later;
- DLRL (Data Local Reconstruction Layer), an optional high-level API that allows a simple integration of the Service into the application layer.

In summary we can say that the advantages of this infrastructure according to [19] are:

- It is based on a simple publish/subscribe communication paradigm;
- it has a flexible and adaptive architecture that supports auto-discovery of new stale endpoint applications;
- the low overhead can be used with high-performance systems;
- it has a deterministic data delivery;
- it is dynamically scalable;
- it provides an efficient use of transport bandwidth
- it supports one to one, one to many, many to one and many to many communications;
- and it has a large number of configuration parameters that provide to the developers a complete control of each message in the system.

The information flows with the aid of the constructors as it is shown in figure 1. The Publisher and DataWriter are on the sending side while the Subscriber and DataReader are on the receiving side. The Topics are used to provide the basic connection between Publishers and Subscribers. The Topic of a given Publisher on one node must match the Topic of an associated Subscriber on any other node. If the Topic does not match, communication will not take place. The Publisher is responsible for the distribution of the different data types, and the DataWriter is used to communicate to a Publisher the existence and value of data. Meanwhile the Subscriber is responsible for receiving the publishing data and making it available to the receiving application and the DataReader is used to access the received data [16].

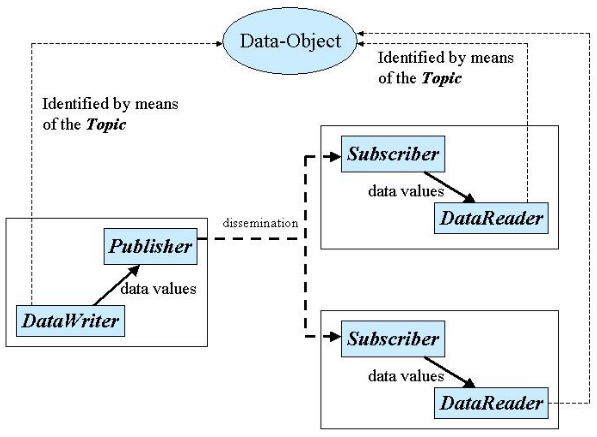

Fig. 1. DDS Entities [19]

B. Quality of service in DDS

One of the important aspects to consider is the Quality of Service (QoS), which is a concept used to specify certain behavior of a service. QoS provide the ability to control and limit the use of resources like network bandwidth, memory, reliability, timeliness, and persistence, among others.

The DDS QoS model is a set of classes which are derived from QoSPolicy. DDS provides USER_DATA QoS policy, TOPIC_DATA QoS policy, DURABILITY QoS policy, DEADLINE QoS policy and other policies. Further details regarding these policies can be studied in [16].

IV. Architectural Overview of a Middleware For Real-Time Wireless Embedded Systems

The proposed architecture is shown in the figure 2, where the PaRTiKle real-time OS is executed directly on the ARM or x86 hardware; figures 3 and 4 show how different kinds of nodes can communicate with other devices using a ZigBee communication module; The µDDS middleware sits between the node and user applications, and the PaRTiKle OS.

