An Industrial Embedded Control System Design Process

Pieter J. Mosterman
The MathWorks, Inc.
Natick, MA 01760
pieter.mosterman@mathworks.com

Sameer Prabhu
The MathWorks, Inc.
Novi, MI 48375
sameer.prabhu@mathworks.com

Tom Erkkinen
The MathWorks, Inc.
Novi, MI 48375
tom.erkkinen@mathworks.com

Abstract

Embedded software provides a flexible and powerful means to differentiate products and achieve competitive advantage. However, the design of embedded software tends to be difficult to manage. This paper describes use of model-based design to help manage the design of embedded control systems. It presents typical design configurations, such as rapid prototyping, software-in-the-loop, processor-in-the-loop, and hardware-in-the-loop, and discusses their differences and usages. Finally, an overview of some of the aspects typical in developing embedded control systems using model-based design is given.

1 Introduction

Embedded software has become indispensible in consumer products ranging from automobiles to cell phones. For example, it is commonly estimated that 90% of all innovations in the automotive industry are in embedded software, while 40% of the overall cost of premium automobiles is incurred by software and electronics [2].

In the automotive industry, software adds functionality such as central door-lock, anti-lock braking, and even 'by-wire' technologies (e.g. brake-by-wire) that are now common in aerospace. Software is even used to configure and control hardware differently, e.g., to trade off comfort and performance. Software is becoming the main differentiator between products in most areas where it is used, such as cell phones, TVs, coffee makers, and washers.

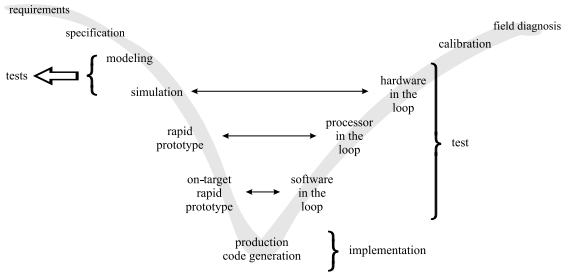

Figure 1: Industrial design process.

With the benefits that software brings comes the difficulty of managing the design process. Software is notoriously difficult to design, despite many efforts to establish a more controllable software design paradigm. In business applications, this difficulty often results in time-to-market determining the release cycle, rather than the quality of the product.

For embedded control applications, however, this trade-off cannot be applied. Quality is of primary importance. Therefore, it is necessary to establish a design process that, as far as possible, allows control over the new product introduction process. To structure a predictable design process the prevalent systematic approach in automotive and aerospace industry is model based. This approach is illustrated in Fig. 1 using a 'V'-shaped trajectory.

Moving downward along the left stroke of the 'V', the design starts with a set of requirements that are transformed into a system specification. This sys-

tem specification is then decomposed into subsystems, each with its own well-defined interface specification and internal behavior. The decomposition process commences hierarchically until the component level is arrived at. Throughout this process, typically computer-aided systems engineering (CASE) tools can be used, though modeling tools such as Simulink® [18] are increasingly being applied. The advantage of the latter is that it results in 'executable specifications,' i.e., the specification is given using a modeling language with a formal semantics which allows its dynamic behavior to be generated, and early validation of initial requirements becomes possible. The test cases used to validate the requirements can be reused to test the system implementation.

Hierarchically deriving specifications results in system modules with well-defined interface behavior. These modules can be implemented in hardware or software. Often this is done by external suppliers to the original equipment manufacturer (OEM). These suppliers specialize in a given field, e.g., automatic transmissions, and construct the requested component according to the specification. In the 'V', this is the bottom of the left stroke, just before the design process starts moving up along the right stroke.

The implemented components are then composed together according to the design to realize the subsystems. Those subsystems are then composed together, tested, and tuned, until, eventually, the system is constructed, tested, and calibrated. This step is at the top of the right-hand stroke of the 'V'. The process is 'V'-shaped because there is a correspondence between the level of decomposition in the left-hand design phase and the right-hand implementation phase.

While moving down and back up the design 'V', iterations may occur. It is desirable for these iterations to occur locally, preferably within one level of decomposition. If not, the iteration cost grows quickly, especially if the implementation of a subsystem requires revisiting its specifications, e.g., because the requirements cannot be met. This implies moving back from the right-hand stroke of the design 'V' to the left-hand stroke, where it may affect the design of other subsystems and components, a potentially expensive proposition.

