A Model-Based Design Methodology for Cyber-Physical Systems

Jeff C. Jensen
National Instruments
Berkeley, CA 94704
E-mail: jjensen@ni.com

Danica H. Chang
University of California, Berkeley
Berkeley, CA 94720
E-mail: danicachang@berkeley.edu

Edward A. Lee
University of California, Berkeley
Berkeley, CA 94720
E-mail: eal@eecs.berkeley.edu

***Abstract*—Model-based design is a powerful design technique for cyber-physical systems, but too often literature assumes knowledge of a methodology without reference to an explicit design process, instead focusing on isolated steps such as simulation, software synthesis, or verification. We combine these steps into an explicit and holistic methodology for model-based design of cyber-physical systems from abstraction to architecture, and from concept to realization. We decompose model-based design into ten fundamental steps, describe and evaluate an iterative design methodology, and evaluate this methodology in the development of a cyber-physical system.**

***Index Terms*—model-based design, cyber-physical systems, embedded systems, LabVIEW, Ptolemy II.**

I. Introduction

Model-based design (MBD) [1]–[3] emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems. A complete model of a cyber-physical system represents the coupling of its environment, physical processes, and embedded computations. Modeled systems may be tested and simulated offline [4], enabling developers to verify the logic of their application, assumptions about its environment, and end-to-end (*i.e.* closed-loop) behavior.

The design of a complex cyber-physical system — especially one with heterogeneous subsystems distributed across networks — is a demanding task. Commonly employed design techniques are sophisticated and include mathematical modeling of physical systems, formal models of computation, simulation of heterogeneous systems, software synthesis, verification, validation, and testing. We have yet to find a set of sequential steps that, if followed carefully and correctly, encompasses each of these design techniques and sufficiently governs the development of a complex cyber-physical system. Instead, we propose a set of steps, not necessarily sequential but necessarily codependent, that facilitates the co-evolution of a model of a cyber-physical system with its realization. Our focus is on design methodology, and for each step we offer only a cursory introduction to a vast field of research. Since no model can ever be complete [5], a practical implementation of this methodology is to iteratively visit each step until design requirements are met.

II. Model-Based Design in Ten Steps

A. MBD Step 1: State the Problem

Use simple language to describe the problem to be solved, without the use of mathematics or technical terminology. This is the "elevator speech" for the project and is a handy reference for developers, collaborators, colleagues and experts, vendors, and machine shops. Developers of large or safety-critical applications should also write a project plan consisting of requirement tracking, metrics, formal testing processes, and (most importantly) a process for peer review. Given the multidisciplinary nature of cyber-physical systems, this step is necessary to effectively communicate design requirements.

B. MBD Step 2: Model Physical Processes

A first iteration of physical modeling should establish basic observations and insight into relevant physical systems, such as the environment in which the cyber-physical system resides, or physical processes to be controlled. Models of physical processes are simplified representations of real systems, and are usually in the form of systems of differential equations or Laplace transfer functions. What may begin as simple mathematical models may need to be refined following development of a control algorithm, specification of hardware, and testing of components and subsystems.

C. MBD Step 3: Characterize the Problem

Isolate fixed parameters, adjustable parameters, and variables to be controlled. Identify quantities that characterize physical processes, such as configuration spaces, safety limitations, input and output sets, saturation points, and modal behavior. Understand how a physical process may interact with a computation, including end-to-end latency requirements, fault conditions, and reactions to noise and quantization.

D. MBD Step 4: Derive a Control Algorithm

Determine conditions under which physical processes are controllable and derive a suitable control algorithm to be executed by an embedded computer. Use the problem characterization to specify requirements on latencies, delays, sampling rates, jitter, and quantization so that the physical dynamics of interest can be accurately measured and suitably controlled; these requirements must be satisfied by the computational platform used. In highly distributed applications, or systems that are globally asynchronous but locally synchronous, it may be necessary to select models of computation before a control algorithm can be derived. Revisit this step after selecting models of computation and specifying hardware to determine the impact of latency jitter or variable sampling rates introduced by an asynchronous model of computation, or saturation or other nonlinear artifacts introduced by hardware.

