2013

Exploring the Microsoft .NET Micro Framework for Prototyping Applied Wireless Sensor Networks

Jay D. Carlson
University of Nebraska-Lincoln, jcarlson@unl.edu

Mateusz Mittek
University of Nebraska-Lincoln, mmittek@gmail.com

Lance C. Pérez
University of Nebraska-Lincoln, lperez@unl.edu

Electro/Information Technology (EIT), 2013 IEEE International Conference on
Year: 2013
Pages: 1 - 6, DOI: 10.1109/EIT.2013.6632701

Exploring the Microsoft .NET Micro Framework for Prototyping Applied Wireless Sensor Networks

Jay D. Carlson, Mateusz Mittek, and Lance C. Pérez
University of Nebraska-Lincoln, Department of Electrical Engineering
Lincoln, NE 68588-0511
Email: {jcarlson, mmittek, lperez}@unl.edu

***Abstract*—Most Wireless Sensor Network platforms – such as the Mica, Iris, and Telos B families of motes – use low-power 8-bit microprocessors which have limited memory and processing capabilities, thus requiring researchers to implement communication protocols and data processing routines using low-level programming practices that are tedious and cumbersome. Rich features available in modern desktop operating systems – such as threads, memory management, and exception-handling – are largely absent. The Microsoft .NET Micro Framework implements a scaled-back .NET framework suitable for development on low-cost, low-power wireless sensors, while providing developers a rapid software development environment for prototyping embedded applications. Here, this technology is explored by comparing performance characteristics with those of traditional 8-bit platforms, as well as Sun SPOT, a popular platform that also uses a managed-language runtime. The .NET Micro Framework platform was found to offer researchers the most flexibility in terms of hardware and software prototyping.**

I. Introduction

General-purpose Wireless Sensor Network (WSN) platforms typically used by the research community for prototyping are based around 8-bit or 16-bit microcontrollers, such as the AVR ATmega and Texas Instruments MSP430 lines of microcontrollers. Early work has shown the advantages of 8-bit microcontrollers and their suitability for power-efficient WSN development. While WSN research has been extremely expansive in scope, relatively few general-purpose WSN hardware platforms have been developed. Some popular platforms are [1]:

- *Mica2/MicaZ* – second and third generation of motes designed by Crossbow Technology.
- *Cricket* – platform developed at MIT for indoor, ultrasound / RF (TDOA-based) localization with a design similar to Mica2.
- *IRIS* – fourth, improved generation of motes available from Crossbow Technology.
- *TelosB* – WSN modules developed by researchers at UC Berkeley and available from Crossbow Technology.
- *Sun SPOT* – Small Programmable Object Technology (SPOT) motes developed by Sun as a prototyping platform running Java Virtual Machine.
- *EZ430-RF2500* – MSP430 Wireless Development Tool made by Texas Instruments as a WSN development platform.

Without adequate system abstraction, developing a software application for embedded devices requires significant knowledge of hardware resources and functionality. Attempts have been made to increase productivity by providing a toolkit containing commonly used functions, as well as an application programming interface (API) to hardware functions.

The Mica, Cricket, IRIS and TelosB motes run TinyOS, a popular development environment for embedded WSN development. While TinyOS contains a hardware abstraction layer (HAL), a rudimentary task scheduler, and an event-driven architecture, it lacks features present in desktop application development, like memory management, threading, object-oriented design patterns, and exception handling. Sun (now Oracle) created the Sun SPOT platform to bring a rich Java-based programming environment to the embedded world. Just like the PC implementation of Java, Sun SPOT uses a virtual machine to execute intermediate bytecode. By doing this, Sun SPOT provides an environment that performs memory management, type checking, exception-handling, threading, and other functionality present in an operating system. This comes at a cost: since the intermediate bytecode is interpreted instead of executed natively, system performance is reduced significantly. Although Sun SPOT hardware is expensive, this platform provides a robust framework for rapid WSN application prototyping.

