Demo Abstract: Run-time Compilation of Bytecode in Wireless Sensor Networks

Joshua Ellul
Electronics and Computer Science
University of Southampton
je07r@ecs.soton.ac.uk

Kirk Martinez
Electronics and Computer Science
University of Southampton
km@ecs.soton.ac.uk

ABSTRACT

Recent work on virtual machines for wireless sensor networks has demonstrated the benefits of using a Java programming paradigm for resource constrained sensor networks. Results have shown that a virtual machine approach greatly suffers from interpretation overheads. We present run-time compilation of bytecode which leverages from a compact platform independent bytecode application encoding as well as an efficient program execution platform by converting bytecode to native code in situ.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and Embedded Systems; D.3.4 [**Programming Languages**]: Processors—*Compilers*

General Terms

Languages, Experimentation, Performance

Keywords

Java, Compilers, Wireless Sensor Networks, Bytecode

1. INTRODUCTION

The learning curve involved to begin development for sensor networks is quite steep, since not only does one have to understand the embedded systems paradigm, but new languages and tools must be adopted.

Recent work on Java compatible virtual machines for wireless sensor networks[2][1] attempt to alleviate the sensor network programming paradigm shift by allowing developers to create Java applications which are converted to bytecode and interpreted on sensor nodes. Results presented from the Darjeeling Virtual Machine [1] demonstrate the high execution costs incurred when using an interpreted virtual machine.

Outside the realm of sensor networks, the study of efficiently executing Java has been exhaustively researched. Techniques proposed include Just-in-Time compilation, bytecode optimization and also processors designed specifically to execute Java bytecode. Due to the resource constraints imposed in sensor networks, it is widely accepted that compiling bytecode to native code on sensor nodes is impossible or infeasible [5][3][7][6][4].

Copyright is held by the author/owner(s).
IPSN'10, April 12–16, 2010, Stockholm, Sweden.
ACM 978-1-60558-955-8/10/04.

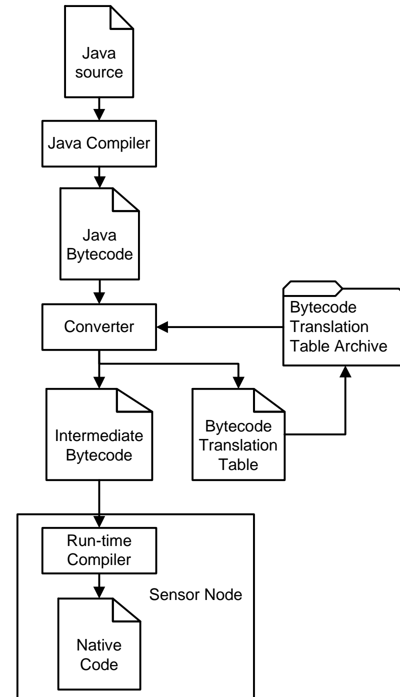

Figure 1: The translation process from Java source code down to native code compiled on the sensor node.

Our contribution is that we demonstrate that run-time compilation of bytecode to native code is both feasible and practical, and a substantial decrease in execution overhead can be achieved with a simplistic compilation model.

In this demo we will demonstrate the development environment and full process of compiling Java source code to native code on a sensor node. The demo will be in the form of a hands-on approach allowing the audience to program sensor applications in Java.

2. COMPILATION PROCESS

Figure 1 depicts our proposed compilation process of bytecode. The Java program is first compiled using the standard Java compiler. The Java bytecode generated is then passed through our converter which generates our interme-

diate bytecode. This process involves resolving constants, class names and function signatures to a more compact symbolic representation. The final step involves compiling bytecode to native code on the resource constrained sensor node.

In order to minimize footprint of the system, we have implemented a simplistic run-time compiler which translates bytecode to the respective native code without any additional optimizations. To achieve such a simplistic translation, a run-time operand stack is implemented which mimics the Java bytecode operand stack. Items are pushed and popped to the run-time operand stack natively with single `PUSH` or `POP` native code commands, thus, our run-time operand stack operates on top of the microcontroller's stack.

