

Caterpillar Automatic Code Generation

Jeffrey M. Thate and Larry E. Kendrick
Caterpillar, Inc.

Siva Nadarajah
The MathWorks, Inc.

Copyright © 2004 SAE International

Abstract

Automatic code generation from models is actively used at Caterpillar for powertrain and machine control development. This technology was needed to satisfy the industry's demands for both increased software feature content, and its added complexity, and a short turn-around time. A pilot development effort was employed initially to roll out this new technology and shape the deployment strategy. As a result of a series of successful projects involving rapid prototyping and production code generation, Caterpillar will deploy MathWorks modeling and code generation products as their department-wide production development capability.

The data collected indicated a reduction of person hours by a factor of 2 to 4 depending on the project and a reduction of calendar time by a factor of greater than 2.

This paper discusses the challenges, results, and lessons learned, during this pilot effort from the perspectives of both Caterpillar and The MathWorks.

Introduction

This objective of this paper is to describe the activity and results of an ongoing effort by CAT Electronics¹ and The MathWorks² to implement an automatic code generation capability for production embedded systems. The stage for this activity was set by an initial effort that accelerated the development of control algorithms for CAT engines and machines. The

solution was based on the control system development process, which is iterative by nature and requires analysis, simulation, and testing capabilities.

This paper describes the key steps being used to achieve the solution and is presented as follows:

1. Initial Investigation and Roll-out
2. Projects and Applications
3. Results and Benefits
4. Lessons Learned
5. Future Plans

Initial Investigation and Roll-out

Utilizing hand-coded software in our previous process had caused long, time-consuming, iteration cycles that imposed severe limits on the number of iterations we could perform to develop a system. This in turn required control system designers to make final design decisions without adequate information. The solution identified and implemented was to utilize a model-based, rapid-prototyping capability.

With a model-based, rapid-prototyping capability, strategy models are graphically defined and tested against plant models using simulation. Code is then automatically generated from the graphical model and executed on rapid-prototyping hardware, which is separate from the Electronic Control Module (ECM). The rapid-prototyping configuration that CAT implemented is shown in Figure 1.

