Computer-Automated Multiparadigm Modeling in Control Systems Technology

Pieter J. Mosterman, *Member, IEEE*, Janos Sztipanovits, *Fellow, IEEE*, and Sebastian Engell, *Member, IEEE*

***Abstract*—The use of model-based technologies has made it imperative for the development of a feedback control system to deal with many different tasks such as: plant modeling in all its variety; model reduction to achieve a complexity or level of abstraction suitable for the design task at hand; synthesis of control laws that vary from discrete event reactive control to continuous model predictive control, their analyses, and testing; design of the implementation; modeling of the computational platform and its operating system; analysis of the implementation effects; software synthesis for different platforms to facilitate rapid prototyping, hardware-in-the-loop simulation, etc. Throughout these tasks, different formalisms are used that are very domain specific (e.g., tailored to electrical circuits, multibody systems, reactive control algorithms, communication protocols) and that often need to be coupled, integrated, and transformed (e.g., a block diagram control law in the continuous domain has to be discretized and then implemented in software). Significant improvements in many aspects (performance, cost, development time) of the development process can therefore be achieved by: 1) relating and integrating these different formalisms; 2) automatic derivation of different levels of modeling abstractions; and 3) rigorous and tailored design of the different formalisms by capturing the domain (or meta) knowledge. The emerging field of computer automated multiparadigm modeling (CAMPaM), presented in this paper in the context of control system design, aims to develop a domain-independent formal framework that leverages and unifies different activities in each of these three dimensions.**

***Index Terms*—Computer-aided control system design, control engineering, design automation, embedded software design, metamodeling, model integrated computing, multiparadigm modeling, systems engineering.**

I. Introduction

CONTROL system development is an inherently multidisciplinary activity. Typical stages of this process are: 1) modeling of the (existing or envisioned) plant; 2) model reduction to generate a model of appropriate complexity; 3) control algorithm design (e.g., inverse dynamics, pole placement); 4) test of the control algorithm using the original plant model; 5) implementation design; and 6) realization. Note the difference between design and development: A controller is initially designed and then re-designed as details of the realization/implementation are taken into account. This process of repeatedly modifying the design is the development process.

The complexity of systems and the need for using different engineering disciplines in the different design stages motivated the use of the *divide et impera*, or divide and rule principle. Application of this principle reduces complexity by means of: 1) *abstraction* to remove irrelevant detail; 2) *partitioning* into subsystems and components (chunks with a complexity that can be handled without further decomposition); and 3) *hierarchy* to organize the system [1].

In this paper the controller design is decomposed in three hierarchical structures: 1) functional hierarchy; 2) implementation hierarchy; and 3) the realization hierarchy. The functional hierarchy describes *what* the system is supposed to do and so captures system behavior. An implementation is a means of achieving an end and as such the implementation hierarchy describes *how* the intended functionality is achieved. The realization is the actual *physical* system.

Because these hierarchies are closely related and even entangled, an inherent characteristic of system development is the need for incremental refinement in different but related directions. In control systems, this requires repeated modifications of the original control algorithm as it is combined with additional implementation functionality induced by system integration and realization needs such as data validation, scaling, sampling rates, and numerical accuracy.

To streamline this iteration process and facilitate analysis and synthesis, sophisticated modeling approaches are used for each of the aspects.

- To model the functional aspects of the control algorithm and its supporting functionality high-level languages are used such as block diagrams [2], statecharts [3], communicating sequential processes (CSP) [4], Grafcet [5], synchronous languages [6], data-flow diagrams, control-flow diagrams, and discrete event systems specification (DEVS) [7].
- The controller implementation may be modeled using the unified modeling language (UML) [8] and the plant implementation with Modelica [9], VHSIC (very high speed integrated circuit) hardware description language (VHDL) [10], or bond graphs [11], [12].
- The realization may include physical models that can be, e.g., scale models, mockups, and prototypes.

Combining these aspects and addressing the inherent heterogeneity poses demanding integration requirements on modeling languages.

Manuscript received April 29, 2002; revised July 8, 2003. Manuscript received in final form August 26, 2003. Recommended by Editor M. Bodson. The work of J. Sztipanovits was supported by NSF ITR Grant CCR-0 225 610.

P. J. Mosterman is with the Real-Time and Simulation Technologies Group, The MathWorks, Inc., Natick, MA 01760 USA (e-mail: pieter_j_mosterman@mathworks.com).

J. Sztipanovits is with the Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37203 USA (e-mail: sztipaj@vuse.vanderbilt.edu).

S. Engell is with the Process Control Laboratory, University of Dortmund, Dortmund D-44221, Germany (e-mail: s.engell@bci.uni-dortmund.de).

Digital Object Identifier 10.1109/TCST.2004.824280

The prolific use of embedded code has another important implication for control system design; today, systems exist with many interacting embedded controllers at different levels in the system hierarchy. As a consequence, verification and validation[1] (V&V) of the control system has to account for a number of complicating aspects: 1) within a level in the functional hierarchy, the parts of the controller are analyzed not only individually but also in combination; 2) between levels in the functional hierarchy, the V&V process mixes subsystems at different levels; and 3) across phases in the development cycle, the effects of selected implementation technologies and design approaches are taken into account.

This paper aims to introduce these "systems concerns" in controller design. It presents the phases of the design process using a power window controller as an example and shows that much of the effort in controller design must be devoted to the functionally correct realization of the control algorithms. Given the multitude of formalisms used in the functional design in describing the implementation of the control algorithms and in representing the heterogeneous platforms realizing the controller, a comprehensive method is required to integrate these representations.

Specifically, customized (maximally constrained) domain-specific formalisms and their supporting development environments should be easy to design and adapt. Models in different formalisms that capture different levels of detail and that view different aspects of the system, need to be managed. Translation between them is necessary to allow: comprehensive analyses, presenting results in different domain-specific views, synthesis of heterogeneous models, and automated changes in model abstraction. And, finally, vendor supplied models and different "best-of-class" tools have to be coupled.

Computer automated multiparadigm modeling (CAMPaM) tries to achieve this feat by leveraging and combining theory and methodology in three orthogonal dimensions: 1) support for multiple formalisms; 2) support for different levels of abstraction; and 3) support for generating tailored formalisms from a model of the modeling formalisms, i.e., a meta-model. Though the actual design process is domain specific, it is possible to construct such a domain independent formal framework. For the particular task of control system development, this framework then includes theory, tools, and methods for addressing the issues of optimal controller design across and between levels of functionality, implementation, and realization.

This paper presents an overview of some of the issues involved in the design and development of complex systems and how CAMPaM can be applied. Section II presents an example system, an electronic window in automobiles, to relate the concepts and notions. Section III generalizes the concepts exemplified by the power window system. This introduces the importance of modeling and Section IV gives an overview of its uses. Section V then distills the requirements for a comprehensive model-based approach to system design and presents the field of CAMPaM and how it addresses the issues put forward in the previous section. Finally, in Section VI conclusions of this overview are drawn.

[1]In this paper, verify is interpreted as checking domain laws and validate as checking correctness of execution against the requirements. Verification then checks whether a model was built correctly, while validation checks whether the correct model was built.

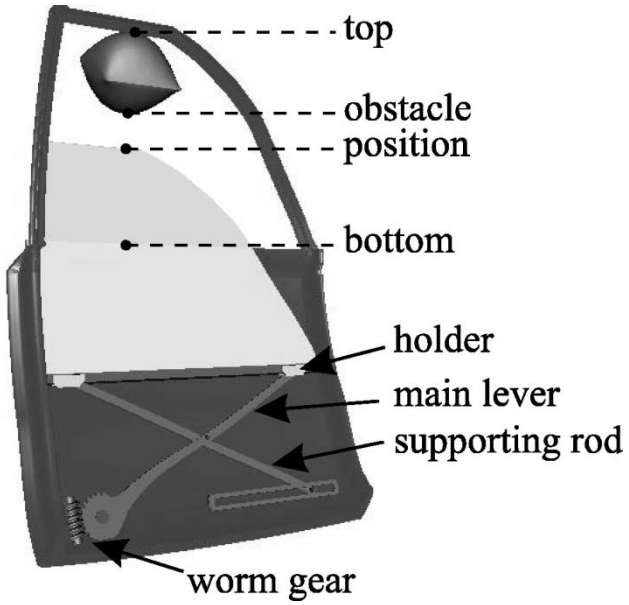

Fig. 1. Power window mechanics.