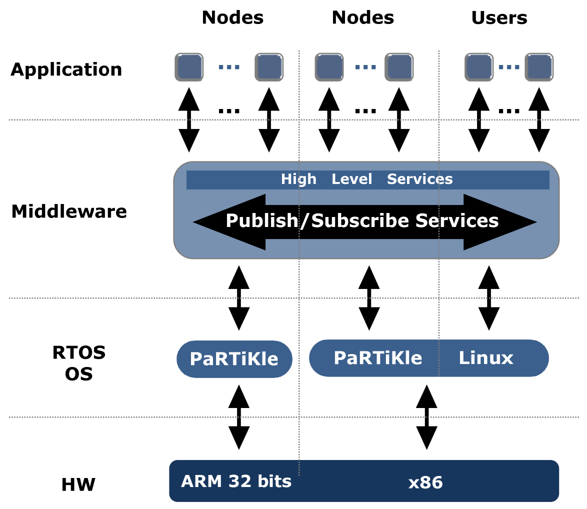

Fig. 2. Global Architecture

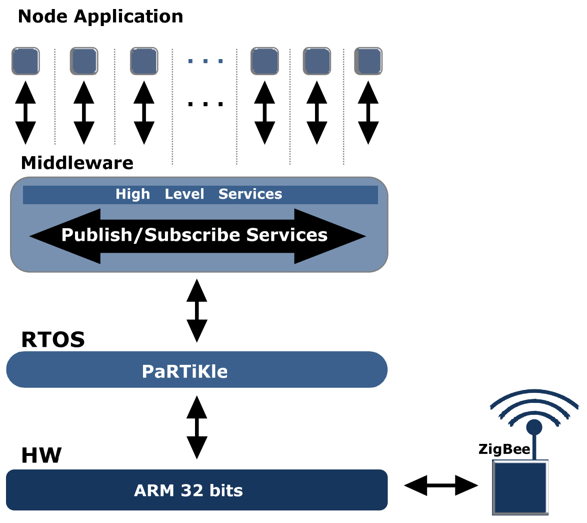

Fig. 3. Communication Node Architecture

A. PaRTiKle OS Architecture

PaRTiKle [18]-[23] is a recent embedded real-time operating system designed to be compatible with the POSIX 5.1 standard. PaRTiKle has been designed bearing the following ideas in mind:

- it must be as portable, configurable and maintainable as possible.
- it must support multiple execution environments, thus allowing to execute the same application code (without any modification) under different environments, such as, in a bare machine, a Linux regular process and as a hypervisor domain.
- it must support multiple programming languages; currently PaRTiKle supports Ada, C, C++ and Java

PaRTiKle has been designed to support applications with real-time requirements, providing features such as full preemptability, minimal interrupt latencies, and all the necessary synchronization primitives, scheduling policies, and interrupt handling mechanisms needed for this type of applications. Figure 5 shows the PaRTiKle architecture that has been designed as a real kernel with a clean and well defined separation between kernel and application execution spaces. All kernel services are provided via a single entry point, which improves the robustness and also greatly simplifies the work to port PaRTiKle to other architectures and environments.

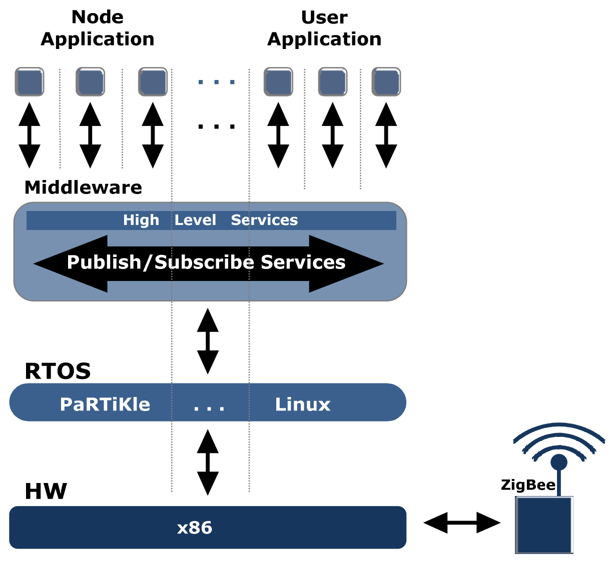

Fig. 4. Advanced Node Architecture

B. µDDS: A Middleware for real-time wireless embedded systems

µDDS is a publish/subscribe middleware for real-time wireless embedded systems based on DDS specification and implements a subset of standard interfaces for event subscriptions and publication to be used by applications. Applications implemented on top of µDDS can disseminate and collect data through a publish/subscribe interface provided by the middleware. Different routing protocols can be used to implement the overlay network; the middleware is currently implemented on 802.15.4 standard devices which can support star, tree and mesh topologies.