**CONTROL ANALYSIS & DESIGN
ALGORITHM DESIGN PROCESS
Using Autocode**

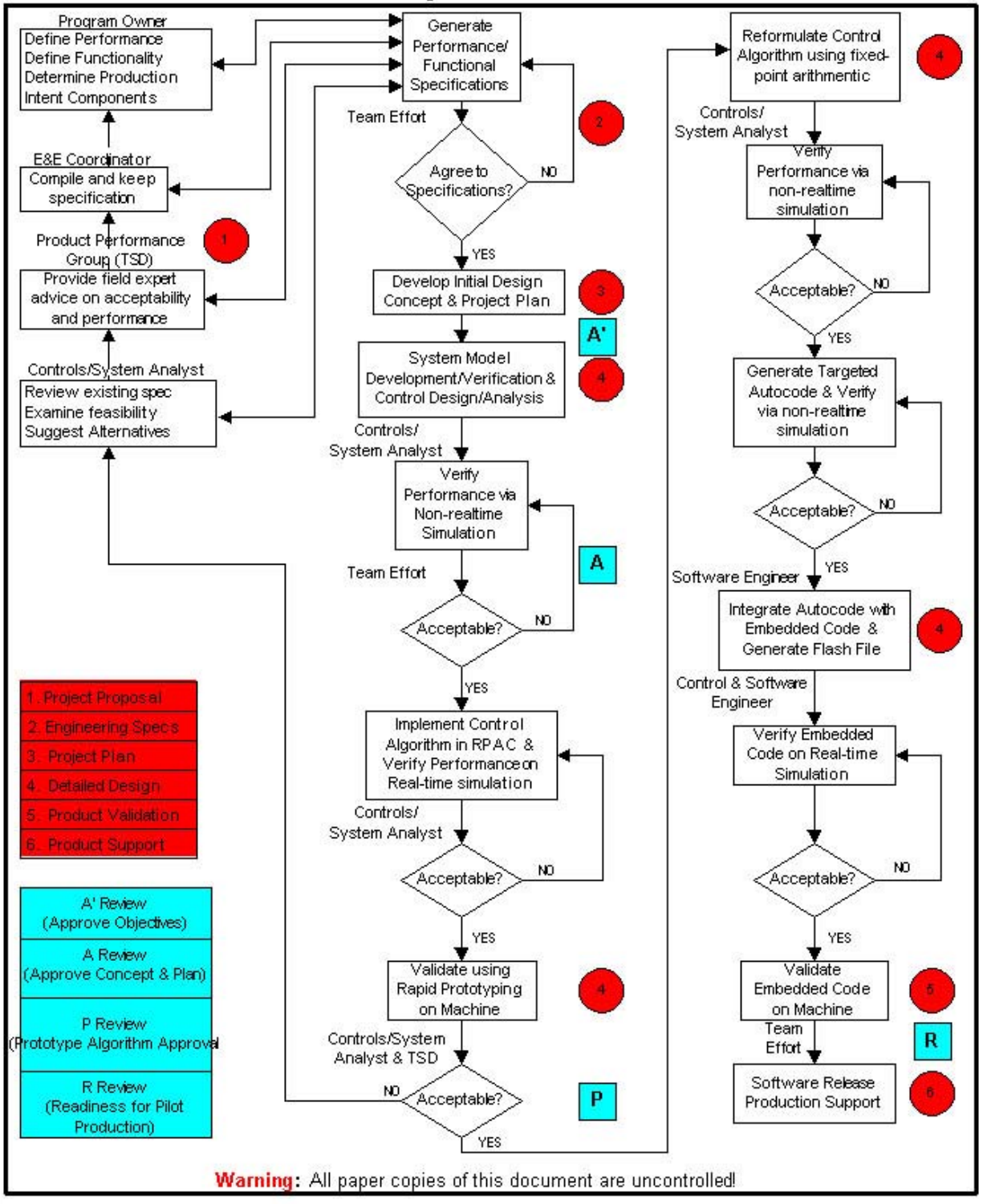


Figure 2: CAT ECM Development Process

In looking at tools to implement this process it was recognized that not only was the tool set required to handle analysis, simulation rapid prototyping, code generation, data acquisition and analysis, but the tool set must be modifiable to meet our unique coding and system design and integration standards. This means there must be willingness and capability on the part of the tool supplier to adapt the standard tool set as needed.

We selected The MathWorks as our tool supplier and settled on the tool chain of MATLAB®/Simulink®/Stateflow®/Real-Time Workshop Embedded Coder. The standard capabilities of the tool set were an important factor in our decision, but we also appreciated the open nature of the MathWorks environment and the availability of technical support through MathWorks Consulting services.

Projects and Applications

A series of pilot projects of increasing complexity were initiated in order to develop a production autocoding capability. Each pilot activity was intended to generate useful results in order to justify continued development.

The pilot activities are:

- Manually Integrated Automatically Coded Functions
- Manually Integrated Automatically Coded Applications
- Automatically generating code for Functions (automated integration)
- Autocoding Applications (automated integration)

Manually Integrated Automatically Coded Functions

The first pilots were focused on Autocoding functions of modest complexity and integrating them into a handwritten application. The code was expected to conform reasonably well with CAT coding standards, be reasonably efficient in terms of execution time and memory requirements, and be easy to associate the generated code with the defining Simulink subsystems (in other words, it had to be readable).

The conversion of the rapid-prototyping model into a form suitable for an embedded application required conversion of floating-point data to a discrete fixed-point implementation. Once converted to the discrete format the issue of code output format must be addressed. The code output format from The MathWorks code-generation tools was sufficiently configurable to CAT's coding standards and code readability objectives were easily achieved. Execution time was initially longer than expected. Investigation showed that the problem was associated with the standard lookup table techniques. The Simulink lookup table capability was extremely flexible and general, but this led to some inefficiencies. CAT Electronics had over time developed very efficient code for providing lookup tables. Working with MathWorks consultants, a modification to the tools was made that allowed the use of CAT Electronics lookup code. Once this was accomplished the efficiency objectives were met.

