Towards Two-Level Formal Modeling of Computer-Based Systems

Gabor Karsai
(Vanderbilt University, Nashville TN
gabor@isis.vanderbilt.edu)

Greg Nordstrom
(Vanderbilt University, Nashville TN
gnordstr@isis.vanderbilt.edu)

Akos Ledeczi
(Vanderbilt University, Nashville TN
akos@isis.vanderbilt.edu)

Janos Sztipanovits
(Vanderbilt University, Nashville TN
sztipaj@isis.vanderbilt.edu)

Abstract: Embedded Computer-based Systems are becoming highly complex and hard to implement because of the large number of concerns the designers have to address. These systems are tightly coupled to their environments and this requires an integrated view that encompasses both the information system and its physical surroundings. Therefore, mathematical analysis of these systems necessitates formal modeling of both "sides" and their interaction. There exist a number of suitable modeling techniques for describing the information system component and the physical environment, but the best choice changes from domain to domain. In this paper, we propose a two-level approach to modeling that introduces a meta-level representation. Meta-level models define modeling languages, but they can also be used to capture subtle interactions between domain level models. We will show how the two-level approach can be supported with computational tools, and what kind of novel capabilities are offered.
Category: D.2.2 Tools and Techniques

1 The Need

A Computer-based System (CBS) is essentially a *physical* system that consists of an information processing (IP) component, a physical environment (PE), and a sensing and actuation mechanism that establishes the interface between the two. Figure 1 illustrates this statement. The behavior of the resulting system is determined by all the components in this ensemble: the hardware and the software of the information processing component, the interfaces to the physical processes, the physical environment, and the interaction among all of these. We argue that to develop the engineering science of these systems one needs an integrated approach, where all aspects of the design can be analyzed.

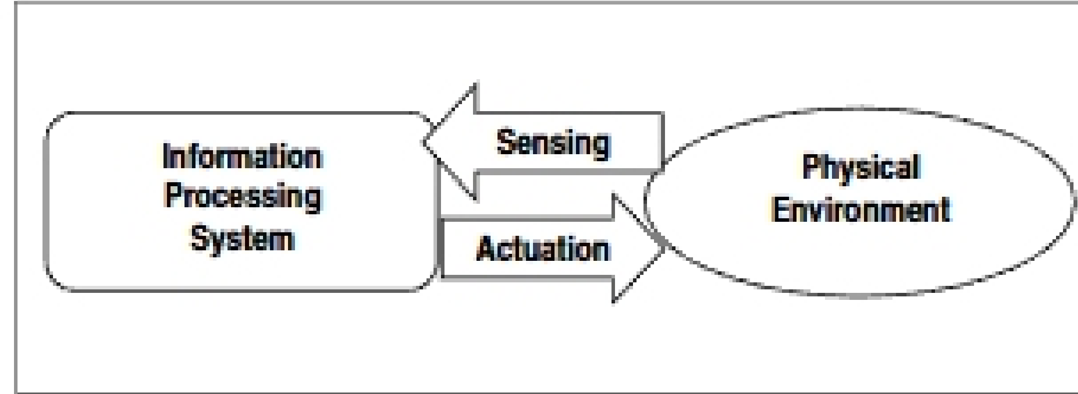

Figure 1: A Computer-based System

In any engineering discipline the rigorous analysis of a design artifact happens through the manipulation and analysis of mathematical objects, called *models*. Frequently physical prototypes are also built for experimentation, but still the analysis—and the understanding—happens with the help of the mathematical objects. We need a similar approach to CBS, such that we can build models of the systems. These models, by the very nature of the CBS, have to be able to represent both the IP and the PE components, together with the interaction between the two.

An illustrative example can be found in the area of digital avionics. Let us consider a fly-by-wire system that transforms pilot commands and data from environmental inputs (e.g. from air data computers and motion sensors) into actuator commands that act on the control surfaces of the aircraft. When *designing* such a system, one uses the knowledge of control theory, aircraft dynamics, and other engineering disciplines to establish the control laws, to calculate the controller gains, etc. The physical environment: aircraft body dynamics, actuator dynamics, etc. determine how the information processing component should behave. When *implementing* such a component one works with software abstractions: modules, tasks, synchronization, floating-point and fixed-point variables, task timing, jitter, etc. *The essential problem of CBS is the subtle interaction between the IP and PE of the system.* When a hardware or software implementation decision is made, that will have an impact in terms of the physical environment. For instance, selecting a particular fixed-point representation for a physical quantity determines what the expected maximum and minimum value of that quantity is. The IP will simply not work if these assumptions are violated by the physical environment. On the other hand, time constants determined by the physical environment will have an impact on the hardware and software implementation. This leads to a vicious circle of interaction, where changes on one side impact the other and vice versa. In order to understand CBS it is not sufficient to model just the IP or just the PE components, we need techniques for simultaneous modeling that also support capturing the interactions.

Naturally, we want to analyze, validate, and predict the behavior of the integrated system from these models. Hence, the modeling language should be rich enough to capture all these aspects. Additionally, if feasible, we would like to *synthesize* the implementation of the system from the model. By synthesis we mean a process that leads from design models and component libraries to automatically generated software and hardware components. This last step is made possible by the development of various design automation algorithms and tools. Design automation is

very successful in the hardware world but recently software synthesis tools have also become available.
In this paper, we want to address the following questions: What is the right way to model CBS? What is the "modeling language" to be used? *We argue that there is no single modeling language, which would satisfy the requirements of all CBS.* Instead, we propose a two-level approach, where area-specific modeling tools are used for creating domain-specific models, and these tools are represented in terms of (and built from) a higher-level *meta*-model.

2 The vision

Except for trivial cases, CBS are closely related to mature engineering disciplines. An industrial instrumentation and control system relies on signal processing, control engineering, and electronics. An on-line problem-solving environment for chemical manufacturing uses process engineering knowledge. An avionics system employs the concepts of aerodynamics, control theory, fault tolerant computing, and others. Arguably, CBS are *always* used in an engineering context, where the solution provided by the CBS must fit in.
In order to help design the hardware and the software for a CBS, one must use *domain-specific* terminology, concepts, and techniques. By domain, we mean the larger engineering discipline where the CBS belongs. CBS are often the result of cooperation between domain engineers and hardware and software designers. We argue that the common language used by these participants should be that of the *domain*, and not necessarily that of computer engineering.
Modeling languages that capture interesting properties of software systems (e.g. UML) are very rarely suitable for modeling the entire system. Note that the "entire system" includes not only the hardware and the software, but the environment as well. While there are some aspects of UML that make it suitable for modeling dynamic, reactive systems (e.g. state charts), it is inadequate for capturing models in the form of Laplace transforms or differential equations. Mature engineering disciplines (e.g. control theory or chemical engineering) have their own languages and forcing the use of another modeling language is not acceptable.
To summarize, we emphasize the need for domain-specific modeling languages in order to properly model all aspects of CBS. As CBS are used in widely differing domains, there is a potentially very large number of modeling languages that can—and should—be used. To phrase it differently: the modeling languages for CBS changes from domain to domain.
Another aspect of CBS is their *integrated* nature. They integrate different disciplines: hardware design, software engineering, performance modeling and engineering, in addition to the "base" domain engineering discipline. When one creates models for these systems, it is unavoidable that these models be integrated. For example, models of the software architecture should be considered in conjunction with the models of the hardware system to determine end-to-end timing latencies. Therefore, while an engineering modeling language dominates the modeling process, one must also address the issue of integrating these models with models that are closer to the domain of computer engineering. We argue that integration of models is not only an