ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

A Java-Compatible Multi-Thread Middleware for an Experimental Wireless Sensor Network Platform

[1]CARLOS COTA, [2]LEOCUNDO AGUILAR, [3]GUILLERMO LICEA

[1,2,3] Facultad de Ciencias Químicas e Ingeniería, Universidad Autónoma de Baja California.

Abstract: **The software development for the Wireless Sensor Networks is complicated. Mainly by the programming languages and tools existing for it, which are unconventional to the well-known for the PC system developments. However, the use of middleware helps to this activity, increasing the abstraction level existing in the platform and making possible only think about the software requirements to the programmer. The present work describes the implementation of a multi-thread middleware which adopts the Java programming language and its standard class library for the threads programming. Helped by a Java RT kernel which complies with the Java Real-Time Specification Group. Also is reported the optimization for a better performance in the Java bytecode interpretation.**

I. INTRODUCTION

The Wireless Sensor Networks (WSN) are one of the biggest convergences between digital electronic, micro-electro mechanics and wireless communications. These networks are composed by dozens or thousands of nodes, which are small compute pieces equipped with sensors, and some case actuators. With this devices can collect, process and communicate wirelessly information about the environment where are located [1]. Together makes the network, obviously with a defined role shown in the Figure 1. At the units which are composed the wireless sensor network is called WSN-node.

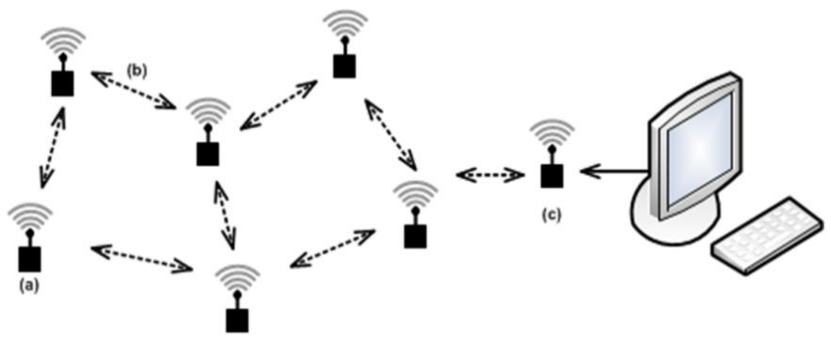


Figure 1: WSN elements: (a) Sensor Node, (b) Wireless communication, (c) Gateway and (d) Monitoring and management station.

The technology advance has become possible to have more and more of them embedded on our environment. Helping to have optimal factories, better production, food harvest, inclusive, early earthquakes detection. The possibilities are enormous, so believes the technology industries is betting on them. This kind of networks are embedded to the physical space where are they. But has an entirely different function to the others networks kind, in which are working with user

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

captured data. In the WSN, works from acquired data by each WSN-node into the network. The main attractive is that we are enabled to connect the physical world with the existing information systems or develop a new nature applications with these features. However, nowadays there still are doing researches about them for purpose of define hardware, software and communications standards [2], [3], [4].

A field where the WSN pose many challenges is about the software application development that run on them. It is not only the scarcity of the WSN-node resources, but also the particularities of the WSN applications that make software design and development in the WSN as open research issue[5].

This work is focused about the WSN-node programming, approaching it from the middleware perspective, for to make complete hardware abstraction. The idea is to use a high-level programming language as Java and the existing programming tools for develop the system to be executed by the WSN-node, using the middleware (installed in WSN-node) as final target. That is for to hide the complexity existing in the final platform, in other words, the programmer only must be think about the application and forgetting about the hardware issues. The middleware is of a Virtual Machine type and is capable to execute a subset of the Java bytecode.

II. RELATED WORK

The WSN software plays an important role in the WSN. Like any computer-based system it must be programmed, but on there does not exists the same resources presented on a conventional PC or distributed system (the most similar kind of systems to WSN), where the application layer is built atop a largely immutable, application-agnostic network stack, for the WSN that is entirely realized in software, therefore malleable, and expected to be tailored to the needs of the application [6].

For the development of application software for the WSN-node, is desirable a component which can helps on this process (shown in the Figure 2), where first, each functional block the components must be identified and included. On this phase, the chosen components must be interconnected and the dependencies resolved. Next, during compilation process, the executable is created. Finally, during evaluation phase, the created node application can be downloaded to the node and executed [7]. This WSN application development uses the components as blocks.

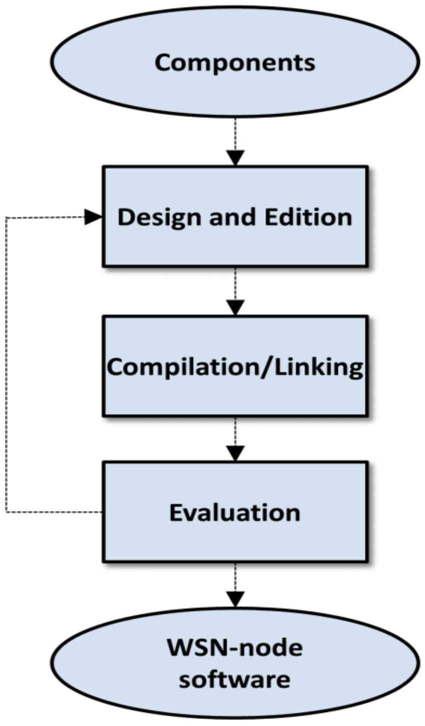


Figure 2: WSN-node software development cycle.

Faced to the challenge presented by the software development for the WSN-node, has been developed various partial, incomplete or incompatible solutions together. While the software reusability concept has become an essential part of the development in other distributed environments. An approach which solves the complexity about the software development and the code reuse is the middleware-based.

A. MIDDLEWARE

The middleware is an intermediate-layer which helps the programmer to develop an application, making the complete hardware and operating systems abstraction. With this, the programmer only thinks about the requirements to be satisfied by the program to be executing in the middleware, itself which is responsible for solve the complexity existing in the hardware (CPU, batteries, RF and others issues) [8]. In brief, the middleware is the responsible for:

- Gives the appropriate interfaces for a variety of possible applications.
- Provide for a run-time environment which coordinate and support various applications.
- Methods to achieve an adaptive and efficient use of the system resources.

Also allows the code reuse with its respective services, as can be the data upgrade, so that the programmer can develop and execute an application without take care for hardware complexity or native functions [9], in a nutshell, the middleware manages the complexity of the hardware instead of the application (see Figure 3).