Microsoft's answer to Sun SPOT is a stripped-down version of .NET Framework, known as .NET Micro Framework (NETMF), which provides the same conveniences and trade-offs as Sun SPOT does. Here, we examine these three development environments as they pertain to WSN development. In the following section, an architectural overview of each platform is presented; then, commonly used specimens of each platform are evaluated for power consumption, and performance.

II. Background

The software environment used to develop WSN apps is a crucial component of any mote platform. In the following, software components of popular platforms are described and compared in a systematic manner.

A. TinyOS

TinyOS is an open-source development environment which provides a rich library supporting commonly used mote hardware platforms. In TinyOS, every piece of code is encapsulated as a *component*. TinyOS applications are almost universally

978-1-4673-5208-6/13/$31.00 ©2013 IEEE

written in nesC [2], which extends the C programming language by providing structure for the design paradigm of TinyOS. NesC introduces the idea of *wiring* which allows accessing the functionality provided by *components* through *interfaces*. TinyOS provides *generic components,* which abstract peripherals available on all supported platforms (such as timers or accessing the LEDs). Platform-dependent functionality can be provided in *components* developed explicitly for that platform.

Since many general-purpose WSN motes provide a connector allowing custom-designed sensor hardware to be interfaced, TinyOS's HAL separates the concept of a *platform* (which is the base hardware containing the processor, memory, and a programming interface), with that of the Sensor Board (which contains auxiliary peripherals).

The most commonly used sensor boards (for example, the MTS300 boards designed to work with Mica-family motes and equipped with light, temperature, acceleration and acoustic sensors) are supported out-of-the-box. Support for custom-designed sensor boards can be added to TinyOS by the user. Similarly, support for new hardware platforms can be added to TinyOS.

TinyOS is not a true operating system; it does not provide memory management or threading. Because of this, multitasking has to be implemented by the user in the application layer by breaking large tasks into smaller chunks that can be called individually.

TinyOS devices can interact with PC applications using external libraries available for C/C++, Java and Python.

B. Sun SPOT (Squawk Virtual Machine)

As a response to the limitations of TinyOS devices, Sun Microsystems introduced the Sun SPOT as a new (now open-source) WSN platform. The platform uses a managed execution environment [3], and was designed around the following ideas [4]:

- The mote hardware platform is a completely new embedded device with a 180 MHz 32-bit ARM microprocessor and 512k of RAM
- A provided *sensor board* enables measuring temperature, light, three-axis acceleration, and has I/O pins, buttons and LEDs
- The mote software is a small Java Virtual Machine (Squawk) which runs directly on the processor as an operating system.
- Software is typically developed using the NetBeans integrated development environment (IDE) with additional building utilities (e.g. Apache Ant).
- Utilities such as *Solarium* allow the user to observe network traffic and manage the remote devices.

The Sun SPOT runtime environment consists of:

- Application Layer – user-written code.
- Java Class Library – libraries provided by SPOT environment.
- Runtime control – loader, verifier, garbage collector, interpreter, scheduler, runtime compiler.
- Device Driver Architecture – HAL written in Java.
- I/O Library and Native code – written in C.

Squawk comes with a complete solution for memory management, garbage collection, threading, nested calls support and thread synchronization. Additionally, multiple *midlets* (user applications) can be stored on the device, sharing the hardware resources and working completely separately.

In general, Squawk can be installed on any 32-bit mote with a minimum of 8KB of RAM, 32KB of EEPROM and 160K of ROM memory [3] as it is fully compatible with the CLDC standard [5]. In practice, though, the runtime has been ported to very few platforms other than Sun SPOT. Sun SPOT technology also supports features like remote programming, network management and visualization tools. All of the above made Sun SPOT technology an almost perfect candidate for rapid WSN development and testing.[1]

C. .NET Micro Framework

The .NET Micro Framework (NETMF) is a software framework designed for rapid embedded application development. Like the Sun SPOT platform (and unlike most other software frameworks used for embedded systems), software written for NETMF (usually in Microsoft's C# language) is compiled to intermediate bytecode, called the Common Intermediate Language (CIL), which is then executed on a stack-based virtual machine (the common language runtime – CLR) running on the embedded processor.

