Dynamic Reconfiguration for Software and Hardware Heterogeneous Real-time WSN

Fabien Mieyeville, Mihai Galos, David Navarro
Ecole Centrale Lyon, Institute of Nanotechnology of Lyon
INL-UMR5270, CNRS, Ecole Centrale de Lyon, Ecully, F-69134, France
fabien.mieyeville@ec-lyon.fr, mihai.galos@ec-lyon.fr, david.navarro@ec-lyon.fr

***Abstract*—Wireless Sensor Network (WSN) technology has imposed itself in civilian and industrial applications as a promising technology for wireless monitoring due to its wireless connectivity, removing many hardware constraints. Initially used in low-frequency sampling applications, the increasing performances of electronic circuits has driven WSNs to integrate more powerful computation units, paving the way for a new generation of applications based on distributed computation. These new applications (process control, active control, visual surveillance, multimedia streaming) involving medium to heavy computation present real-time requirements at node level where reactivity becomes a primary concern as well as at the network level where latency must be bounded. In this paper, we present the implementation of a high-level language MinTax coupled with an in-situ compilation solution for real time Operating Systems enabling energy-aware dynamic reconfiguration while supporting hardware heterogeneity in Wireless Sensor Networks.**

***Keywords*-Wireless Sensor Network; dynamic reconfiguration; MinTax; real time; in-situ compilation.**

I. Introduction

Wireless Sensor Networks are highly distributed self-organized systems. A wireless sensor network is made of a large number of scattered tiny low-cost devices featuring strong constraints in terms of energy, processing, communications and memory capabilities. Common applications of WSN deployed on a given space are data collection from sensor node measurements that are transmitted to a specific node called the sink node. Typical deployment of wireless sensor network can be seen Figure 1.

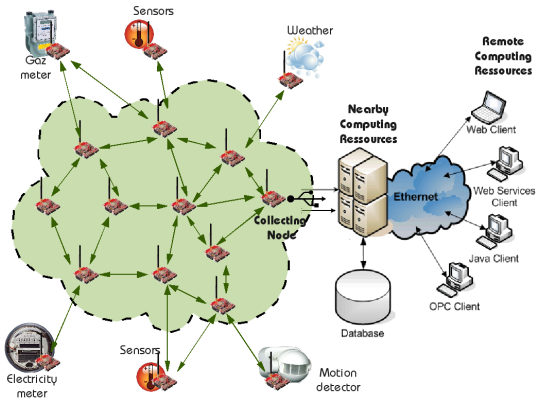

Fig. 1. Classical deployment of wireless sensor network.

First generation of Wireless Sensor Networks has been deployed in applications to remove wired connections and to offer new approaches in the physical deployment of distributed systems. Hardware node platforms of those WSNs offer limited computation ability, small memory capacity and energy constraints are so high that local demanding computations are alleviated (or sometimes optimized [1]) restraining WSN node activities to the sensing and data transmitting tasks. Yet with the advance of microelectronic technology, the new generation of hardware WSN nodes offers improved performances in power management domain as well as in computation field. Consequently, local processing on WSN node can be considered in a WSN design process flow. Thus, WSNs are now disseminating into the fields of high performance networked applications such as process control, multimedia streaming [2] and active control [3]. In this last domain, numerous successful implementations can be found, particularly in Structural Health Monitoring (SHM) [4] where realizations are numerous and distributed computation possibilities offered by WSNs are beginning to be explored [5]. These performance-critical applications require bounded delay latency and then can be referred to as real-time applications that generate new design constraints in WSN compared with conventional WSN applications.

In this paper, we will demonstrate the capability of a reconfiguration solution based on a high-level language called MinTax [6], [7] for current real-time applications. This solution takes benefit from new hardware node architecture to reduce reconfiguration consumption in WSN by in-situ compilation at node level and can be easily deployed on hardware and software heterogeneous nodes architecture in a real-time context. This paper falls in four parts: after a presentation of WSN reconfiguration state of the art, we will develop the stakes in real-time WSN (RTWSN). Then, we will develop MinTax and in-situ compilation and demonstrate by experiment that it can be used in a RTWSNs.