E. MBD Step 5: Select Models of Computation

A *model of computation* is a set of allowable instructions used in a computation along with rules that govern the interaction, communication, and control flow of a set of computational components [4]. A formal model of computation defines semantics that often result in greater analyzability and the potential to simulate cyber-physical systems through the use of heterogeneous modeling tools. Models described by formal models of computation may be easier to analyze with respect to determinism, execution time, state reachability, memory usage, and latency [6], [7]. These software dynamics alter the evolution of a cyber-physical system, and if modeled may be generalized and used in an MBD workflow. The inherent complexity of many cyber-physical systems often necessitates the composition of multiple models of computation. Advantages of using a specific model of computation depend on its semantics, whether timing constructs are used, and whether it is Turing-complete.

F. MBD Step 6: Specify Hardware

Select hardware that is capable of withstanding the environment, interacting with the modeled physical systems, and implementing the control algorithm. For each component, consider its input and output bandwidths, delay from input to output, power usage, measurement resolutions and rates, and mechanical parameters such as form factor, rejection of electrical interference, durability, and lifespan. Mechanical actuators should be capable of producing forces and torques in excess of minimum values derived from earlier problem characterizations. Consider and model the impacts of using cost-effective substitutes for ideal parts; keep in mind that manufacturer specifications are not always accurate, and that hardware components should be independently tested.

Selection of an embedded computer may hinge on a deeper understanding of latency and execution time requirements of control algorithms, worst-case execution time measurements of synthesized software, and reasoning as to how software will interact with a specific hardware architecture. This step may require several iterations with software design and simulation before an embedded computer can be selected with confidence.

G. MBD Step 7: Simulate

Solve the problem using a desktop simulation tool. If multiple models of computation are to be used, simulation and synthesis tools must allow the compositions of and interactions between multiple models of computation. Depending on the robustness of the development environment, incorporate models of sensors, actuators, and physical processes. Use platform-based design to separate application logic and architecture-specific software into modular components, which can improve code portability, reduce the impact of changing hardware components, and allow components to be reused in other contexts [8].

Models of individual components and subsystems are as important as a complete end-to-end model. Component models provide a test harness for construction, verification of synthesized software, and testing. If no one modeling tool can completely describe the system, then for each subsystem use the modeling tool that best captures its dynamics. While disjoint simulations cannot represent relationships between signals that cross subsystem boundaries, or the behavior of compositions of these subsystems, the exercise facilitates co-iteration of physical modeling, simulation, and testing. Ptolemy II is a versatile tool for researching heterogeneity [4], allowing developers to easily create new models of computation and simulate their behavior.

Many simulation tools exist, but most are limited to only a few models of computation and are unable to capture the interactions between heterogeneous systems. In our case study, we use Ptolemy II and LabVIEW. The heterogeneous modeling capabilities of Ptolemy II are well-known [9]. LabVIEW is a capable tool in this realm: continuous systems are expressed as ordinary differential equations or differential algebraic equations, and discrete systems are expressed as difference equations, in the LabVIEW Control, Design, and Simulation Module; concurrent state machines are expressed in models created in the LabVIEW Statechart Module (which implements a variant of Harel's Statecharts); imperative expressions are expressed as formula nodes (a subset of ANSI C) or MathScript nodes (compatible with scripts created by developers using The Mathworks, Inc. MATLAB software and others); data acquisition and program flow are expressed in structured dataflow, which is general enough to allow the composition of each of these models of computation [10].

H. MBD Step 8: Construct

Build the device according to specifications, taking note where exceptions have been made that may impact earlier modeling. Plan construction in a way that allows individual components and subsystems to be tested against theoretical models, which facilitates co-iteration between simulation and testing.

I. MBD Step 9: Synthesize Software

Code synthesizers are sometimes incorporated into desktop simulation environments, examples of which are LabVIEW and Ptolemy II. They may directly support the embedded computer used, or generic code may be synthesized and tied to handwritten, architecture-specific code. Unlike many tools, models written in LabVIEW are natively executable across many platforms without knowledge of architecture-specific instruction sets or drivers, including desktop computers (for simulation or data acquisition), real-time processors, FPGAs, and ARM-based microcontrollers. LabVIEW models may target custom platforms through arbitrary C code generation. If code synthesis is infeasible or unavailable, handwritten code should carefully follow the selected models of computation.