II. Power Window Control System

Controller design is inherently a "systems problem." This means that the design process includes issues that arise from partitioning large-scale systems and realizing them with physical components having a variety of nonfunctional attributes such as power consumption, fault behavior, and dimensions.

A. Partitioning to Manage Complexity

Consider a power window control system as used in automobiles to open and close the windows. The windows are moved by motors that are controlled by up and down commands generated by the driver and the passengers pressing switches. These switches are typically located elsewhere in the vehicle and connect to the window control module via a controller area network (CAN) [13]-based communication bus. To efficiently design the window control mechanism, it can be partitioned into four subsystems: 1) the embedded controller; 2) signal conditioning hardware; 3) the electrical actuator; and 4) the lift mechanism.

The electrical actuator is typically a motor of the dc type, and is not decomposed any further in the controller design process, i.e., it is considered a primitive *component*. Similarly, the lift mechanism, shown in Fig. 1, at this level is typically considered as a primitive component by the control design engineer. An essential part of the system requirements is the ability to accommodate an obstacle between the window and its frame.

In contrast to the dc motor and lift mechanism, the embedded controller subsystem is a compound object. It consists of a hardware part and software part that are separate but closely related.

The power window control system is part of a larger CAN bus connected automobile electronics system that also controls, e.g., the opening and closing of a sun roof and moving the headlights up and down. So, it is a subsystem itself and interacts with the window control switches. These, again, are subsystems that can be decomposed into the switch hardware, signal conditioning hardware in the form of a voltage divider to transform the battery

TABLE I
POWER WINDOW REQUIREMENTS

ID	Description
1	The window has to be fully opened and fully closed within 4 [s].
2	If the up or down command is issued for at least 200 [ms] and at most 1 [s], the window has to be automatically opened or closed completely, respectively.
3	The force to detect when an obstacle is present should be less than 100 [N].
4	When an obstacle is present, the window should be lowered by approximately 10 [cm].

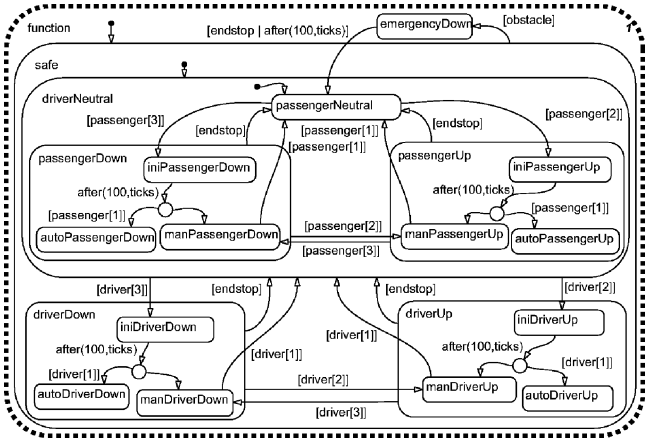

Fig. 2. Discrete event model of window control.

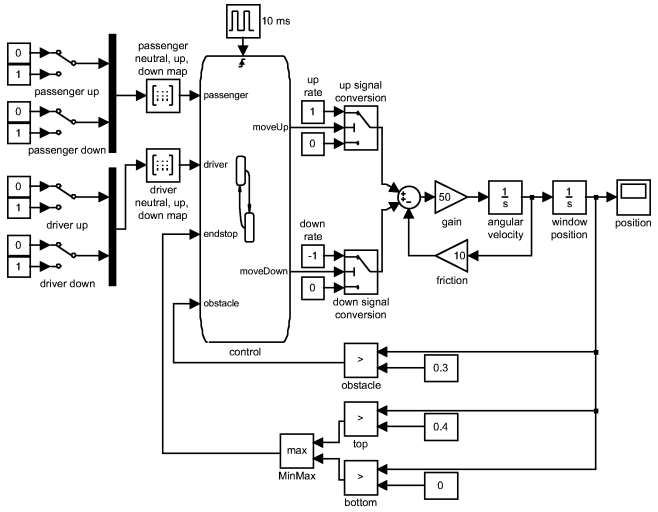

Fig. 3. Hybrid model of window control.

voltage to within the 0–5 [V] input range of the analog to digital (AD) converter and the micro-controller that communicates the commands to the CAN bus.

B. Development Cycle

It is first stepped through the development of the power window system and the effects of the different stages on the functional aspect, i.e., the design of the behavior of the controller, are pointed out.

1) Initial Specification: The requirements that are the basis for the design are typically written down in natural language, possibly annotated with graphics. These requirements often are not very rigorous, and, consequently, contain inconsistencies and ambiguities. For the power window control system, the requirements pertinent to the discussion that follows are stated in Table I.

2) Discrete Event Behavior: An initial design may focus on the discrete event control aspects of the requirements. The statechart [3] formalism is intuitive and efficient for capturing the reactive behavior. Fig. 2 shows a statechart in Stateflow® [14] that captures the transitions between *neutral, up*, and *down*; the precedence of driver commands over the passenger commands; the automatic and manual up and down modes; and the priority of an emergency state over the safe state. This is achieved by a hierarchical structuring of state transition diagrams. At the top level, *function*, there is an *emergencyDown* state and a *safe* state. The *safe* state is decomposed into a *driverNeutral* state, a *driverUp* state, and a *driverDown* state. The latter two are decomposed in three other states each, that implement the auto-up and auto-down behavior. A similar structure is found for the *passengerUp* and *passengerDown* states that are part of the *driverNeutral* state, i.e., when the driver does not issue a command, the passenger may assume control. The driver and passenger commands are coded as arrays with entry 1 representing the *neutral* command (i.e., no control signal is issued), entry 2 the *window up* command, and entry 3 the *window down* command. The *endstop* and *obstacle* events correspond to the window reaching the top or bottom of the frame and the window reaching an obstacle, respectively.

When an event occurs that deactivates a state that is decomposed into child states, this state is departed irrespective of the active child state. When a state with child states is entered without explicitly modeling which of the child states is activated, the one with the *initial transition* (depicted by a solid circle connected to an arrow) is activated. The mathematical model of these hierarchical state transition diagrams is the finite state machine formalism (see, e.g., [15]) extended by hierarchy and parallellism.

3) Temporal Behavior: Once the discrete event control has been designed and validated against the respective requirements, the temporal requirements have to be analyzed. The controller is realized by means of a microprocessor that runs at a 10 [ms] sampling rate. Since the discrete event controller is connected to a plant (the lift mechanism), which is modeled as continuous dynamics, the controller and the plant constitute a *hybrid dynamic system* [16]–[19]. The Simulink® [2] model of the hybrid dynamic system is shown in Fig. 3. This is a block diagram typically used in control engineering which on a semantic level executes as a set of differential equations. The $(1/s)$ block indicates continuous integration, showing that the plant is modeled as a second order system of differential equations. The triangular shaped blocks represent multiplications with a constant (50 for the *gain* block and 10 for the *friction* block). Other blocks are constant sources (square with a numeric value), inequality relations (square with the $>$ operator), a block to select the maximum of two inputs

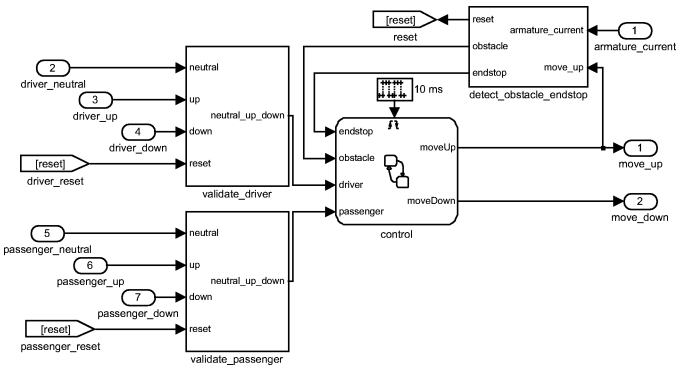

Fig. 4. System integration requires interface functionality.

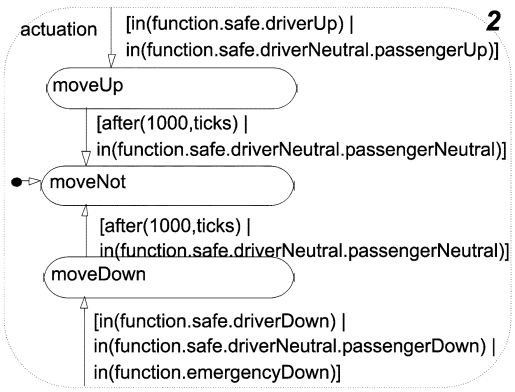

Fig. 5. Additional actuation functionality.