II. Reconfiguration: brief overview

A. A complex problem

Reprogramming a whole network can be addressed in numerous ways. The taxonomy for programming model (cf. Figure 2) established by R. Sugihara and R. Gupta [8] demonstrates the different hierarchical levels at stake in programming of WSN nodes. Our solution falls in the platform-centric category of the node-level abstraction. We will then focus on the reconfiguration of the node itself and the cost minimization

 ISBN: 978-1-61208-207-3

of this task while complying with WSN design constraints (including real-time aspects) and not on the global process of disseminating the reconfiguration through the whole network [9] .

B. Challenges and stakes in WSN reconfiguration

The main challenges in WSN reconfiguration are [10]: (1) the energy cost of reconfiguration should be as small as possible, (2) minimization of the size of the code and small necessary memory usage to perform the reconfiguration because of the constrained hardware architectures used in WSN and last (3) the size of the code to be updated or created should be kept minimal. This last aspect has a huge impact on reconfiguration in two ways. First, minimizing the size of the code sent by RF to reconfigure node minimizes the consumption of the node. Secondly, wireless communications being unreliable due to possible signal collisions, interferences, and packet contentions, a short reconfiguration code improves the probability of being successfully received.

To those commonly recognized challenges we add the support for heterogeneity of end systems. WSNs are deployed for a long period of time: nodes can be replaced by new architectures with different hardware and software specifications compared with the initial deployment. Hence any reconfiguration solution for long-term deployed WSNs should provide support for both hardware and software heterogeneity.

Now that challenges are clearly established, we will put into context the common existing solutions for performing reconfiguration at node level.

C. Current solutions for WSN reconfiguration at node level

Three main approaches for dynamic reconfiguration are usually identified [7] as follows: use of machine code, use of bytecode for a Virtual Machine and diff-based approaches.

The first category is either associated with Operating Systems (OS-es) that can dynamically load/unload modules (modular) or those that cannot (monolithic). Monolithic OS-es are statically compiled and globally optimized into a single executable image. Part of the code cannot be reprogrammed independently and requires a complete rewriting of the entire code resident on the node. Modular OS-es offer a partial reprogramming paradigm that can take a part (module) of the functionality and link it to existing functionalities already running on the node. In general, operating systems do not offer support for hardware heterogeneous WSN and offer at best limited support for real-time [11]. The updates consist of machine code for a particular instruction set.

The second category is associated with Virtual Machines, which execute an intermediate form of information called bytecode. This bytecode is decoded and the resulting instructions are executed. If they offer a promise of hardware heterogeneity, they are either tied to a particular OS or deployed as a standalone solution. In the context of a software heterogeneous WSN, they do not provide support for different Operating Systems. Moreover, since the code is interpreted (decoded) on every run, the execution of a functionality implies overhead which means a greater energy expenditure.

The last category, the diff-based approach makes use of a difference computing algorithm that runs on the PC and generates a delta-file. This delta file contains the modifications between two versions of the software, the one already present on the node and the newer version. Then, this delta file is sent to the node and is added by a resident program to the targeted functions (improvement or creation). Functions being committed to be increased in size, a slop region between them can be provided so as to easily accommodate to those code size modifications. The diff-based approaches offer the advantage of extremely small updates, but do not offer either hardware or software heterogeneity.

D. Performances metric

Current reconfiguration solutions are based on a code processed by cross-compilation and then disseminated through the network. Hence, size of the code to be transmitted is critical since the RF transceiver is the most consuming part in WSN node. The metrics that are commonly used to evaluate performances of a reconfiguration solution are completion time, energy consumption and memory usage [10]. First, reprogramming a node is a non-trivial task which necessitates a quick completion so as to keep disruption of embedded software on the nodes to minimum, especially in case of partial reprogramming of a network. Secondly, since energy is at the heart of any WSN design process, reprogramming must ensure that the node can keep on working after the reconfiguration has been performed: this process must consume as little as possible so as not to exhaust the node. Lastly, embedded architectures in WSN nodes offer limited memory capacity: the program and data memories used in the reprogramming process must be kept to a minimum.

E. Specifications for an optimal reconfiguration solution

From our point of view, a good reconfiguration solution must provide the high level approach of a Virtual-Machine that can offer support for hardware heterogeneity and must minimize the size of the code to be sent by radio-frequency with the effectiveness of code-machine that reduces execution consumption of the code. Furthermore, the energy cost of reconfiguration must be minimal so any monolithic operating system should be avoided. Our solution that will be developed in the next sections mixes those three aspects. To those classical WSN specifications, we will add the constraints introduced by real-time applications that will be developed in the following section.