Assuming the code synthesizer produces code that faithfully executes the semantics of the models of computation used, the logic of synthesized code is correct by construction. Timing behavior, however, must still be verified, as code generators and compilers may introduce software timing artifacts, and hardware features such as pipelines and caches may introduce jitter. Other constraints such as memory footprint or processor utilization may also require independent verification. Timing and other constraints should be verified against existing models.

J. MBD Step 10: Verify, and Validate, and Test

Configure adjustable parameters to create test environments that are as simple as possible, and test each component and subsystem independently. Computational systems may be isolated from physical systems via hardware-in-the-loop testing, where programmable hardware such as embedded computers or FPGAs simulate the feedback from physical or other computational processes. Measurements of execution time and latency can be used to refine previous models, and unexpected test results may point to errors in modeling or implementation.

Formal verification and validation give insight into the behavior of an algorithm over all or certain combinations of its inputs, or over the course of time. Precisely state requirements and translate them into a formal specification for verification and validation. List invariants that should be verified during testing. Verification and validation are perhaps the most difficult aspects in the design of a cyber-physical system.

III. Case Study: The Tunneling Ball Device

The Tunneling Ball Device (TBD) [11] is a cyber-physical system whose operation demands hardware and real-time embedded computing that deliver high-precision sensing and actuation. Computations are triggered by a combination of sporadic, periodic, and quasi-periodic events. Signals present reflect those in an automotive engine control unit for control of fuel injection, ignition timing, and valve retraction of an automotive engine. The system is naturally extensible to a distributed platform, presenting an interesting example for modeling distributed cyber-physical systems. For the purposes of demonstrating our design methodology, we do not consider a distributed implementation of this system.

A. TBD Step 1: State the Problem

Steel ball bearings are dropped one at a time at sporadic intervals towards a fixed drop target located below a spinning disc. The disc has been bored through at two opposite ends, and the ball will pass ("tunnel") through untouched if the disc is correctly aligned at the time of impact. Should the disc be improperly rotated, the ball will collide with the disc. The device must sense when a ball is dropped, track the position of the disc, and adjust the trajectory of the disc so that balls tunnel through the disc untouched. Only one ball will be above the disc at any time, and between drops the disc must maintain constant speed. The disc must not stop at any time, and changes in rate should be minimal.

B. TBD Step 2: Model Physical Processes

1) Kinematics of a Ball in Freefall: A ball is modeled as a tuple of its initial altitude, initial velocity, and time at which it is detected above the drop target, $\beta = (z_0, v_0, t_0) \in \mathrm{B}$, where $\mathrm{B} \subset \mathbb{R}^2_+ \times \mathbb{R}$ is the set of all possible ball drop events. Let $z : \mathrm{B} \times \mathbb{R} \longrightarrow \mathbb{R}$ be the altitude from the center of the ball to the center of the disc,

$$z(\beta, t) = z_0 - v_0(t - t_0) - \frac{1}{2}g(t - t_0)^2 \tag{1}$$

where g is constant acceleration due to gravity [12].

A ball with radius r_b may first contact the disc at arrival time $T_a(\beta)$, pass through the center of the disc at time $T_c(\beta)$, depart the disc at time $T_d(\beta)$, and is known to be above the disc for time $\Delta T(\beta)$, where $T_a, T_c, T_d : \mathrm{B} \longrightarrow \mathbb{R}$ follow from (1), and $\Delta T : \mathrm{B} \longrightarrow \mathbb{R}_+$ is defined $\Delta T(\beta) = T_a(\beta) - t_0$. The *drop interval* $[t_0, T_a(\beta)]$ is the duration for which a ball is known to be above the disc, and the *impact interval* $[T_a(\beta), T_d(\beta)]$ is the interval over which the ball may contact the disc. The *impact radius* $R_I : \mathrm{B} \times \mathbb{R} \longrightarrow \mathbb{R}_+$ is the widest horizontal slice of the ball that may be passing through the disc:

$$R_I(\beta, t) = \begin{cases} 0 & \text{if } |z(\beta,t)| > r_\mathrm{b} + \frac{h}{2} \\ r_\mathrm{b} & \text{if } |z(\beta,t)| < \frac{h}{2} \\ \sqrt{r_b^2 - \left(|z(\beta,t)| - \frac{h}{2}\right)^2} & \text{otherwise,} \end{cases} \tag{2}$$

