On-the-fly Dynamic Reprogramming Mechanism for Increasing the Energy Efficiency and Supporting Multi-Experimental Capabilities in WSNs

Gabriel Mujica, Victor Rosello, Jorge Portilla, Teresa Riesgo

Abstract—**Remote reprogramming capabilities are one of the major concerns in WSN platforms due to the limitations and constraints that low power wireless nodes poses, especially when energy efficiency during the reprogramming process is a critical factor for extending the battery life of the devices. Moreover, WSNs are based on low-rate protocols in which as greater the amount of data is sent, the more the possibility to lose packets during the transmitting process is. In order to overcome these limitations, in this work a novel on-the-fly reprogramming technique for modifying and updating the application running on the wireless sensor nodes is designed and implemented, based on a partial reprogramming mechanism that significantly reduces the size of the files to be downloaded to the nodes, therefore diminishing their power/time consumption. This powerful mechanism also addresses multi-experimental capabilities because it provides the possibility to download, manage, test and debug multiple applications into the wireless nodes, based on a memory map segmentation of the core. Being an on-the-fly reprogramming process, no additional resources to store and download the configuration file are needed.**

Keywords — Wireless Sensor Networks, Hardware-Software co-design, dynamic reprogramming, energy efficiency.

I. INTRODUCTION

Remote reprogramming capabilities for in-field WSN are key aspects in the maintainability and operability of the nodes in order to achieve the autonomy of the network, even more when wireless sensor devices are deployed in very remote places or there are difficulties getting physical access to them. Moreover, being able to modify or update the application running in a node or even in a set of nodes in a fast and reliable way can led important reductions on time, energy and cost of maintenance. These updates can cover application parameter changes (e.g. acquisition times), add or modify some functionalities to a full new application code, or replacing the old application code with a new one. However, reprogramming remotely wireless sensor nodes implies several challenges and limitations due to the inherent features of a WSN. First of all, the low data rate of these networks poses some constraints because the network is not prepared to handle a great amount of data, therefore the likelihood of losing packets during the runtime reprogramming process is higher. In this context, the more number of hops among the network the data packets need to do, the more the probability for the reprogramming process is to fail. Following this concept, is clear that as long as the configuration file is greater, the number of data packets that will be lost is higher. Moreover, as the amount of information is bigger, the power/time that the radio has to consume in order to send/receive data becomes more important, taking into account that the communication module of a wireless sensor node is commonly a key factor on the power consumption of the device, thus their autonomy might be penalized.

One solution to reduce the risk of failures during the remote reconfiguration process is to implement a collaborative reprogramming mechanism in which the configuration file to be downloaded into the node is divided in several blocks that are temporarily stored in near nodes so that the integrity of the program is assured by stages. After the cooperative nodes receive the portion of the programs, these are sent to the final device to be reprogrammed. This technique could be useful in large scale deployments in which the number of hops to raise the target is high. However, this requires a dynamic assignment of collaborative groups of nodes depending on the location of the final target as well as keeps some data memory space in each node to save the fragment of the configuration file. Those factors increase the complexity of the reprogramming algorithm whereas the method is time and power consuming, even more when the number of nodes to be updated is important.

An alternative to update the node functionalities is to change partially the running program depending on the type of modification to be carried out. In this way, the work here proposed aims to define a complete reprogramming architecture within the wireless nodes in order to optimize not only the reconfiguration mechanism of the device (based on having lower power consumption and less consuming wireless network wideband), but also to provide an efficient way to handle the hardware/software resources of the node by combining relocation of functional blocks and libraries into the program memory with the concept of multi-application programming. This approach is mainly focused on providing a remote reprogramming mechanism in which users can code, download and update their custom applications without being necessary to recompile and resend the whole programming file to the remote node, so that partial reprogramming is fully supported. This technique is especially powerful when more than one independent application has to be downloaded into the node (i.e. testbeds and debugging scenarios).