Although managed code introduces performance handicaps, it has several advantages when compared to native code [8]:

1) *Platform-agnostic* – one compiled file can be executed on different physical platforms.
2) *Managed memory* – accessing null variables and out-of-range array indices throws an exception.
3) *Code security* – the virtual machine can check the authenticity of managed code before execution.

Although these are useful conveniences, a major benefit of NETMF is that it provides a rich API and featureset that is reminiscent of developing a desktop application, with full support for event-driven programming, threading, exception handling, graphical UI development, and robust object-oriented features.

In addition to this rich programming environment, NETMF has built-in support for peripherals and features commonly used in embedded platforms, such as general-purpose input/ouput (GPIO), pulse-width modulation (PWM), analog-to-digital and digital-to-analog conversion (ADC, DAC), and several serial communication protocols, including asynchronous serial, serial peripheral interface (SPI), and inter-IC communication (I^2C).

The layer model of NETMF consists of the following components [9]:

1) *Application Layer* – All user-written code (including the application and libraries)

[1]Regarding platform availability: It is unknown if Oracle will continue supporting Sun SPOT; the source code for the platform is no longer available, and the support forums have been down for at least several months [6]. Oracle appears to be repositioning the platform for high-end embedded platforms based on ARM application processors [7].

2) *Library Layer* – System libraries, usually written in managed code.
3) *Runtime Components Layer* – Contains the CLR implementation, as well as the hardware abstraction layer (HAL) and peripheral access layer (PAL). These components are written in C/C++ and must be compiled for the specified hardware platform.

These features make NETMF suitable for developing a wide range of embedded systems [10], [11].

1) Firmware: Traditional embedded development environments, like TinyOS, are not permanently installed on the target; rather, applications are compiled inside the environment, and the resulting binary image is uploaded to the device. This contrasts with Sun SPOT and NETMF, in which the firmware (which contains the runtime components) lives permanently on the hardware. The NETMF firmware source code for a particular device is called a *port*, and is written in C/C++.

This is a powerful way of abstracting the platform, and strictly separates the application from the hardware and library implementation; the class library system also allows strong sandboxing of code into modules that can be tested and compiled separately. And since all NETMF binary code is hardware-independent, different developers can write, test, and debug their software on any hardware of their choosing.

Because of the high degree of hardware abstraction, the particular firmware is greatly dependent on the hardware platform, and any hardware changes related to the processor and memory require the NETMF firmware to be recompiled and reloaded onto the device.

When compared to Sun SPOT's Squawk VM, which is almost entirely written in Java (with a small amount of native code executing on the device to set up the VM and execute bytecode), the majority of a NETMF firmware image is written in C/C++, including all hardware abstraction libraries, the peripheral access libraries (PAL), the CLR itself (which includes the garbage collector, memory management, threading, timers, stacks, interop support, and initialization), and most CLR libraries, including graphics, hardware, networking, debugger, time, touch, diagnostics and messaging.

To add support for a new piece of hardware, a developer creates a new *port* using the Microsoft-provided *porting kit,* which contains all the NETMF source code, plus a build toolchain and tools to automatically create the skeleton structure of the port. A *port* is generally separated into a *target*, which contains source code for a particular processor and its peripherals, and a *solution*, which is a combination of a target and system libraries. Hierarchically, a solution is a child of a target: many solutions may share the same target (perhaps they have different RAM configurations), and a child may override or augment functionality a target provides.