When the OEM hands over the specifications for a component to its supplier, it is important to know the supplier can feasibly implement the specifications. There are many ways to validate feasibility against a possible implementation and prevent divergence. Specifically, the industrial design process employs configurations known as hardware-in-the-loop, processor-in-the-loop, and rapid prototyping.

This paper discusses the use of design configurations throughout the model-based design of embedded control systems. In Section 2, the idiosyncracies of embedded software, in particular with respect to desktop applications, are given. Section 3 briefly introduces the example of a power window system used in modern automobiles. In Section 4 the different configurations used in the industrial design process are presented. Section 5 discusses issues that need to be addressed in the design of embedded control systems. Section 6 presents conclusions.

2 On Embedded Software

With the widespread availability of embedded computing power, functionality that was typically implemented in hardware is increasingly being moved into software. For example, in safety-critical systems such as aircraft, a number of sensors may be used to obtain one measured quantity. The redundant sensor readings have to be evaluated and consolidated to arrive at one value that has the largest probability of being the true value [13]. This 'voting' functionality is now implemented in software with superior flexibility compared to the original hardware implementations. Even the low-level feedback control is implemented in software, as a result, the originally designed continuous-time control law must be discretized so that it can execute at a given sample rate.

Contrary to the additional complications to implement continuous-time control laws in an embedded controller, discrete event reactive behavior is perfectly suited for embedded controllers. It is, therefore, no surprise that such functionality is increasingly being used and the size of embedded code dedicated to this type of control is rapidly growing. For example, complex redundancy management schemes

require extensive control that is of a discrete event reactive nature [17].

Even though it is software, the embedded nature of the deployed code prohibits the use of design methodologies developed for software projects such as desktop applications. A well established and much discussed differentiator is the 'real-time' nature of embedded control systems [10]. In software design paradigms, time is often the first thing that is abstracted away and all that matters is the logical correctness of a program. For real-time applications, however, a logically correct but tardy answer is still a failure. Furthermore, in computer science much effort is devoted to ensuring that a program terminates, while in embedded applications, a program should never terminate.

These considerations necessitate a design process that is distinctly different from more traditional software design, for example, because the componentization of software is not an efficient approach. Given the gradual refinement of a design, it would mean that one design component corresponds to a number of software components each with varying levels of detail. For example, at one point, a designed control law may switch from a floating-point implementation to a fixed-point implementation; therefore, a different software component would have to be selected. Maintaining a database of related components is time-consuming at best and very error-prone as each update to one component needs to be included in its related ones as well.

An exacerbating phenomenon is the practice of repartitioning the software as more detail becomes available. For example, in the later design stages when implementation effects such as computing resources and their need for scheduling and managing shared memory become available, rigorous changes to earlier software may become necessary. This all stems from the refining nature of embedded software design as opposed to a composing nature of more classical software systems. Earlier versions of embedded software with limited functionality cannot simply be extended by adding more components; the components need to be refined. In the process, even component repartitioning may be required, e.g., to achieve context-related optimizations.

To address these needs, higher-level design methods are a necessity. Rather than executing much of the system design at a software level, automatic code generators can be exploited to include implementation aspects at the model level. In addition, tailoring automatic code generators provides great flexibility in terms of facilitating platforms used for intermediate validation stages. Furthermore, it becomes possible for software and systems engineers to communicate at the same level and in the same language, which greatly reduces the turn-around time for any defects found in the design when implemented in software [20]. On the other end, using a high-level language supports high-level analyses such as reachability and coverage and ties in seamlessly with test vector generation.

Given that the ultimate goal of embedded systems is the interaction of embedded computing power with the physical world, it is important to validate the controller design in this context as soon as possible. A prominent role in this is played by the real-time aspects of the embedded system. Controlling the dynamics of the physical world requires precise timing. It is imperative that the scheduled timing is found to be correct and feasible early in the design process.

3 A Power Window System

This paper will describe the different design configurations during the development process using a power window system as employed in modern vehicles. Figure 2 shows the door of an automobile. Its window is intended to be opened and closed by means of an electric motor. The operation of this motor is controlled by a microprocessor that translates driver and passenger commands into electrical signals to move the window up and down.

A number of *transducers* transfer energy between physical domains. The microprocessor operates in the electrical domain; therefore, its input and output are electrical signals. The window operates in the mechanical domain. A transducer converts power between the electrical and mechanical domains.

