THE ENGINEERING of (MIXED TECHNOLOGY) SYSTEMS

Graham R. Hellestrand

Chief Executive Officer, Vast Systems Technology Corporation

1. OVERVIEW

Mixed technology used in the definition of systems is tautological. Components of systems may be single technology elements or may encompass several technologies, such as electronic, software and mechanical, but systems incorporate several technologies. A consequence is that systems encompass interfaces between components of various technologies and the real world. In general, systems are too complex to admit to being modeled using a single modeling technique which leaves systems to be modeled using techniques typically having inconsistent views of time and signal and state values.

2. SYSTEMS IN THE *CIRCUITS AND SYSTEMS* DOMAIN

Systems of interest to the Circuits and Systems community tend to possess elements dominantly constituted from the following hierarchy of six technologies:

- analog electronics - the fundamental circuits, and the data acquiring/driving and interfacing transducers (transforming real world symbols to those interpretable in computational domains);
- digital electronics - the fast, functional transducers of discrete symbols;
- processors (conventional CPUs and specialized processors, such a signal processors and devices attached to other processor) - the slow behavioral and functional transducers of discrete symbols and the typical centers of control effecting the state maintenance and transitions of the system;
- executive software (kernels or operating systems) which manages the states of the various processors (including attached devices) to present a standard machine (augmented ensemble of processors). The standard machine interprets *executable* application software, and provides interfaces by which to govern the interactions amongst the ensemble of processors and digital and analog components constituting the system, and between the system and the real world;
- re-targetable compilers which map application software to *executable* forms interpretable by candidate, target machines; and
- application software which form the computational domains of systems, assuming the existence of underlying interpretive machines.

In addition, real systems directly interface with mechanical and optical subsystems via actuators, signal transducers, and sensors, adding another two technologies to the hierarchy of system technologies.

The province of software engineering (called computer science) is largely the building of correct useful computational domains, re-targetable compilers, the

executive software and contributing to the architecture of the underlying interpretive machines. This requires intimate knowledge of, and skill in using, possibly all of the latter three technologies in the dominant technology hierarchy, together with an understanding, at least, of the processor technology. The province of hardware engineering (called electrical engineering) is largely the building of correct useful interpretive machines and their interfaces with the real world. This requires intimate knowledge of the first three technologies in the hierarchy, understanding the constraints of, and mechanisms for, effectively interfacing with the mechanical and optical contrivances emulating the real world, and an understanding of the software engineering domain. The province of systems engineering (called computer engineering) is largely the building of correct useful (mixed technology) systems and subsumes software and hardware engineering. This requires intimate knowledge of some technologies and understanding of the remaining technologies, determined by the requirements of the target system, together with the ability to manage the integration of systems constituted from components incorporating various technologies.

Major and complex tasks of systems engineering are the requirement specification and the architectural specification of the target system, the correct design of system subsystems and components (elements) together with (possibly several) appropriate implementation technologies, the definition of interfaces between elements, the integration of the elements into the system, and the production of correct, reliable and economically viable artefacts. The use of methodologies appropriate for managing the design processes, such as hierarchical top-down partitioning and testing, the definition of testable interfaces between the hierarchically deduced modules including trans-technology interfaces, and the bottom up synthesis and testing of all elements into a functionally consistent, reliable, testable and operable unit. In this model of the methodology, elements of different technologies are likely to be confined within individual modules, which may be deeply embedded within the system itself. The use of synthesizable specification notations is a fundamental dream (some would say requirement) for engineering correct (mixed technology) systems. In this context, synthesis subsumes the capabilities of modeling and simulation directly from specification notations.

In practice, systems in the *CAS* domain are dominated by software engineering requirements and the hardware engineering requirements are predominantly digital. Systems, as such, are rarely treated adequately in universities, at the undergraduate or postgraduate level. The language with which the industrial and academic communities communicate about systems is different. The lack of migration between the academic and industrial communities means that academics largely lack systems experience.

3. A PREDOMINANTLY DIGITAL SYSTEM AND ITS MODEL

The overview of a system hides much of the complexity and detail contained within each of the components. It is these details which produce the overwhelming complexity which is characteristic of the many modern systems. This section details some levels of this complexity for dominantly digital systems.

To reiterate, in the predominantly digital context, a system is comprised of hardware - an ensemble of application specific analog and digital hardware, computer systems (central processing unit (CPU), processor bus, memory) and special purpose processors, application software – tasks being threads and processes, and a kernel or operating system. Figure 1 below illustrates the relationships between system constituents.