For example, the STM32F4-series processor from ST is an example of an actively-developed targets available for use [12]. Although the STM32F4 has a parallel camera interface on some parts, the NETMF port does not currently implement this functionality (most likely since NETMF does not have a well-defined camera interface standard, and any implementation would require a significant amount of RAM consumed as a framebuffer, plus a significant number of pins). However, a developer who wishes to integrate a camera into a WSN mote using NETMF could implement this functionality as a library, and then create a new solution that uses the STM32F4 target, as well as the implemented camera library. Again, this is all done in native code, which is compiled into a firmware image and loaded on the device. Once the firmware has been developed, it does not need to be changed or updated for different applications (unless new core hardware featured are required).

In general, most users will purchase NETMF-ready hardware that already has NETMF firmware installed on the device. Once the firmware is installed, the device appears in Windows as a .NET Debugging Platform, and Visual Studio can upload NETMF applications to it. The user does not need any knowledge of the underlying firmware or C/C++ source code to develop applications on this platform.

NETMF ports are usually built by vendors of development boards who wish to add value to existing products by adding support for NETMF, or by designing hardware specifically designed for NETMF.

2) Native Code and Interops: One of the biggest advantages NETMF has over other managed platforms (including Sun SPOT) is its support for creating interop libraries. When the managed runtime is simply too slow to perform a time-critical operation, or when there is no managed interface to an important hardware peripheral, an interop class library can be created. Interop libraries are C# managed libraries that declare classes and functions which are then implemented in native code (C/C++). By implementing time-critical code in the native execution environment, this allows the user to write the majority of her application in a managed environment, without making large sacrifices in terms of performance.

3) NETMF Hardware: Just like Sun SPOT, NETMF is open-source and designed to be highly-portable to a wide variety of 32-bit architectures, both little- and big-endian. Currently, NETMF can target ARM processors from a variety of manufacturers, as well as the Renesas SuperH processors and the Analog Devices Blackfin DSP. There is also an emulator that runs inside of Microsoft Windows.

One example of a WSN hardware platform that supports NETMF is the Crossbow Imote2 [13], which contains a PXA271 processor capable of running at 13-416 MHz, 32 MB of SDRAM and 32 MB of flash memory. The Imote2 has a CC2420 802.15.4-capable transceiver on-board[14].

4) Internet of Things: Because NETMF has an agile development environment, plus a fully-implemented transmission control protocol and internet protocol (TCP/IP) stack capable of communicating with Internet-enabled computers, servers and other devices, it is an ideal platform for developing WSN and other embedded applications that involve internet communication. Also built into NETMF is a full web server library, as well as a web client library, which are capable of communicating using the hypertext transfer protocol (HTTP), which is the ubiquitous communication standard used by the world wide web. By leveraging this technology, relatively simple WSN motes can capture and send data to centralized servers or other computers, which can use proven server-side methods and components to analyze and store sensor data [15].

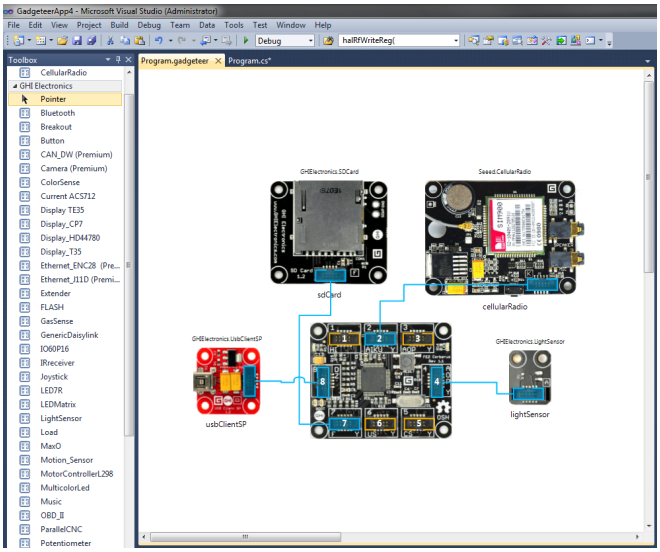

Figure 1. Visual Studio's .NET Gadgeteer designer interface provides a visual way for users to connect multiple modules together, while automatically adding appropriate support libraries to the project and initializing them with the correct port mappings.