A DC motor serves as the *actuator* to move the window up and down. This actuator contains a trans-

Figure 2: Power window.

ducer that transforms electrical voltages and currents into mechanical torques and velocities. The power required to move the window up and down, however, exceeds the power that the microprocessor can supply; therefore, signal conditioning hardware is used to 'amp up' the electrical signal produced by the microprocessor. This hardware consists of an electrical amplifier and some passive circuitry connected to the vehicle battery.

To control the window, the microprocessor takes in driver and passenger commands from a Controller Area Network (CAN) [4] used in automobiles. In addition, it uses the armature current as drawn by the DC motor. This current is measured by a *sensor* that connects to the DC motor and produces a voltage across a small resistance in the armature line. This voltage needs to be scaled within the range that the microprocessor can handle and of sufficiently low impedance. This, again, requires signal conditioning hardware, but now on the sensor side.

All this leads to the configuration of the embedded control system shown in Fig. 3. The microprocessor contains the *controller* functionality while the *plant* represents the door, including the window and its lift mechanism. As discussed, sensors and actuators are required to connect the low-power electrical domain of the controller to the high-power mechanical domain of the plant.

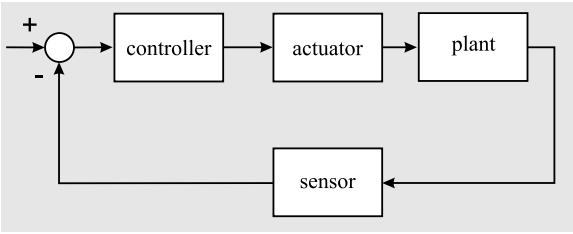

Figure 3: General embedded control system configuration.

The configuration in Fig. 3 is a general template for embedded control systems and is used prevalently in control engineering [7]. In general, it is part of hierarchical control structures which is discussed in detail for the power window elsewhere [12].

4 Configurations Throughout the Design Process

Throughout the 'V'-shaped design process shown in Fig. 1, different system configurations are used based on the general template in Fig. 3. The configurations can be differentiated based on (i) the hardware used (varying from general-purpose computers to dedicated target processors), (ii) whether it is operated in real-time, (iii) if a model is used or embedded code, and (iv) how the first three factors apply to the plant and controller.

4.1 Modeling

In case the plant to be controlled already exists, which often is the case in, for example, the automotive industry, a model is generally obtained to facilitate control law synthesis and other design and analysis methods. Many plant modeling techniques exist to derive a model structure (e.g., bond graphs [14]) but plant measurements are necessary to determine and calibrate the model parameters.

Depending on the plant, modeling can proceed from first principles where model parameters that capture the dynamics of the model are determined

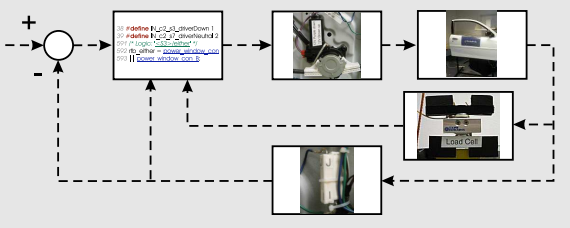

Figure 4: Modeling.

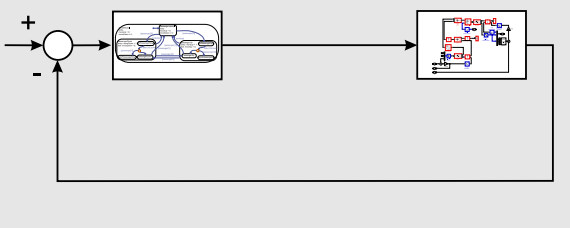

Figure 5: Simulation.

by static measures. For example, in case of a mechanical system, it is possible to derive the center of gravity of a body and its inertia from its shape and weight. This approach requires complete knowledge of the internals of a plant, and so can be termed a 'white-box' approach to modeling. At the other extreme, such detailed information about parts internal to the plant may not be available, or not be helpful, and a 'black-box' approach to modeling may be applied. In this approach, the system behavior is mapped onto a transfer function of a predetermined complexity. This requires elaborate system identification, including extensive measurements of plant variables.