(square with **max**), a switch block to select one of two inputs (the *up signal conversion* and *down signal conversion* blocks), a table to map two inputs onto an array with three elements (the *driver neutral, up, down map*, and the *passenger neutral, up, down map* blocks), a multiplexing block that takes two inputs and moves them into a two dimensional signal (thick solid vertical bar), and, finally, the statechart control as shown in Fig. 2 (the block named *control*). The controller is triggered by a pulse with 10 [ms] period. In this setup, the *endstop* and *obstacle* events are generated from the plant model based on the frame top (40 [cm]) and bottom (0 [cm]) and the position of the obstacle (30 [cm]).

The safety requirement is that the window has to be rolled down 10 [cm] when an obstacle is detected. Given that the controller runs at a 10 [ms] sampling rate, simulation shows that the window is rolled down 10 [cm] by a timed transition from the *emergencyDown* state that is activated after 1 [s], or 100 clock ticks of 10 [ms], thus satisfying the requirement.

4) Effects of System Integration on the Functional Design: When integrating the discrete event controller with the rest of the power window control system, additional considerations are required to ensure that the actual driver and passenger commands are consistent with the assumptions on which the state model was based. For example, if for whatever reason the CAN bus communicates both *up* and *down* commands from, e.g., the driver, only one may be executed. This may cause an anomalous situation to arise that needs to be prevented. So, additional functionality is required to make sure that only one of the three possible commands (*up, down, neutral*) is issued by the driver and the passenger. This adds control logic to the controller that will default to the down command in such a situation.

Fig. 4 shows how the original discrete event controller is embedded in the extended functionality. The *detect_obstacle_endstop* data analysis process infers the presence of an obstacle or whether the top or bottom of the window frame is reached. In this design stage, this may be based on a position thresholding scheme, i.e., if the window position reaches 40 [cm], it is at the top of the frame, the bottom is at 0 [cm] and the object can be positioned anywhere in between.

In addition, the original control is extended by a parallel statechart with the actuation control functionality, as shown in Fig. 5. For example, when the state transition diagram in Fig. 2 is in either the *driverUp* or *passengerUp* state, the state transition diagram in Fig. 5 moves into the *moveUp* state. This state transition

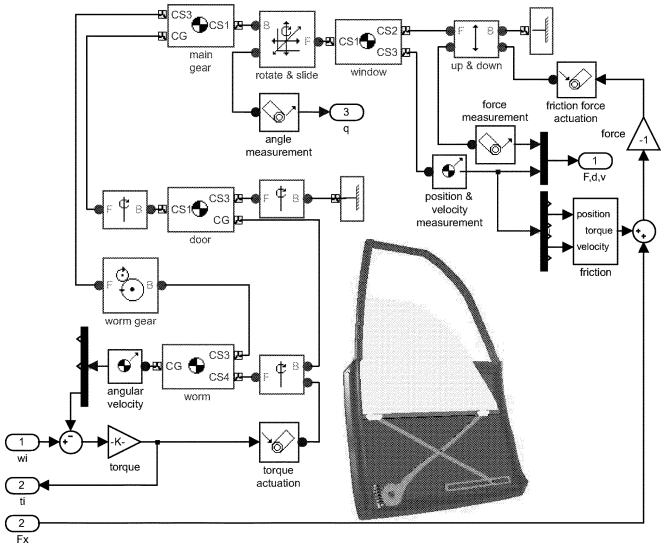

Fig. 6. Power effects in the mechanical domain.

diagram has two purposes: First, it translates the system state into an output command (*moveUp* and *moveDown*) and second it ensures that after a given time-out (here 1000 ticks, or 10 [s]) the system resets to its neutral state, e.g., when the detection of the frame top or bottom fails and the dc motor is not turned off appropriately.

5) Refinements of the Implementation: After a first integration pass, a more detailed plant model can be designed by the plant design engineers and coupled to the control subsystem. High level modeling languages such as bond graphs [11], Modelica [9], The MathWorks blocksets [20], [21], VHDL-AMS [22], and Adams [23] allow intuitive and quick modeling of the electrical circuitry and the window mechanism. Fig. 6 shows a multibody model of the window mechanism using the SimMechanics. The difference between the angular velocity, wi, as generated by the dc motor and the angular velocity of the worm gear generates a torque modeled by the torsional damper constant, K. This torque rotates the worm of the *worm gear* that causes the *main gear* that models the main rod of the lift mechanism to rotate. The two parts of the worm gear are kept in place by the *door* body that may rotate around the vertical axis modeled by a revolute joint. The *main gear* connects to the *window* body through a sliding and rotating joint, *rotate & slide*. The *window* itself may move up and down with respect to ground by

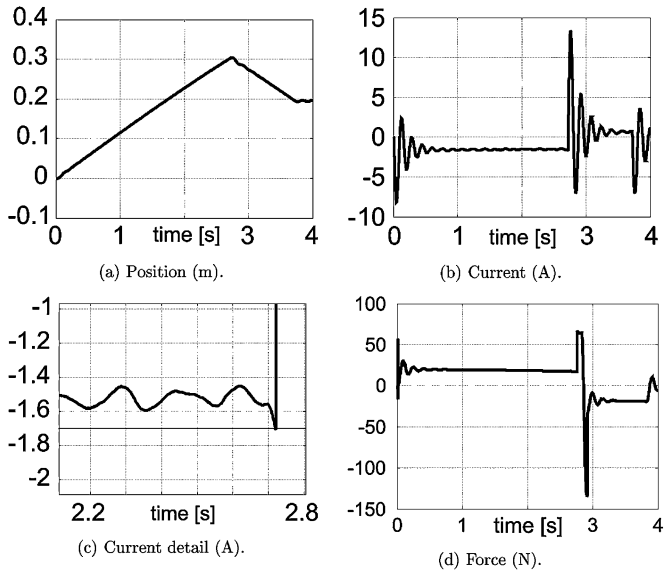

Fig. 7. Power window control system simulation.

means of a prismatic joint. Kinetic and dynamic variables are obtained by "measurement" and "actuation" blocks. The model includes viscous friction as a function of the window position. This friction is very much vehicle dependent and changes over time. Note that this model, though it applies a formalism different from the block diagrams shown earlier, also executes as a system of differential equations.

The mapping onto an implementation architecture reveals that the obstacle and endstop detection logic of the controller subsystem has to be modified to use the armature current as drawn by the dc motor that moves the window, instead of the window position (simply because the latter is not a measured quantity in the actual system). This affects the control algorithm profoundly, since the control logic that detects an obstacle and the endstops cannot be position based.

Simulation of the combined discrete event control and the continuous plant behavior is the basis for devising a control algorithm that obeys the 100 [N] force limit. Here, a straightforward thresholding scheme is applied, where the current thresholds are synthesized from simulation.[2]

Fig. 7(a) shows the window position over a 4 [s] timespan when the driver initially commands it to move up. At 30 [cm] an obstacle is detected because the armature current, shown in Fig. 7(b), exceeds the 1.7 [A] threshold. Fig. 7(c) shows how little margin is available in terms of current deviation for differentiating between the normal operation and 100 [N] force limit [shown in Fig. 7(d)]. Upon detection of the obstacle, the window is automatically lowered by 10 [cm]. The negative force to reverse window movement does exceed 100 [N] in magnitude but since this force is negative, it does not violate the force requirement.

The necessity to use the armature current measurement complicates the control algorithm design. Simulation studies

[2]In the spirit of this paper, the complexity does not originate from the control algorithm so much, rather it is brought about by the increasingly detailed functionality of the plant as well as the controller implementation. It should be clear that the model-based approach does lend itself perfectly to handle these effects for much more complex control algorithms as well.

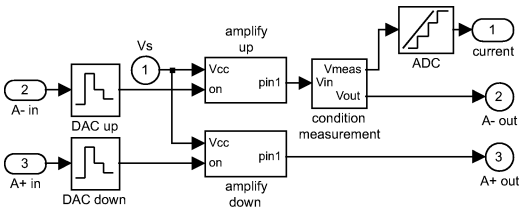

Fig. 8. Data acquisition effects in the power domain.