The remote reprogramming mechanism is an on-the-fly process, which means that the partial programming files are sent over-the-air to the final nodes and, at the same time, saved into the program memory, so that additional resources, mechanisms and time to store/download data are not needed.

II. PROPOSED APPROACH & APPLICATION SCENARIOS

In [1] node reprogramming scenarios are classified according to the size of the update, frequency and optimization. The concept and implementation proposed in this work aims to be included as a major contribution to two main application scenarios:

A. Wireless sensor network testing platform (WSN Testbeds)

As part of the WSN research lines, the Center of Industrial Electronics (CEI) is working on developing a complete infrastructure for supporting experiments in a real environment under test [2]. The main goal is to provide a fully controlled wireless sensor network based on an Ethernet/Wi-Fi backchannel interface that allows users to debug and maintain the deployed devices without interfering in the main wireless communication, so that the reliability of the nodes, runtime performance, fidelity of protocols and the efficiency of the network can be tested and, therefore, properly optimized according to experimental results and real measurements obtained from the testbed platform.

In this way, the Cookie-based WSN testbed (called CookieLab) has to provide the possibility to reprogram remotely every node through the backchannel interface, allowing users to download configuration files as well as preprogramming experiments, so that different tests can run and/or be updated at the same time. In order to implement properly this powerful feature, the remote reprogramming technique proposed in this work provides a complete *memory map architecture* of the functional blocks that control and support the hardware infrastructure and the debugging tasks (that is called ***static-support segment***), the linking mechanism to correlate the user dynamic applications with the supporting blocks (that is called ***linking segment***) and the multi-experiment segment in which users can create, download and update their custom algorithms and experiments by using the supporting blocks (that is called ***dynamic application segment***). More than one application could be downloaded into the nodes in order to be run under test, so an experiment support chooser must also be included in the architecture (called ***schedule segment***).

B. Final application (in-field reprogramming scenarios)

During In-field application deployments, nodes must have the possibility to be updated in case their behavior needs to be modified or some special algorithms/functional blocks should be replaced, so the remote reprogramming mechanism must be assured, increasing then the autonomy of the network. Due to the fact that the memory map allows changing specific blocks or segment without reprogramming the whole system, the decrease of the time/power consumption during the reconfiguration process is also highlighted. Moreover, if new functional blocks of the static-support segment need to be added in order to enhance additional specific features, support over-the-air debugging tasks, or fit specific application-dependent requirements, it is possible to manage the memory map structure to include them without modifying the rest of the functionalities.

This dynamic programming platform allows users to code their Cookie-based algorithms and high level application functionalities without including full libraries of the platform into their projects. They can use the provided HW-SW support platform from a high abstraction level, so that the memory map structure and the multi-experiment features will be handled transparently.

As it is well known, most of the wireless sensor network applications are so far based on commercial nodes such as TelosB or MicaZ, which are intended to be used together with TinyOS or other operating systems. In contrast, this work proposes to take advantage of the modularity and flexibility of the Cookie based hardware-software platform [3] to increase the dynamism of functionalities and its runtime performance.

As explained in [4], the Wireless Sensor Cookie Node structure includes a microcontroller as the main processing element within the platform. Furthermore, the processing layers that have been designed up to now are based on the very well-known 8051 architecture (such as the ADuC841 from analog Devices [5] or the C8051F930 from Silicon Lab [6]), hence, the implementation carried out and described in the following lines has been made under this technology, but the concept is fully portable to other architectures by changing low level functionalities. Both of them include 64KB flash program memory as well as 4KB extended data memory. The complete multi-experimental support platform is based on these two main resources; nevertheless the mechanism concept to relocate and distribute the functional blocks as well as the partial reprogramming is not fully technology dependent.

III. REMOTE REPROGRAMMING IN THE STATE OF THE ART

During the last years many works have been focused on providing remote programming capabilities, covering aspects such as reduce or optimize the size of the code, dynamic linking in nodes, data delivery, node resources usage, etc.