The configuration in Fig. 4 illustrates this for the system identification of a power window mechanism. The block at the top right represents the plant, i.e., the mechanism inside the door to lift the window as well as the window itself and its rails. The block that connects to the input of the plant represents the actuators. In this case there is one DC motor that moves the window up and down. Two types of sensors are shown to connect to the output of the plant block. One measures the current drawn by the DC motor. The output of this block feeds back to the input of the controller, i.e., it is used for control purposes. At the same time, the measured current is logged for parameter identification purposes, indicated by connecting it separately at the bottom of the controller block. Another type of sensor may be present that is not used to control the plant, but whose values are logged only for parameter identification. This type of sensor is represented by the other block connected to the plant output, in this case a load cell.

Note that in many cases, there is no need for, at least a rudimentary, controller to have the plant operate at different operating points selected to provide measurement data. Also, the control and data logging functionality could be implemented on different computing platforms. The broken lines in Fig. 4 indicate that this is a real-time measurement setup.

4.2 Controller Design and Synthesis

Once a model of the plant is available, it can be used in the control law design process. Typically, extensive system identification methods may result in detailed models with many non-linearities, some of which are empirical in nature. In fact, look-up tables are found to be indispensible in industrial models and form a key ingredient of intellectual property.

To synthesize a controller from such models, the detailed model has to be simplified. For many synthesis methods, it needs to be linearized and its order reduced, either because the synthesis methods are based on linear systems theory or because otherwise the controller would be of too high an order. For example, a finite element approach to modeling may quickly lead to a model with several hundred states.

The control law design exploits a modeling and simulation configuration that is illustrated in Fig. 5 for the power window system. The controller block on the left contains a model of the designed control law. The plant block on the right contains the model of the plant as derived after the system identification.

This is a completely functional setup in which no implementation aspects of the eventual embedded nature are included. No actuation and sensor effects are accounted for yet, and dynamic behavior analyses

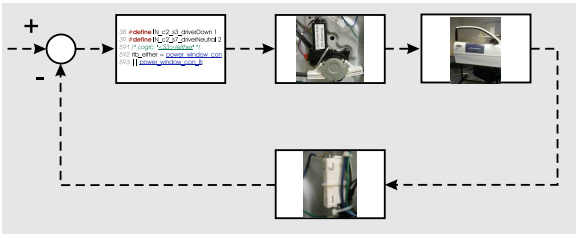

Figure 6: Rapid prototyping.

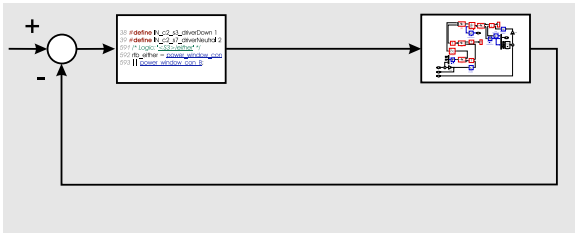

Figure 7: Software-in-the-loop.

rely on non real-time simulation, which allows, for example, the use of variable-step integration methods.

4.3 Tuning

Once the control law has been designed, a rapid prototyping configuration, as shown in Fig. 6, is used to tune the control law. Again, additional sensors may be present to measure values that are part of the requirements but are not necessary for the control law design. In case of the power window design, a load cell may be included to measure the force on a potential object that is trapped between the window and the frame. This force should never exceed 100 [N] according to the requirements

For higher fidelity tuning, one could execute the control law code on the actual embedded target. This on-target rapid prototyping scenario lets developers tune the functional performance while making sure that the code fits within the resources of the embedded system (e.g., RAM/ROM). Advances in code generation technology now allow for highly efficient code that can first be used for prototyping then redeployed for mass-production systems.

4.4 Test

Once the controller has been designed, tuned, and verified, the system goes through a number of test configurations to verify the correctness of the embedded code.

Testing it by feeding the controller artificial inputs is a complicated task, especially since the controller output depends on its internal state and, therefore, an extensive history of previous input. An important aspect of this exhaustive testing is a coverage analysis [1]. Depending on its level of detail, such analysis determines whether all parts of the embedded code are indeed executed in response to the test cases that were derived based on the requirements. Any code that is present but not needed to satisfy the requirements is superfluous and should be removed.

Software-in-the-Loop The first stage verifies that the generated algorithm code satisfies the requirements and produces results that are bit-true to the fixed-point simulation. In this *software-in-the-loop* configuration, shown in Fig. 7, code that is generated from the controller model is connected to a plant model. No sensor and actuator effects are included and the system is tested in simulated time. At this time, target effects such as fixed-point data types are emulated in software.