V. Application Development with µDDS

Figure 6 shows the development model of a µDDS application on PaRTiKle platform. There are three main elements of the development model, µDDS Middleware and µDDS library, user application and PaRTiKle's Kernel. To build a µDDS application running on PaRTiKle, the figure 6 sketches how it can do it, PaRTiKle provides a bash script, named mkkernel, for ease of building process, basically the steps performed by this script are: links the application with µDDS middleware, then the script links the resulting object together with the kernel object to create the executable containing all the components (.prtk).

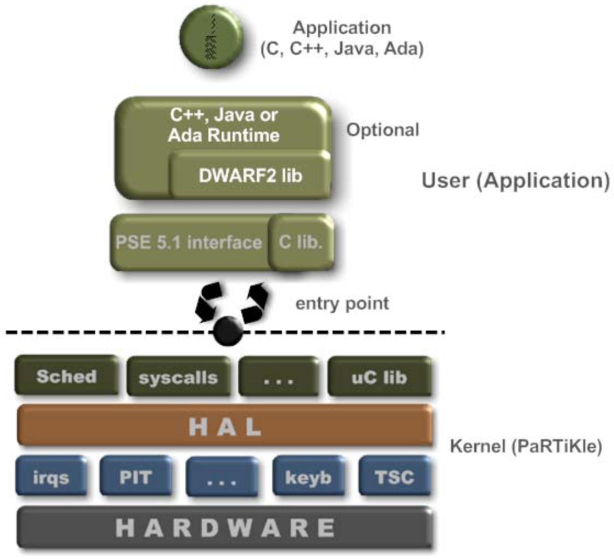

Fig. 5. PaRTiKle Architecture

The mkkernel script requires the following parameters:

$ mkkernel -f <output> <file1.o> [<file2.o> ...]

where <output> is the name of the target executable once the process of building the application has concluded, and <file1.o>, <file2.o>, etc., are the object files obtained when the application is compiled using GCC with the option -c. The steps performed by this script are:

1. It links the application against the user "C" library and the suitable run-time (the run-time is selected depending on the language used to implement the application).
2. It turns every application's symbol into a local symbol, except user entry point.
3. Eventually, the script links the resulting object file together with the kernel object file to create the executive (*.prtk).

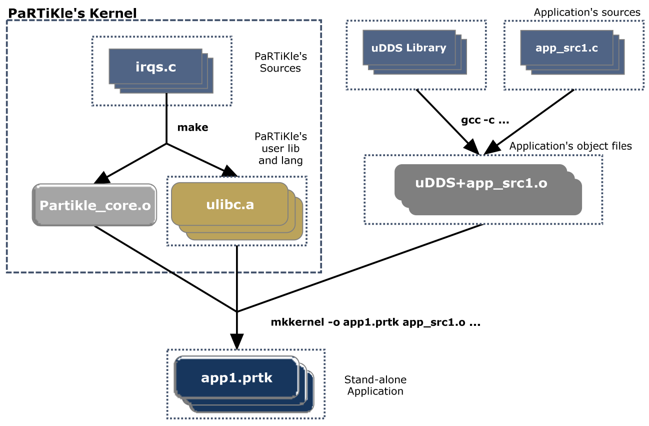

Fig. 6. Building process of a µDDS application

For development model in Real-Time Java a compiled application includes the GCJ runtime with the javax.realtime classes and the native methods as shown in the figure 7. GCJ, which is the GCC Java compiler, links with the libgcj by default, which includes a complete java runtime on the order of some megabytes with a set of characteristics that are not necessary for small platforms. The idea is to remove the any functionality which is not necessary for safety critical systems and also not necessary for real-time systems with hard timing restrictions.

We started from scratch, copying the source code necessary to the directories where we had installed GCJ on PaRTiKle. The code was compiled generating a new, reduced, version of libgcj. The size obtained is in the range of hundreds of kilobytes for the complete application, which is composed of the application code (code developed by the end user), µDDS Middleware and the PaRTiKle operating system. The reduced version of libgcj is a subset of the runtime support and the CLASSPATH. Finally, this version of libgcj has all of the support to execute applications using the porting and adaption of jRate on the PaRTiKle OS, a subset of java.lang and java.io, support for thread execution, and OS interfaces. The garbage collector, support for graphical interfaces, runtime classloading, bytecode interpretation, reflection, finalization, serialization, file and network I/O, and many parts of java.lang, java.util, and java.io, that were not considered essential, have been removed, for real-time and safety critical applications.