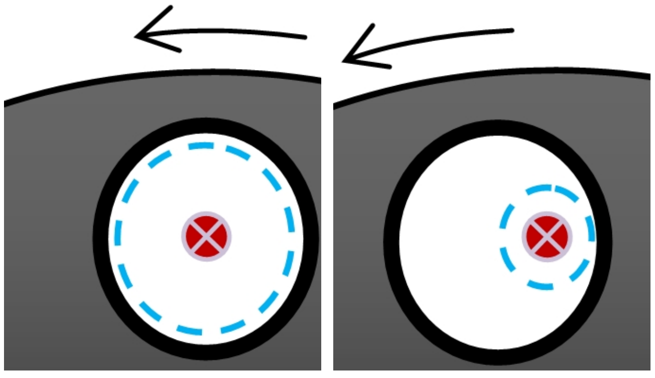
where h denotes the thickness of the disc.

2) Kinematics of a Rotating Disc: Let $\vartheta = [\mathbb{R} \longrightarrow (-\pi, \pi]]$ be the set of functions that describe the rotation of a disc over time. Note that all angle arithmetic is wrapped to $(-\pi, \pi]$. The disc has two doors bored at opposing ends, each with radius r_door and centered at a distance r_drop from the axis of rotation. A coordinate system is fixed so that the doors on the disc are centered above the drop target at rotation 0 and π. The Euclidean distance $d : \vartheta \times \mathbb{R} \longrightarrow [0, \sqrt{2}r_\mathrm{drop}]$ from the drop target to the center of the nearest of two doors is

$$d(\theta, t) = 2r_\mathrm{drop} \sin\left(\tfrac{1}{2}\min\{|\theta(t)|, |\pi - \theta(t)|\}\right). \tag{3}$$

As the disc rotates, the doors pass over the drop target exposing a tunnel through which a ball may pass. The *tunnel radius* $R_T : \vartheta \times \mathbb{R} \longrightarrow [0, r_\mathrm{door}]$ is the largest allowable impact radius at time t (Fig. 1):

$$R_T(\theta, t) = \begin{cases} r_\mathrm{door} - d(\theta, t) & \text{if } d(\theta,t) \le r_\mathrm{door} \\ 0 & \text{if } d(\theta,t) > r_\mathrm{door}. \end{cases} \tag{4}$$



(a) Disc rotated such that a door is centered over the drop target, yielding an optimal tunnel. (b) Disc rotated such that a door is offset from the drop target, yielding a sub-optimal tunnel.

Fig. 1. Disc rotations showing optimal and sub-optimal tunnels.

3) Dynamics of a DC Motor with Load: To find a mechanism to control the position of the disc, we recursively apply MBD to model and characterize a disc with an inertial load, derive a PID control algorithm, evaluate hardware such as DC and AC brushed and brushless motors, and simulate using the continuous model of computation in the LabVIEW Control, Design, and Simulation Module. We conclude that a DC brushed motor is a sufficient control mechanism and save our models for later code synthesis and subsystem testing.