This configuration does not operate in real-time. After the controller has computed its new output values from its input, the simulated plant is moved one step forward in time and the computed output of the plant is then immediately used as input to the controller to ensure quick and efficient simulation.

Processor-in-the-Loop Next, controller code is generated and tested on the target platform to ensure that the compiler works properly and to test the linker and loader. This configuration, shown in Fig. 8, is referred to as *processor-in-the-loop*. This configuration still does not operate in real time.

At this point, on-target debugging is often used to track down possible defects in the embedded code. This is a much more difficult task than analyzing the

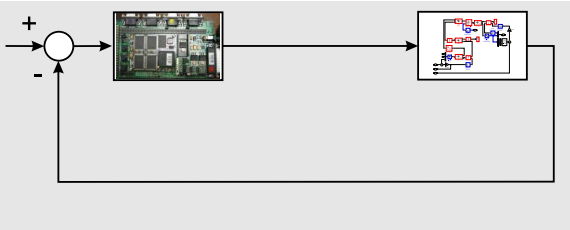

Figure 8: Processor-in-the-loop.

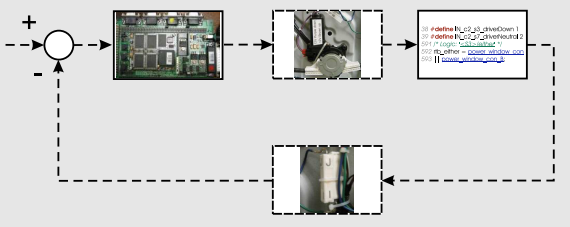

Figure 9: Hardware-in-the-loop.

code in the software-in-the-loop configuration, where debugging facilities such as setting breakpoints and stepping through the code are available.

Hardware-in-the-Loop In the penultimate configuration, the so-called *hardware-in-the-loop*, the generated code is run on the actual target processor connected to a real-time simulation of the plant as shown in Fig. 9. The plant simulation model contains as much detail as possible. The configuration may even include actual sensor and actuator hardware, e.g., in case their behavior would be difficult to model in sufficient detail (the sensor and actuator blocks are, therefore, shown with a broken outline). This is especially true if the data acquisition hardware applies embedded processors itself, in which case the system cannot simply be stopped to make measurements and continued one more step. Also, peripherals such as the actual power supply for the controller, actuators, and sensors, are connected and tested at this point.

For example, in the power window system, the DC motor used to position the window may be driven by a pulse width modulated control signal. To derive a sufficiently detailed model of this motor may be an arduous task, if not impossible. Instead, at this point in the design phase, the actual DC motor can be included in the test configuration. The output of the designed controller then connects to the DC motor input. The angular velocity that it produces in return is used as input to a real-time model of the window lift mechanism. Note that this requires the DC motor to be connected to a load that mimics the force it would experience when moving the actual window.

In addition to the sensors and actuators that may be included in this configuration, real-time simulations of interacting systems, or at least their effect on the design, may be included. For example, in case of the power window, commands to move the window up and down are generated by a switch that could be located in the center console of the vehicle. A controller area network communicates the commands to the actual controller, but this network is used for more functionality in that vehicle (e.g., opening and closing the sun roof, moving the headlights up and down, and centrally locking the doors). All these systems generate network traffic and, therefore, may affect performance. How performance is affected is all tested in the hardware-in-the-loop configuration, i.e., immediately before the final implementation, though in general many of these effects are best included in earlier analyses as well.

4.5 Calibration

When it has passed all tests and the system is properly integrated, the control code is deemed correct. At this point, the actual system can be realized, as shown in Fig. 10. Now that the actual hardware and software are available and connected in their final configuration, the embedded control system needs to be calibrated and tuned. This is still an elaborate process involving teams of experts and trained engineers using sophisticated calibration tools [11].

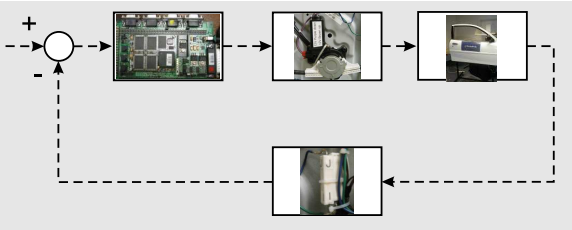

Figure 10: Final integration.

