Computer-based system design inextricably combines system and software development. An embedded software component, whose logical behavior is defined in some computer language, is “instantiated into” a physical behavior on a computing platform. The instantiation of logical behavior into physical behavior is complicated by the following factors:

1. Physical behavior is directly influenced by the detailed physical characteristics of the devices involved (physical architecture, instruction execution speed, communication bandwidth and others).
2. Modern processor architectures introduce complex interactions between the code and essential physical characteristics of the device (speed, power dissipation, etc.).
3. Lower layers of typical software architectures (RTOS scheduler, memory managers, middleware services) interact with application code in producing the net physical behavior.
4. Properties of physically instantiated software components interfere with each other due to the use of shared resources (processors, buses, physical memory devices, etc.).

Analysis of essential physical characteristics of designs can be significantly simplified by over design: we use enough resources to minimize or eliminate the need for resource sharing (computation, communication) or consider hard to compute physical properties unessential (e.g. power). Unfortunately, in most practical cases, efficiency and application circumstances force us to explicitly design for physicality, which requires deep modeling not only the functional structure and behavior of software but also the physical structure and behavior of the distributed computing platform and their interactions.

The cost of modeling on this level of detail is a major concern. The development of detailed enough models to compute all interesting physical properties of computer-based systems can be cost prohibitive without sophisticated technology supporting model-based design.

Modeling, model analysis, model transformation and model-based code generation play fundamental role in integrated systems/software design. All of these steps use models, which are formal, composable and manipulable during the design process. The modeling languages, in which models are expressed, are domain-specific, offering system designers modeling constructs and syntax that are close to their application domain. In model-based design, domain-specific modeling languages (DSMLs) are used to capture the structural and behavioral aspects of embedded software and systems. Their semantics emphasize concurrency, communication abstractions, temporal and other physical properties.

The goal of this talk is to describe our approach to model-based design, which is based on an integrated framework called Model-Integrated Computing (MIC) [1]. MIC includes theoretical foundations for specifying the syntax and semantics of DSML-s, and provides a meta-programmable tool suite for modeling, model transformation, code generation and tool integration [2]. The approaches and tools discussed are used in a wide range projects focusing on different categories of computer-based systems [3].

References

[3] www.isis.vanderbilt.edu