III. Reconfiguration for real-time WSN

A. Real-time WSN

Defining the real-time capacity of a wireless network requires a two-fold approach: if a quantitative notion of real-time capacity is often related to the amount of real-time data that the network is able to route in their deadlines, reactivity of the nodes themselves makes the real-time aspect fundamental in the node hardware platform itself (particularly in active control applications [5]). Most works in the field of RTWSNs focus on the quality of service with strong emphasis on the protocols [9], [12]: that is the reason why characterizing real-time ability of WSNs and evaluating performances is often linked to metrics related to network wireless communication

 ISBN: 978-1-61208-207-3

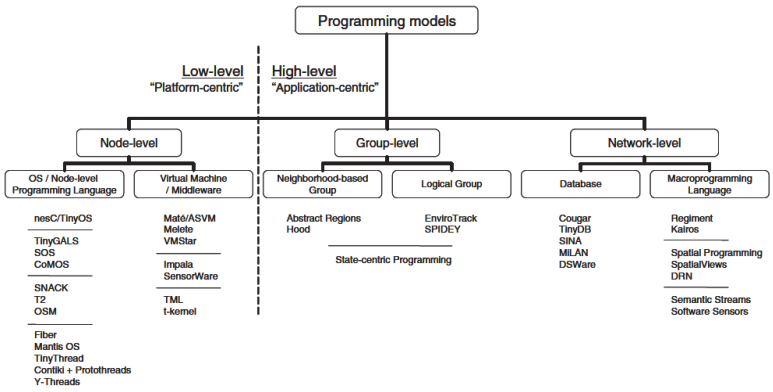

Fig. 2. A taxonomy of programming for WSN [8].

performances [13]. If such an approach is sufficient in traditional WSN applications based on low-frequency sampling rate, more demanding applications such as active control [5] or multimedia sensors [2], requiring local computation, necessitate real-time management at node level implemented in operating systems features [14], [15].

Very scarcely used in conventional WSN application, real-time is being increasingly used in new WSN applications and is very application-dependent. Then, implemented solutions will differ, both on the protocol aspect [16], [17] and at node level. Real time constraints support of software can be achieved by implementing adequate scheduling policies [12].

B. Real-time operating systems

Among existing operating systems for WSN, very few offer support for real-time [11], [18]. So as to establish the adequacy of our solution with real-time WSN, we have selected the following operating systems including energy-aware Real-Time Operating System (RTOS) kernels offering preemptive multithreading based on a traditional programming paradigm: FreeRTOS [19]. While Nano-RK (not presented in this work but currently being implemented) is the only RTOS dedicated to WSN and offering energy-awareness, FreeRTOS offers a small footprint and targets limited computation architecture: they can easily be deployed for WSN and are commonly used in WSN community [20].

FreeRTOS is a portable, open source, mini Real Time Kernel. FreeRTOS code base is small (classical kernel size from 4kBytes to 9kBytes) and is mostly written in standard C. Each task is assigned a priority and tasks with the same priority share the CPU time in a round-robin fashion. The FreeRTOS scheduler is preemptive so as to meet real-time behaviour required by the system. FreeRTOS is often used in WSN applications [20] and offers an extensive hardware heterogeneity support.

C. Reconfiguration specifications for real-time WSN

A real-time system must perform a set of actions within a certain time interval. In RTOS, tasks are executed periodically and must be completed within their deadlines. The way a RTOS manages the concurrent programming of these tasks is set by its scheduling policy that may be based on priority assignments. Reconfiguration of a real-time WSN node is critical since the new tasks to be embedded must comply by the existing deadlines. In this paper, we made the assumption that the application to be programmed is validated on a node so as to ensure the real-time integrity of the resulting programming. In particular, latencies of the system after reprogramming node must be verified on a test node before reconfiguration deployment. The resulting constraints that must be respected are the following: the RF communication must be shortened so as to ensure a viable transmission of code to be implemented and the duration of the reconfiguration of the node must be kept as small as possible.

IV. MinTax for real-time WSNs

MinTax [6], [7] is a high-level programming language designed and tailored precisely for energy-aware software updates in Wireless Sensor Networks. It is compiled dynamically on the node (in-situ) after its deployment. The resulting machine code is then written to the microcontroller memory and is available as a new functionality.