Deluge [7] is the base reference for wireless sensor reprogramming, which is the main TinyOS reprogramming technique. It is mainly a delivery mechanism in which the main goal is to disseminate large objects into the whole network in a quick and reliable way. During the object dissemination the received packets are stored in the flash memory, once the full image has been properly stored, Deluge calls the bootloader and the new application is loaded from the flash to the program memory. It is intended to be used for reprogramming the whole memory of the node, hence the new application must contain the Deluge component in order to be able to reprogram again. It focuses its best effort on a fast data delivery but not on the size of the object, and only supports monolithic TinyOS image reprogramming.

On the side of size optimization there are mechanisms that are based on calculate the difference between the previous and the new application to be downloaded (INP[8], Zephyr [9], among others). In this case, instead of updating full programs, scripts are created off-line, which contain operations that must be performed in the program memory to create the new application, based on the current application that is running in the node. These works are focused mainly on obtaining the

best mechanism to optimize the size of the script and usually have hardware dependencies because they operate from a low level, even if they are designed for specific operating systems. Furthermore, this method also has an intensive flash memory use, because it needs at least three different components stored in the external memory: the application base, the updated script and the new application, Zephyr adds an extra area for the reprogramming control application. In some cases, if the difference between the old ad the new application are too big, the scrip size is similar to the whole application object.

Some techniques support dynamic linking of applications, such as the ContikiOS Run-time dynamic linking mechanism [10]. This is an in-node run-time dynamic linker, in which the loader and the reallocate mechanism are implemented and integrated in ContikiOS. Object files are read using the Contiki file system, which makes the dynamic linker unaware of the physical location of the file. The object files for the application updates use the standard ELF [11], and C-ELF (compact ELF). Reallocation depends on the CPU architecture so it is not fully portable and must be ported if the hardware platform is not included in the Contiki distribution. This method supports reprogramming at any level of the software, from the core to the application and does not require re-building elements that will not be modified. However, the size of the updates is usually bigger than the previous technique due to the indirection tables that have to be included. Another important reprogramming technique is the modularization of the applications, where only new modules are sent and these replace or join the current application running in the node. A good example of this technique is the solution presented in [12]. This work adds a level of modularity to TinyOS based applications. The modularity is achieved by dividing the applications in two parts, one of them static and the other exchangeable. The last one is intended to be updated remotely. One of the main drawbacks is that for starting-up correctly the application and due to fact that different TinyModules have different sizes, the reset vector handler must be changed dynamically every time a new module is applied. This requires modifying the bootloader that depends on the hardware architecture. At present, there are commercial WSN solutions that support remote reprogramming, such as Waspmote [13]. This platform uses different code dissemination techniques for the different communications available. On the node side it supports the storage of several applications on the external memory (SD card) using the file management system of the software platform. This technique supports multi-application capabilities but The SD must be validated before being fully compliant with the file management of the platform. It is not designed to support partial updates.

According to these relevant works, it is important to highlight that the solution presented in this work does not require any operating system supports for its implementation, although it is platform oriented and a network protocol to reach the destination nodes is needed. In contrast to [12], not only the application can be updated but also support libraries and drivers can be exchanged, so the approach is fully focused on modularity. Moreover, the reprogramming engine is integrated into the bootloader segment, so that loading a reprogramming into the program memory each time, such as in [9] is not required, nor increase the size of the new application with the reprogramming component such as in [7].

IV. ON-THE-FLY REPROGRAMMING MECHANISM

In contraposition to other solutions found in the state of the art that are based on saving the whole programming files into an external memory before reprogramming the node's core, in this work an *on-the-fly* reprogramming technique is proposed. This poses important advantages towards reducing the complexity of the remote reprogramming mechanism because the reconfiguration process is done in runtime whenever it is needed, without wasting additional data space and power consumption within the nodes. However, some constraints must be taken into account, because using low rate wireless sensor communication for reprogramming remotely the nodes could cause failures during the on-the-fly process, hence producing malfunction of the devices. Moreover, the remote reprogramming process must be compliant with several standard interfaces and formats that are defined as follows.