A standard DC brushed motor with torque constant K_τ, armature resistance R, armature inductance L, damping coefficient b, back-electromotive force constant K_B, input voltage amplification K_A, and net inertia J is modeled by the system of linear differential equations [13]

$$\tau(t) = K_\tau i(t) \tag{5}$$

$$K_G \tau(t) = b\frac{d\theta(t)}{dt} + J\frac{d^2\theta(t)}{dt^2} \tag{6}$$

$$K_A v(t) = Ri(t) + L\frac{di(t)}{dt} + K_B\frac{d\theta(t)}{dt} \tag{7}$$

where $v : \mathbb{R} \longrightarrow \mathbb{R}$ is voltage applied to the armature coil, $i : \mathbb{R} \longrightarrow \mathbb{R}$ is the current induced by this voltage, $\tau : \mathbb{R} \longrightarrow \mathbb{R}$ is the torque produced by the motor, and $\theta \in \vartheta$ is the rotation of the disc. Moving to the frequency domain, the transfer function of the system is

$$\frac{\Theta(s)}{V(s)} = \frac{K_A K_G K_\tau}{JLs^3 + (RJ + Lb)s^2 + (Rb + K_G^2 K_B K_\tau)s}, \tag{8}$$

where s is the Laplace complex variable [13].

C. TBD Step 3: Characterize the Problem

The Tunneling Ball problem is characterized by six fundamental quantities: a worst-case drop (minimum drop time coinciding with maximum correction angle), the minimum torque that can accommodate a worst-case drop, the minimum voltage required to produce this torque, lower and upper bounds on disc rate, conditions for success, and trajectories that yield success. We translate these quantities into physical parameters used to select appropriate hardware for the device.

1) Worst-Case Drop: The initial velocity of a ball is bounded, so let maximum initial velocity be defined as

$$v_{\text{max}} = \max_{\beta \in \mathrm{B}} v_0 \tag{9}$$

and minimum drop time be defined as

$$t_{\text{min}} = \min_{\beta \in \mathrm{B}} \Delta T(\beta). \tag{10}$$

For a ball to pass through the disc, a tunnel must be present at the time of impact, likely requiring the position of the disc when the ball arrives be altered from its original trajectory. The center of a door is never more than one-quarter rotation away from the drop target, hence maximum position error

$$\theta_{\text{max}} = \frac{\pi}{2}. \tag{11}$$

A *worst-case drop* is tuple $(\beta_{\text{worst}}, \theta_{\text{worst}}) \in \mathrm{B} \times \vartheta$ such that $\Delta T(\beta_{\text{worst}}) = t_{\text{min}}$ and $\theta_{\text{worst}}(T_c(\beta_{\text{worst}})) = \theta_{\text{max}}$, which corresponds to the minimum amount of time to correct for the maximum position error.

2) Minimum Torque: The trajectory with the least maximum torque that adjusts for a worst-case drop is given by Maupertuis' principle of classical mechanics [14], and is the result of applying a constant torque τ_{min} over the drop interval. Given the motor and disc are at steady-state at time $t_0 = 0$ with constant angular velocity $\omega_0 = \frac{d\theta}{dt}(0)$, and solving motor equations (5)–(7) subject to $\Delta T(\beta) = t_{\text{min}}$, $\tau(t) = \tau_{\text{min}}$, $\theta(0) = \theta_{\text{max}}$, and $\theta(t_{\text{min}}) = 0$, for $t \geq t_0$,

$$\tau_{\text{min}} = \left(\frac{b}{K_G}\right) \frac{\theta_{\text{max}}}{t_{\text{min}} + \frac{J}{b}\left(1 - e^{-\frac{b}{J}t_{\text{min}}}\right)}. \tag{12}$$

3) Minimum Voltage: The minimum voltage v_{min} is the voltage applied to the motor necessary to produce steady-state torque τ_{min}. Substituting $\tau(t) = \tau_{\text{min}}$ into the motor equations,

$$K_A v_{\text{min}} = \left[\frac{R}{K_\tau} + \frac{K_B K_G}{b}\left(1 - e^{-\frac{b}{J}t_{\text{min}}}\right)\right]\tau_{\text{min}} + K_B \omega_0 e^{-\frac{b}{J}t_{\text{min}}}. \tag{13}$$

4) Rate Bounds: The minimum rate ω_{min} at which the disc must spin to accommodate a worst-case drop follows from minimum torque:

$$\omega_{\text{min}} = \omega_0 + \left(\frac{K_G}{b}\tau_{\text{min}} - \omega_0\right)\left(1 - e^{-\frac{b}{J}t_{\text{min}}}\right). \tag{14}$$