Its high-level semantics mean that the code is not dependent on the underlying node hardware, as is machine code. As a consequence, when an update on a hardware heterogeneous WSN is performed, a single update for all present architectures is broadcast once. MinTax can also be considered as a generic reconfiguration method for WSN from a hardware point of view. Furthermore, the MinTax compiler residing on the node does not interact with the Operating System that may be present on the node, and the update written in MinTax does not contain any information pertaining to the OS, software heterogeneity is thus supported. What is more, because the updates need to take a short amount of time, they take the form of modules that a modular OS links to its kernel and makes them coexist with functionalities already present on the node. The MinTax compiler supports modular compilation and software heterogeneity.

A. Features of MinTax

MinTax was inspired from C, and offers a subset of the C language. Function calls, with up to 4 bytes as formal parameters, as well as returns are present. Iteration clauses such as “while” and “for”, as well as branching (“if” and “switch-case”) clauses are also supported. Other features include analog/digital port read/write (analog read and PWN output) and arithmetic clauses. Future versions of MinTax will probably support matrix manipulation, which would significantly improve the scope of the applications that can be deployed using it.

 ISBN: 978-1-61208-207-3

B. The MinTax compiler

A classical compiler is composed of an Analysis and Synthesis part, presented in Figure 3. The Analysis part is responsible for reading the input file, divide it into atoms (or indivisible parts) and constructing a parse tree. This tree contains information on the order the instructions are to be interpreted. The splitting into atoms is called lexical analysis and the construction of the parse tree is done by the syntactical and semantical analysis. Next, the Synthesis part is the part that does the actual machine code generation. In the case of

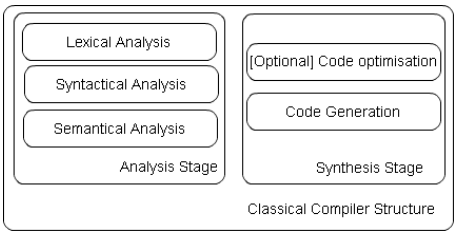

Fig. 3. Classical compiler structure

the MinTax compiler, the lexical analyser is generated using a program called Re2C [21] (regular expression to C). Lexical rules are given in the form of regular expressions, and a C file is generated with the language's corresponding lexical analyser. In what concerns the Semantical and Syntactical part, the parser is generated with a parser generator called Lemon [22]. Lemon takes semantic rules in the form of an input file in which the grammar of the MinTax language has been implemented. During the parsing process, a Symbol Table with the all the symbols present in the MinTax file (functions, variables) is created. It will be used to generate the code in the generation phase.

For the Synthesis part, the instruction set for every supported architecture (AVR and MSP430) has been implemented as function primitives. These primitives take formal parameters as arguments and generate the instructions they model.

The process described above is described in Figure 4.

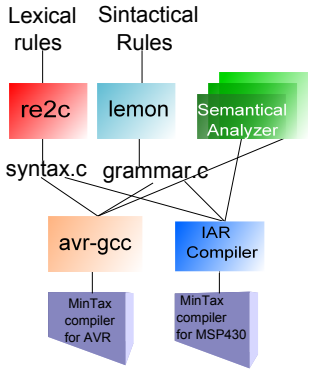

Fig. 4. The process required to compile the MinTax compiler

For a more detailed presentation of MinTax features, the reader may refer to previous works [6], [7].

C. Supported hardware architectures

One of the key feature of MinTax is its support of heterogeneity: the major microcontroller families of WSNs are then supported: the AVR ATMEGA128 family and MSP430 and MSP430x architectures. A cross-compilation feature is also provided so as to enable development and debugging on computer before deployment. Some platforms supported by the MinTax compiler are presented in Table I.

TABLE I
MINTAX COMPILER SUPPORT FOR COMMERCIAL WSN PLATFORMS

WSN Node	Flash usage	RAM usage	RF Transceiver
Mica2	26kB	150bytes	CC1000, 433Mhz
AVRRaven	26kB	150bytes	AT86RF230, 2.4Ghz
Zolertia Z1	22kB	168bytes	CC2420, 2.4Ghz

V. EXPERIMENTAL VALIDATION

We have validated our approach on a point-to-point communication between two different hardware platforms implementing two different real-time OSes (funkOS and FreeRTOS).