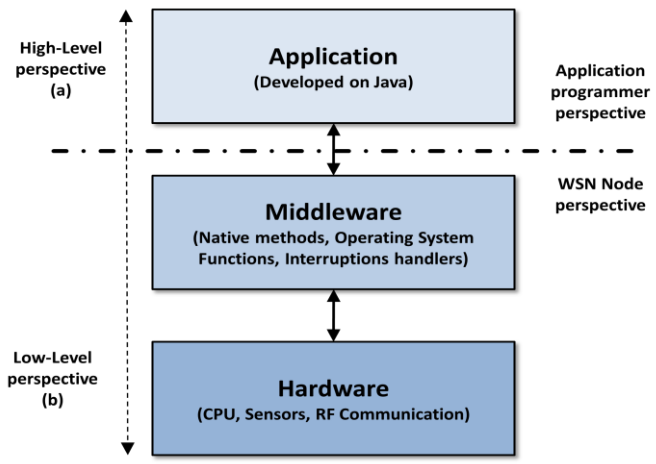

High-Level perspective (a)
Application
(Developed on Java)
Application programmer perspective
WSN Node perspective
Middleware
(Native methods, Operating System Functions, Interruptions handlers)
Low-Level perspective (b)
Hardware
(CPU, Sensors, RF Communication)

Figure 3: Development perspectives: (a) High-Level perspectives involve the use of high level languages like Java, while (b) Low-Level perspectives use mnemonic languages like assembler.

The use of middleware for easiest application development is not recent. They has been used by the traditional systems as a bridge between operating systems and applications [10]. However, the existing (as DCOM. CORBA) no are appropriated for the WSN requirements. For this they have been created different kinds of middleware since the birth of WSN, which can be classified by its programming paradigms shown Figure 4.

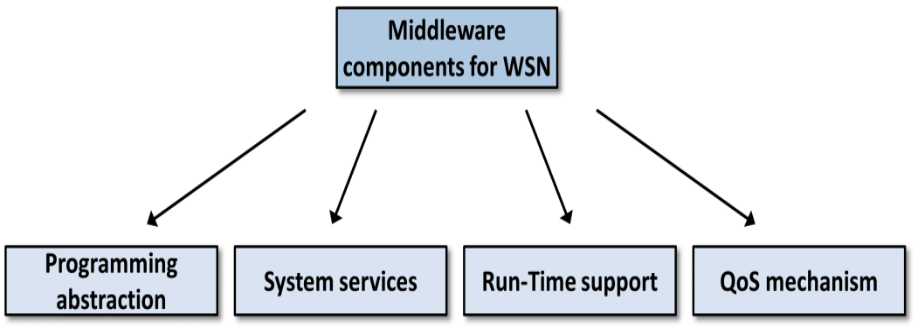


Figure 4: Middleware components for WSN.

There are three different abstraction middleware levels into the WSN, classified from existing WSN researches work: node, network and infrastructure [8]. We are focused on the node abstraction through a middleware of a virtual machine type.

B. VIRTUAL MACHINE

The virtual machine (VM) includes the interpreters and the mobile agents. The main features for this kind of middleware is the flexibility, which allows to developers make applications divides by modules to be interpreted by a virtual machine.

The virtual machine manages the energy consumption in addition to the use of resources. However, the technology is complex and the instruction interpretation introduce an extra processing charge [10]. Some examples of approaches made are Maté [11] and Squawk [12], each one provide the abstraction about the hardware or operating system below of them and adopts a particular programming language. For example, on the Maté it runs over TinyOS [13] operating system and implements a TinyScript which is designed to hide the event-driven behavior of TinyOS [14] or the Squawk which runs without an operating system and use the Java as programming language but only exists on the Sun Small Programmable Object Technology (SPOT) [15].

Both examples showing the necessity for learn a WSN programming languages. Because them not use, with exception of Squawk, a well-known programming language at the majority by programmers. The idea about to use these programming languages is facilitate the transition of PC systems programmers to WSN programming and allow to develop this new type of information systems. On the other hand, the Java language implemented on Squawk VM, is a good option for the WSN programming, but it requires the use of an embedded virtual machine on a specific hardware (Sun Spot). The obvious response is to create a Java-Based virtual machine as middleware for the WSN.

C. JAVA AS PROGRAMMING LANGUAGE

The Java programming language originally was created to be used for embedded applications under the name of Oak by James Gosling [16]. However, after years experimentation and adding significant contributions of collaborators it was used for Internet programming, a hybrid environment where its philosophy “compile once, run anywhere” allows the code reuse and abstraction of the network complexity (a necessary feature for the programming on the WSN). It was designed to be simply to allow at many programmers can use with fluency, has many similarities to C++, but without the complexity manage of memory (all in Java is a reference) and including ideas from another programming languages.

Nowadays, his developers and various professionals area thinks about Java as a mature language for use on a diversity of scopes. One of them is the WSN programming, because are for embedded systems into environments (like the origin scope of Java) and made the necessary abstraction both programming and hardware complexity. But its adoption requires of a compatible Java virtual machine.

D. THE JAVA VIRTUAL MACHINE

The Java virtual machine (JVM) is the cornerstone about the use of the Java technology platform. That brings hardware and software independence, a small compiled code called bytecode and secure execution for the user code. This is a complete abstract computer with its set of instructions and use of memory [17]. Today, the JVM and its components as Java 2 SDK and JRE can be emulated the set of instructions over Win32, GNU/Linux, Solaris and MacOS, but there is not a particular technology for to be implemented over an operating system or hardware.

The JVM unknown about the Java programming language, but rather about a binary format called *classfile*. Inside of this format are the instructions, a symbol table and other auxiliary information for the correct execution of the program. In order to ensure the security, the JVM imposes hard formats and structural limitations to the *classfile* format code. However, any language with the expressed functionality of a valid *classfile* can be executed by the JVM.

III. PROPOSED WORK.

The WSN-node programming is inevitable, for this reason, is valuable adopt a programming language with a syntaxes more easy to manage, making the learning curve shortest, thus the programmer can be focused only about the software application development. Obviously, our proposal includes the Java as application programming language, which are suitable because is easy to learn.

The adoption de Java as programming language require of a compatible Java virtual machine as middleware installed into the WSN-node. This middleware must deal with the complexity existing in the WSN-node, in addition to give support the multi thread a valuable feature inside of the language programming, because the use of threats has an optimal use of the CPU processing and allow divide the task into small activities has share the scarce resources.

The experimentation about the use of Java and the adoption of the Java-based middleware was made on an own designed WSN-node platform called LiSANDRA.

A. THE LASIANDRA PLATFORM

The LISANDRA networks has a multilevel hierarchical architecture as shown on Figure 5, and it is composed at the most basic layer by the node-S (sensor node) which can only senses and transmit small data packets, the node-SR which was previously associated to form a group of sensor cluster and the node-G which is a gateway between it and 802.11 b/g networks commonly called Wi-Fi.