As an example, let's consider the compilation of the bytecode for the Java assignment `a = b + c`, where the Java compiler has referenced the local variables a, b and c by 1, 2 and 3 respectively. This will be compiled to the following bytecode:

```
iload_2
iload_3
iadd
istore_1
```

The run-time compiler will then produce the following native code commands for the above generated bytecode:

```
    ; iload_2
0   PUSH 0x0002(R11)

    ; iload_3
1   PUSH 0x0004(R11)

    ; iadd
2   POP R10
3   POP R9
4   ADD.W R10,R9
5   PUSH R9

    ; istore_1
6   POP R10
7   MOV.W R10,0x0000(R11)
```

The `iload_2` bytecode instruction is translated to a native *pushing* of the variable referenced by 2, i.e. b. The same applies for the loading of variable 3. The `iadd` bytecode instruction results in two native code *poppings* into registers 10 and 9 which are then added together and the result stored in register 9 by the `ADD.W` native command. The result, to mimic the bytecode operand stack, has to be put on the run-time operand stack, and thus, the result which was stored in register 9 is pused onto the run-time operand stack. Finally, the result is set to be stored into variable 1, i.e. a, by *popping* the result just pushed onto the stack into register 10 and then moving the value of register 10 to the memory position of variable 0.

3. EVALUATION

Table 1 displays some benchmark tests performed with C implementations, Run-Time Compilation (RTC) and the Darjeeling Virtual Machine (DVM).

As can be seen from the results our implementation is only slower by 4.7 to 7.5 times the native C code implementation, whilst the Darjeeling virtual machine ranges from 30.4 to 96.5 times slower than the native code implementation.

Table 1: Performance Benchmarks

Test	C	RTC	DVM
Bubble sort 16	0.2s	1.5s	19.3s
Bubble sort 32	0.3s	1.8s	23.3s
MD5	13.1s	61.6s	399.7s

4. CONCLUSIONS

Our work demonstrates that run-time compilation of bytecode is both feasible and practical for resource constrained devices such as sensor networks. The run-time compiled code achieves execution speeds of up to a magnitude less than that of interpretted code. We believe that our work will open up new areas of research for run-time compilers for sensor networks. We plan to continue our research with further evaluation and analysis of the generated native code. Also, debugging of bytecode on a virtual machine running on resource constrained devices can prove to be very difficult. Thus, We would like to look into reverse native code to bytecode techniques in aim of achieving direct debugging of bytecode on sensor node hardware.

5. ACKNOWLEDGMENTS

The authors would like to thank the Government of Malta for supporting this research through the Malta Government Scholarship Scheme ME 367/07/7/

6. REFERENCES

[1] N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich VM for the resource poor. In *SenSys09*, Berkeley, CA, nov 2009.

[2] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov. Mote runner: A multi-language virtual machine for small embedded devices. *Sensor Technologies and Applications, International Conference on*, 0:117–125, 2009.

[3] J. Koshy and R. Pandey. Vmstar: Synthesizing scalable runtime environments for sensor networks. In *In In Proceedings of the third international Conference on Embedded Networked Sensor Systems (Sensys*, pages 243–254. ACM Press, 2005.

[4] J. Koshy, I. Wirjawan, R. Pandey, and Y. Ramin. Balancing computation and communication costs: The case for hybrid execution in sensor networks. *Ad Hoc Netw.*, 6(8):1185–1200, 2008.

[5] D. Palmer. A virtual machine generator for heterogeneous smart spaces. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.

[6] R. Pandey and J. Koshy. A software framework for integrated sensor network applications. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 11, New York, NY, USA, 2006. ACM.

[7] B. L. Titzer, J. Auerbach, D. F. Bacon, and J. Palsberg. The exovm system for automatic vm and application reduction. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 352–362, New York, NY, USA, 2007. ACM.