A. Hardware and software configuration

A Zolertia Z1 node under funkOS [7] sends a functionality written in MinTax to an AVR Raven node from ATMEL running the FreeRTOS kernel. This node receives the functionality, compiles the function and links it to its existing functions. In the same time it resends the received data to another node so as to emulate a dissemination. Measurements are processed through the use of a home-made current-sense amplification circuit, which enables us to correlate measurements with the different phases of microcontroller activity. This RTOS offering no support for RF communication, we have implemented an IEEE 802.15.4 RF packet format since it is the common protocol used in real-time WSN [9].

B. Code example

The code implemented is a loop corresponding to the following pseudo-code: *count to led until 11, send RF, delay*. This example uses a great number of functionalities of MinTax (calculation, jump, use of hardware feature of microcontroller, etc.) and illustrates the robustness of this high-level language solution. The MinTax translation of this pseudo-code is the following (code is commented as pseudo-code to help the understanding of MinTax syntax):

```
 inputMinTax[]={
"bCa{" // function b, char parameter a
"Ea<11;"  //if a>11
"s@a;"  // call send a
"#" //end if
"};" //end function b

"aa{" //function aa
"Wa<11;" //while a<11
"a+;" // increment a
"#" //end while
"};" //end function aa


"a{" //function a
"WI" //while true
"b=$a;"  //read port a
"b@a;"  // send
"b+;" // increment
"$a=b;" // while porta
"aa@;"  // delay
"#" //end while true
"};" //end function a


};
```

The function realized by this code has a compiled sized (machine code obtained by cross-compilation) of 96 bytes. The MinTax formulation of this code occupies 60 bytes. On this quite simple solution, the MinTax abstraction enables a gain of 33% on the size, which should result in reduced RF transceiver consumption.

 ISBN: 978-1-61208-207-3

TABLE II
DETAILED OF DURATION AND ENERGY CONSUMPTION OF REPROGRAMMING PROCESS.

State	Duration	Power	(mJ)	Total (mJ)
RX	40,8	127,15	4,986	17,547
Compile	111,28	83,33	9,273	
Flash	23,3	80,66	1,879	
TX	9,28	130,01	1,207	

C. Experimental results

Figure 5 shows the activity on the microcontroller of the AVR Raven node and durations and energy consumption of each state are summarized in the Table II. Beyond the success of reprogrammation of the function that is seamless merged with other existing functionalities, a thorough analysis of these results enables us to evaluate the performances of our solution. Four successive phases are to be observed:

1) the receiving of the data whose shortness enables to solicit at minimal the consuming RF transceiver,
2) the code is compiled by the in-situ compiler and linked to the FreeRTOS kernel,
3) the generated machine code is written in the flash memory,
4) broadcast of the MinTax code to the other nodes so as to reconfigure the whole network. This step is accomplished at the end of the process so as to validate the integrity of the received file. Hence, no corrupted data, and then no useless configuration file, is emitted through the network at the cost of a slight latency in the global reconfiguration process. This implementation offers a supplementary robustness to the reconfiguration process.

At the end of the RF-broadcasting, the experimental traces show the execution of the new functionality.

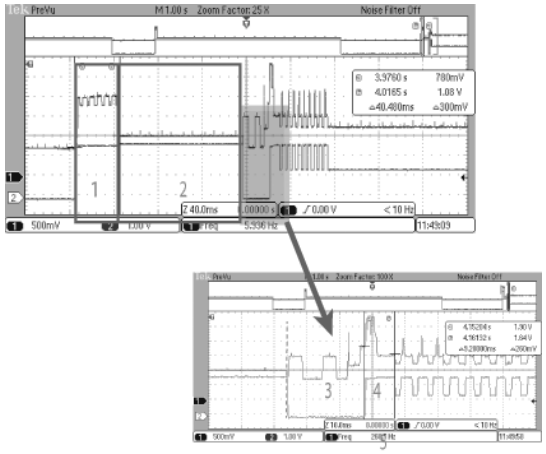

Fig. 5. Physical measurement of the reconfiguration process.