A. *Communication interface*

From the processing element point of view, the configuration file will be received through the standard interface that is used to control the communication module, which is typically a serial protocol. In figure 1 the main communication layers that are implemented in the Cookie platform are summarized, with their corresponding protocol to establish the connection to the processing layer.

Figure 1. Cookie Communication layers

B. *Configuration file*

In order to standardize the node reprogramming process, the format of the configuration file to be downloaded into the core has to be the Intel HEX [14], which is widely used to program microcontrollers, EEPROMS and other programmable chips. As the format of the HEX file is based on ASCII code, the corresponding conversion into bytes must be done afterwards the code is received character by character through the serial interface, especially because the format spends 2 bytes, i.e. "75", for coding 0x75. Nevertheless, in order to reduce much more the amount of data to be transferred-received, a pseudo Hex code format could be used (directly code in bytes).

C. *Flash memory writing*

The 8051-based microcontrollers included in the Cookie platform are based on 64 KB flash memory, divided in 1024 pages of 64 bytes each. The read process of the flash memory is the same as the data memory access. However, due to the fact that the program memory is writing protected, a specific

procedure to update it must be carried out. First, the *ULOAD* configuration mode that allows users to refer to the program memory when the addressing mode commands are used must be enabled. Second, the specific address that will be modified must be specified in two bytes following by erasing the corresponding byte. Finally, the byte to be updated is written. This procedure must be followed in order to avoid failures or erratic behavior during the configuration process.

According to these three main standard features the fundamental structure of the programmable processing core architecture were defined in order to assure the correct remote reconfiguration/updating of the application running on the wireless node, in which two main areas have been delimited. One of them is the *dynamic area,* in which the new program will be downloaded, and the second one is the static/protected area, in which the *bootloader support* is saved. This definition prevents to block the microcontroller during the reprogramming process in case the configuration file is not properly received, hence having the protected part as a recovery mechanism. This *bootloader support platform* (defined as a fully compliant Cookie-based library) includes the configuration of the serial protocols to receive the programming file, the HEX format decoder and the writing procedure to access and update the flash memory.

Due to the fact that the configuration file is byte by byte remotely received and directly stored into the flash memory in runtime, there are two main parallel mechanisms in order to detect and prevent possible errors during the *on-the-fly* reprogramming process. First of all, to prevent the lost packets effects during the configuration process, a checksum calculation is performed after finishing the reprogramming process. In case the configuration file is not properly received, the bootloader requests a second attempt to receive it, so the application will not run until the checksum is validated.

The second mechanism that is running at the same time with the configuration process is the *Cookie-defined Watch-dog block*, which detects connectivity problems to the network, radio signal problems or power supply problems. A specific timeout could be specified as a threshold in order to trigger the *Cookie-watch-dog* functionalities. The bootloader/recovery segment is protected so it cannot be corrupted. In case of malfunctions or failures when the user application is running, the Cookie-watch-dog or specific debugging tasks can launch the recovery segment in order to update or restore the reprogramming file remotely. The general architecture of the remote reprogramming structure and the bootloader/recovery segment is shown in figure 2.

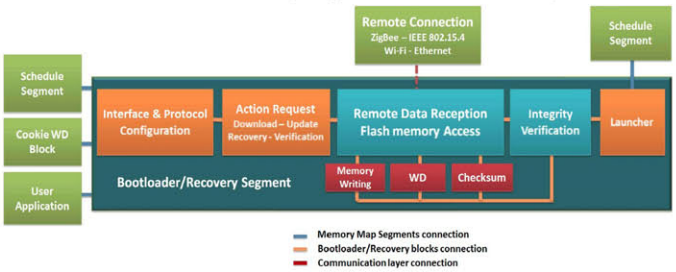

Figure 2. Remote reprogramming architecture.