D. .NET Gadgeteer

While NETMF does an excellent job of abstracting the processor architecture and peripherals, it does not attempt to abstract the entirety of the hardware platform, which, for wireless sensor network motes, often consists of components like sensors, transducers, and wireless transceivers in addition to the processor. This means the user of the platform must understand the communication signals on the board, and how to interface with these modules.

The hardware developer can address this issue by creating a board-support library (BSL) that abstracts the hardware platform, providing a high-level interface to the system. This is how TinyOS is designed. As the hardware is modified and updated, the BSL can be updated to reflect these changes.

This methodology works well when designing monolithic devices which have defined functionality. However, during the prototyping stage, hardware implementations may change rapidly. To enable faster prototyping, Microsoft developed .NET Gadgeteer, a unified hardware and software prototyping standard consisting of processor boards (mainboards) and plug-in auxiliary boards (modules). Mainboards typically consist of nothing but the bare essentials for a system to boot (including the processor, optional external RAM and/or flash memory, crystal circuitry). Modules can consist of displays, buttons, LEDs, wireless transceivers, external storage, sensors, battery power, cameras, and any other circuit imaginable. Once the system is assembled from modules and a motherboard, the system can be drawn in Visual Studio's Gadgeteer designer pane, which automatically imports the correct software packages into the project based on the designed embedded system.

Each connection point is called a *socket.* To account for varying connectivity requirements, there are a dozen socket types, with each type supporting some combination of analog pins, plain digital GPIO, and interfaces such as SPI, i^2C, UART, USB, LCD, and CAN. An example Gadgeteer deployment is shown in Figure 1. In this diagram, a FEZ Cerberus mainboard made by GHI is connected to a GSM cellular radio, a temperature/humidity sensor, and an SD card reader, and a USB device port. Because Gadgeteer automatically adds the appropriate BSL packages and instantiates all of these devices as objects, the user only needs to write the "glue" code to build a simple datalogging application.

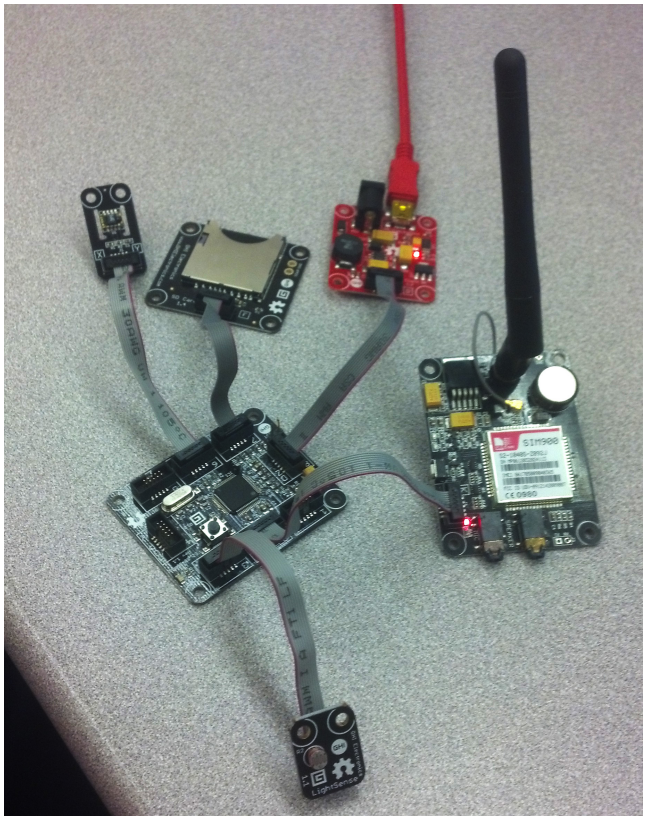

Figure 2. Example Gadgeteer system. By simply connecting these modules together, an entire WSN datalogging platform has been prototyped in hardware, without the user being required to know any details of the hardware implementation.

