Concurrency on and off the sensor network node

Matthew C. Jadud, Christian L. Jacobsen, Damian J. Dimmich

1. INTRODUCTION

The Transterpreter is a small, portable run-time for the occam-π programming language[14, 2]. As a language, occam-π provides powerful constructs for safely managing concurrency and parallelism in a framework derived from Hoare's Communicating Sequential Process algebra (CSP), a model of concurrency with message-passing synchronisation primitives[12]. Given the fundamentally parallel nature of wireless sensor networks (WSN) and their nodes, we believe there is great value in beginning with a well-defined concurrency model for reasoning about, and ultimately authoring, software on the network. Here we introduce our run-time in light of other popular environments and languages for WSN applications, and some thoughts on possible future directions given our experiences.

2. OPERATING SYSTEMS FOR SENSOR NETWORKS

Broadly speaking, software developed for wireless sensor networks either works as a custom application tailored to the hardware, or relies on services provided by an operating system. Where an OS layer is employed, these are either open-source projects[4, 1, 11] or commercial solutions[20, 9, 19, 3]. In all cases, they provide different guarantees; for example, eCos[4], an embedded version of the Linux kernel, provides a rich, POSIX environment for programmers, while VxWorks, a proprietary run-time system, delivers hard real-time guarantees to programmers developing for small devices.

In almost all cases, these run-time environments provide an impoverished model of concurrency. Most operating systems for WSNs provide an event-based model of concurrency, and rarely provide any safety for the programmer working in this space. Race hazards, deadlock, livelock, and all the other dangers of concurrent execution face a programmer working in the constrained space of a sensor mote. While environments like TinyOS[11] and the corresponding language nesC[8] provide some compile-time race condition checks, the TinyOS concurrency model of *events* and *tasks* is difficult to reason about—either formally (using modern tools for verification) or informally (as the TinyOS programming model is rather unique).

2.1 On models and algebras

The CSP algebra, developed by Tony Hoare, has provided a sound model for reasoning about concurrency for thirty years. A fundamentally cooperative model of concurrency, it has been adapted into the real-time space[17, 18], as well as crossed into many other paradigms. Furthermore, it was implemented in hardware in the form of the Transputer[21]; this implementation of the model (in the form of the occam2 programming language[13]) is well documented, and occam2 programs (as well as many programs written in occam-π) are formally verifiable.

Using existing documentation, we have built a "Transputer interpreter," or the *Transterpreter*[14]. This virtual machine (VM) for the occam-π programming language requires approximately 12KB of space on 16-bit architectures, and executes a concise (space-conserving) Huffman-encoded byte code. This VM is designed to be portable and executes code on all major desktop platforms, and has been ported to some PDAs, mobile phones, and other small devices. It has been most actively used at Kent in teaching robotics both on the LEGO Mindstorms and the Pioneer3 robotics platforms[6, 15].

Unlike TinyOS, Mate[16], Mantis[1], eCos, or other virtual machines intended for small devices, we have made a focus of providing well-reasoned support for concurrency. While a cooperative model of concurrency can provide some challenges in the context of hard real-time operation, it eases many other programmer tasks in the face of concurrent and parallel systems. We believe it is important to explore the use of concurrent programming languages (or languages that provide powerful and appropriate abstractions for managing parallelism and concurrency) in the embedded systems space.

3. MANAGING CONCURRENCY ON THE WSN

The occam-π programming language makes it trivial to execute processes in parallel: we simply declare them in a `PAR` block. To handle communication between processes, we make use of the CSP idioms of sending (`!`) and receiving (`?`) data over a unidirectional, blocking, point-to-point channel be-

tween two concurrent processes. To handle multiple inputs to a process at once, we ALTernate over many inputs, and deal with them as they are ready. What makes these abstractions particularly powerful is that they are not limited to a single WSN node; we might just as easily be expressing communications between a radio driver and a buffer in our software as between two processes running on two separate nodes in the network. The consistency of this model, both on a node and between nodes, is a boon (in our experience) to developers. For example, we have no need for "patterns," as described by Gray et. al. with respect to TinyOS[7]; our concurrency model already has natural concurrency patterns, and we can implement them directly in occam-π.