VI. Implementation and Evaluation

To study the performance of publish/subscribe systems we implement the µDDS as our middleware base for wireless embedded applications. The purpose of our experiment is to run latency and throughput performance tests with different message size and number of subscribers in a practical environment.

The test was performed using eight devices based on an ARM 7TDMI-S processor running at 60 Mhz, model LPC2136 with 32 Kbytes RAM and 256 Kbytes EEPROM, and a MaxStream XBee Pro with 802.15.4 protocol stack for the wireless communication.

A. Throughput test

In this test, the publisher sends data where the size varies from 16 bytes to 4Kb and is sent to one or more subscriber applications. The throughput is the total number of messages received per second by all the subscribers in the system divided by message fanout. The test code contains two applications: one for the publishing node and the other for the subscribing node(s). The publisher applications are started first followed by the subscriber applications, then the publication application sends a burst of data and repeats the cycle for a specified duration. Figure 8 shows the Throughput performance results of the test.

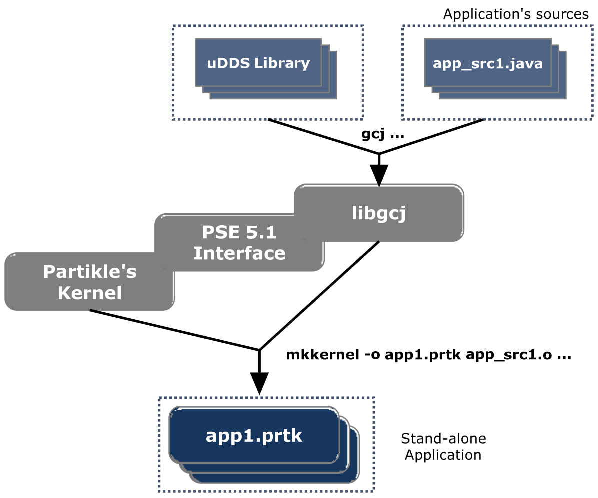

Fig. 7. Java application build process

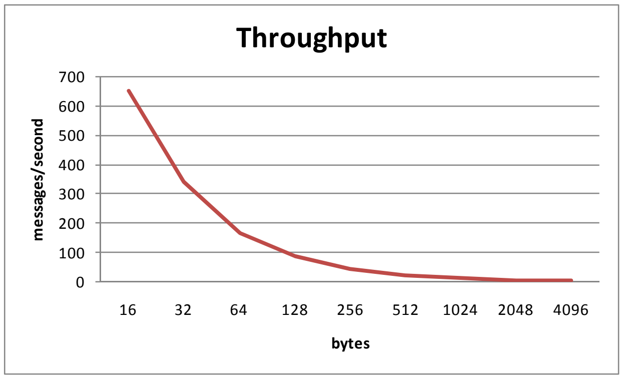

Fig. 8. Throughput Test

B. *Latency test*

Similarly to the throughput test, the publisher sends data with varying size from 16 bytes to 4 Kb to one or more subscriber applications in which the latency is estimated as a half of the round-trip time of a message. The test code contains two applications: one for the publishing node, one for the subscribing node(s). The publisher application is started firts, followed by each subscriber, then the publisher starts publishing data. The test ends when all the messages have been sent and the same number of replies has been received by the publisher. Figure 9 shows the latency results.

C. *QoS Test*

For the QoS policy, µDDS implement some of the QoS model of DDS, such as deadline and time based filter. The Deadline QoS indicates the minimum rate at which a DataWriter will send data. The Time-Based Filter provides a way to set a minimum separation period, which is used to specify that a DataReader wants new messages no more often than this time period; according to the specification, if the value of this QoS policie is 0, it means that the DataReader wants all values.