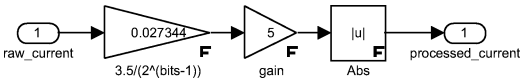

Fig. 9. Fixed-point data acquisition computations.

showed that the magnitude of the current transients when starting to move the window (or reversing its direction) tend to be of the order of 10 [A]. This is much higher than the maximum current allowed to prevent a force of more than 100 [N] on an obstacle, which is of the magnitude of 1.7 [A]. An extensive and sophisticated control algorithm that robustly satisfies the requirements can be designed given the available models (e.g., a learning feedforward neural net approach may be applied [24]). However, it would be beyond the scope of this paper to discuss this in detail. Here it suffices to implement a statechart-based control module to delay the 1.7 [A] threshold detection till the current transients have died out. Simulation shows this to be the case 1 [s] after the *moveUp* command has been issued which corresponds to 100 clock ticks in this statechart control module. The shortcoming of this implementation is that an obstacle cannot be detected within one second after initiating the up command. Note that the *moveUp* command has to be included as an additional input to the detection module, while a more sophisticated control algorithm would affect the core algorithm more profoundly.

6) Data Acquisition Functionality: Further implementation aspects are included by modeling the effects of AD conversion (time discretization, conversion delay, amplitude quantization) and signal conditioning. For example, to measure the armature current, a small resistance is included in the voltage source line across which the voltage is measured. In general, to increase the measurement range, a shunt resistor may have to be included. Based on the resistance values, the current can be computed. The controller design repeatedly goes through such phases of extensive testing and validation against the requirements, which may lead to additional refinements and modifications.

In addition to making the plant model more realistic, the controller model is brought closer to an implementation in much the same way. To improve the computation performance of the implementation, typically fixed-point computations are applied. This means that for the power window, several floating-point operations need to be transformed to fixed-point. Furthermore, the AD signal has to be scaled to correctly compute the armature current and its absolute value is used in the control (see Fig. 9).

The implementation decisions again affect the control algorithm performance. In this particular case, when transforming the floating-point computations to fixed-point, it was determined that 8 bits would not provide sufficient resolution given the range of the armature current (approximately ± 15 [A]) to

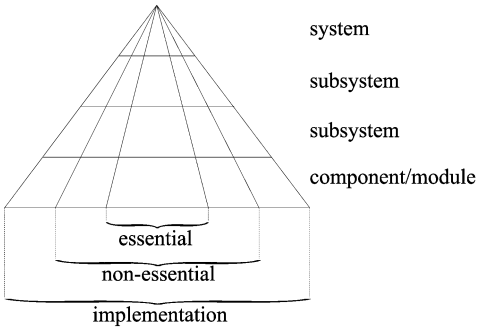

Fig. 10. Control design across levels in the design hierarchy.

be able to implement a control sufficiently sensitive to obey the 100 [N] force constraint. Instead, a 16 bit implementation had to be chosen.

7) Summary: Once all implementation effects that may affect the satisfaction of the requirements are accounted for, the controller code can be automatically generated from the controller model. This results in a faithful representation of the modeled behavior. Sophisticated compiler technology generates controller code more efficiently than its manually coded counterpart. This code itself can even be more efficient if linear time (or at least quadratic) synthesis algorithms exist because their systematic optimization of the code scales well.

In general, a control system contains many such control algorithms that are functionally connected. As discussed, there is control for data validation, actuation time out, and delayed obstacle detection. In addition, the power window system may use pulse width modulation (PWM) to control the angular velocity of the dc motor, the setpoint of which is then generated by the discrete event control of the power window module, but also there is a control module that takes the switch voltages and translates these into *up, down*, and *neutral* commands. Furthermore, other distributed controllers that use the CAN communication bus may affect the performance of the window control system. To coherently analyze such systems and rule out inconsistencies in the combined behavior requires not only combining the different models and formalisms used, but also abstracting to a level of detail that is amenable to analysis. CAMPaM aims at providing tools and methods to support such a coherent control system analysis.

III. Control System Development Process

The development process as sketched for the power window control exemplifies the general process of control system development, typically structured along two dimensions: 1) the system functionality is considered at increasing levels of detail and 2) at each level in the hierarchy, the system description is partitioned into subsystems [1].

Graphically, this can be depicted by a triangle where the system requirements are given at the top and these are refined through several levels of specification (see Fig. 10). It is important to note that the triangular representation includes actual extensions of functionality by nonessential and implementation parts. So, the design process is not simply a refinement hierarchy in the sense that it merely refines and presents more detail

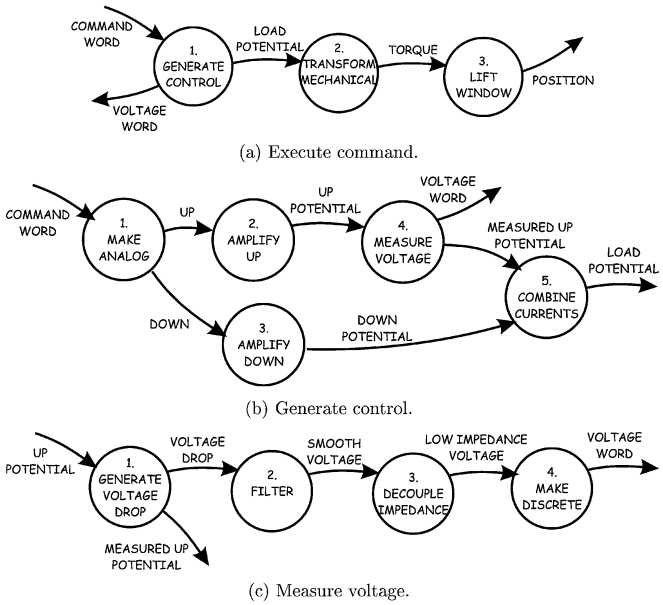

Fig. 11. Levels in the functional hierarchy.

of the higher level processes. In terms of the power window control system, the data validation functionality, data acquisition functionality (e.g., filtering and scaling), but also the fixed-point functionality (quantization and overflow) are added as nonessential and implementation artifacts, while the functional behavior is detailed by specifying how exactly the obstacle detection takes place (in this case by current thresholding).

In contrast to the traditional waterfall approach, system development does not progress monotonically toward increasing levels of detail, e.g., [25]. Instead, an inherent characteristic of the structure of the development process as shown in Fig. 10 is the need for iterations as more functional detail becomes known. These iterations make the overall design cost difficult to contain. Especially iterations that span several design levels can become expensive. For the power window, the initial endstop detection design based on the availability of the window position had to be revised because of cost saving reasons that only allow the dc motor armature current to be measured.

These design hierarchies exist both for the system functionality and implementation. In Figs. 11 and 12, parts of the functionality and implementation hierarchies of the power window system are illustrated, respectively. In Fig. 11 a structured analysis (SA) method [26], [27] is used to functionally capture the operation of the plant with sensors and actuators included. At the first level in Fig. 11(a), it is captured how a command from the controller has to be transformed into a control signal for the actuator that bridges the electrical and mechanical domain to lift the window. One level down, in Fig. 11(b) the functionality to generate the control of the actuator is captured as a process that makes the controller command analog, amplifies it, and then delivers it to the actuator, while monitoring the requested current by means of a voltage drop. The functionality to generate a voltage that relates to the current drawn is shown in Fig. 11(c). It first smoothes the signal and then makes it low impedance so it can be connected to the analog to digital conversion.

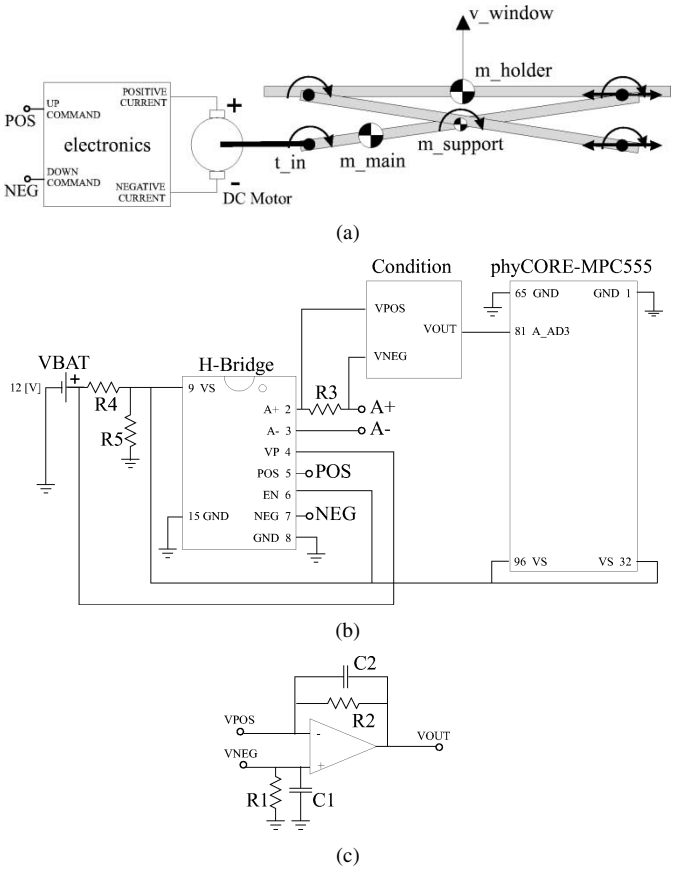

Fig. 12. Levels in the implementation hierarchy. (a) Plant. (b) Electronics. (c) Condition.