If the disc is rotating too fast, it may be impossible for a ball to tunnel through. An exact bound on disc rate follows from the kinematics equations but is somewhat cumbersome; a conservative bound ω_{max} guarantees the tunnel is always larger than the radius of the ball over the impact interval:

$$\omega_{\text{max}} = \frac{2\sin^{-1}\left(\frac{r_{\text{door}} - r_b}{2r_{\text{drop}}}\right)}{\max_{\beta \in \mathrm{B}} \Delta T(\beta)}. \tag{15}$$

5) Success: A ball successfully tunnels through the disc if its impact radius is smaller than the tunnel radius for all times within the impact interval. The predicate $\mathbf{S} : \mathrm{B} \times \vartheta \longrightarrow \{\mathrm{true}, \mathrm{false}\}$ is the conditional for success given a ball and a trajectory,

$$\mathbf{S}(\beta, \theta) \Leftrightarrow \big[t \in [T_a(\beta), T_d(\beta)] \Rightarrow R_I(\beta, t) \leq R_T(\theta, t)\big]. \tag{16}$$

6) Correction Trajectory: A disc trajectory $\theta \in \vartheta$ is a *correction trajectory for* β if and only if $\beta \in \mathrm{B}$ and $\mathbf{S}(\beta, \theta)$. At the time a ball drop is detected, the disc must follow a correction trajectory or the ball will impact the disc.

D. TBD Step 4: Derive a Control Algorithm

Position control of a DC motor with an inertial load is a simple and well-known problem. We opt for PID control, and tune its gains using LabVIEW (Fig. 2). To derive a trajectory planning algorithm, we assume the motor and disc are at steady-state at time t_0 with constant angular velocity ω_0 when a ball is dropped, that tracking error is negligible, that motor transients are negligible (which restricts correction trajectories to those with a fixed rate), and that the ball does not accelerate over the impact interval (hence the impact radius is symmetric). Under these assumptions, the optimal solution to the Tunneling Ball problem is to center the nearest door over the drop target as the center of the ball passes through. The *uncorrected final angle* $\theta_f : \mathrm{B} \times \vartheta \longrightarrow (-\pi, \pi]$

$$\theta_f(\beta, \theta) = \theta(t_0) + \omega_0 \Delta T(\beta) \tag{17}$$

is the projected rotation of the disc at the time of impact. The *correction angle* $\theta_c : \mathrm{B} \times \vartheta \longrightarrow [-\theta_{\max}, \theta_{\max}]$

$$\theta_c(\beta, \theta) = \begin{cases} -\theta_f(\beta, \theta) & \text{if } |\theta_f(\beta, \theta)| \leq \frac{\pi}{2} \\ \pi - \theta_f(\beta, \theta) & \text{if } |\theta_f(\beta, \theta)| > \frac{\pi}{2}. \end{cases} \tag{18}$$

is the angle by which the final trajectory must be altered to center a door over the drop target at impact time. The planning algorithm $\Psi : \mathrm{B} \times \vartheta \longrightarrow \vartheta$ that produces the optimal correction trajectory by applying a constant change in rate when a ball is detected is

$$(\Psi(\beta, \theta))\,(t) = \begin{cases} \theta(t) & \text{if } t < t_0 \\ \theta(t_0) + \left[\omega_0 + \frac{\theta_c(\beta,\theta)}{\Delta T(\beta)}\right] t & \text{if } t \geq t_0. \end{cases} \tag{19}$$

E. TBD Step 5: Select Models of Computation

PTIDES [15], which faithfully executes discrete-event (DE) semantics [16], extends DE by specifying timing constraints at sensor and actuator boundaries, which are used to produce real-time guarantees through static analysis of the model. PTIDES models are readily deployed to embedded targets while still benefiting from the powerful simulation features of DE. PTIDES is an inherently distributed model of computation, which allows the modeling of systems that are separated by network boundaries. Given the discrete, mixed-signal nature of inputs and the potential opportunities for distributed computation, we find PTIDES to be the best fit to solve the Tunneling Ball problem.