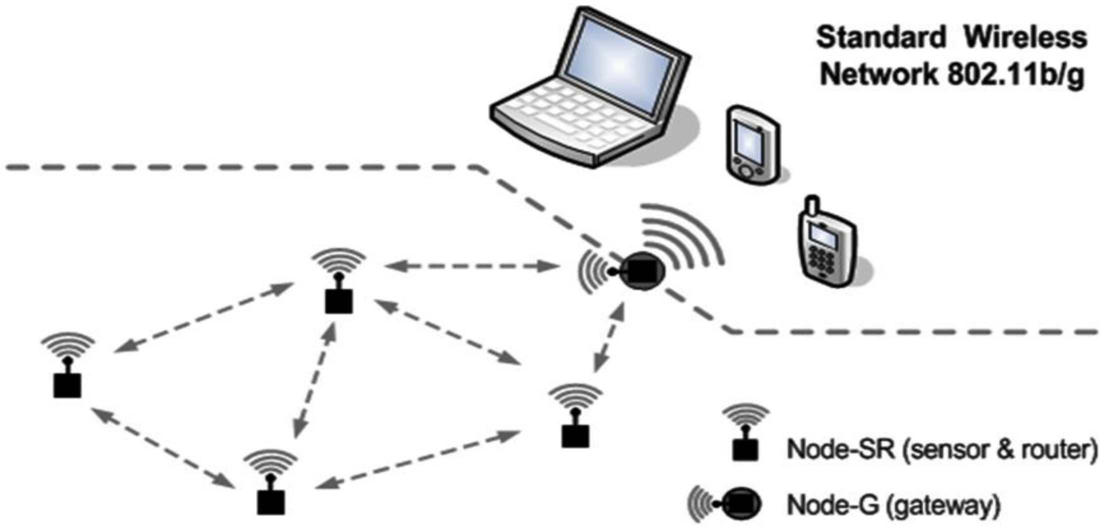

Standard Wireless Network 802.11b/g
Node-SR (sensor & router)
Node-G (gateway)

Figure 5: LISANDRA Architecture.

In the node-SR (see Figure 6) is located the middleware, because there is where the application programmed going to be store. The other nodes only require the network configuration.

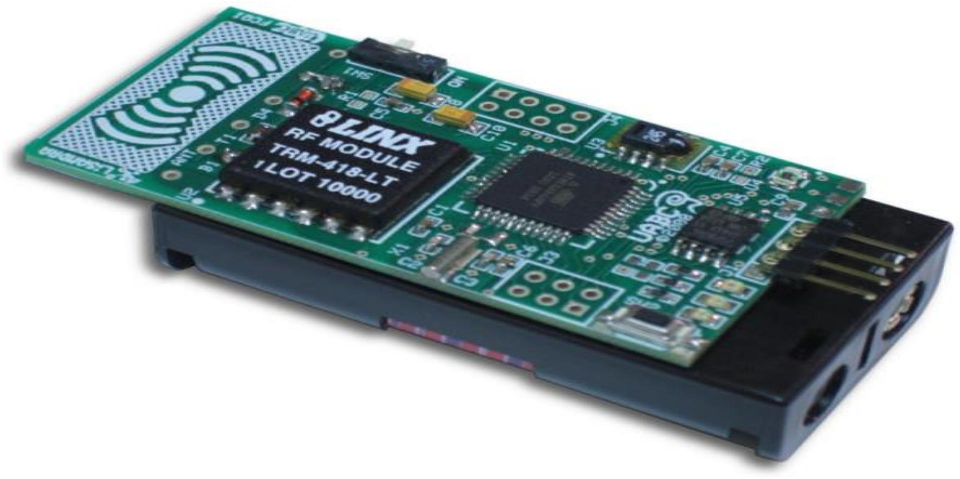


Figure 6. LiSANDRA node-SR module.

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

The node-SR (see Figure 7) has two AA rechargeable batteries as power supply, as processor unit uses the microcontroller AtMega1284p [18] which has 128 KB FLASH program memory and 16 KB RAM memory. Also 1 MB external serial memory for system data storage. The sense unit is composed by a SHT11 temperature and humidity sensor [19] and a TSL2550T light sensor [20]. However, it also has I/O lines that can be used as an interface for other kind of sensors. For communication, it has a RF transceiver TRM-418-LT [21] and a RS232 for the platform programming or download store data to the PC. On the other hand, there is a monolithic firmware composed by the application and management subsystems for the hardware units. The subsystems manage the communication and sense all process; it is all encapsulated to a kernel level called LEARN [22].

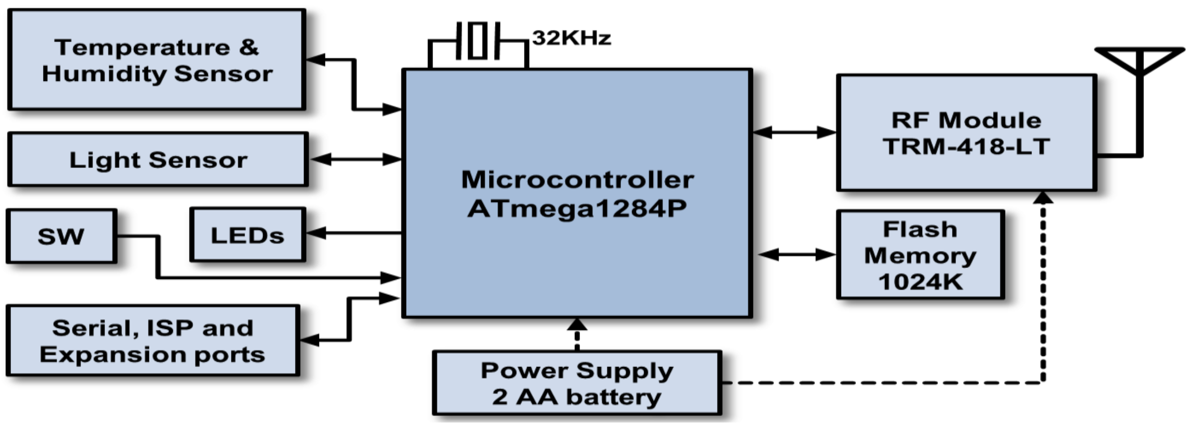

Temperature & Humidity Sensor
Light Sensor
SW
LEDs
Serial, ISP and Expansion ports
32KHz
Microcontroller ATmega1284P
RF Module TRM-418-LT
Flash Memory 1024K
Power Supply 2 AA battery

Figure 7: Node-SR block diagram.