V. MEMORY SEGMENTATION & MULTI-EXPERIMENT PLATFORM

The on-the-fly reprogramming technique proposed is a powerful and less resource consuming mechanism to download remotely the configuration file into the main processing element of the Cookie platform. Once the reconfiguration process has been defined, it is important to remind that as the configuration file size grows, the time/power consumed by the nodes increases, thus the autonomy could be penalized. This is the main reason why an alternative approach to reduce the amount of data transmitted to the nodes in order to modify or change their configuration file has been taken into account. As it is defined in [3], the CookieLibs platform provides a complete support of the Cookie nodes in order to manage and control their behavior and functionalities within the wireless network, which means that the functional blocks and drivers that are part of the platform must be considered as an integral part of the system. Moreover, the Cookie-based testbed infrastructure is fully supported by this set of functionalities in order to debug, maintain and experiment with the under-test pre-deployed network. The SW support platform has to be allocated into the node as a fundamental part for operation of the applications; hence the CookieLibs are preinstalled in the node before being deployed. This means that in case the application running on the nodes has to be changed or updated, the support platform could be unaltered, so the reprogramming file to be downloaded will be much smaller. If a CookieLibs functional block should be modified or new ones have to be added, it is not necessary to send the full set of libraries again.

In this context, the solution proposed in this work defines a complete memory map architecture in which the program code is reallocated in modular segments so that a partial reprogramming of the Cookie Nodes can be performed remotely, reducing the time and the energy cost during the remote reprogramming process. When a complete application is programmed from a high abstraction level, the compiler is in charge of managing and handling the processing architecture in order to allocate the functional blocks in different areas as much optimized as possible. However, partial reprogramming requires low level optimizations and specific directives in order to achieve the segmentation, flexibility and modularity of the functional blocks without impacting the performance and the resource consumption within the processing element. In this way, the memory map architecture has been divided into five main segments: four of them for supporting the HW-SW platform as well as the organization and optimization of the memory map structure; and a dynamic segment in which users can load their customs applications without enter in low abstraction level details, so that fast prototyping and debugging is also enhanced. This modularity allows users to modify specific functionalities or even segments partially without changing the rest of the system. Each segment is described as follows.

A. *Static Support Segment*

Both the protected part (bootloader support segment) and the CookieLibs functionalities and drivers are allocated in this

segment. In this way, there are two main distribution possibilities: the first one aiming to be applied to final functionalities. In this case, the final versions of functional blocks are reallocated, from a specific memory address, one after the other in order to optimize as much as possible the resources of the program memory. The second option is attended to functional blocks that can be modified (add or optimize its structure) so their size could increase along the time. In this case, each functionality has its memory address and space assigned in order to prevent overlapping, so optimizations could carry out up to certain point (preliminary support functions to be tested before final versions). Libraries for controlling communication layers, peripherals, analog/digital data processing functions, among others are included in this segment. Furthermore, low level directives and addressing configurations are managed by this segment.

B. *Linking Segment*

This segment establishes the correlation between the user dynamic segment and the static segment, working as a router. From the point of view of the dynamic segment, each function has its own fixed address within the memory map. The management of these addresses is done by the linking segment, which assigns a relative address to all of the functional blocks. When a function is called from the application segment, the linker correlates the relative address called with the corresponding address where the functional block is placed. The functionality is then called and executed. In case a library is replaced in the static segment, this is a transparent process for user applications, because the relative addresses are always the same. Therefore, it is possible to change any particular block or driver without being needed for users to change their specific applications.

C. *Dynamic application segment*

This is the part of the memory map structure in which users can download their own specific applications. More than only one application could be allocate in this segment, allowing the possibility to save several configuration profiles or, in case of the testbed infrastructure, configuring different experiments to be executed at different stages of a test. Due to the fact that the static segment and the linking segment are part of the pre-deployed HW-SW support platform, users only need to include the linking segment information to their application project as a system file. In this way, one of the greatest advantages of this memory map segmentation is that users can code their specific application using the CookieLibs support platform without being necessary to include the whole library objects into the projects they are working on. The link segment information provided by the platform is enough to use the functional blocks and the Cookie controllers in their custom applications. Developers can work independently programming their experiments or final applications without interfering with the rest of the memory architecture.