Gadgeteer is a useful system for prototyping embedded devices, especially those with wireless connectivity [16].

III. Platform Evaluation

To gauge the advantages and disadvantages of NETMF, two NETMF device (FEZ Cerberus and FEZ Hydra, GHI Electronics) were compared with a Oracle Sun SPOT device and a TinyOS device (TelosB, Crossbow).

The FEZ Cerberus is an entry-level NETMF Gadgeteer board that uses an STM32F4-series Cortex M4 microcontroller from ST. The Cerberus has 192 kB of RAM, and 1 MB of flash memory. The FEZ Hydra runs at 200 MHz and has 4 MB of flash memory and 64 MB of RAM, plus an LCD controller and more six more Gadgeteer sockets.

Although the performance assessment does not include any wireless testing, to make more accurate power consumption comparisons of hardware, these two NETMF boards were paired with a custom-designed 900 MHz ISM-band transceiver that uses a CC1101 RF transceiver IC from Texas Instruments.

Hardware Platform	Telos B	Sun SPOT	FEZ Cerberus	FEZ Hydra
Software Platform	TinyOS	Sun Java	NETMF	NETMF
Processor	8 MHZ MSP430F1611	180 MHz ARM920T	168 MHz Cortex-M4	200 MHz ARM926
Cost	$100	$399	$52	$102
Power Consumtion	13.53 mW	293 mW	191 mW	514 mW
Bubble Sort 100	61.59 ms	17.97 ms	107.5 ms (managed) 0.35 ms (native)	118.2 ms
Bubble Sort 1000	2021 ms	1373 ms	9158 ms (managed) 32.80 ms (native)	9752 ms
Floating-Point	2067 ms	10 ms	78 ms	28 ms

Table I
WSN MOTE COMPARISONS

The Sun SPOT is monolithic WSN platform based around a 180 MHz ARM920T processor and has 512 KB of RAM and 4 MB of flash memory. It has a built-in 2.4 GHz IEEE 802.15.4-compliant radio.

The Telos B uses an 8 MHz Texas Instruments MSP430F1611 with 10KB of RAM, 48 KB of application flash memory and 1 MB of flash memory that can be used for storage. The Telos B has a 2.4 GHz radio.

Three parameters were assessed: cost, performance, and ease-of-use. These results are presented in Table I, and discussed further below.

A. Cost

The Sun SPOT development kit, which includes two Sun SPOT devices and a base station, costs $399 [17]. The Telos B mote costs $100 without any on-board sensors [1]. The FEZ Cerberus costs $30 [18], while the FEZ Hydra costs $80 [19]. For the computer to communicate with the FEZ devices, a USB Gadgeteer module (which supplies a USB device connector and 3.3V regulator) must also be purchased; this costs $10 [20]. To add wireless capability, a CC1101-based 900 MHz RF transceiver module was designed and built. The cost of this module is approximately $12 per unit in small quantities. All three platforms have software which is freely available for download, and none of the devices require external programming hardware.

B. Performance

To test the performance of these three platforms, two software routines were written to gauge both memory read/write performance and processor performance.

1) Bubble Sort: A bubble sort routine[21] for randomly-generated arrays of size 100 and 1000 was written to test memory access, write and comparison performance. The test was repeated 1000 times for randomly-generated arrays of numbers; these results were averaged together, and is shown in Table I (rows labeled "Bubble Sort 100" and "Bubble Sort 1000"). The native integer datatype was used for each platform – that is, 16-bit integers were used for the Telos B platform, while 32-bit integers were used for the other three platforms.