B. MIDDLEWARE DESIGN CONSIDERATIONS:

The WSN-nodes abstraction is the main requirement for the use of the middleware on the WSN deployment. They are small, cheap, low-power consumption, also of to have sensors, enough memory for application program, processing unit and obviously a wireless communication. In resume, they are composed by hardware blocks (see Figure 8) which its particular complexity must be hide by the middleware.

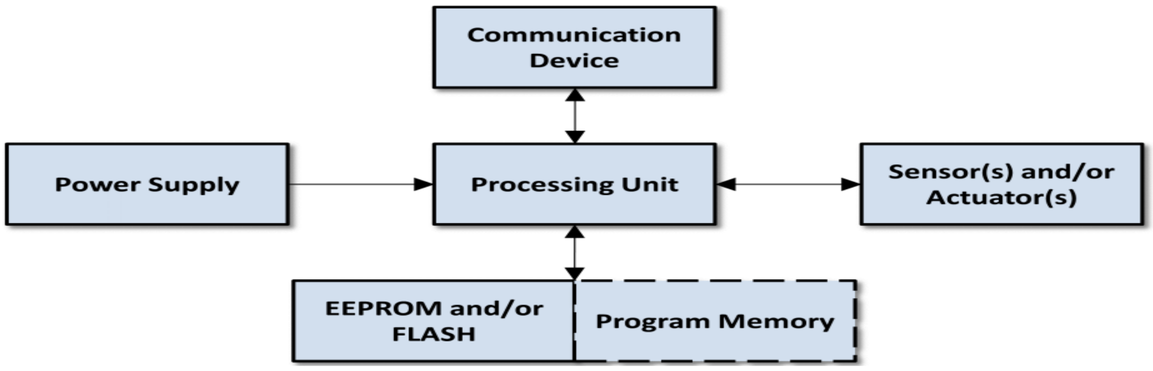


Figure 8: Generic block diagram of a WSN-node.

The access to the hardware of the node-SR is through native methods, they are software pieces embedded into the middleware code for the resources manage and are called from the Java application, making use of a native object.

The use of threads is part of the Java language, its implementation into the middleware must adopt it without a special code consideration.

C. JAVA-COMPATIBLE MIDDLEWARE:

The high-level programming languages like for this case (Java) are compiled to equivalent low-level programs which are executed on a given machine. The process of compiling a programs has many stages. It first analyses the input program to look for errors of various kinds. If none of these are present the program will be for translation. The translation process itself has different stages, making many different transformations on the form of the program in order to move it towards the equivalent low-level program which the compiler will write as its output. A Java source code at be compiled produce a classfile which must be interpreted by the VM. This simple generated program has the instruction in Java *bytecode*, an example is shown in the Figure 9 where is illustrated a form of loop.

```
//java source code
...
      Public void for99(){
        for(int i=0;i<99;i++);
      }
...
//bytecode translation
Method void for99()
      0   iconst_0              //the integer constant zero is pushed on top of the stack.
      1   istore_1              //the top of the stack is stored into variable number one.
      2   goto 8                        //jump to line 8, avoiding increment i before the first
          comparison.
      5 iinc 1 1                //increment local variable one by 1 (i++).
      8 iload_i                 //read the current value of i and push it on top of the stack.
      9 bipush 99               //push the integer constant 99 on top of the stack.
      11 if_icmplt 5            //compare the top two items on the stack and jump if need be
      (i<99).
      14 return                 //return void when the end of the method is reached.
```

Figure 9: *For* loop expressed on Java source code and its corresponding bytecode

The bytecode instructions show a *for* loop which manipulate a stack of operands and the memory where the values of variables are stored. The instructions *iconst*, *iload* and *bipush* push operands on top of the operand stack. The *istore* instruction and the *iinc* instruction update the memory. The instruction **goto** and **if_icmplt** (if integer compare less than) cause transfers of control to the numbered line. These are must be interpreted by a Java-compatible virtual machine.
Exists a previous work about the development of a middleware [23], however this first proposal was very limited, only was capable of execute the bytecode without include the mechanisms to manage threads, application classes and among others. That was made to experiment.
The actual Java-compatible middleware has three specialized software modules, also is helped by a LEARN-based kernel (called Java RT) focused in the Java for Real-Time treatment, threads manager and a set of native methods (rather than being part of the middleware).

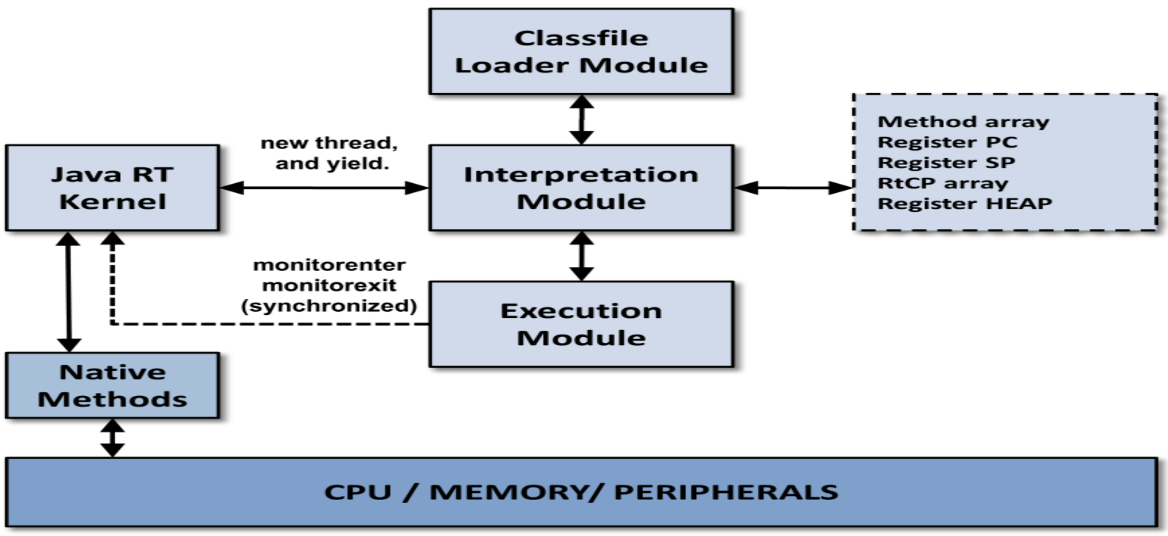
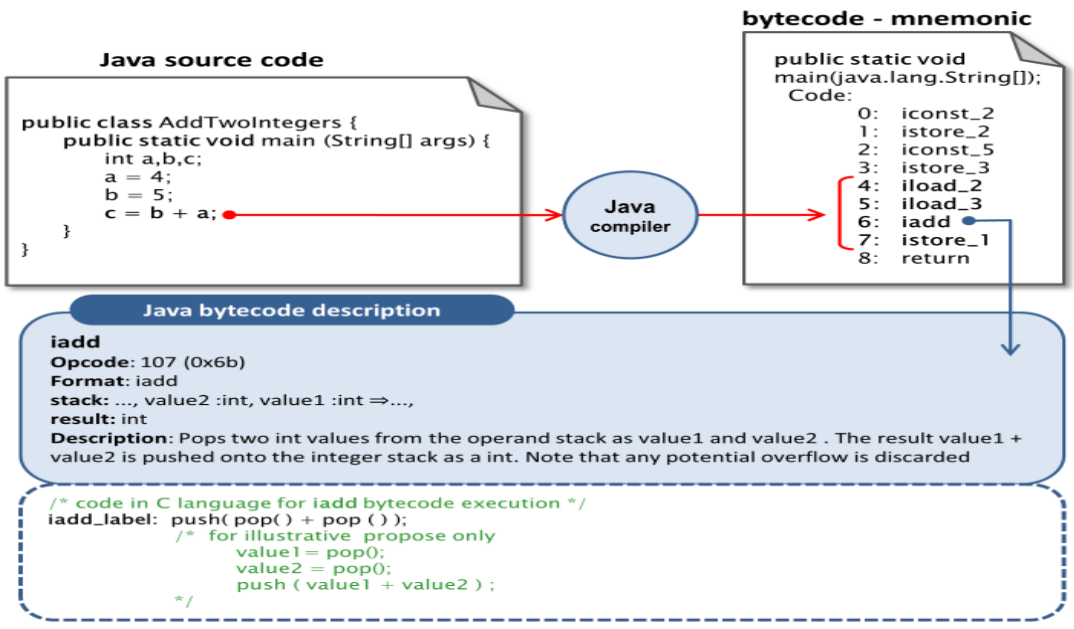