The automatically coded model was then tested in a simulation environment utilizing the Simulink S-function capability to insure that the conversion process had not introduced errors into the strategy. In order to integrate the automatic code with the handwritten environment, a handwritten "wrapper" was required. The creation of such wrappers was a reasonably simple task but obviously one which took some time and required significant software development skills. Even with the need to develop wrappers for the automatically coded functions, the efficiencies achieved were substantial and the approach was used to accelerate development of multiple critical product development programs including:

- Advanced Combustion Technology
- Track Type Tractor (TTT) Auto Carry
- Backhoe Loader (BHL) Swing Compensation
- BHL Implement Cross Modulation

Example of Manually Integrated Automatically Coded Functions

CAT ACERT Program

The purpose of the ACERT Program was to develop an engine control system to enable Caterpillar, Inc. diesel engines to meet the EPA's 2004 on-highway truck emissions regulations. The overall project involved

development of new combustion technology, new fuel system components, and new monitoring and control strategies. Initial plans for the project called for the use of control system rapid-prototyping technology, but not production automatic code. Due to the very compressed time schedule it became apparent that it would be necessary to pull forward use of production automatic code. The approach taken was to develop and automatically code specific features and then manually integrate the automatic code into the total application. Features selected for development included: smart wastegate control, independent valve actuation (IVA), IVA diagnostics, cold start fuel trimming strategy, torque limiting, and derate strategy.

Manually Integrated Automatically Coded Applications

Based on the success with developing automatically coded functions, pilot projects with a significantly higher level of complexity were undertaken. These projects were primarily for external customers and had very tight time and resource constraints. The objective in these pilots was to develop all the code associated with application-specific functionality using the model-based development process and tools.

Platform services (basic I/O, operating system, datalink, etc.) were lifted from previous hand-coded projects with limited additional hand-code development to provide services, such as unique I/O, not already developed. This general approach has become central to our long term approach to application development. Software is divided into two basic categories: platform services and application functionality. Code associated with platform services is handwritten and is intended to be common across all applications. Code associated with application functionality is automatic coded and is developed using model based process and tools.

Projects completed to date include HUMVEE engine and vehicle control, engine control for a European truck manufacturer, and a controls implementation for a refuse hauler. These projects again required that a handwritten wrapper be developed to provide the interface between the platform services and the application functionality. These projects utilized

the simulation development, rapid prototyping, and targeted automatic code capabilities provided by The MathWorks tools. They have demonstrated a dramatic reduction in both development time and resources. Data was collected from similar projects that utilized our traditional development and the results showed a reduction in time of a factor of two to four and a reduction in resources by a factor of approximately four. This was accomplished without the benefit of an infrastructure intended to support model-based software development.

Example of Manually Integrated Automatically Coded Application

HUMVEE

The project involved development of a complete engine monitoring and control system. Production automatic code was used for the engine monitoring and control functionality including: engine governing (speed and torque), timing control, boost control, EGR control, cruise control, sensor data processing, and emissions diagnostics (OBDII). Use of production automatic code made it possible for the project to meet all performance objectives, maintain an aggressive schedule, and bring the project in under budget.

Automatically generating Code for Functions (Automated Integration)

The objective of the next set of pilots was to eliminate the need to write hand-code wrappers in order to integrate automatically coded functions into an application. To accomplish this there must exist a stable defined interface that the automatic code generation can target. The approach taken was to target the interface CAT Electronics developed to facilitate integration of hand-coded functions. This interface is referred to as the CAT Package Architecture.

CAT Package Architecture Overview

Caterpillar, Inc. has implemented an in-house software architecture standard for hand coding in which an application consists of several functional components (features) with a well defined interface boundary. For instance, if we consider a diesel engine controller as an application then spark control, fuel control and idle speed control represent functional components as shown in Figure 3.