5 Aspects in Embedded Control System Design

Given the overview of configurations used in the design of embedded control systems, a number of aspects can be established that need to be addressed throughout the entire process from requirements capture to field deployment.

5.1 Heterogeneity

An efficient design process enables navigation through the many different views of the system. This is achieved by cross-referencing system design aspects that have a manifestation in different forms.

For example, a particular requirement such as "the force of a power window against a trapped object should never exceed 100 [N]" should result in a test case that is used throughout the system design. The computed force is inspected in simulation but it should also be a test case when the first realization is available. The requirement typically results in a specification, in this case, a threshold for the armature current drawn by the DC motor, and this threshold is present in the different models (for the different levels of detail) that are used and the different code (for the different targets) that is generated. Linking all related aspects greatly improves traceability, important for all embedded systems, crucial for safety-critical systems.

The different stages during embedded control system design employ different languages to capture the characteristic of interest. For example, in the early design stages, the high-level functional behavior of the system can be well captured by 'bubble charts' as found in Structured Analysis methods [9, 21]. However, to design the feedback control part of an embedded control system, time-based block diagrams such as those available in Simulink® [18] are typically used, while the reactive behavior is well captured by state transition diagrams that are extended with hierarchy, concurrency, and broadcasting facilities such as found in Stateflow® [19].

In the implementation phases, schematics are typically used and computer-aided design (CAD) tools are common. In software, the implementation aspects pertain to issues such as scheduling and functional partitioning, which can be well captured by dedicated tools in Simulink. Eventually, the actual programming language is the language of choice. In general, this is C.

To bring all of this together, cross-referencing is the first step. A more sophisticated approach is the use of model transformation technology. For example, Real-time Workshop® [15] automatically generates C-code from a time-based block diagram, including hyperlinked references to the original block diagram components. This transformation can be tailored to produce code that fits the desired appearance and structure.

Model transformation technology usage is growing, and research predicts major advances in this area, supporting, for example, the design by generating tests from the model [5], transforming between high-level formalisms [3, 6, 16], and supporting exchange between design tools [8].

5.2 Embeddedness

Embedded control systems combine hardware with software. This implies the need to support such integrated design. In particular, the design of an embedded controller moves through a number of stages with different goals. For example, in the early design stages, code may be generated that is not highly optimized but merely used to validate correctness of the algorithm in software. This code typically is run on much more powerful and flexible hardware than that used in the eventual product.

In later design stages, the code is optimized and a

different target is used to run the code, one that is closer to what is used in the final product. Though the code changes through these stages, the original model on which it is based should not. Therefore, facilities need to be available to generate code for different targets from the same model.

In addition to the embedded code that needs to be produced, the embedded nature of the computing power requires interaction between low-power logic electronics and the high-power plant. Furthermore, the code run on the microprocessor lacks standard user interface features of desktop computers, such as a keyboard entry facility and a monitor. This complicates verifying that the embedded code functions as designed. Especially in the case of incorrect output, it is difficult to determine the underlying defect from the input and output of the embedded processor.

This issue is resolved by communicating data about the execution of the embedded code from the embedded 'target' processor to a 'host' computer, e.g., by means of an RS-232 communication channel. This approach allows the retrieval of information about how the embedded code executes and supports features such as on-target parameter tuning.

For example, once the power window system has been realized, it may violate the maximum force of 100 [N] requirement because the armature current threshold was set too high. This parameter can now be changed in the original high-level model on the host, and the new value communicated to the target.

5.3 The Real-time Nature of Embedded Control Systems

Related to the embeddedness, but a different dimension is the real-time characteristic of embedded control systems. While in the processor-in-the-loop configuration behavior is not real time, it is critical in many of the other configurations.

For example, in the modeling phase, a rudimentary controller may be necessary to allow taking measurements that are used to compute model parameters. This rudimentary controller has to run in real time to operate the plant at a desired operating point.

This requires a real-time operating system (RTOS), and so the widely used desktop operating system such as Microsoft Windows cannot be used. Typically, this results in a target-host configuration where the desktop computer is used as a host to connect to a target that is running an RTOS and the control software. In sophisticated modeling environments, the communication link is made as transparent as possible, so it appears as though the high-level models that are run on the host are really the front end to the code ran on the target. This approach greatly alleviates the otherwise unwieldy task of managing the embedded code.