Figure 10: Java-compatible middleware internal architecture.

The modules which compose the middleware (see Figure 10) are: first is a *classfile* loader, where is realized the tasks of verification, preparation and resolution of the binary data existing into the *classfile* and used in the initialization of the internal VM registers (program counter, stack pointer, run-time constant pool, heap, main thread, etc.). Also some optimizations are executed (explained later) with the purpose of a better performance at the Java application execution and the extras *classfile* included in the unique image.

The second one is the interpretation module, where the Java bytecode nature is defined. For example, if is an instruction as new thread, yield or listener registration this is envoy to the Java RT for his resolution. Otherwise, this is an instruction given to be the next module. This and the next module uses the VM internal registers to help in the *classfile* data access or for the correct execution of the application. The last one, is the execution module, this translate the given Java bytecode to a set of hardware instructions to be executed, it also if require to have access to the hardware it call the native methods contained into the Java RT kernel.

Is well known than the use of a middleware protects the hardware from an incorrect or unexpected access by the executing application, for example, referring on hardware management, when possible to access registers by memory is mapped in a careless manner. Moreover, its use implies the introduction of code overload, where a simple instruction of Java bytecode must be executed by various native CPU instructions. A simple explanation of interpretation process overload is shown in Figure 11; the addition of two integers, in the Java programming language, results into a group of bytecode operations and then it in another group of native instructions.



```
public class AddTwoIntegers {
      public static void main (String[] args) {
            int a,b,c;
            a = 4;
            b = 5;
            c = b + a;
      }
}
```

```
public static void
main(java.lang.String[]);
  Code:
      0:   iconst_2
      1:   istore_2
      2:   iconst_5
      3:   istore_3
      4:   iload_2
      5:   iload_3
      6:   iadd
      7:   istore_1
      8:   return
```

iadd
Opcode: 107 (0x6b)
Format: iadd
stack: ..., value2 :int, value1 :int ⇒...,
result: int
Description: Pops two int values from the operand stack as value1 and value2 . The result value1 + value2 is pushed onto the integer stack as a int. Note that any potential overflow is discarded

```
/* code in C language for iadd bytecode execution */
iadd_label:  push( pop( ) + pop ( ) );
                  /*  for illustrative  propose only
                      value1 = pop();
                      value2 = pop();
                      push ( value1 + value2 ) ;
                  */
```

Figure 11: Java bytecode equivalence

The execution code overload introduced by the JVM is an average of ten C code instructions by one bytecode. But this can be solved by different techniques like just in time compiler or a way-ahead-of-time Java compiler. However, this is endurable if we desire to have features lie the application security and its platform independence.

The Java-compatible middleware gives a complete hardware abstraction. The programmer only thinks about the application without take care about the hardware or system platform complexity, besides of, if have experience in the Java language, can feel comfortable during the development.

D. MULTI-THREAD SCHEDULING:

A thread is a basic unit of CPU utilization, it comprises a thread ID, a program counter, a register set, and a stack all them as *Thread Context Block*. An important feature about its use is that shares with other threads belonging to the same process, its code section, data section, and other platform resources. If a process has multiple threads of control, it can perform more than one task at a time.

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

The benefits of multithreaded programming applied to embedded system can be broken down into three major categories listed as: responsiveness, resource sharing and economy [24].

The Thread Context Block implemented for the middleware, shown in the Figure 12, comprises *State*, where stored the transition state of the thread, *priority* which determines the execution order, the CPU *registers* as Instruction Pointer (IP) and stack pointer (SP), the segment of the Thread *Data* and the *Locals* variables for the execution of the Thread.

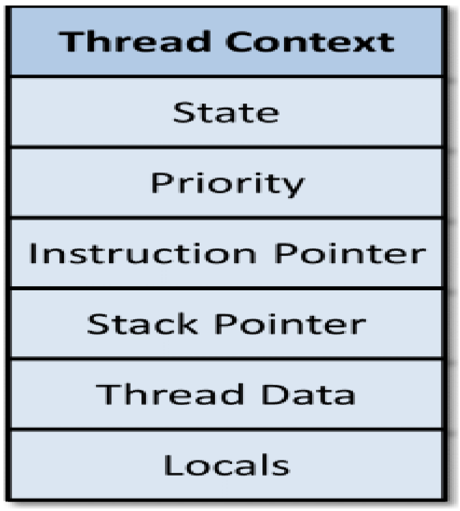


Figure 12: Thread Context Block.

The kernel, switches between the threads, making use of the information content into context block (see Figure 13). For it, saves the VM state on the context block corresponding to the thread which have exhausted its time processor usage, check the schedule (with the policy describe later) for the next thread to be executed, locates the context block indicated and restore the saved context into the VM state and continues the execution while is no exhausted the assigned time.