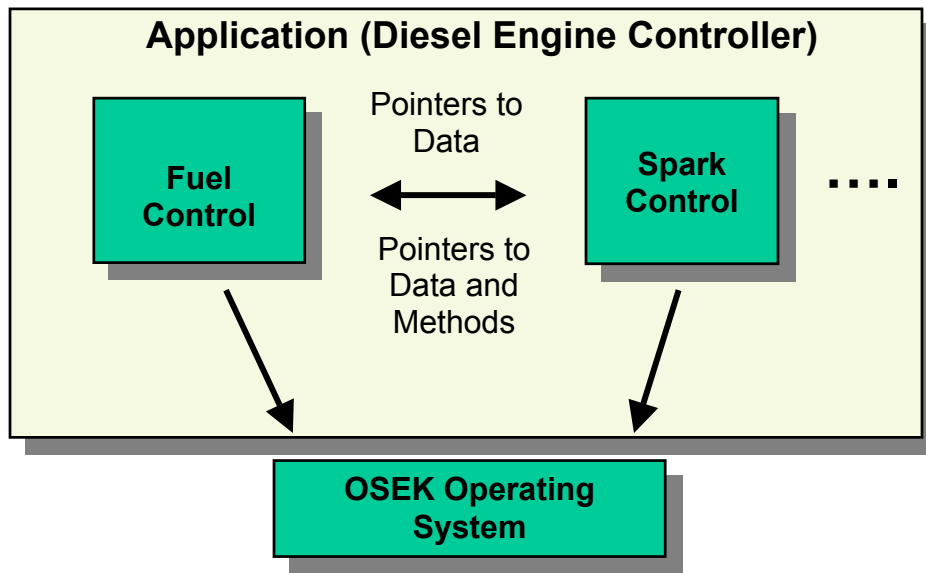


Figure 3: Functional Components

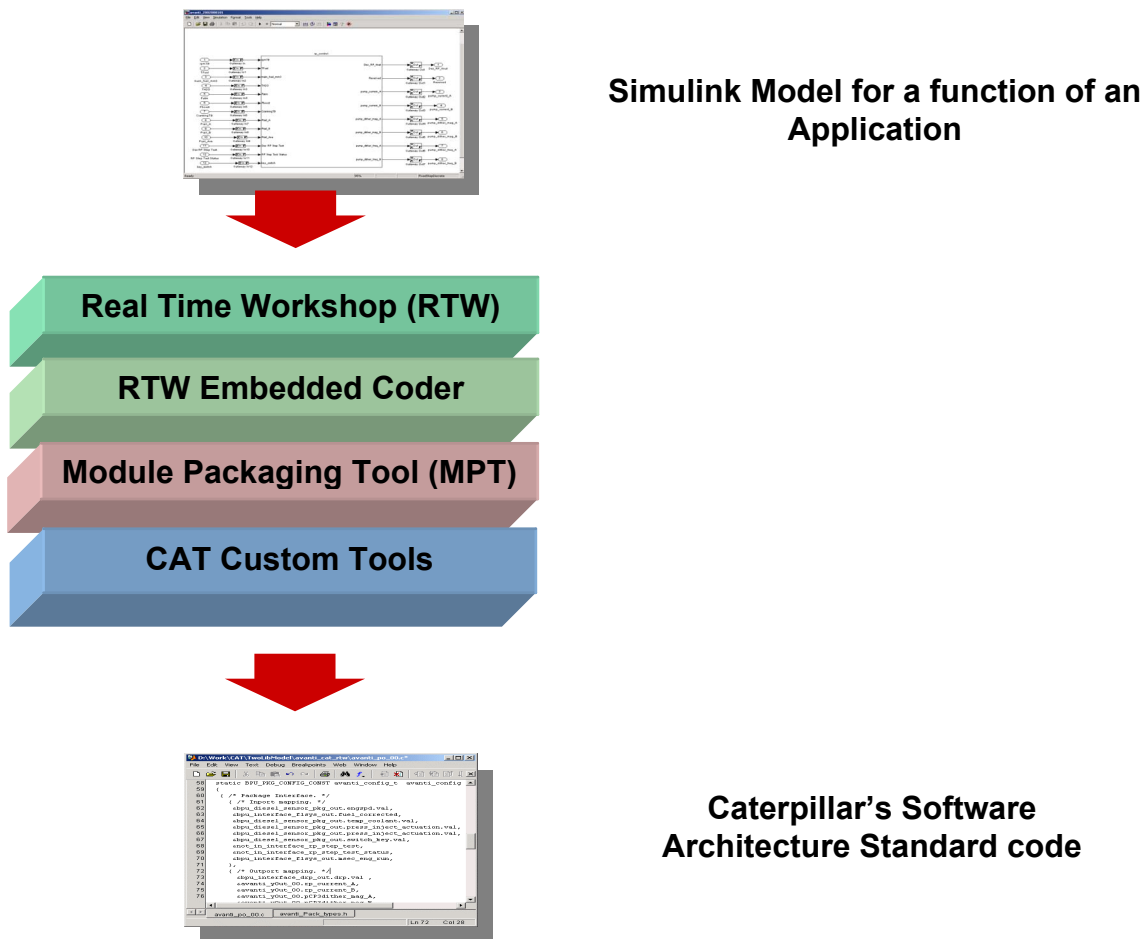


Figure 4: Code Generation General Steps

CAT's software architecture standard defines the interface between functions (or features) as well as the interface between a function and the Operating System. Each function consists of an initialization method, main methods, and configuration data. There is a separate main method for each execution rate of the function.