Furthermore, the computing power of the host can be used to aid in the design activities. For example, when calibration is required, the host can be exploited to generate the optimal set of measurements that need to be made to produce the calibration tables with highest fidelity.

6 Conclusions

To manage and control the design of embedded software, a model-based approach that is shaped like a 'V' is employed in industry. Model-based design allows for a systematic decomposition of the overall system, while increasingly adding detail to the system specifications. Once the low-level specifications have reached the level of detail that allows an implementation, the components can be integrated while ensuring the integration proceeds according to the specifications.

During this process, a number of design configurations are employed: rapid prototyping, on-target rapid prototyping, software-in-the-loop, processor-in-the-loop, and hardware-in-the-loop. The characteristics and application of these configurations was presented. In addition, a number of aspects particular to the design of embedded control system design have been categorized.

7 Acknowledgments

The authors gratefully acknowledge helpful discussions with Jason Ghidella on the topic.

References

[1] William J. Aldrich. Using model coverage analysis to improve the controls development process. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit*, Monterey, CA, August 2002.

[2] ICSE 2004 workshop on Software Engineering for Automotive Systems. Call for Papers, May 2004.

[3] Luciano Baresi, Alessandro Orso, and Mauro Pezzè. Introducing Formal Specification Methods in Industrial Practice. In *19th International Conference on Software Engineering (ICSE'97)*, pages 56–66, Boston, MA, May 1997.

[4] CAN specification. Technical Report, 1991. Robert Bosch GmbH.

[5] Mirko Conrad, Heiko Dörr, Ingo Stürmer, and Andy Schürr. Graph transformations for model-based testing. In M. Glinz and G. Mller-Luschnat, editors, *Proceedings Modellierung 2002*, pages 39–50, Tutzing, March 2002. GI-Edition Lecture Notes in Informatics P-12.

[6] Juan de Lara and Hans Vangheluwe. Defining Visual Notations and Their Manipulation through Meta-Modelling and Graph Transformation. *Journal of Visual Languages and Computing*, 15(3), June 2004.

[7] Richard C. Dorf. *Modern Control Systems*. Addison Wesley Publishing Co., Reading, MA, 1987.

[8] Rony G. Flatscher. Metamodeling in EIA/CDIF meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation*, 12(4), 2002.

[9] Derek J. Hatley and Imtiaz Pirbhai. *Strategies for Real-Time Systems Specification*. Dorset House Publishing Co., New York, New York, 1988.

[10] Edward A. Lee. What's ahead for embedded software. *Computer*, 33(9):18–26, September 2000.

[11] Model-Based Calibration. *Model-Based Calibration User's Guide*. The MathWorks, Natick, MA, 2002.

[12] Pieter J. Mosterman, Janos Sztipanovits, and Sebastian Engell. Computer automated multiparadigm modeling in control systems technology. *IEEE Transactions on Control System Technology*, 12(2), March 2004.

[13] Stephen Osder. Practical view of redundancy management application and theory. *Journal of Guidance, Control, and Dynamics*, 22(1):12–21, January-February 1999.

[14] Henry M. Paynter. *Analysis and Design of Engineering Systems*. The M.I.T. Press, Cambridge, Massachusetts, 1961.

[15] Real-Time Workshop. *Real-Time Workshop User's Guide*. The MathWorks, Natick, MA, 2002.

[16] Manuel A. Pereira Remelhe. Simulation and Visualization Support for User-defined Formalisms Using Meta-Modeling and Hierarchical Formalism Transformation. In *Proceedings of the IEEE International Conference on Control Applications*, Mexico City, Mexico, September 2001.

[17] Manuel A. Pereira Remelhe, Sebastian Engell, Martin Otter, André Deparade, and Pieter J.

Mosterman. An environment for the integrated modelling of systems with complex continuous and discrete dynamics. In S. Engell, G. Frehse, and E. Schnieder, editors, *Modelling, Analysis, and Design of Hybrid Systems*, pages 83–105. Springer-Verlag, Berlin, 2002.

[18] Simulink. *Using Simulink*. The MathWorks, Natick, MA, January 2002.

[19] Stateflow. *Stateflow User's Guide*. The MathWorks, Natick, MA, 2002.

[20] Steve Vestal. Software Architecture Workshop, July 1994.

[21] Paul T. Ward and Stephen J. Mellor. *Structured Development for Real-Time Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.