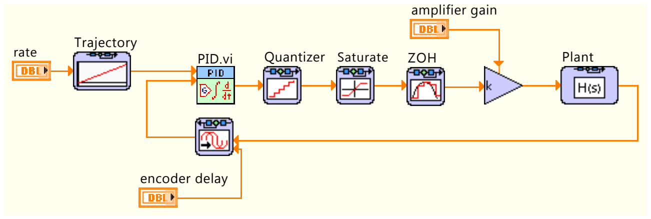


Fig. 2. LabVIEW model of the motor and PID controller.

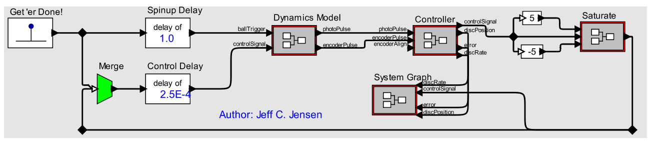


Fig. 3. Top-level view of the Tunneling Ball Device in Ptolemy II.

F. TBD Step 6: Specify Hardware

Characterizing the problem required a mechanism for rotating an inertial load, and a recursive application of MBD verified a DC brushed motor would suffice. The six fundamental quantities from the problem characterization are the basis for selecting a motor, encoder, H-bridge, and embedded microcontroller.

The drop sensor will be constructed from two sequential optical sensors at a fixed altitude above the drop target. As the ball passes through the first optical sensor, the time of the event is recorded and compared to the time the ball passes through the second optical sensor, which is the time t_0 when the ball is known to be a fixed altitude above the disc.

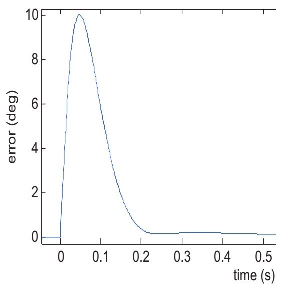
G. TBD Step 7: Simulate

Nonlinear operations such as quantization, saturation, and sampling are difficult to model mathematically; however, simulating their effects is straightforward in LabVIEW. Our LabVIEW model for position control of the motor incorporates the motor transfer function, sampling rate and quantization of a digital controller, latency and quantization of a PWM generator, and voltage saturation.

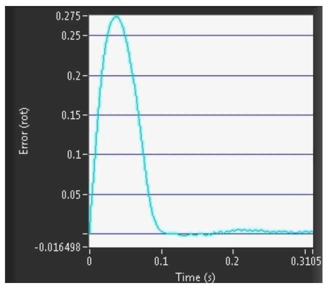
The entire end-to-end heterogeneous system is modeled in Ptolemy II, making use of its DE, PTIDES, and Continuous models of computation. The top-level view (Fig. 3) shows a drop generator, a controller, and physical dynamics actors connected in a feedback loop.

H. TBD Step 8: Construct

A housing to enclose the motor, disc, and dropped balls is constructed. As schematics for the device are drawn, design tradeoffs are easily considered by changing simulation parameters. During construction, individual components are tested against simulations before integrating into larger subsystems. Measurements of the constructed device are fed back into simulation for verification.



(a) Theoretical tracking error. The graph shows a settling time of 200ms and steady-state error of 0.1% of a rotation.



(b) Experimental tracking error. The graph shows a settling time of 100ms and steady-state error of 0.5% of a rotation.

Fig. 4. Theoretical predictions and experimental results for trajectory planning of the Tunneling Ball Device. The graphs show tracking error versus time. The system is at rest ($\omega_0 = 0$rps) at time $t = 0$s, and is given a new trajectory with $\omega_0 = 16$rps. The scales of the vertical axes are different.

I. TBD Step 9: Synthesize Software

The first iteration of the software is written by hand in C, drawing from many aspects of PTIDES. Kinematics lookup tables are generated from models of ball and disc kinematics. We leave formal software synthesis from PTIDES as future work.

J. TBD Step 10: Verify, and Validate, and Test

The success predicate (16) is a first step towards formal verification. Given a worst-case drop and our planning algorithm, we find a necessary and sufficient condition for which the device will *always* succeed,

$$\omega_0 + \frac{\theta_{\max}}{t_{\min}} < \omega_{\max}. \quad (20)$$

Such a condition can be used to validate software using formal temporal logic. We leave formal verification of the TBD as future work on the PTIDES model of computation.