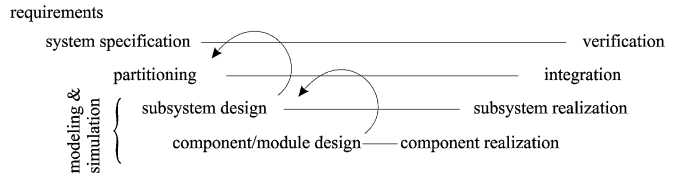

Fig. 13. The V process of system development.

In Fig. 12 part of the implementation hierarchy is shown. Fig. 12(a) shows electronics is used to drive a dc motor that powers the lift mechanism to move the window up and down. The electronics part of this is detailed in Fig. 12(b) which shows that the up and down commands are amplified by an H-bridge, the output of which is conditioned and sent to an AD converter. The signal conditioning is shown at the next level, Fig. 12(c). It smoothes the signal with a low-pass filter and decouples it by means of an operational amplifier so it is of low impedance and can be used as input to an AD converter.

Note that the AD conversion that is functionally part of the measurement process in the implementation resides on a phyCORE printed circuit board [28] that is populated with the MPC555 controller. Also note that the functionality in Fig. 11(b) is represented by a Simulink model in Fig. 8 that can be executed to quickly test the specifications.

The progress from system design to realization can be portrayed as a V-type activity [29] (see Fig. 13). Going down along the left branch, the system is specified at increasing levels of detail, both functionally as well as in terms of its implementation. Moving up along the other branch, the realization is created as it starts at the component level and proceeds to integrate increasingly comprehensive subsystems.

The system integration may again reveal discrepancies between (subsystem) requirements and realization, which may result in even more costly iterations between the branches of the V. For the power window control design, such an iteration would result from the fixed-point implementation if hardware testing was chosen over a model-based approach since an eight-bit processor would have to be programmed and connected to find out that it does not provide sufficient resolution to adhere to the 100 [N] maximum force. To avoid these iterations as much as possible, sufficiently detailed implementation models are desired that support rapid prototyping, hardware-in-the-loop simulation, software-in-the-loop simulation, and code deployment to allow early validation of subsystem design.

IV. Modeling as a Key Element of Control System Technology

The richness of the control tasks in large systems implies the need for richness and expressiveness in modeling technology. In fact, modeling is an inseparable part of control technology. This section reviews the requirements for modeling technology introduced by the systems approach to controller design.

A. Modeling as Part of Control Technology

Because an increasing number of realization effects can be captured in implementation models nowadays, the required iterations between the functional, implementation and realization hierarchies are significantly reduced.

The use of high-level modeling languages simplifies (in simpler domains even removes) the hard task for control engineers to produce dynamic models in terms of differential equations or state diagrams. Software engineers, however, use different formalisms and notions than control design engineers. These differences have to be reconciled at a systems level which allows a more systematic approach to modeling that is less error prone and applicable without the need to "manually" translate domain-specific representations to more abstract mathematical notations of difference or differential equations or finite automata (e.g., a schematic of an electronics circuit such as in SPICE [30] or a multibody topology in SimMechanics [20]).

Over time, many different formalisms have been developed that address problems in specific fields. To support controller design, it is crucial to develop modeling languages for both the controller and the plant that are closely related to the application domain. This allows the design to use precise models and in return yields high-quality control. In addition, if possible, the formalism should be designed such that it facilitates computer aided analysis and synthesis. For example, Petri net models allow for static deadlock analysis [31], [32] and statechart models can be coupled with verification tools.

Models of the same subsystem or component may be completely different, depending on the aspect of interest. For example, fault detection and isolation may only be interested in approximate propagation times of failures [33], whereas hardware-in-the-loop simulation would require real-time

numerical data generation (e.g., [34]). Yet links exist between the different models, and control design at a system level needs to account for these different views and levels of abstraction. The varying levels of detail and different perspectives that are modeled with these languages need to be related as well and the consistency among the modeling aspects needs to be automatically maintained.

B. Model-Based Design Technology

Model-based design technologies in controller design use models as the primary interface for moving design toward realization. While modeling itself is evidently the most important area in model-based design, three important model-based technologies are: 1) code generation; 2) simulation; and 3) test-vector generation.

Model-based code generation allows the high-level modeling of control laws and computation platforms and transforms them into embedded code via a form of model transformation. It is an enabling technology for safe, dynamic system reconfiguration as changes to the embedded code are made at the model level by a reconfiguration controller and not at the software level, which is much more error prone. Because of their more intuitive and comprehensive nature, high-level modeling languages allow the human designer to work with much more complex problems that what was previously possible. In addition, if control code can be automatically generated, rapid control prototyping [35] is facilitated to quickly study the behavior of the core control algorithm when embedded in a realistic environment [36].

Numerical simulation is still the only analysis technology capable of handling the complexity that arises from modern modeling approaches, mainly because verification techniques do not yet scale up to this level of detail (particularly with mixed continuous and discrete time behaviors [17]–[19]). It is an important enabler for controller design and can be the basis for using sophisticated search techniques to obtain robustness against parameter variations in multiple objective parameter synthesis (MOPS) [37] where parameter is to be interpreted in a broad sense, e.g., the control algorithm structure.

Even though certified compilers may cover a large part of the system validation effort and move it to earlier stages, system testing will always be required, especially for safety critical systems. Using model-based approaches, test vector generation can be automated to a large extent [14].

C. Additional Challenges

Considering the system as a whole, or at least comprehensive parts of it, in controller design results in additional challenges and opportunities. Two areas are discussed here: 1) reconfigurable control and 2) interlevel optimization.

1) Reconfigurable Controllers: The use of embedded controllers facilitates radical changes of the control algorithms on the fly. New, flexible hardware architectures even allow hardware reconfiguration during operation [38]–[40]. These new opportunites are all based on the extended use of modeling in the design process. Not only the plant and controller dynamics are modeled, but their modalities, the related configurations, the utilized resources, the architecture of the resources, and all of their interdependencies.

2) Interlevel Optimization in Control Design: When considering control at a systems level, it becomes critical to analyze and verify the correctness of the low-level control loops scattered throughout the system in combination with and across component and subsystem boundaries. A system-level control design perspective takes these level crossing effects into account and avoids overdesign by considering the coupled systems in combination with one another.

Note that, though modularization inherently leads to some degree of overdesign and performance degradation, partitioning into subsystems and components with well-defined interfaces is an indispensible means in system design to structure the process and support modification and exchange of subsystems, especially in the initial design stages.

Another interesting example for level crossing design is synthesis of controllers under constraints introduced by the effects of realization resources [41].

V. Computer Automated Multiparadigm Modeling

The general discussion in Section IV indicates the need for a model-based framework to manage the complexity of the control system development. A set of requirements for this framework is distilled first, after which it is discussed how CAMPaM addresses these.

A. Requirements

The extended interpretation and use of modeling leads to the following requirements in modeling technology.

1) Use of different, domain-specific modeling languages (computer aided design data, dynamics models, implementation technology model) needs to be coupled and related (including legacy models).
2) Translation between modeling formalisms is necessary to perform comprehensive analyses, to present results in different domain-specific views, and for synthesis of heterogeneous models.
3) Models provided by (at least) first tier vendors have to be connected to each other and to the in-house designed models, at least to the point where they function in a cosimulation setup.
4) Automated changes in model abstraction must be supported. This refers to model reduction (e.g., [42]) as much as augmentation (e.g., [43]) and also includes controller order reduction.
5) Customized domain-specific formalisms and their supporting development environments should be designed and adapted with little effort to support the evolutionary process of formalism changes.
6) Domain-specific constraints (e.g., the fan-out of digital logic) need to be included as inherent to the formalism.
7) Tools that are best-of-class with respect to specific analyses in the control design process have to be integrated so that they can operate on the same models.
8) Models that capture different levels of detail and view different aspects of the same subsystem need to be managed.

