Position Paper

National Workshop for Research on Transportation Cyber-Physical Systems: Automotive, Aviation, and Rail

Time-Oriented Programming

Frank Vahid, UC Riverside

951-827-4710, vahid@cs.ucr.edu, http://www.cs.ucr.edu/~vahid

Advances in embedded system software, including event and data based computation models, high-level simulation and analysis, concurrent programming concepts, and real time scheduling, can take decades to have a significant impact on practice. A key barrier to adoption of new techniques is the tendency for engineers to fall back to their first-learned techniques. Such tendencies are ubiquitous, and explains in part why, for example, soda and fast food companies pay American high-schools to allow on-campus sales (thus increasing the odds that those students will fall back to consuming those products as adults), as well as in engineers preferring the text editors, operating systems, and programming languages learned in college. More importantly, the entire perspective of software development formed in college tends to serve as the means by which an engineer understands and assimilates future techniques; changing that perspective can take many years. For example, early introduction of objects via object-oriented programming can, some argue, encourage engineers to give more importance to organization than to algorithms and thus enhance the ability to build large complex software projects.

The present introduction of programming with an emphasis on data-oriented languages like C, C++, or Java, without a concept of time or of computation models, inhibits adoption of key advances in embedded system software. Algorithm-oriented or object-oriented programming does not present time as a first class system aspect, whereas in cyber-physical embedded computing systems, time is perhaps the most important aspect. While many advances for cyber-physical software will be proposed and made in the coming years, their adoption will be hampered if the proper foundation for their adoption is not built during the teaching of software development to engineering students. Adoption of advances will continue to be limited to highly-specialized industry sub-groups who invest tremendous resources into training their engineers to utilize sophisticated embedded software techniques.

We believe that *explicit time management* is a fundamental concept that should be taught early in a computing curriculum as part of a programming approach we refer to as *time-oriented programming*. Such management can be achieved via a computation model of *synchronous state machines*, which can be captured in state machine languages or in more common languages like C, C++, or Java. Some claim that such concepts are too advanced for young students. We have found instead that, with proper abstractions (as is done for all programming), the concepts can be easily understood by young students, and that the result is a solid understanding of how to write real-time programs, with or without an RTOS, and an understanding that computation models are far more significant than programming languages.

The first abstraction involves a virtual microcontroller – an exceptionally clean microprocessor with direct access to I/O pins and with a built-in timer plus ISR functionality. The student immediately sees how, with an ISR called automatically every X milliseconds as configured by the programmer, accurate time-based input and output can be achieved in C code.

The second abstraction is the synchronous state machine. It is easy to show the limitations of the sequential program model (used by C, C++, and Java) for creating time-oriented code. In contrast, state machines with a known clock period (which is entirely distinct from a microprocessor's clock period), which we call synchronous state machines, naturally express timed behavior. Each state performs its action once and then the system stays in that state until the next clock tick, when transitions are evaluated and one of them taken. Declaration of variables of varying data types, sequential-program instructions used for actions (using the combinational restrictions for sequential code as is common in hardware description language approaches) and complex conditions, together mean that synchronous state machines are on par with common programming languages like C with respect to expressivity. Additions to the transition constructs, akin to the branching constructions in Algorithm State Machines whose roots lie in digital design, can further improve expressivity while maintaining the synchronous state functionality. To obtain specific timing behavior (such as holding an output signal for a particular time, or repeating an action at a particular rate), the developer adjusts the state machine's period and/or introduces a variable to maintain counts as certain states execute and hence to count clock ticks—in other words, the developer *explicitly manages time* in his/her description. Such explicit time management cultivates a perspective of time as a first class system aspect. Multiple concurrent synchronous state machines can be synchronized by using the same clock period, or can utilize asynchronous communication mechanisms for communication.

