Challenge in Embedded Systems Design

Vasantray Jani[1]
PDDC student, LDCE, Ahmedabad
[1]*callvasant@gmail.com*

Abstract

We summarize some current trends in embedded systems design and point out some of their characteristics, such as the chasm between analytical and computational models, and the gap between safetycritical and best-effort engineering practices. We call for a coherent scientific foundation for embedded systems design, and we discuss a few key demands on such a foundation: the need for encompassing several manifestations of heterogeneity, and the need for constructivity in design. We believe that the development of a satisfactory Embedded Systems Design Science provides a timely challenge and opportunity for reinvigorating computer science.

1. Introduction

An embedded software system is sometimes defined as a computing system that interacts with the physical world. This definition is incomplete, because every software system, once it is up and running, interacts with the physical world. More precisely, what is meant is that an embedded software system has non-functional requirements, which concern the system's interaction with the physical world. Computer Science is going through a maturing period. There is a perception that many of the original, defining problems of Computer Science either have been solved, or require an unforeseeable breakthrough (such as the P versus NP question). It is a reflection of this view that many of the currently advocated challenges for Computer Science research push existing technology to the limits (e.g., the semantic web [4]; the verifying compiler [15]; sensor networks [6]), to new application areas (such as biology [12]), or to a combination of both (e.g., nanotechnologies; quantum computing). Not surprisingly, many of the brightest students no longer aim to become computer scientists, but choose to enter directly into the life sciences or nanoengineering [8]. A language-based approach is centered on a particular programming language with a particular target run-time system. Examples include Ada and, more recently, RT-Java [5]. The synchronous languages, such as Lustre and Esterel [11], embody an abstract hardware semantics (synchronicity) within different kinds of software structures (functional; imperative). More recent modeling languages, such as UML [20] and AADL [10], attempt to be more generic in their choice of semantics and thus bring extensions in two directions: independence from a particular programming language; and emphasis on system architecture as a means to organize computation, communication, and constraints. An alternative is to develop high-level programming languages that can express reaction constraints, together with compilers that guarantee the preservation of the reaction constraints on a given execution platform [13]. The application code running on a given platform, however, is a dynamical system that can be modeled as a timed or hybrid automaton [1]. We are not the first to emphasize the need for encompassing heterogeneity in systems design. Much recent attention has focused on so-called ‘metamodels,’ which are semantic frameworks for expressing different models and their interoperation [2, 9, 3]. A related class of correct-by-construction techniques is focused on the use of component interfaces [7].

There are two interfaces of a software system with the physical world: the environment and the platform. The environment includes the human users of the system, possibly a physical plant that is controlled by the system, and other application software processes that interact with the system. The platform consists of software and hardware components that implement a virtual machine on which the system is executed; it includes the operating system and network, with specific scheduling and communication mechanisms.

In the following we will lay out what we see as the Embedded Systems Design Challenge. In our opinion, the Embedded Systems Design Challenge raises not only technology questions, but more importantly, it requires the building of a new scientific foundation —a foundation that systematically and even-handedly integrates, from the bottom up, computation and physicality [14].

2. The Embedded Systems Design Problem

What is an embedded system? An embedded system is an engineering artefact involving computation that is subject to physical constraints. The physical constraints arise through two kinds of interactions of computational processes with the physical world: (1) reaction to a physical environment, and (2) execution on a physical platform. Accordingly, the two types of physical constraints are reaction constraints and execution constraints. Common reaction constraints specify deadlines, throughput, and jitter; they originate from the behavioral requirements of the system. Common execution constraints put bounds on available processor speeds, power, and hardware failure rates; they originate from the implementation requirements of the system. Reaction constraints are studied in control theory; execution constraints, in computer engineering. Gaining control of the interplay of computation with both kinds of constraints, so as to meet a given set of requirements, is the key to embedded systems design. Systems design in general. Systems design is the process of deriving, from requirements, a model from which a system can be generated more or less automatically. A model is an abstract representation of a system. For example, software design is the process of deriving a program that can be compiled; hardware design, the process of deriving a hardware description from which a circuit can be synthesized. In both domains, the design process usually mixes bottom-up and top-down activities: the reuse and adaptation of existing component models; and the successive refinement of architectural models in order to meet the given requirements.