B. Three Dimensions of CAMPaM

The emerging field of CAMPaM aims to develop a formal framework that leverages and unifies different activities in each of the following three dimensions: 1) support for multiple formalisms; 2) support for multiple levels of abstraction; and 3) meta modeling [44]. In this sense, a modeling paradigm consists of a set of formalisms, several levels of abstraction addressed by the different formalisms, domain constraints, and formal interrelationships between the formalisms, abstractions, and constraints.

1) Support for Multiple Formalisms: A domain-specific modeling language (L) is a five-tuple of concrete syntax (C), abstract syntax (A), semantic domain (S) and semantic and syntactic mappings (MS and MC, respectively) [45]: $L = \langle C, A, S, \mathrm{MS}, \mathrm{MC} \rangle$. The concrete syntax C defines the specific (textual or graphical) notation used to express models, which may be graphical, textual, or mixed. The abstract syntax A defines the concepts, relationships, and integrity constraints available in the language. Thus, the abstract syntax determines all the (syntactically) correct "sentences" (in this case: models) that can be built.[3] The semantic domain S is usually defined by means of some mathematical formalism in terms of which the meaning of the models is explained. The $\mathrm{MC}\colon A \rightarrow C$ mapping assigns syntactic constructs (graphical, textual, or both) to the elements of the abstract syntax. The $\mathrm{MS}\colon A \rightarrow S$ semantic mapping relates syntactic concepts to those of the semantic domain.

A domain-specific formalism is effective, because it describes the domain models most precisely and most intuitively. Requirements 1 and 2 are concerned with the combining and relating of formalisms, which can be implemented at three distinct levels.

- The most basic level is the numerical level. Here, models in different formalisms are combined by producing data for each of them in a unified representation, i.e., an execution trace.
- One level up, data structures may be exchanged between formalisms. This requires each of the formalisms to produce and consume data in a commonly understood format. This approach is taken, e.g., by Ptolemy [46], where data tokens are exchanged that can have a complex structure.
- At the highest level, models in one formalism can be translated into a representation using another formalism. For example, causally augmented bond graphs can be transformed into ordinary differential equations [11] and translations between different DEVS dialects [47] may be performed under closedness conditions.

As translations between the different formalisms become available, a formalism transformation graph can be designed with a partial ordering much like a type lattice in software [48]. This graph allows decisions about which translation path to take to combine different formalisms. Domains where this activity takes place are, e.g., in cosimulation, modeling and simulation tools (to arrive at the common differential equations executable format), and multi-agent applications. This addresses Requirement 3.

[3]It is important to note that the abstract syntax includes semantic elements as well. The integrity constraints, which define well-formedness rules for the models, are frequently called "static semantics."

2) Different Levels of Abstraction: Orthogonal to the multiformalism dimension is the level of detail of the model. A model is designed to solve a particular problem and as such, there is not one unique model. Different levels of control design in the system hierarchy require models with a different degree of complexity (e.g., [42]). Requirement 4 is addressed by formal methods for abstraction and refinement, e.g., using quotient spaces to produce bisimulations of control systems at different levels of detail [49]. Especially in discrete systems, abstraction, and refinement are key factors for success [50]. Further approaches, e.g., power-based [51], are investigated.

Along with model reduction, model augmentation is important as well [52]. There has been work done in automatically finding the desired level of detail of (bond graph) models that are composed into a larger system of which behavior with a prescribed bandwidth is to be investigated [43].

3) Meta-Modeling: The quick generation and evolution of modeling environments (Requirement 5) demands that the modeling formalisms are modeled as well. Formal models of domain-specific formalisms (called meta-models) enable the construction of meta-programmable tool environments that can be turned into a domain-specific environment by means of the meta-models [53], [54].

The specification of the abstract syntax of domain-specific modeling languages requires a meta-language that can express concepts, relationships, and integrity constraints. In model-integrated computing (MIC) [45], UML [8] class diagrams and the object constraint language (OCL) are used as meta-language. The semantic domain and semantic mapping defines the semantics of a modeling language. The role of semantics is to describe the meaning of models in precise, usually mathematical terms. Naturally, models might have different interesting properties, therefore, a modeling language might have a multitude of semantic mappings associated with it. For example, structural and behavioral semantics are frequently used interpretations of modeling languages. The structural semantics of a modeling language describes the meaning of the models in terms of the composition of possible model instances: structural semantics is frequently called instance semantics. Accordingly, the semantic domain for structural semantics is defined by some form of set-relational mathematics. The behavioral semantics describes the evolution of the state of the modeled artifact along some time model. Hence, behavioral semantics is formally modeled by mathematical structures representing some form of dynamics, such as hybrid dynamic systems [16]–[19].

There are two frequently used methods for specifying semantics: 1) the meta-modeling approach and 2) the translational approach. In the meta-modeling approach, the semantics is defined by a meta-language that already has a well-defined semantics. For example, the UML/OCL meta-language that can be used for defining the abstract syntax of a modeling language has a structural semantics: it describes the possible component structure of valid, syntactically correct domain models. The semantics of this meta-language can be represented by means of a formal language, which enables the precise definition of sets and relations on sets. By providing the formal semantics for UML class diagrams and OCL, say, in Z [55], the meta-model of domain-specific languages specifies not only their abstract

syntax, but their structural semantics as well, which addresses Requirement 6.

The translational approach specifies semantics via specifying the mapping between a modeling language and another modeling language with well-defined semantics.

The recent UML effort relies heavily on (so-called 'loose' [56]) meta-models. Other work applies these notions to software architectures [57]. More specific control system technological efforts are the generic modeling environment (GME) [58] and the domain modeling environment (DoME) [53], projects to quickly instantiate formalisms and even entire (real-time capable) tools [59], [60]. In control systems design, meta-models can be used to express and validate domain constraints (e.g., conservation of energy or deadlock freeness). This allows the use of formal methods to be applied to a much smaller model (the meta-model) instead of an entire component model, subsystem model, or even a complete system model, thereby avoiding tractability pitfalls. The verified meta-characteristics help restrict the design space and add to the validity of the models based on the particular formalism.

Using meta-modeling for formalism and tool design has several benefits. Besides the support for quickly instantiating tools and formalisms, it allows to conveniently migrate the formalism or tool along with the user requirements. In addition, an explicit meta-model of formalisms makes it easier to synthesize these together.

C. The Cross-Product Space

Developments in each of the branches of the V in Fig. 13 are important in the overall structure of multiparadigm modeling. One of the interesting problems is to transcend the application domains and to find independent frameworks to support each of the individual branches. In addition, there is a lot to be gained from leveraging the individual results by combining them across the multiparadigm modeling dimensions.

The case data interchange format (CDIF) [61], for example, uses meta models of data formats to facilitate data exchange between formalisms. The meta data allows tools to import data that is not understood in all its facets, but of which it has some knowledge at a meta level, which satisfies Requirement 7. At present, graph transformations are one of the most important technologies to translate between formalisms and the production rules for the translation are typically specified at the meta level [62].

Another important application that addresses Requirement 8 by cutting across all three dimensions is the combined use of formalisms in system modeling. This comes in four incarnations: 1) multiple views allow embedding different aspects of a model; 2) combining different formalisms to interact with each other; 3) heterogeneous refinement where more detail of a process is represented in a different formalism; and 4) layering of system models through different levels of the implementation process [63]. This is the most comprehensive use of computer automated multiparadigm modeling and applies very much to the control design at a system level.

Tools, techniques, methods, and applications of these multiparadigm modeling notions in the field of control system technology are the topic of this special issue that contains state of the art presentations. Details on the theory and methodology can be found elsewhere [44].

VI. Conclusion

The design of large control systems centers around the "core control algorithms." These are control algorithms as designed from the initial specifications and may range from being of a discrete event reactive nature to nonlinear model predictive control (e.g., [64], [65]).