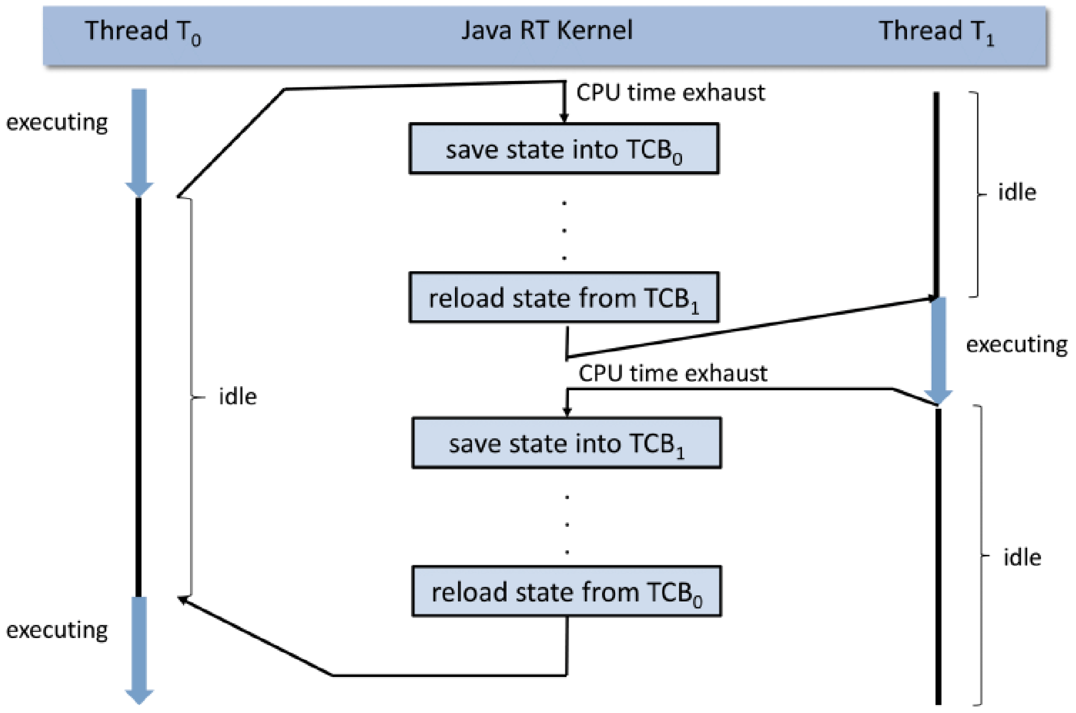


Figure 13: CPU switches from thread to thread.

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

The threads programming is part of the Java standard library which provides of a rich set of features for its creation and management, also is the fundamental model of program execution in a Java program. The idea of make it viable for the WSN-node programming allow to the application development be as small autonomous pieces of code. Each one of the threads with its own action and isolated its memory resource and context processor, saved inside of the VM stack, sharing only in some cases the program code. They are native to the Java RT Kernel and this is the responsible for its correct work in the multi-thread environment.

The specification for the JVM (from is based this VM) has loosely defined scheduling policy that simply states that each thread has a priority and the higher-priority threads will run in preference to threads with lower priorities. No says if the scheduling policy must be preemptive, or the time-sliced using a round-robin scheduler. The decision is up to the particular implementation of the JVM.

The scheduling implementation uses the preemptive and priority based scheduling algorithm, this means that all Java threads have a priority and the thread with the highest priority is schedule to run first run. The execution priority is according to the Real-Time Specification for Java [25].

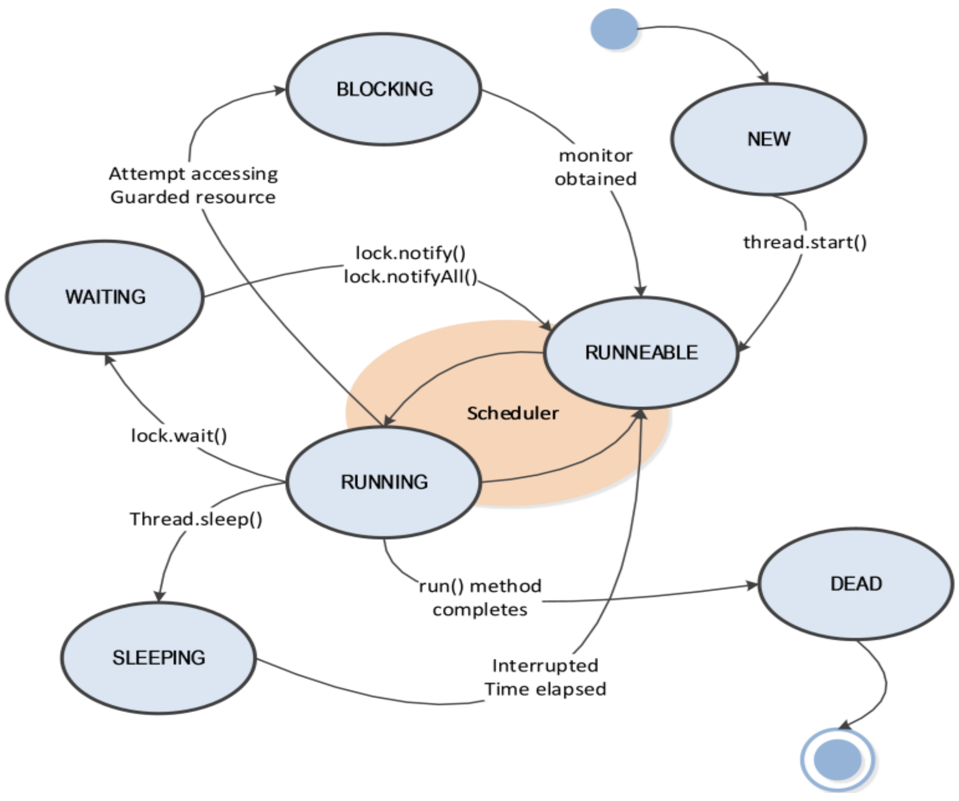


Figure: 14 Thread states and transition diagram.

The amount of CPU time (quantum) by thread is fixed on base time of 1 millisecond. Depends of the priority the assigned time. Once finished his time, the running thread yield the CPU to another thread of the same priority or lowest.

The state diagram, shown in the Figure 14, has the transitions between all thread states. Running and runnable are the states where the scheduler, based in the execution time, is the unique involved: the other states (blocking, waiting and sleeping) are traced by the execution thread condition, where it can be requiring a synchronized resource or voluntarily yield the processor or wait for a timer overflow. A none desired state is the dead because it is finished *run()* method of the thread and there is not mechanism for recover its reserved thread memory.