That said, there are challenges for principled languages and run-times. For example, hard real-time constraints are always a challenge, in any language—unless you evolve your language to explicitly support time[10]. While our run-time provides a clean abstraction for communications—which can easily be used for inter-node communications—robustly hiding the complexities and dangers of wireless communications between nodes "in the wild" is a challenging abstraction to get right. Also, developing a portable virtual machine requires trade-offs, and power consumption is one: is a byte code interpreter running on a small device too power hungry for use in the general case? Our initial tests suggest that this is not a significant concern, but there is a subject for future work and discussion.

Lastly, using a "different" language brings its own set of challenges. To leverage existing code, we provide facilities for bridging from occam-π to "foreign code" (C libraries, etc.)[5]; this is a fragile process, as the C-code may violate invariants our compiler previously guaranteed regarding safety in the face of concurrency. And change never comes freely; despite being a fundamentally unsafe tool, programmers may prefer to implement using eCos and C, as opposed to learning a new language that forces them to "think differently," regardless of the safety such a change might bring.

4. REFERENCES

[1] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. Mantis: system support for multimodal networks of in-situ sensors. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 50–59, New York, NY, USA, 2003. ACM Press.

[2] F. R. M. Barnes and P. H. Welch. Communicating Mobile Processes. In *Communicating Process Architectures 2004*, pages 201–218, 2004.

[3] T. Brusehaver. Linux in air traffic control. *Linux J.*, 2004(117):10, 2004.

[4] C. Curley. Open source software for real-time solutions. *Linux J.*, 1999(66es):33, 1999.

[5] D. J. Dimmich and C. L. Jacobsen. A Foreign Function Interface Generator for occam-pi. In J. Broenink, H. Roebbers, J. Sunter, P. Welch, and D. Wood, editors, *Communicating Process Architectures 2005*, pages 235–248, Amsterdam, The Netherlands, September 2005. IOS Press.

[6] D. J. Dimmich, C. L. Jacobsen, M. C. Jadud, and A. T. Sampson. The robodeb player/stage/transterpreter virtual machine. http://robodeb.transterpreter.org, 2006.

[7] D. Gay, P. Levis, and D. Culler. Software design patterns for tinyos. In *LCTES'05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 40–49, New York, NY, USA, 2005. ACM Press.

[8] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM Press.

[9] Green Hills Software. http://www.ghs.com/products/velosity.html, 2006.

[10] K. Hammond and G. Michaelson. Predictable space behaviour in fsm-hume, 2002.

[11] J. Hill. A software architecture supporting networked sensors, 2000.

[12] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., 1985.

[13] Inmos Limited. *occam2 Reference Manual*. Prentice Hall, 1984. ISBN: 0-13-629312-3.

[14] C. L. Jacobsen and M. C. Jadud. The Transterpreter: A Transputer Interpreter. In *Communicating Process Architectures 2004*, pages 99–107, 2004.

[15] C. L. Jacobsen and M. C. Jadud. Towards concrete concurrency: occam-pi on the lego mindstorms. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 431–435, New York, NY, USA, 2005. ACM Press.

[16] P. Levis and D. Culler. Mate: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA, 2002. ACM Press.

[17] G. Lowe. Prioritized and probablistic models of timed csp.

[18] G. Lowe. Relating the prioritized model of timed csp to the timed failures model, 1992.

[19] QNX Software Systems. http://www.qnx.com/, 2006.

[20] VXWorks. http://www.windriver.com/, 2006.

[21] C. Whitby-Strevens. The transputer. In *ISCA '85: Proceedings of the 12th annual international symposium on Computer architecture*, pages 292–300, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.