In general, the bulk of the engineering effort in control system design deals with taking the core algorithm to a realization. This is especially true for embedded control systems because of the effects of embedded technologies on implementation and performance (think of fixed-point resolution, scheduling effects, analog-to-digital conversion delays, etc.). The difficulties are further exacerbated by the increasing use of heterogeneous implementation platforms (using digital signal processors, application specific integrated circuits, field programmable gate arrays, etc.).

To provide automated support for control system design, the implementation effects have to be modeled in much detail. This requires high-level modeling languages with well-defined formalisms (both semantics and syntax) for the functional, implementation, and realization aspects of the design process.

In addition, safety and performance requirements mandate analysis of the interaction of the different core control algorithms and their embedding data analysis functionality at various levels of detail. This necessitates a comprehensive analysis of behavior that involves various subsystems simultaneously. These subsystems may be at different levels of detail and even in different stages of the design process, i.e., subsystems in their implementation phase may be mixed with those in their early functional specification stages.

To facilitate this need, modeling formalisms tailored to a particular domain need to be quickly designed and easily evolved as the needs of the design engineers change. These formalisms should include domain-specific constraints to make the designed models inherently better. Models in different formalisms need to be coupled and related. This can occur at the model level, data level, and numerical level (cosimulation) and allows the use of tools with analysis capabilities best suited to the problem at hand. In addition, models have to be translated between different formalisms and different levels of abstraction, and the use of different views on different aspects of the system needs to be supported.

These requirements span three dimensions: 1) support for multiformalism modeling; 2) translation between different levels of abstraction; and 3) meta-modeling, i.e., modeling the modeling formalism. Computer automated multiparadigm modeling strives for further results in each of these dimensions in order to make them domain independent and leverage their individual merits in the cross product space.

Acknowledgment

The authors would like to acknowledge the suggestions of and discussions with many people, in particular, G. Grübel, M. Mestchian, and H. Vangheluwe. They would also like to thank D. Hull and S. Prabhu for their help with the realization of the power window system.

REFERENCES

[1] K. Wijbrans, "Twente Hierarchical Embedded Systems Implementation by Simulation: A Structured Method for Controller Realization," Ph.D. dissertation, Univ. Twente, Enschede, The Netherlands, 1993.

[2] *Using Simulink*. Natick, MA: The MathWorks, Inc., Simulink, Jan. 2002.

[3] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Programming*, vol. 8, pp. 231–274, 1987.

[4] C. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978.

[5] *Petri Nets & Grafcet*, 1992.

[6] N. Halbwachs, "Synchronous programming of reactive systems, a tutorial and commented bibliography," in *Proc. 10th Int. Conf. Computer-Aided Verification (CAV'98)*. Vancouver, BC, June 1998, pp. 1–16. LNCS 1427.

[7] B. P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. San Diego, CA: Academic, 1990.

[8] OMG Unified Modeling Language Specification (1999, June). [Online]. Available: http://www.omg.org/

[9] M. M. Tiller, *Introduction to Physical Modeling with Modelica*. Boston, MA: Kluwer, 2001.

[10] IEEE Standard 1076.1-1999 (1999, Mar.). [Online]. Available: http://www.vhdl.org

[11] D. Karnopp, D. Margolis, and R. Rosenberg, *Systems Dynamics: A Unified Approach*, 2nd ed. New York: Wiley, 1990.

[12] H. M. Paynter, *Analysis and Design of Engineering Systems*. Cambridge, MA: MIT Press, 1961.

[13] R. Bosch, "CAN Specification,", Stuttgart, Germany, Tech. Rep., 1991.

[14] *Stateflow User's Guide*. Natick, MA: The MathWorks, Inc., 2002.

[15] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978.

[16] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory," *IEEE Trans. Automat. Contr.*, vol. 43, pp. 31–45, Jan. 1998.

[17] M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, Eds., *Hybrid Systems: Computation and Control*. ser. Lecture Notes Comput. Sci.: Springer-Verlag, Mar. 2001, vol. 2034.

[18] N. Lynch and B. Krogh, Eds., *Hybrid Systems: Computation and Control*. ser. Lecture Notes Comput. Sci.. Berlin, Germany: Springer-Verlag, Mar. 2000, vol. 1790.

[19] F. W. Vaandrager and J. H. van Schuppen, *Hybrid Systems: Computation and Control*, ser. Lecture Notes Comput. Sci.. Berlin, Germany: Springer-Verlag, Mar. 1999, vol. 1569.

[20] *SimMechanics User's Guide*. Natick, MA: The MathWorks, Inc., 2002.

[21] *SimPowerSystems User's Guide*. Natick, MA: The MathWorks, Inc., 2002.

[22] E. Christen, "The VHDL 1076.1 language for mixed-signal design," *Elect. Eng. Times*, vol. , 1997.

[23] *User Manual*. Ann Arbor, MI: Mech. Dynam., 2002. Automat. Dynam. Anal. Mech. Syst. (ADAMS).

[24] B. J. de Kruif and T. J. A. de Vries, "On using a support vector machine in learning feed-forward control," in *Proc. 2001 IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics*, Como, Italy, July 2001, pp. 272–277.

[25] S.-T. Levi and A. K. Agrawala, *Real Time Systems Design*. New York: McGraw-Hill, 1990.

[26] D. J. Hatley and I. Pirbhai, *Strategies for Real-Time Systems Specification*. New York: Dorset House, 1988.

[27] P. T. Ward and S. J. Mellor, *Structured Development for Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[28] *phyCORE-MPC555*, Mainz, Germany: PHYTEC Meßtechnik GmbH, 2000.

[29] K. D. Müller-Glaser, G. Frick, E. Sax, and M. Kühl, "Multiparadigm modeling in embedded systems design," *IEEE Trans. Contr. Syst. Technol.*, 2003.

[30] *SPICE User's Guide*. Berkeley, CA: SPICE, EECS Dept., Univ. California, 2002.

[31] T. Murata, "Petri nets: Properties, analysis, and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[32] P. H. Starke, *Analyze von Petri-Netz-Modellen*. Stuttgart, Germany: B.G. Teubner, 1990.

[33] A. Misra, "Sensor-Based Diagnosis of Dynamical Systems," Ph.D. dissertation, Vanderbilt Univ., Nashville, TN, 1994.

[34] M. Otter, C. Schlegel, and H. Elmqvist, "Modeling and realtime simulation of an automatic gearbox using modelica," in *Proc. ESS'97*, Passau, Germany, Oct. 1997, pp. 115–121.

[35] R. Otterbach, T. Pöhlmann, A. Rökgauer, and J. Vater, "DS1103 PPC Controller Board—Rapid Prototyping with Combined RISC and DSP Power for Motion Control," dSPACE GmbH, Paderborn, Germany, Tech. Rep., May 1998.

[36] J. Sztipanovits, G. Karsai, and T. Bapty, "Self-adaptive software for signal processing," *Commun. ACM*, vol. 41, no. 5, pp. 55–68, May 1998.

[37] H.-D. Joos, "A methodology for multi-objective design assessment and flight control synthesis tuning," *Aerosp. Sci. Technol.*, vol. 3, no. 3, pp. 161–176, 1999.

[38] S. Kumar, D. Bhatt, S. Vestal, B. Wren, J. Shackleton, H. Shirley, R. Bhatt, J. Golusky, M. Vojta, J. Fischer, S. Crago, B. Schott, R. Parker, and G. Gardner, "ADAPTERS," in *Proc. 2nd Annu. Military and Aerospace Applications of Programmable Devices and Technologies Conf.*, Laurel, MD, Sept. 1999.

[39] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, and S. Asaad, "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," Vanderbilt Univ., Nashville, TN, Tech. Rep. ISIS-99-01, 2000.

[40] H. Garcia, A. Ray, and R. Edwards, "A reconfigurable hybrid system and its application to power plant control," *IEEE Trans. Contr. Syst. Technol.*, vol. 3, no. 2, June 1995.

[41] L. Palopoli, C. Pinello, A. S. Vincentelli, L. Elghaougi, and A. Bicchi, "Synthesis of robust control systems under resource constraints," in *Lecture Notes in Computer Science, Hybrid Systems: Computation and Control*, C. Tomlin and M. Greenstreet, Eds. Berlin, Germany: Springer-Verlag, Mar. 2002, pp. 337–350.

[42] H. Mann, "A versatile modeling and simulation tool for mechatronics control system development," *Proc. 1996 IEEE Symp. Computer Aided Control System Design*, pp. 524–529, 1996.