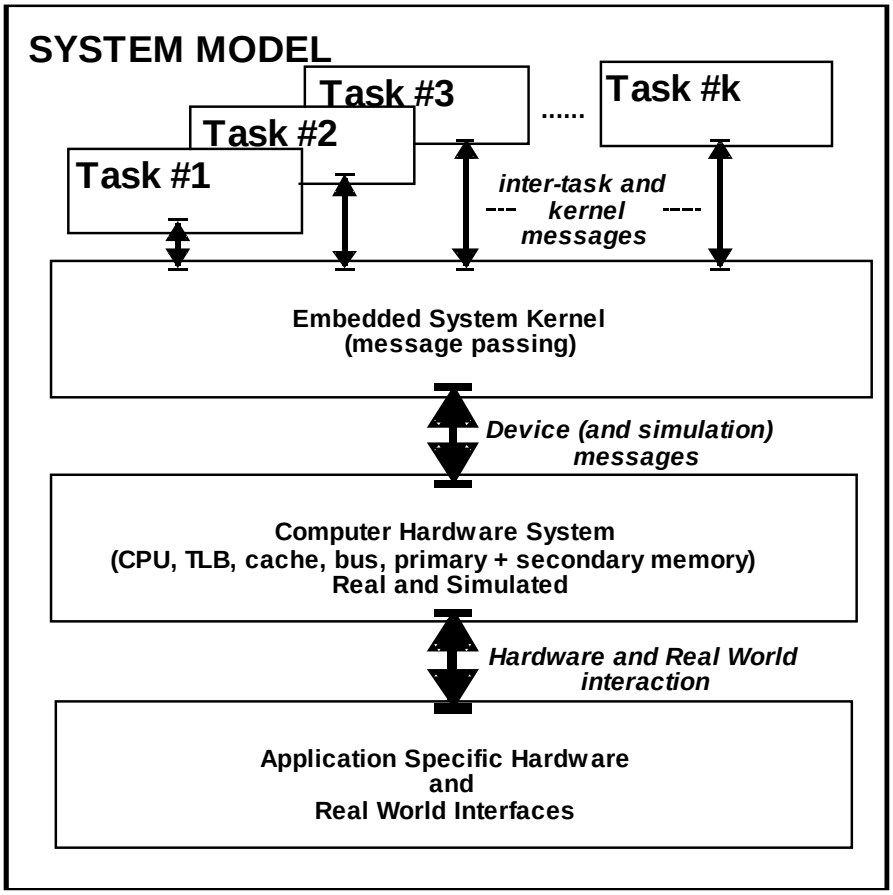

Figure 1. The System Model

The specification of a hardware-software system architecture naturally exists within a hardware specification. The rationale is that application software operates within an environment defined by a processor architecture, the kernel running on the processor, and the associated system such as the processor bus and primary and secondary storage. The Processor system and its bus and storage paraphernalia constitute a virtual processor model (VPM) which is modeled within a hardware module embedded in a hardware architectural specification.

4. THE REQUIREMENTS OF EFFECTIVELY MODELING SYSTEMS

For a model of a system to be useful it must support analysis, synthesis and simulation, and ideally verification in a formal sense.

To effectively and accurately model a system of application and kernel code being interpreted by a computer system whilst concurrently and intimately interacting with application specific digital and analog hardware, many of the details of the

VPM must be judiciously chosen. A modern processor described at the register transfer level may require hundreds to thousands of events in a simulator to model one instruction interpretation cycle. Using such models up to 100 machine instruction interpretation cycles can be simulated in a second on a fast workstation. If the processor model is simplified, companies report the simulation of up to 5,000 instruction interpretation cycles per second. To test a modest embedded system just initializing a relatively sophisticated kernel may require the simulation of 10^9 instructions interpretation cycles, or a simple image processing algorithm may require the simulation of 10^{11} cycles. The simulator would be busy for 50 hours in the first instance and for about 200 days in the second and many such simulations are required for testing systems. For simulation tools to address the problem of modeling and testing systems they need a 10^4 to 10^5 increase in speed beyond the available tools.

The issue of system simulation performance is moot if the accuracy of timing in hardware simulation and across the hardware-software and hardware-mechanical interface is sacrificed to gain the performance. In systems development the hierarchy of interaction boundaries between:

A. the real world and the analog (interface) hardware;
B. the digital and analog hardware;
C. the application specific digital hardware and the computer system hardware,
D. the computer system hardware and the kernel (operating system) software;
E. the kernel software and the application software (potentially multi-tasking); and
F. the application software and the real-world

is where most of the difficult design and testing problems lie. In digital systems the integrity of interfaces A and B is assumed.

These boundaries require considerable specification, analysis and simulation ingenuity to allow effective modeling and interactive, accurate testing. Across boundaries C and D, detailed hardware events need to be simulated. While concurrently across boundaries D, E and F the few instructions of application code which invoked the many instructions of the kernel system call code, also needs to be simulated. And the integrity of time, symbol transformation and the causality of events across all boundaries B-F must be guaranteed, if validating the correctness of systems is the goal. The event-time discrepancy between real and simulated models over boundaries B-F is about 10^6. If continuous time is also to be modeled for boundaries A and B, the event-time discrepancy may increase by another factor of 10^3 -10^4. Modeling boundaries A-F with integrity remains a difficult, open question.