Position control is independently verified (Fig. 4) before moving to a system-wide test. The first system-wide test of the Tunneling Ball Device is a success: with initial disc rate $\omega_0 = 8$rps, a randomly dropped ball passes through untouched. The test is replicable and succeeds for rates of up to $\omega_0 = 16$rps, nearing the theoretical bound for successful operation. We consider the late-night proving of the device to be an illustration of the power of model-based design.

IV. Summary

We introduced model-based design methodology for cyber-physical systems and evaluated it through the development of the TBD. It is our strong belief that the Tunneling Ball problem would have been extremely difficult (if not impossible) to solve without the use of modeling and simulation. MBD proved critical in nearly every aspect of development, and especially in device construction, hardware selection, and selection of the model of computation. The methodology invokes powerful modeling theory in a strongly pedagogical application of mathematics, engineering, and computer science. Each of the steps investigated in this report is only a preview of a vast field of research, but the formalization into codependent steps offers an innovative approach to the design of cyber-physical systems.

Acknowledgment

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET) and #0931843 (ActionWebs), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

References

[1] C. Brooks, C. Cheng, T. Feng, E.A. Lee, and R. von Hanxleden, “Model Engineering Using Multimodeling”, in 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 08), September 2008.

[2] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, “Developing Applications Using Model-Driven Design Environments,” *IEEE Computer*, vol. 39, no. 2, pp. 33, February 2006.

[3] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, “Model-Integrated Development of Embedded Software,” Proceedings of the IEEE, vol. 91, no. 1, January, 2003.

[4] J. Eker, J. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, “Taming Heterogeneity - The Ptolemy Approach,” *Proceeedings of the IEEE*, vol. 91, no. 1. January, 2003.

[5] N. Cutland, *Computability: An Introduction to Recursive Function Theory*. Cambridge, MA: Cambridge University Press, 1997, pp. 100-112, 149-156.

[6] E.A. Lee, “Computing needs time,” *ACM Communications*, vol. 52, no. 5, pp. 70-79, May 2009.

[7] J. Eidson, E.A. Lee, S. Matic, S.A. Seshia, and J. Zou, “Time-centric Models for Designing Embedded Cyber-Physical Systems,” University of California, Berkeley, Technical Memorandum. UCB/EECS-2009-135, October 2009.

[8] K. Keutzer, A.R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-Level Design: Orthogonalization of Concerns and Platform-Based Design,” *IEEE Transactions*, vol. 19, no. 12. December 2000.

[9] A. Goderis, C. Brooks, I. Altintas, E.A. Lee, and C. Goble, “Heterogeneous Composition of Models of Computation,” Future Generation Computer Systems 25(5): 552-560.

[10] J. Kodosky, J. MacCrisken, and G. Rymar, “Visual Programming using Structured Data Flow,” IEEE Workshop on Visual Languages, IEEE Computer Society Press, Kobe, Japan, pp. 3439. October, 1991.

[11] J.C. Jensen, “Elements of Model-Based Design,” University of California, Berkeley, Technical Memorandum. UCB/EECS-2010-19, February, 2010.

[12] H. Young, *Sears & Zemanskys University Physics: with Modern Physics*, 11th ed. San Francisco, CA: Pearson, 2004, pp. 58-60, 178-181, 334, 345-346.

[13] S. Shinners, *Modern Control System Theory and Design*, 1st ed. New York, NY: Wiley Interscience, 1992, pp. 143-159, 256-258.

[14] E. Corinaldesi, *Classical Mechanics for Physics Graduate Students*. Boston, MA: World Scientific Publishing, 1999, pp. 12-13.

[15] P. Derler, T. Feng, and E.A. Lee, “PTIDES: A Programming Model for Distributed Real-Time Embedded Systems,” University of California, Berkeley, EECS Technical Report. EECS-2008-72, May 2008.

[16] G. Fishman, *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer-Verlag, 2001.