D. MinTax and in-situ compiler performances analysis

MinTax used with in-situ compilation enables the fast reconfiguration of node in approximately 184.66 ms with less than 18 mJ. The transmission duration and energy budget is from far inferior to the receive phase: indeed, if we use an IEEE 802.15.4 protocol for initial sending of data from Zolertia Z1 to AVR Raven node, the resending of data is accomplished without any protocol layer.This choice was made to evaluate the cost of the protocol overhead on our solution. As a result, the strong overhead necessary for an IEEE 802.15.4 protocol-based communication stands for 75% of the energy necessary to the RF transmission of the reconfiguration data. This cost has obviously a strong impact on the reconfiguration process. Hence, if MinTax enables a gain of 33% on the size of the transmitted code, it represents only a gain of 5ms on the 40.8ms of the global RF communication. MinTax should then be used with a lightweight RF protocol such as the RF layer used for Contiki [23]. Moreover, it is important to notice that the presented example is quite simple and the advantages of our solution grow with the size of the application code to reconfigure. If for small applications, compilation could remain consuming, for big applications, compilation energy will be smaller compared with transceiver energy expenditure since offering a power consumption reduced by one third.

Otherwise, the global performances of MinTax present an improvement for reconfiguration compared with the literature. For example, the reconfiguration of a blink application using run-time linking of ELF files in the Contiki operating system [24] takes 972ms for a total consumption of 19.92mJ.

E. Compression implementation for improved performances

To improve latency performances of MinTax in a real-time applications context, we have implemented the support for compression. After the generation of the MinTax code, we proceed to a compression using a Huffman-based compression algorithm.

This approach has been used with the example previously described. From a size of code of 60 bytes for MinTax, we obtain a size of 49 bytes, i.e., a gain of 12%. From the previous conclusion, it is obvious that this gain will not have a huge impact on the reconfiguration cost in a classical WSN configuration using a IEEE 802.15.4 algorithm. We, nevertheless, performed the experimentation so as to evaluate the cost of the decompression on the reconfiguration process. The experiment results are to be seen on Figure 6.

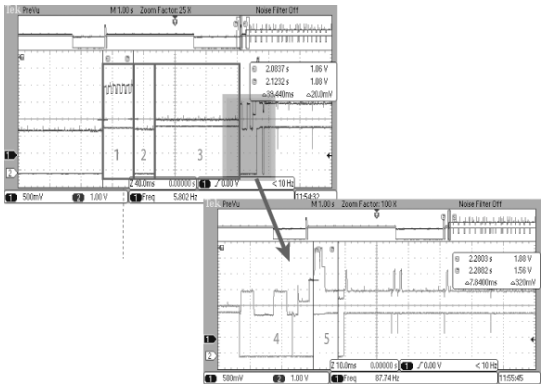

Fig. 6. Physical measurement of the reconfiguration process using compression.

The decompression phase is added to the previous behaviour, resulting in five successive phases to be observed: (1)

TABLE III
DETAILED OF DURATION AND ENERGY CONSUMPTION OF REPROGRAMMING PROCESS USING COMPRESSION.

State	Duration (ms)	Power (mW)	Energy (mJ)	Total (mJ)
RX	39,44	126,43	4,986	18,129
Decompress	25,84	52,46	1,356	
Compile	108,24	82,03	8,879	
Flash	23,44	78,91	1,850	
TX	7,84	134,96	1,058	

receiving of the compressed MinTax code, (2) decompression of the archive by a resident algorithm on the node, (3) compilation of the code, (4) writing of the generated machine code in the flash memory and (5) broadcast of the MinTax code to the other nodes so as to reconfigure the whole network.

Durations and energy consumption of each state are summarized in the Table III.

From a global point of view, the overall performance of the reconfiguration is very similar to the precedent section where no compression was applied. The decompression cost is not really high in term of energy consumption (only 7.4% of the global energy consumption) but is more impacting on the latency with 25.84ms, the same duration of the flash writing, that is to say about 12.6% of the total duration of the reconfiguration (204.76 ms).

The loss of performances compared with the case where no compression is implemented is about 15% as well on energy consumption as on duration of reconfiguration. Yet the slightly reduced duration time for receiving and transmitting of the reconfiguration code could justify the use of compression. Furthermore, the application case is here quite simple. If we consider more complex applications such as distributed computation on node requiring reconfiguration, the over-cost of compression should be absorbed by the gain in code size and in duration of dissemination.