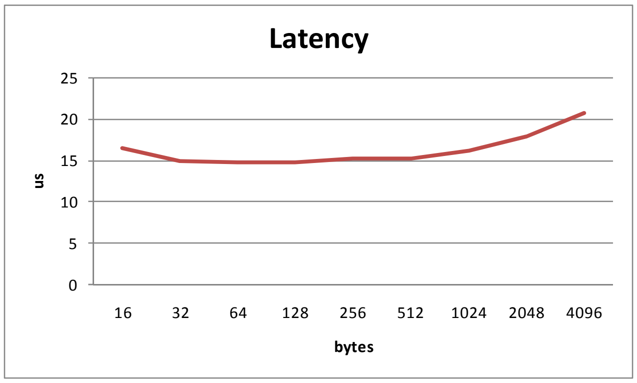

Fig. 9. Latency Test

Figure 10 shows the performance of dealing with notifications per second with different Topics and TIME_BASED_FILTER (TBF) settings. The result shows that µDDS achieves maximal processing capacity for notifications in a second when Topic Number grow up to 600 and TIME_BASED_FILTER = 1.

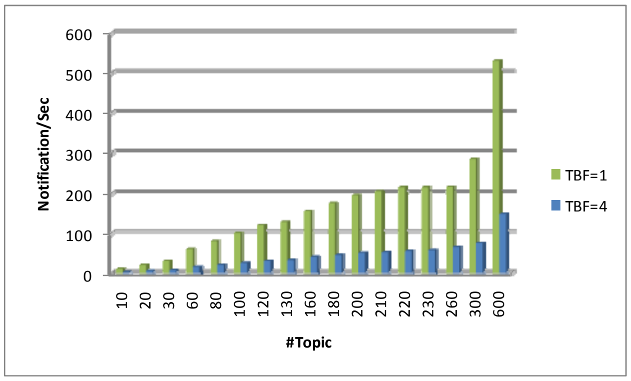

Fig. 10. TIME BASED FILTER QoS Test

Figure 11 shows the Deadline Missed Ratio as a function of the number of Topics while considering different DEADLINE periods. The result show that Deadline increases slowly when Topic Number is bigger than 160 for Deadline = 2 and 200 for Deadline = 4.

These results have been obtained with a service that just run with topic number ranging from 10 to 500, which was observed 20 times in order to obtain the average value.

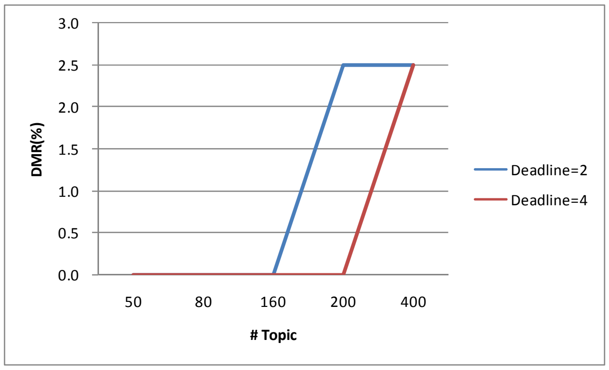

Fig. 11. DEADLINE Test

VII. Conclusions

In this paper, a new DDS compatible real-time middleware has been presented; µDDS is a publish/subscribe middleware that allows real-time wireless embedded applications to interoperate with each other, and is capable of supporting different QoS levels for various applications. The combination of a compact Java environment, the µDDS and the PaRTiKle OS, has resulted in a very small footprint, low latency, and highly reliable platform for time critical Java applications. In order to validate and evaluate the performance of our implementation, several tests have been designed and performed. All the testing realized in this work, shows that the performance of this implementation is very efficient, achieving very good results in throughput, latency and the QoS performed. Evaluations results demonstrate that µDDS is lightweight and efficient, and the use of the µDDS middleware simplifies the development process of real-time wireless embedded publish/subscribe applications.

In the future we will provide our software architecture for other hardware architectures such as XScale and PPC, and we will implement the memory model TLSF which is supported by the PaRTiKle operating system, for which we must integrate the adaptation and implementation of RTSJ that we realized.