The Telos B, running TinyOS, showed excellent performance. Since TinyOS performs no memory management or code execution, the bubble sort algorithm essentially ran as straight, uninterrupted C code, which is easily translatable to high-speed machine instructions.

The Sun SPOT showed excellent performance, considering the device runs a managed language environment. The debugger in the Squawk VM uses a light-weight serial interface that does not require as many updates, which reduces the interrupt rate. This may have contributed to the drastically different performance measurements.

One of the main advantages of NETMF mentioned in the previous section is that time-critical code can be written in native C/C++ and called directly from the managed environment using interop routines. For the FEZ Cerberus, this algorithm was written both in managed code (C#, executing in the CLR runtime), and in native code (using an interop function implemented in C). Our results show that, on average, the managed code version of the Bubble Sort 100 algorithm took 832 times longer to execute than the native code version, and the Bubble Sort 1000 took 279 times longer to execute in the managed environment than in native code. This demonstrates the significant overhead incurred by the managed environment.

2) Floating-Point: To test processor computational ability, an arbitrary algorithm was devised which performs 10 floating-point operations per loop iteration. This is iterated 100 times. The entire loop's execution time was recorded. This was repeated 1000 times, and these results were averaged together. The Telos B's MSP430 has no floating point unit, so its performance is substantially worse than the powerful ARM processors. This test also requires significantly less memory management, which reduces the managed runtime overhead.

C. Power Consumption

To assess power consumption, each device was hooked up to a 3.3V power supply (BK Precision, 1760A). A digital multimeter (Fluke 45) was placed in-line with the power supply to measure current accurately. All devices were programmed with the Bubble Sort 100 algorithm. Again, the Telos B mote shows excellent performance, with the entire system only consuming 4.1 mA of current.

D. Ease of Use

Both the NETMF and the Sun SPOT platform provides a comparable development experience; Java and C# are comparable in terms of language constructs and built-in language functionality. Both platforms support high-level event-oriented programming techniques, and have full object-oriented support, including a robust class inheritance model.

Both platforms have a robust debugger that can be used to step through code, display variables and object structures, and communicate textual debug statements back to the PC. Installing the development environments in Windows is trivial, and once the environments are installed, applications can be developed and deployed immediately.

The TinyOS-enabled Telos B is substantially more difficult to develop on, since it relies on loosely-coupled open source tools that come from different vendors. Software updates often break functionality, and the environment has to be carefully configured once the software is installed, which requires working knowledge of a UNIX-based system. The recommended way of developing applications is to install a Linux-based virtual machine on the development computer that has been pre-configured. This is tedious and cumbersome. Deploying applications can be done over the USB-to-serial converter built into the platform, although this provides no debugging functionality at all. To properly debug code, a JTAG emulator must be purchased and interfaced with the target. The GDB debugger only has experimental support for TinyOS's NesC constructs, so typically, the translated C code is used as source code for the debugging look-up. The embedded JTAG interface has several limitations that managed environments do not have, such as limited number of break points, and generally slow operations for variable watches and references.

IV. Conclusion

The .NET Micro Framework extended with .NET Gadgeteer hardware provides an extremely easy-to-use, low-cost development platform for rapid WSN prototyping. Results show that this ease of use comes at the cost of significant overhead, but in the prototyping stage of development, this is usually acceptable. Furthermore, monolithic platforms such as Sun SPOT lock the user into a specific hardware configuration, which may not be acceptable for the desired application. Although all four motes provide interfacing to allow custom-designed sensor boards, the .NET Gadgeteer boards running NETMF are the most flexible. As a platform, NETMF (with Gadgeteer devices) provides instant access to a plethora of sensors and functionality, and the performance limitations of NETMF can largely be mitigated by moving time-critical code into native assemblies via the interop functionality mentioned. This method of optimization proved invaluable in the Bubble Sort tests conducted.