E. INTERPRETER OPTIMIZATION.

The use of a middleware implies the introduction of code overload, where a simple instruction of Java bytecode must be executed by various native CPU instructions. This execution code overload introduced by the VM use, is an average of ten C code instructions by one bytecode. But this can be solved by different techniques like just in time compiler or a

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

way-ahead-of-time Java compiler. However, this is endurable if we desire to have features lie the application security and its platform independence.

The interpretation, is made by a module that reads the instruction to be executed and prepare the middleware registers for its correct execution. In order to have a correct idea about it, was taken measurement a of the CPU cycles of some of the instructions. Especially those to call to methods inside of classes (static, virtual and special) which are complex because require of a late binding to a class code. Also of simplest bytecode as *iinc* (increment an integer variable value), *goto* (set the program counter to a specific instruction address) and *jsr* (branch to specific instruction) which are routines easy to interpret. The results are showed in the Figure 15.

As is shown (see Figure 15, light blue bars), before optimization, the bytecode to call methods requires over 2500 CPU cycles according to AVR-gcc compiler. That is because the class and method name was solved at runtime, each time when it occurs through of the constant pool index (only identifying his class name, where is contained and method name) by the interpreter module. Once solved the class and method, this is searched inside of Run-Time Method reference where are indicated the method address where the program counter must jump (obviously saving before the CPU context). All this is done when it is running the clasfile; hence the concern about the CPU cycles required.

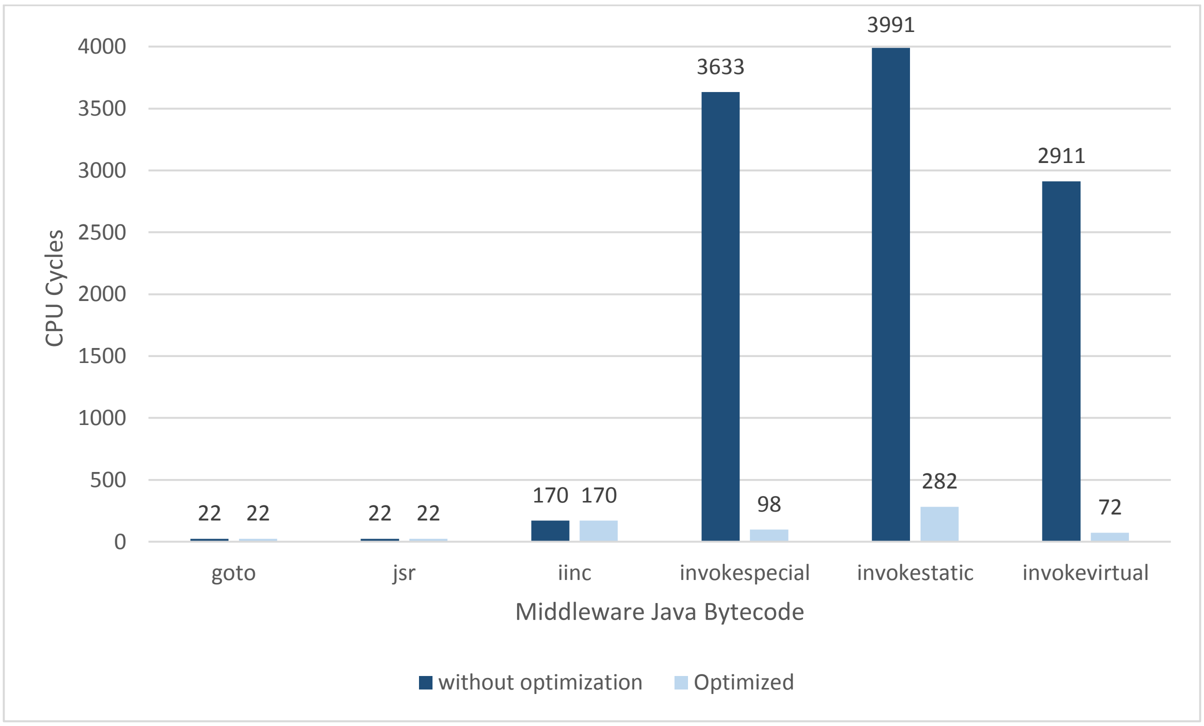


Figure:15 Bytecode interpretation measurements with optimization results.

For a better performance (see Figure 12), inside of the initialization module (see architecture of the proposed VM section), are solved all the indexes to classes and methods, then searches for their localization (if are native or to late binding too) and registers their reference inside of the Run-Time Constant Pool. So that, if is called, can be referenced accessing inside of the same index in the Constant Pool. That make slow the start of the Middleware but makes efficient the *classfile* interpretation reducing the interpretation CPU cycles.

F. CODING EXAMPLES:

The software development for WSN is transparent to the programmer using the Java language, as can see in the Figure 16, the structure of the source class no change anything about the other application developed on this language, also is visible in this example how the resource of the WSN-node is accessible via LiSANDRA object with its methods to operate the present hardware in the platform.

There are two techniques for creating threads in a Java program. One approach is to create a new class that is derivate from the *Thread* class and to override its *run* () method. The other technique is to define a class that implements the *Runnable* interface. However, for practical purposes is only adopted the first technique.

The description of the shown program is: are created two threads where one use the humidity sensor and sends the value through RF every 5 seconds, and the other, uses the temperature sensor and sends the value through RF too. The 5 seconds of delay of each created thread can be cause collision at the RF use, but the Java RT kernel solved with the schedule and monitoring the resources.

```
import LiSANDRA.*;

public class SimpleThreadApp{

  final int FIVE_SECONDS = 5000;
  final String COMM_MEDIA = “radio”;
  final int N1_addr = 138;
  final int N1_port = 25;
  final int N2_addr = 100;
  final int N2_port = 12;

  public static main( String arg[] ){
    HumidityTxt neighborHum = new HumidityTx();
    TemperatureTx neighborTem = new TemperatureTx();
    neighborHum.start();
    neighborTem.start();
  }
  private class HumidityTx extends Thread {
    Socket sock_n1;
    LOutputStream outRF_n1;
    LHumiditySnr humidity = (LHumiditySnr) LiSANDRA.Resources(LHumiditySnr);

    public HumidityTx ( int node_addr, int port ){
      sock_n1 = new Socket( COMM_MEDIA, node_addr, port );
      outRF_n1 = (LOutputStream) sock_n1.getOutputStream();
    }
    void run(){
        while(true){
                outRF_n1.write( “H:” + humidity.read() );
                Thread.sleep( FIVE_SECONDS );
        }
    }
  }

  private class TemperatureTx extends Thread {
    Socket sock_n2;
    LOutputStream outRF_n2;
    LTemperatureSnr temperature = (LTemperaturSnr) LiSANDRA.Resources(LTemperatureSnr);

    public TemperatureTx ( int node_addr, int port ){
      sock_n2 = new Socket( COMM_MEDIA, node_addr, port );
      outRF_n2 = (LOutputStream)sock_n2.getOutputStream();
    }
    void run(){
      while(true){
        outRF_n2.write(“T:” + temperature.read());
        Thread.sleep( FIVE_SECONDS );
      }
    }
  }
}
```

Figure 16: Java application using threads.