D. *Schedule segment*

In order to provide multi-application support, an additional segment for scheduling and selecting applications has been implemented. There are mainly two ways of selecting the application to run. The first one is the *asynchronous assignment*, in which a selection request can be performed remotely from the server application in order to switch from one application to another. This request, as an asynchronous interruption, will execute the scheduler, suspend the current application running and then execute the selected one or receive and download a new programming file, if needed. The second possibility is the *Synchronous or pre-programmed assignment*, which is intended to be used for scheduling application time-slots so that predefined experiments could run during a specific and reserved period of time. This feature allows users, for instance, to download tree different application prototypes and schedule their execution, then conforming a multi-experimental scenario for testing.

The overall architecture of the memory map segments and their correlations is shown in figure 3.

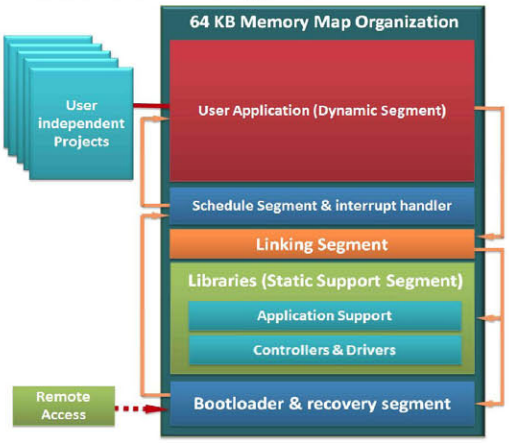

Figure 3. General view of the Memory Map Segmentation.

VI. EXPERIMENTAL RESULTS & TESTS

In order to verify the proposed on-the-fly reprogramming strategies and the memory map architecture for partial reprogramming of wireless nodes, real validation scenarios were set up and carried out. The main goal was to validate the whole implementation by using a wide number of functional blocks and controllers through a custom application. One of these scenarios was to perform Wi-Fi-based experimental tests in order to download remotely the reprogramming file, by using the Low-Power Wi-Fi module included in a Cookie communication layer. Three main configurations were set-up in order to test the remote reprogramming mechanism and the multi-application features. The first program downloaded remotely into the Cookie node was focused on the connection to an access point of the CEI, being necessary to configure all the related parameters such as type of authentication (WEP, WPA, WPA2) and password, DHCP to get a dynamic IP from the server, network mask and channel mask, gateway ID, etc. After the node is already connected to the local network, a TCP communication is established from a remote PC to the device (by using a configured port) in order to update the application running on the device with a new user code.

The second application is related to connect the node with an Ad-hoc network created by other wireless Cookie node, and

after establishing the communication the new application will send LDR measurements every two seconds and enter in low power consumption every five seconds. The reprogramming file is executed, the new configuration of the Wi-Fi is set-up and the sensor measurements are received properly from the remote nodes. The third Wi-Fi-based experiment was related to send the sensor measurements to a web server provided by the manufacture of the Wi-Fi module. Once again, the reprogramming mode command is sent remotely and the configuration process is launched. After the node is properly configured and connected to the network, the LDR value can be monitored by using the HTTP interface.

Table 1 shows the results regarding the size of the partial reprogramming file. In these scenarios the total amount of data that has to be sent through the network in case of reprogramming the whole memory is 34 KB. Instead, the application code transmitted to the sensor node is reduced to only 3 KB when the memory map structure is applied, thus reducing the power/time consumption of the node during the reprogramming process.

TABLE I. REMOTE REPROGRAMMING SCENARIOS.

Experimental Scenarios	Over-the-air data sent/received	
	Whole reprogramming	Partial reprogramming
Ad-Hoc network	37 KB	**2.21 KB**
Access Point connection	36 KB	**2.65 KB**
HTTP configuration	34 KB	**1.81 KB**
ZigBee–Wi-Fi network	42 KB	**5 KB**

As a part of test and validation of the proposed implementation an additional application scenario was taken into account. The experiment has been related to the use of a Cookie-based gateway that implements both the high performance Wi-Fi communication and the low rate ZigBee communication. The main idea was to send remotely the communication file to the Cookie Gateway by using a TCP connection, and then retransmit it to the final ZigBee node, as shown in figure 4. Nodes were updated with a new version of the user application in which temperature, humidity and light intensity parameters were monitored in real time from the remote base station connected to the Cookie gateway.