Increasing performance of the NETMF platform is a difficult problem to solve. One solution would be to implement a global compilation system (that would slow down initial load time, but improve execution performance) [8], however, this removes several desirable aspects of NETMF – most importantly, the modularity of different binary libraries.

Future work will include prototyping an entire WSN for a medical monitoring application using .NET Micro Framework combined with custom transceiver hardware and sensors, and comparing its wireless communication performance with other embedded platforms.

References

[1] M. Johnson, M. Healy, P. Van de Ven, M. Hayes, J. Nelson, T. Newe, and E. Lewis, "A comparative review of wireless sensor network mote technologies," in *Sensors, 2009 IEEE*, pp. 1439 –1442, oct. 2009.

[2] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," *SIGPLAN Not.*, vol. 38, pp. 1–11, May 2003.

[3] Oracle, "The squawk system preliminary draft 2.1," tech. rep., Sun Microsystems Laboratories, 2600 Casey Avenue Mountain View, CA 94043, Sep. 2002.

[4] S. Microsystems, *Sun SPOT Programmer's Manual Release v6.0 (Yellow)*. Oracle, document revision 2.0 ed., Nov. 2010.

[5] Oracle, "Connected limited device configuration (cldc); jsr 30, jsr 139 overview."

[6] Oracle, "Sunspotworld forums are down for maintenance." http://www.sunspotworld.com/down.html, Feb 2013.

[7] T.-M. Grønli, J. Hansen, and G. Ghinea, "Android vs windows mobile vs java me: a comparative study of mobile development environments," in *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, p. 45, ACM, 2010.

[8] O. Sallenave and R. Ducournau, "Efficient compilation of. net programs for embedded systems," in *Proceedings of the Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems*, p. 3, ACM, 2010.

[9] J. Kühner and P. Bnskỳ, *Expert. NET Micro Framework*. Apress, 2009.

[10] R. Hilbrich, "An evaluation of the performance of dpws on embedded devices in a body area network," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 520–525, IEEE, 2010.

[11] O. Krejcar, D. Janckulik, L. Motalova, and K. Musil, "Real time processing of ecg signal on mobile embedded monitoring stations," in *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*, vol. 2, pp. 107–111, IEEE, 2010.

[12] C. Pfister and B. Heeb, "Netmf for stm32." http://netmf4stm32.codeplex.com/, Feburary 2013.

[13] M. U. Ilyas, M. Kim, and H. Radha, "Reducing packet losses in networks of commodity ieee 802.15. 4 sensor motes using cooperative communication and diversity combination," in *INFOCOM 2009, IEEE*, pp. 1818–1826, IEEE, 2009.

[14] J. A. Rice and B. Spencer Jr, "Structural health monitoring sensor development for the imote2 platform," in *The 15th International Symposium on: Smart Structures and Materials & Nondestructive Evaluation and Health Monitoring*, pp. 693234–693234, International Society for Optics and Photonics, 2008.

[15] C. Pfister, *Getting Started with the Internet of Things: Connecting Sensors and Microcontrollers to the Cloud*. O'Reilly Media, 2011.

[16] S. Hodges, S. Taylor, N. Villar, J. Scott, D. Bial, and P. Fischer, "Prototyping connected devices for the internet of things," 2012.

[17] Oracle, "Sun spot java development kit." http://tinyurl.com/ccsfttu, Feb 2013.

[18] G. Electronics, "FEZ Cerberus Mainboard." http://tinyurl.com/bv2ogcr, Feb 2013.

[19] G. Electronics, "FEZ Hydra Mainboard." http://www.ghielectronics.com/catalog/product/328, Feb 2013.

[20] G. Electronics, "Usb client sp module." http://tinyurl.com/buncx2d, Feb 2013.

[21] D. E. Knuth, *The art of computer programming, Vol. 3*, vol. 109. Addison-Wesley, Reading, MA, 1973.