References

[1] R. Aquino, A. González, V. Rangel, M. García, L. A. Villaseñor, A. Edwards-Block, “Wireless Communication Protocol Based on EDF for Wireless Body Sensor Networks,” k. Journal of Applied Scioences and Technology, Vol 6, No 2, August 2008.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor Networks,” IEEE Communications Magazine, pp. 102-114, Aug. 2002.

[3] A. Cerpa, J. Elson, M. Hamilton, and J. Zhao, “Habitat monitoring: application driver for wireless communications technology”. ACM SIGCOMM workshop on data communications in Latin America and the Caribbean, Costa Rica, Apr. 2002.

[4] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar, “Next century challenges: Scalable coordination in sensor networks,” in Mobile Computing and Networking. pp.263-270, 1999.

[5] G. J. Pottie, W. J. Kaiser, “Wireless integrated networks sensors,” Communications of the ACM 43(5):52-58, 2000.

[6] W. Mata, A. González, A. Crespo. “A Proposal for Real-Time Middleware for Wireless Sensor Networks,” Workshop on Sensor Networks and Applications (WseNA'08), Gramado, Brasil. Sept 2008.

[7] W. B. Heinzelman, A. L. Murphy, and H.S. Carvalho, “Middleware to support sensor network applications,” IEEE Network, vol. 18, pp. 6-14, 2004.

[8] D. E. Culler and W. Hong, “Wireless sensor networks – introduction,” Communications ACM, 47(6):30-33, 2004.

[9] P. Bonnet , J. E. Gehrke, P. Seshadri, “Querying the physical world,” IEEE Personal Communications 7(5):10–15, Oct. 2000.

[10] P. Levis and D. Culler, “Mate: 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, Oct. 2002.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” In: ACM SIGOPS operating systems review 34(5):93–104, Dec. 2000.

[12] T. Liu and M. Martonosi, “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, Jun. 2003.

[13] L. St Ville and P. Dickman, “Garnet: a middleware architecture for distributing data streams originating in wireless sensor networks,” In: Proceedings. 23rd International Conference on Distributed Computing Systems Workshops, May. 2003.

[14] MiLAN Project. Available: http://www.futurehealth.rochester.edu/milan

[15] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, “Mires: a publish/subscribe middleware for sensor networks,” Personal and Ubiquitous Computing, 10(1):37–44, Feb. 2006.

[16] OMG, *Data Distribution Service for Real-Time Systems version 1.2*. OMG Technical Document, Jan. 2007.

[17] OMG, *Model Driven Architecture (MDA)*, Document number ormsc/2001-07-01. Technical report, OMG, 2001.

[18] OMG, *Overview and guide to OMG’s architecture*, OMG Technical Document formal/03-06-01, Jun. 2003.

[19] G. Pardo-Castellote, B. Farabaugh, and R. Warren, (2005). *An Introduction to DDS and Data-centric Communications*, Available: http://www.omg.org/news/whitepapers/Intro_To_DDS.pdf.

[20] S. Peiro, M. Masmano, I. Ripoll, and A. Crespo, “PaRTiKle OS, a replacement of the core of RTLinux,” In 9th Real-Time Linux Workshop, 2007.

[21] W. Mata, A. González, R. Aquino, A. Crespo, I. Ripoll, M. Capel, “A Wireless Networked Embedded Sistem with a New Real-Time Kernel PaRTiKle,” Electronics, Robotics and Automotive Mechanics Conference, CERMA 2007. ISBN 0-7695-2974-7, Cuernavaca, México. Sep. 2007.

[22] S. Peiro, M. Masmano, I. Ripoll, A. Crespo. “PaRTiKle LPC, port to the LPC2000,” Tehth Real-Time Linux Workshop, Colotlán, Jalisco México. 2008.

[23] W. Mata, A. González, G. Fuentes, R. Fuentes, A. Crespo, D. Carr, “Porting jRate(RT-Java) to a POSIX Real-Time Linux Kernel,” Tenth Real-Time Linux Workshop, Colotlán, Jalisco México. 2008.