Figure 4. Wi-Fi – ZigBee remote reprogramming.

In terms of timing saving, Table 2 shows the comparison of the experimental results of the reprogramming technique applied to two different Cookie node processing platforms, in which the whole reprogramming and the segmentation architecture were implemented and tested. The partial reprogramming provides an important reduction of the reprogramming process time (almost 30 times less), which has therefore a big impact of the total amount of energy saved.

TABLE II. EXPERIMENTAL COMPARISONS OF THE PROPOSED TECHNIQUE.

Processing layers (Power Consumption during reprogramming)	Whole reprogramming		Partial reprogramming	
	file size	Process time	file size	Process time
ADuC841-Based (36,12 mW)	34 KB	9,1 sec	**1.14 KB**	**336 ms**
C8051F930-Based (12,8 mW)	37 KB	9,9 sec	**1.67 KB**	**392 ms**

VII. CONCLUSIONS AND CONTRIBUTIONS

As verified in the experimental scenarios, the memory map segmentation capabilities reduce the amount of data that has to be sent through the low-rate wireless network, so that two major features are improved. First, by reducing the configuration file sent to the remote node the risks of losing data-packets is much slower, so the effectiveness of the reprogramming procedure is higher (avoiding then the need of wasting time/power resending information). Second, a reduction of the configuration time implies an improvement of the power consumption of the nodes because the energy spent during the transfer/reception process is much less. In terms of the Cookie HW/SW platform usability with the proposed technique, the multi-experimental method allows users to implement their custom applications and experiments in an easy but reliable way from a high abstraction level, without being aware of the low level details of the platform. Moreover, CookieLibs functionalities can be used without including their full objects into the user projects.

REFERENCES

[1] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in Proceedings of SenSys, 2006, p. 15.

[2] G. Mujica, J. Portilla, T. Riesgo, "Poster Abstract: A Reliable Support Tool for Monitoring, Testing and Debugging Wireless Sensor Cookie Nodes", in EWSN'13, Ghent, february 2013.

[3] G. Mujica, V. Rosello, J. Portilla, T. Riesgo, "Hardware-Software Integration Platform for a WSN Testbed Based on Cookies Nodes", in Procedings of IECON'12, Montreal, november 2012.

[4] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A Modular Architecture for Nodes in Wireless Sensor Networks", JUCS, vol. 12, nº 3, pp. 328-339, March 2006.

[5] ADuC841 Microcontroller, Analog Devices, *http://www.analog.com/*

[6] C8051F930 Microcontroller, Silicon Labs, *http://www.silabs.com/*

[7] J. W. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale Categories and Subject Descriptors," Data Management.

[8] D. Culler, "Incremental Network Programming for Wireless Sensors", IEEE SECON 2004., pp. 25–33.

[9] R. K. Panta, S. Bagchi, and S. P. Midkiff, "Efficient incremental code update for sensor networks," *ACM Transactions on Sensor Networks*, vol. 7, no. 4, pp. 1–32, Feb. 2011.

[10] A. Dunkels, "Contiki Resources and Support," 2004. Available: http://www.contiki-os.org/support.html. [Accessed: 30-Apr-2013].

[11] "Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification." Available:

[12] M. Gauger, P. J. Marrón, C. Niedermeier, "TinyModules: code module exchange in TinyOS," in Procedings of INSS'09, Jun. 2009.

[13] Libelium, "Waspmote Technical guide." [Online]. Available: http://www.libelium.com/uploads/2013/02/waspmote-technical_guide_eng.pdf.

[14] ARM Technical Support, "Intel Hex File Format". [Online]. Available: http://www.keil.com/support/docs/1584.