MODELING OF SOFTWARE-HARDWARE COMPLEXES

Position Statement

Nikil Dutt
ACES Laboratory, Center for Embedded Computer Systems.
Donald Bren School of Information and Computer Sciences.
University of California, Irvine, CA 92697-3435, USA.
dutt@uci.edu

Complex embedded systems typically comprise a large number of interconnected software and hardware components operating in highly dynamic environments that need to deliver user-specified QoS under demanding design constraints: performance, energy consumption, system cost, system reliability, etc. As the software content of such embedded systems continues to increase, it would appear that the boundary between reasoning about hardware versus software begins to blur. However, modeling strategies for such software-hardware complexes need to *holistically* embody key characteristics of both hardware and software domains in order to effectively capture various facets of design, analysis, verification and synthesis. It becomes even more important in the context of effective co-design of such software-hardware complexes. This is true for the entire gamut of embedded systems that range in size and complexity from "small" embedded Systems-on-Chip (SoCs) to large "system-of-systems" that comprise multiple, distributed, heterogeneous embedded systems working in consort to deliver a desired set of services for the end user.

Such complex embedded systems have several distinguishing characteristics, including, but not limited to:

- *Complexity and scale*. The distributed nature of emerging embedded systems poses significant challenges for managing the complexity of interactions among distributed components, and managing the computational and network resources to keep the system coordinated.

- *Dynamic and uncertain operating environments*. Embedded systems often operate in dynamic and uncertain environments due to a variety of reasons, including unpredictable user requests, hardware and software resource failures, environmental noise, and incomplete knowledge of the system operating state.
- *Stringent timing requirements*. Timing plays a critical role in the correct performance of any embedded system. Indeed, timing is part of key functionality for mission-critical applications that must execute tasks reliably within stringent deadlines.
- *Energy awareness*. Modern embedded systems (regardless of their scale and complexity) increasingly need to address energy as a first-class design constraint.
- *Cross-Layer interactions*. Complex embedded systems are highly networked, and involve end-to-end interactions among multiple layers (application, middleware, network, OS, hardware architecture) in a distributed environment. Thus there is a need for a holistic modeling mechanism that captures such cross-layer interactions.

To operate such systems effectively while maintaining the desired QoS, multiple performance-related parameters must be dynamically tuned to adapt to changing application modes and operating conditions. In addition to performance management, the system's operational constraints and timing characteristics must be verified at design time to ensure its correctness, feasibility, and safety. To verify correct operation, the designer must make necessary (and sometimes strong) assumptions about the services provided by the underlying execution platform, such as guaranteed task execution rates and deadlines, reliable message communication between system components, etc. These requirements must therefore be satisfied by the execution platform (i.e., the middleware, operating systems, and networks) to ensure that the management framework performs correctly at runtime.

Model-based techniques have recently been proposed as a promising approach for capturing, managing and refining overall system behavior in a reliable and efficient manner. Domain specific models can be generated to match the idiosyncrasies of specific application domains, while a generic model-based framework can provide a backbone to address a variety of problems (e.g., power management, efficient resource allocation and provisioning) using well-established concepts and techniques.

The software and hardware communities have much to give, as well as to learn from each other. A holistic integration of best practices and proven technologies from each domain is only the start; we will need to develop modeling techniques that are simultaneously domain-aware and platform-

compliant, in order to tame the ever increasing complexity and challenges posed by tomorrow’s software-hardware complexes.