VI. CONCLUSION AND PERSPECTIVE

We have here successfully demonstrated the use of MinTax for real-time operating systems with hardware heterogeneous support. Furthermore, the energy cost of reconfiguration is kept minimal and the duration of the reconfiguration is small compared with the current reprogramming solutions, making it particularly suitable for real-time systems. The latencies introduced by reconfiguration (key parameter in real-time systems) are currently being explored so as to establish the upper-bounds: more complex functions such as distributed computation algorithms being used in active control applications are under test since MinTax support high complexity [6] code involving loop, conditional evaluations, nesting, etc.. This work will validate the quality of services offered by MinTax as well as the validity of compression approach that will take benefit from the increased size code. MinTax is also currently being developed for Nano-RK [14] a popular reservation-based real-time operating. The support of this RTOS should offer to MinTax and its compiling solution an extended coverage of RTOS for WSN.

REFERENCES

[1] L. Gu *et al.*, "Lightweight detection and classification for wireless sensor networks in realistic environments," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 205–217.

[2] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, pp. 921–960, 2007.

[3] I. Akyildiz and M. Vuran, *Wireless sensor networks*. John Wiley & Sons Inc, 2010.

[4] J. Lynch and K. Loh, "A summary review of wireless sensors and sensor networks for structural health monitoring," *Shock and Vibration Digest*, vol. 38, no. 2, pp. 91–130, 2006.

[5] Y. Wang *et al.*, "Decentralized civil structural control using real-time wireless sensing and embedded computing," *Smart Structures and Systems*, vol. 3, no. 3, pp. 321–340, 2007.

[6] M. Galos *et al.*, "Energy-aware software updates in heterogeneous wireless sensor networks," in *9th IEEE International NEWCAS Conference*, June 2011.

[7] M. Galos *et al.*, "Reprogramming hardware-software heterogeneous wireless sensor networks," in *The 14th International Symposium on Wireless Personal Multimedia Communications (WPMC'11)*, Brest, France, Oct. 2011.

[8] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sen. Netw.*, vol. 4, pp. 8:1–8:29, April 2008.

[9] X. Feng *et al.*, "A survey of adaptive and real-time protocols based on IEEE 802.15. 4," *International Journal of Distributed Sensor Networks*, vol. 2011, 2011.

[10] Q. Wang, Y. Zhu, and L. Cheng, "Reprogramming wireless sensor networks: challenges and approaches," *IEEE Network*, vol. 20, no. 3, pp. 48–55, 2006.

[11] M. O. Farooq and T. Kunz, "Operating Systems for Wireless Sensor Networks: A Survey," *Sensors*, vol. 11, no. 6, pp. 5900–5930, May 2011.

[12] O. Chipara, C. Lu, and G. Roman, "Real-time query scheduling for wireless sensor networks," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. Ieee, 2007, pp. 389–399.

[13] P. Pagano *et al.*, "Simulating real-time aspects of wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, p. 2, 2010.

[14] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-rk: An energy-aware resource-centric rtos for sensor networks," *26th IEEE International RealTime Systems Symposium RTSS05*, vol. 0, pp. 256–265, 2005.

[15] X. L. K. Z. R. C. W. Dong, C. Chen and J. Bu, "Fit: A flexible, lightweight, and real-time scheduling system for wireless sensor platforms," *IEEE Trans. Parallel Distributed Syst. (TPDS)*, vol. 21, no. 1, pp. 126–138, 2010.

[16] S. Bansal, D. Juneja, and S. Mukherjee, "An analysis of real time routing protocols for wireless sensor networks," *International Journal of Engineering Science*, vol. 3, 2011.

[17] Z. Teng and K. Kim, "A survey on real-time mac protocols in wireless sensor networks," *Communications and Network*, vol. 2, no. 2, pp. 104–112, 2010.

[18] A. M. V. Reddy *et al.*, "Wireless sensor network operating systems; a survey," *Int. J. Sen. Netw.*, vol. 5, pp. 236–255, August 2009.

[19] R. Barry, *FreeRTOS Reference Manual — API Functions and. Configuration Options*. Bristol, UK: Real Time Engineers Ltd., 2010.

[20] A. Schoofs *et al.*, "A framework for time-controlled and portable wsn applications," *Sensor Applications, Experimentation, and Logistics*, pp. 126–144, 2010.

[21] P. Bumbulis and D. D. Cowan, "Re2c: a more versatile scanner generator," *ACM Lett. Program. Lang. Syst.*, vol. 2, no. 1-4, pp. 70–84, 1993.

[22] "The Lemon Parser-Generator," in *http://www.hwaci.com/sw/lemon/ (accessed August 11, 2012)..*

[23] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 455–462.

[24] A. Dunkels *et al.*, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 15–28.

 ISBN: 978-1-61208-207-3