[43] J. B. Ferris and J. L. Stein, "Development of proper models of hybrid systems: A bond graph formulation," in *Proc. 1995 Int. Conf. Bond Graph Modeling and Simulation (ICBGM'95)*, vol. 27, F. E. Cellier and J. J. Granda, Eds., Jan. 1995, pp. 43–48.

[44] P. J. Mosterman and H. Vangheluwe, Eds., *Special Issue on Computer Automated Multi-Paradigm Modeling*, ser. ACM Trans. Modeling Comput. Simulat., 2003, vol. 12.

[45] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proc. IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.

[46] J. Davis II, R. Galicia, M. Goel, C. Hylands, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong, *Ptolemy II—Heterogeneous Concurrent Modeling and Design in Java*. Berkeley, CA: Dept. Elect. Eng. Comput. Sci., Univ. California, 1999. version 0.1.1.

[47] F. J. Barros, "Modeling formalisms for dynamic structure systems," *ACM Trans. Modeling Comput. Simulat.*, vol. 7, no. 4, pp. 501–515, 1997.

[48] H. Vangheluwe, "DEVS as a common denominator for multi-formalism hybrid system modeling," in *Proc. IEEE Int. Symp. Computer Aided Control System Design*, Anchorage, AK, Sept. 2000, pp. 129–134.

[49] G. J. Pappas, G. Lafferriere, and S. Sastry, "Hierarchically Consistent Control Systems," Univ. California, Berkeley, CA, Tech. Rep. UCB/ERL M98/16, 1998.

[50] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. Int. Conf. Computer-Aided Verification (CAV'97)*, Chicago, IL, July 2000, pp. 154–169.

[51] W. Minten, S. Vranckx, B. D. Moor, and J. Vandewalle, "Bondlab, a MATLAB based GUI for bond graph modeling," *J. A—Special Issue Computer Aided Control System Design*, vol. 38, no. 3, pp. 11–15, 1997.

[52] J. van Dijk, "On the Role of Bond Graph Causality in Modeling Mechatronic Systems," Ph.D. dissertation, Univ. Twente, Enschede, The Netherlands, 1994.

[53] E. Engstrom and J. Krueger, "A meta-modeler's job is never done: Building and evolving domain-specific tools with DOME," in *Proc. IEEE Int. Symp. Computer Aided Control System Design*, Anchorage, AK, Sept. 2000, pp. 83–88.

[54] G. Karsai, G. Nordstrom, A. Ledeczi, and J. Sztipanovits, "Specifying graphical modeling systems using constraint-based metamodels," in *Proc. IEEE Int. Symp. Computer Aided Control System Design*, Anchorage, AK, Sept. 2000, pp. 89–94.

[55] J. Davis and J. Woodcock, *Using Z: Specification, Refinement, and Proof*, ser. Int. Ser. Comput. Sci.. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[56] C. Atkinson, "Metamodeling for distributed object environments," in *Proc. 1st Int. Enterprise Distributed Object Computing Workshop (EDOC'97)*, Brisbane, Australia, 1997, pp. 90–101.

[57] D. Garlan, R. T. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *Proc. CASCON'97*, Toronto, ON, Canada, Nov. 1997, pp. 169–183.
[58] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti, "On meta-model composition," in *Proc. IEEE Int. Conf. Control Applications*, Mexico City, Mexico, Sept. 2001.
[59] M. A. P. Remelhe, S. Engell, M. Otter, A. Deparade, and P. J. Mosterman, "An environment for the integrated modeling of systems with complex continuous and discrete dynamics," in *Modeling, Analysis, and Design of Hybrid Systems*, S. Engell, G. Frehse, and E. Schnieder, Eds. Berlin, Germany: Springer-Verlag, 2002, vol. 279, pp. 83–105. Lecture Notes Inform. Sci..
[60] J. Sztipanovits, G. Karsai, C. Biegl, T. Bapty, A. Ledeczi, and A. Misra, "MULTIGRAPH: An architecture for model-integrated computing," in *Proc. Int. Conf. Engineering of Complex Computer Systems (ICECCS'95)*, Ft. Lauderdale, FL, Nov. 1995, pp. 361–368.
[61] J. Ernst, "Data interoperability between CACSD and CASE tools using the CDIF family of standards," in *Proc. 1996 Int. Symp. Computer Aided Control System Design*, Dearborn, MI, Sept. 1996, pp. 346–351.
[62] L. Baresi and M. Pezzè, "On formalizing UML with high-level petri nets," in *Concurrent Object-Oriented Programming and Petri Nets*, F. D. Cindio and G. Agha, Eds. Berlin, Germany: Springer-Verlag, 1999, pp. 271–300.
[63] S. Vestal, *Software Architecture Workshop*. Minneapolis, MN: Honeywell Technol. Library, July 1994.
[64] K. J. Åström and B. Wittenmark, *Computer Controlled Systems: Theory and Design*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
[65] A. Bemporad, P. Borodani, and M. Mannelli, "Hybrid control of an automotive robotized gearbox for reduction of consumptions and emissions," in *Hybrid Systems: Computation and Control*, O. Maler and A. Pnueli, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 81–96.

Pieter J. Mosterman (M'95) received the Ph.D. degree in electrical and computer engineering from Vanderbilt University, Nashville, TN, and the M.Sc. degree in electrical engineering from the University of Twente, The Netherlands, in 1991 and 1997, respectively.

He is a Senior Research Scientist in Real-Time and Modeling and Simulation Technologies, The MathWorks, Inc., Natick, MA. Previously, he held a research position at the German Aerospace Center (DLR) in Oberpfaffenhofen. He is Associate Editor of the *International Journal of Applied Intelligence* and was guest editor of a special CAMPaM issue of *ACM Transactions on Modeling and Computer Simulation*. His primary research interests are in hybrid dynamic systems and computer automated multiparadigm modeling (CAMPaM), with principal applications in embedded control systems, training systems, and fault detection, isolation, and reconfiguration. He cochaired the 14th International Workshop on Principles of Diagnosis. He is the Mechatronics Editor of SIMULATION, Transactions of the SCS.

Dr. Mosterman is an Associate Editor of the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY. He designed several modeling and and simulation enviroments, such as the *Electronics Laboratory Simulator* (nominated for The Computerworld Smithsonian Award). He received the Institution of Mechanical Engineers Donald Julius Groen Prize for his paper on HYBRSIM.

Janos Sztipanovits (M'86–SM'90–F'01) graduated from the Technical University of Budapest in 1970 and received the degree of "Candidate of Technical Sciences" from the Hungarian Academy of Sciences, Hungary, in 1980 and the distinguished Doctor degree (Golden Ring of the Republic) in 1982 from the same university.

He is currently the E. Bronson Ingram Distinguished Professor of Engineering in the Electrical Engineering and Computer Science Department, Vanderbilt University, Nashville, TN. He is Founding Director of the Institute for Software Integrated Systems (ISIS), Nashville, TN. Between 1999 and 2001, he worked as Program Manager and Acting Deputy Director of DARPA Information Technology Office. He has published over 150 papers and is the coauthor of two books. During the past two decades, he has conducted research on model-integrated computing, structurally adaptive systems, and embedded software and systems.

Sebastian Engell (M'85) received the Dipl.-Ing. degree in electrical engineering from the Ruhr-Universität Bochum, Bochum, Germany, in 1978, and the Dr.-Ing. degree from the University of Duisburg, Duisberg, Germany, in 1981. In 1987, the University of Duisburg granted him the *venia legendi* in automatic control.

From 1981 to 1984 and 1985 to 1986, he was a Senior Researcher in the Automatic Control Group, Mechanical Engineering Department, University of Duisburg. From 1984 to 1985, he spent a year at McGill University, Montréal, Canada. In 1986, he joined the Fraunhofer-Institut IITB, Karlsruhe, Germany, where he led projects on industrial control and production scheduling. Since 1990 he is Professor of Process Control in the Department of Biochemical and Chemical Engineering, University of Dortmund, Germany. From 1996 to 1999, he served as the Chairman of the Department of Chemical Engineering. Since 2002, he is a Vice-Rector of the University of Dortmund, Dortmund, Germany. He was associate editor of the *European Journal of Control* from 1995 to 2000. He is currently Associate Editor of the *Journal of Process Control* and of *Mathematical and Computer Modeling of Dynamic Systems*. His areas of research are modeling and control of chemical processes, hybrid systems, and scheduling in the process industries.

Dr. Engell was Co-Editor of the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY from 1992 to 2000. He received a Joseph von Fraunhofer Prize in 1991.