Creating a Thread object does not specifically create the new thread; rather is the *start* () method that actually creates the new thread. Calling the *start* () method for the new object does two things: (1) is allocated the memory and initialized a new thread in the VM, and (2) call the *run* () method, making the thread eligible to be run by the VM.

When the example program runs, shown in Figure 16, three threads are created by the VM. The first is the parent thread, which starts execution in the *main* () method. The second thread (*neighborHum*) is created when the *start* () method on the Thread object is invoked. The third (*neighborTem*) is created at the second thread spends his time processor usage.

We must emphasize that the number of threads is limited to 5 because the limited amount of RAM memory available on the platform.

ISSN 2349-7815

International Journal of Recent Research in Electrical and Electronics Engineering (IJRREEE)
Vol. 2, Issue 1, pp: (10-23), Month: January - March 2015, Available at: www.paperpublications.org

IV. CONCLUSION

The software development for the embedded systems as Wireless Sensor Networks is complicated, usually uses a microcontroller compiler, as C language or an uncommon programming language for it (as can be nesC or spin). This is one of research area of the WSN and is crucial for the increasing of the WSN applications and the inclusion of a variety of professionals from distinct areas to electric-electronic. Increasing the abstraction level, allow to the developer be focused only about the software requirements and forget the existing hardware complexity.

The use of Java language is a good option for hide the complexities about WSN programming. With this, the programmer community accustomed to this programming language can emigrate to this new kind of systems development. Mainly with the adoption of the used tools for the PC systems programming. However, to use Java as programming language for WSN, is required a subset from the language specification, as although there are elements relevant to hybrid networks, the WSN nodes will have resources and limited power supply. But the use of the threads as part of the standard library of Java is immutable because is responsibility of the Java RT. Its use allow divide the application into small pieces of code executing.

The middleware is the main component about the increase the abstraction level, although make the Java bytecode introduces code execution overloading. But with the optimization mechanisms can be reduced significantly.

V. FUTURE WORK

This multi-thread middleware offers hardware abstraction and the multiple thread execution helped by a Java RT kernel. However, a better approach for a complete abstraction is the development of application classes for be used into the application to be development. Thus the application can be built from existing code blocks reusing code and accelerating the development.

REFERENCES

[1] Akyildiz I.F., Su W., Sankarasubramaniam Y., and Cayirci E., Wireless Sensor Networks: a survey, Elsevier, Computer Networks, pp-393-422, (2002).

[2] Hac, A., Wireless Sensor Network Designs, John Wiley & Sons, Honolulu, pp (2003).

[3] Callaway E. Jr., and Callaway E., Wireless Sensor Networks: Architectures & Protocols, Auerbach publications, Florida, p- (2003).

[4] Ilyas M., and Mahgoub I., Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems, CRC Press, Florida, pp. Capítulo 13, (2005).

[5] Ana-Belén García-Hernando, Et.Al., Problem Solving for Wireless Sensor Networks, Computer Communications and Networks, Springer.

[6] Gian Pietro Picco, Software Engineering and Wireless Sensor Networks: Happy Marriage or Consensual Divorce?, FoSER 2010.

[7] Jan Blumenthal, Matthias Handy, Wireless Sensor Networks – New Challenges in Software Engineering, In proceedings of the 2003 IEEE Conference on Emerging Technologies and Factory Automation (ETFA´03), pp. 551-556.

[8] Teemu Laukkarinen, Jukka Suhonen, Et. Al., A Survey of Wireless Sensor Network Abstraction for Application Development, International of Distributed Sensor Networks, Hindawi Publishing Corporation, 2012.

[9] Miao-Miao Wang, Jian-Nong Cao, Jing Li and Sajal K. Das, Middleware for Wireless Sensor Networks: A Survey, Journal of Computer Science and Technology, May 2008.

[10] K. Sohraby, D. Minolu, T. Znati, Wireless Sensor Networks Technology. Protocols and Applications, Wiley 2007.

[11] P. Levis and David Culler, Maté: A Tiny Virtual Machine for Sensor Networks, University of California, Berkeley, California.

[12] http://labs.oravle.com/projects/squawk/squawk-rjvm.html March 2011.

[13] P. Levis, S. Madden, Et. Al., TinyOS: An Operating System for Sensor Networks, University of California, Berkeley, California.

[14] Jui-Nan Lin and Jiun-Long Huang, A Virtual Machine-Based Programming Environment for Rapid Sensor Application Development, National Chiao Tung University, Taiwan.

[15] Doug Simon, Cristina Cifuentes, Et. Al., Java on the Bare Metal of Wireless Sensor Devices. The Squawk Java Virtual Machine, Sun Microsystems Laboratories.

[16] James Gosling, Bill Joy, The Java Language Specification. Third Edition, Addison-Wesley, 1996.

[17] Tim Lindholm, Frank Yellin, The Java Virtual Machine Specification, Sun Microsystems, 1999.

[18] Datasheet,8-bit AVR Microcontroller with 128K bytes In-System Programmable Flash, Accessed Nov 16, 2013. http://www.atmel.ca/Images/doc0945.pdf

[19] Datasheet, Humdity and Temperature Sensor IC.

[20] Datasheet, Ambient light Sensor with SMBus Interface.

[21] Linx Technologies, Inc. LT Series Transceiver Module, Data guide.AccessedSept 18, 2010. http://www.linxtechnologies.com

[22] L. Aguilar, G. Licea, “LEARN: Light-weight Embedded Academic Real-Time Nucleus”, Internal technical report, Universidad Autónoma de Baja California, 2000.

[23] Carlos Cota, Leocundo Aguilar, Et. Al. , A Java Compatible Virtual Machine as an Embedded Middleware for Wireless Sensor Networks, CERMA 2010.

[24] Silberschatz, Galvin, Gagne, Operating System Concepts with Java, 7th Edition. John Wiley and Sons, 2007.

[25] Douglas Jense, The Real-Time Specification for Java, Sun Microsystems, 2000.