These abstractions must be implemented to realize real systems. The virtual microcontroller can be emulated on a wide variety of physical platforms, including any of many microcontrollers or FPGAs—we have created several such mappings. Synchronous state machines can be translated to C running on the virtual microcontroller, either automatically, or manually using straightforward templates. The net result is that a synchronous state machine can execute on a precisely-timed physical platform with predictable and consistent results. Multiple concurrent synchronous state machines can be implemented on a single virtual microcontroller as long as the worst case set of states' actions can be completely executed in the given clock period, using traditional WCET and state machine analysis techniques. Eventually, various timing simplifications can be introduced, ultimately leading to timed threading or process models, and providing a perspective better able to absorb additional temporal ideas such as timing specifications, temporal constraint or assertion languages, etc.

A third abstraction involves interfacing with sensors. Sensor interfacing typically involves low-level electronics details that detract from more important issues, while also requiring continual data from the sensor (e.g., a button wires to a microcontroller). A solution is to provide a wrapper around even the most basic sensors, such that each sensor communicates via packets. This is already done for wireless sensors, and something similar is done for sensors on buses wherein sensors are polled or transmit data at a fixed rate, but generally is not the norm for sensors directly connected to a microcontroller's pins. Packet based communication enables sensors to be shut down between transmissions to save power. A formalism to translate packet data to a continuous-time abstraction of the sensor data (0/1 or integer data) is needed, which should include error data as normal data to be expected, and thus for which behavior must be explicitly defined. The virtual microcontroller can be extended to communication via packets over each of its pins, enabling easy plug and play functionality of sensors and the microcontroller, enhancing the time-oriented educational experience by providing for richer functionality.

While the typical research position for this CPS program likely does not focus on education, after two decades of conducting research, I believe that a focus on education ultimately can provide a significant return on investment, leading to a new cadre of engineers having a solid perspective and foundation with which to solve new CPS problems and utilize new CPS solutions. It is an essential, even prerequisite, piece of the puzzle that is often overlooked.

Nevertheless, the concepts described above could conceivably find their way into forming basic implementation platforms in real CPS systems, such as in cars or trains. The advantage would be straightforward analyzable programs with clear timed behavior, at the expense of some cost overhead due to the abstraction layers. However, virtualization is becoming more common even in cost constrained systems, and thus physical realization of the above concepts is conceivable.

Short Biography

Frank Vahid is a Professor of Computer Science and Engineering at the University of California, Riverside; Chair of the Faculty of Engineering at UCR; and Associate Director of the Center for Embedded Computer Systems at UC Irvine.

He received a B.S. in Computer Engineering from the University of Illinois in 1988 graduating with highest honors, and M.S. and Ph.D. degrees from the University of California, Irvine in 1990 and 1994, respectively, where he was an SRC Fellow.

Since 1990, he has co-authored over 120 conference and journal papers, including the best paper award from IEEE Transactions on VLSI in 2000, a DATE conference best paper award, and a DAC conference best paper nomination. He is co-author of the textbooks "Digital Design," "VHDL for Digital Design," "Verilog for Digital Design," and "Embedded System Design" (John Wiley and Sons 2006, 2007, 2007, and 2001, respectively)) and of "Specification and Design of Embedded Systems (Prentice Hall, 1994) He received the Outstanding Teacher of the UCR College of Engineering award in 1997 and the College's Teaching Excellence Award in 2003. He was program and general chair for the IEEE/ACM International Symposium on System Synthesis in 1996 and 1997, respectively, and for the IEEE/ACM International Workshop on Hardware/Software Codesign in 1999 and 2000, and has served on the Steering Committee of Embedded Systems Week since its inception.

He has worked as an engineer for Hewlett-Packard and for AMCC, and has consulted for Motorola and NEC, among other companies. His research is or has been supported by the U.S. National Science Foundation, the Semiconductor Research Corporation, Philips, Motorola, Xilinx, TriMedia, and NEC, among others.

His research emphasizes highly novel self-adapting compute architectures, and creating a generation of electronic sensor blocks that non-experts and experts alike can easily compose to build basic useful sensor-based systems. His teaching emphasis includes seeking to motivate students to build innovative new systems that improve the human condition, and developing the paradigm of time-oriented programming.