5. EFFECTIVELY SIMULATING PREDOMINANTLY DIGITAL SYSTEMS

As may be deduced one key to effective system modeling lies in the capability of the VPM. Another key is the maintenance of causality and relative time across simulation domains – digital hardware, VPM, software - where the relationship between time is not maintained but causality must be. A final key is the modeling of symbols and symbol transitions across domain boundaries. Effective modeling requires the ability to model failure that triggers the requirement for super-symbol

interpretation. The proliferation of symbols representing values of signals, is anathema to fast simulation but is a requirement of accurately modeling failure.

The system depicted below has been built by VaST Systems Technology Corporation and runs software simulations at 10^8 instruction cycles per second whilst maintaining timing accuracy across the hardware-software interface and supports multiple VPMs in a single model. The hardware and VPM modeling domains are clear. The *back-plane kernel* mediates the maintenance of causality across the domains that define their own relative frames of space and time. The capability of the specification notation and the notation interpretation scheme determine the ability of a tool set to support analysis, synthesis and verification, and to model failure.

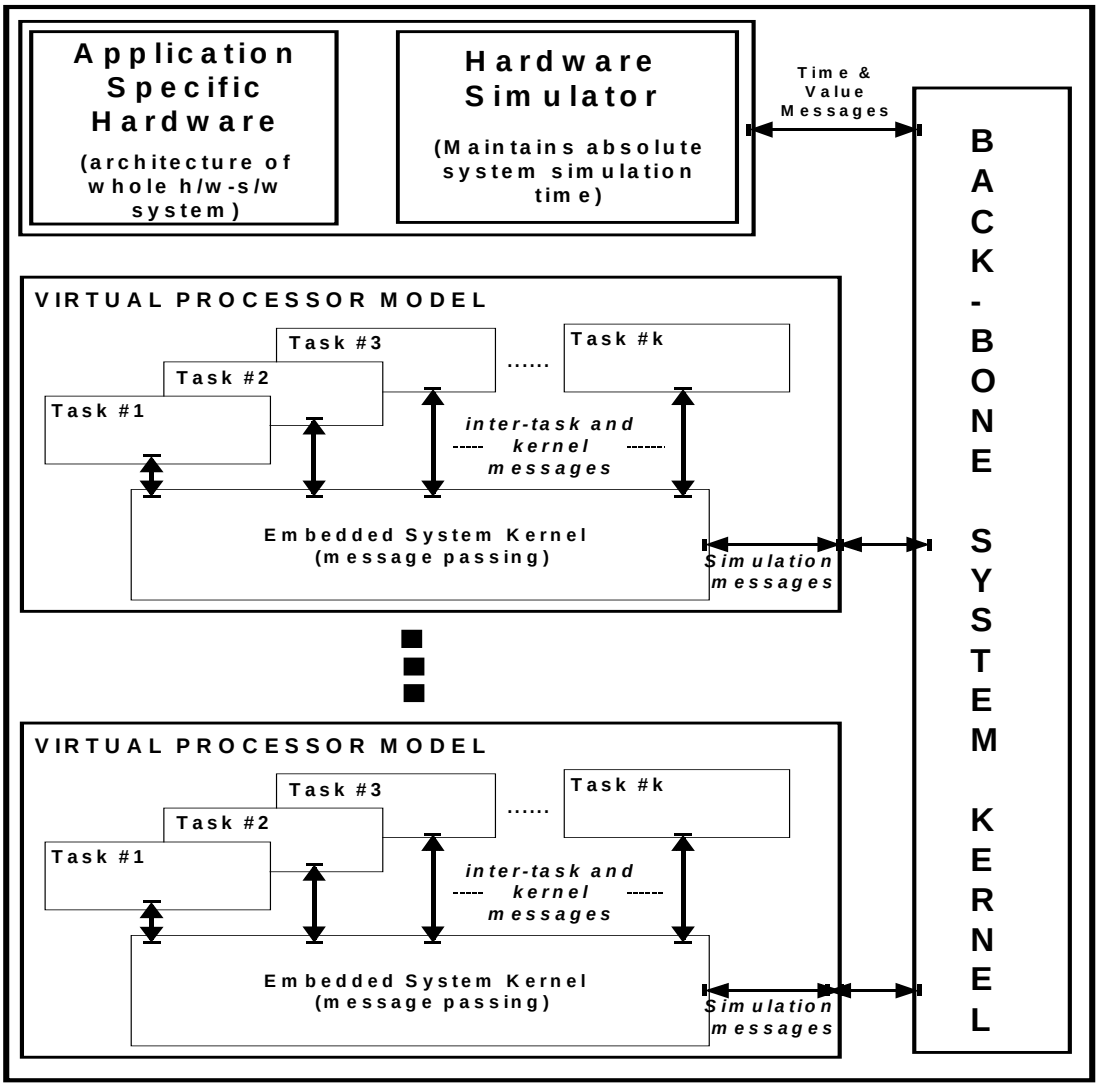

Figure 2. A Hardware-Software System Simulator Model

Models for further technologies may be added to this open-ended system, including: mechanics, continuous processing (such as in chemical engineering), and business systems (work-flow and mixed work-flow/economic modeling.