Embedded systems design. Embedded systems consist of hardware, software, and an environment. This they have in common with most computing systems. However, there is an essential difference between embedded and other computing systems: since embedded systems involve computation that is subject to physical constraints, the powerful separation of computation (software) from physicality (platform and environment), which has been one of the central ideas enabling the science of computing, does not work for embedded systems. Instead, the design of embedded systems requires a holistic approach that integrates essential paradigms from hardware design, software design, and control theory in a consistent manner. We postulate that such a holistic approach cannot be simply an extension of hardware design, nor of software design, but must be based on a new foundation that subsumes techniques from both worlds. This is because current design theories and practices for hardware, and for software, are tailored towards the individual properties of these two domains; indeed, they often use abstractions that are diametrically opposed. To see this, we now have a look at the abstractions that are commonly used in hardware design, and those that are used in software design.

3. Summary

We believe that the challenge of designing embedded systems offers a unique opportunity for reinvigorating Computer Science. The challenge, and thus the opportunity, spans the spectrum from theoretical foundations to engineering practice. To begin with, we need a mathematical basis for systems modeling and analysis which integrates both abstract-machine models and transfer-function models in order to deal with computation and physical constraints in a consistent, operative manner. Based on such a theory, it should be possible to combine practices for critical systems engineering to guarantee functional requirements, with best-effort systems engineering to optimize performance and robustness. The theory, the methodologies, and the tools need to encompass heterogeneous execution and interaction mechanisms for the components of a system, and they need to provide abstractions that isolate the subproblems in design that require human creativity from those that can be automated. This effort is a true grand challenge: it demands paradigmatic departures from the prevailing views on both hardware and software design, and it offers substantial rewards in terms of cost and quality of our future embedded infrastructure.

4. Conclusion

Viewed in this context, the typical RTOS scheduler is just a special case of real time Linux scheduler, or in other words, the RTOS scheduler is the real time Linux scheduler running with the Round Robin. To achieve deterministic output with low latency one should must use RTAI kind of RTOS extension or proprietary RTOS.

5. References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. Theoretical Computer Science, 138(1):3–34, 1995.

[2] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A.L.Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. IEEE Computer, 36(4):45–52, 2003.

[3] K. Balasubramanian, A.S. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema. Developing applications using model-driven design environments. IEEE Computer, 39(2):33–40, 2006.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, 284(5):34–43, 2001.

[5] A. Burns and A. Wellings. Real-Time Systems and Programming Languages. Addison-Wesley, third edition, 2001.

[6] D.E. Culler and W. Hong. Wireless sensor networks. Commununications of the ACM, 47(6):30–33, 2004.

[7] L. de Alfaro and T.A. Henzinger. Interface-based design. In M. Broy, J. Gr¨unbauer, D. Harel, and C.A.R. Hoare, editors, Engineering Theories of Software-intensive Systems, NATO Science Series: Mathematics, Physics, and Chemistry 195, pages 83–104. Springer, 2005.

[8] P.J. Denning and A. McGettrick. Recentering Computer Science. Commununications of the ACM, 48(11):15–19, 2005.

[9] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. Proceedings of the IEEE, 91(1):127–144, 2003.

[10] P.H. Feiler, B. Lewis, and S. Vestal. The SAE Architecture Analysis and Design Language (AADL) Standard: A basis for model-based architecture-driven embedded systems engineering. In Proceedings of the RTAS Workshop on Model-driven Embedded Systems, pages 1–10, 2003.

[11] N. Halbwachs. Synchronous Programming of Reactive Systems. Kluwer Academic Publishers, 1993.

[12] D. Harel. A grand challenge for computing: Full reactive modeling of a multicellular animal. Bulletin of the EATCS, 81:226–235, 2003.

[13] T.A. Henzinger, C.M. Kirsch, M.A.A. Sanvido, and W. Pree. From control models to real-time code using Giotto. IEEE Control Systems Magazine, 23(1):50–64, 2003.

[14] T.A. Henzinger, E.A. Lee, A.L. Sangiovanni-Vincentelli, S.S. Sastry, and J. Sztipanovits. Mission Statement: Center for Hybrid and Embedded Software Systems, University of California, Berkeley, http://chess.eecs.berkeley.edu, 2002.

[15] C.A.R. Hoare. The Verifying Compiler: A grand challenge for computing research. Journal of the ACM, 50(1):63–69, 2003.