It was decided to tailor The MathWorks automatic code generation output to be compatible with this software architecture in order to eliminate the need for developing unique wrappers for each automatically coded feature. As a result there is no need to hand modify the generated code for Operating System, I/O, or application integration. Using this approach, some functions of an application can be automatically generated from a Simulink model while others can be hand coded and, at the end, all of them are easily integrated and coexist seamlessly. Figure 4 illustrates the general steps in the autocoding process.

The necessary tool customization has been completed and the capability utilized on several projects including:

- Compact Common Rail Fuel Control
- Motor grader All-Wheel Drive
- BHL Implement Motion Control
- Motor grader Steering

As would be expected, eliminating the need for custom wrappers has reduced the in-depth software development expertise required to implement an automatically coded function and has further reduced the development time and resources.

Example of Automated Integrated or Automatically Coded Function

Compact Common Rail (CCR) Fuel System Management

The CCR project involved developing the fuel system management control including shot selection, pump control, injection/trimming control, and injection trimming. The code was generated to meet CAT Package Architecture and was thus able to be integrated into the total engine control package without the need for a handwritten wrapper. Use of model-based development allowed direct input from fuel system experts without the need for complex communications between fuel system experts and software developers.

Autocoding Applications (automated integration)

The next step in enhancing the development environment is intended to allow a developer to develop and implement an application completely in a graphical environment. To accomplish this will require development on both the model-based development environment and the platform services portion of the development environment. On the platform side, the concept is to implement a series of platforms, each with a well defined set of platform services and well defined and stable interfaces to these services. On the model-based side, the concept is to incorporate interfaces to the platform services as blocks in the Simulink graphical-development environment.

With these two capabilities, implementation of an application completely from the graphical environment will be possible. Projects are currently underway to develop requirements and technical solutions to implement our vision and an initial implementation is targeted for early 2004.

Results from Pilots

As the Pilots were conducted, an attempt was made to evaluate the benefits being achieved (calendar time reduction and manpower reduction) and to monitor for any unintended ill effects (code quality, efficiency, and readability). No projects were conducted in parallel with competing methodologies and tools, so the approach used was to look at projects that have similar content to the pilot projects but had utilized our traditional development approach. As stated previously, the data collected indicated a reduction of man hours by a factor of from 2 to 4, depending on the project, and a reduction of calendar time by a factor of greater than 2. Because the functional testing is tightly integrated into the strategy development, both in the desk top phase and in the lab/field testing phase, along with the absence of errors in the code generator itself, the quality of the code developed has been judged as being excellent. A limited number of comparisons of execution time and memory utilizations have been conducted. Based on this admittedly limited set of evaluations and the ability to incorporate hand code for critical functions if necessary, code size and computational requirements have been judged to be comparable to our traditional hand-

code developed products. Readability of the generated code has been judged as acceptable and our policy has been to conduct all debugging at the model level and not at the code level.

Lessons Learned

Adoption Varies With Developer Background

Developers who have primarily a control system design background adopt the model-based approach with enthusiasm. Developers with primarily Computer Science backgrounds are uncomfortable with model-based development and require significantly more time and information prior to adopting the methodology.

Integration with Production Systems Requires Tailoring of Tools

Each development organization has a multitude of tools and processes that define the total development environment (version control, coding standards, build process and tools, etc.) In order to achieve maximum efficiency, it is necessary that the model-based development process and tools interface seamlessly with the larger total development environment. In many cases that means the model-based environment must be tailored to the larger environment.

Vendor Support Critical

The need to modify the model-based tools leads immediately to the need for strong vendor support. While it would have been possible for CAT to assume this responsibility, it would have been expensive and inefficient. We understand our applications and our development environment but we are not tool experts. To

acquire the necessary knowledge would have been very time and resource consuming. In addition it would likely lead to approaches that are not in step with the approaches being taken by industry, which leads to future support problems. The vendor obviously has the tool expertise needed and is more in tune to industry trends.

One Step at a Time

Model-based development is a radical departure from traditional development approaches. The steps necessary to completely implement a production automatic code capability are not available from a cookbook and the internal systems that must be accommodated are not clear at the beginning. CAT has had good success with a strategy that incrementally moves from simulation to rapid prototyping to automatically coded functions and finally to automatically coded applications. This approach has limited the complexity of the individual efforts and has facilitated the cultural and infrastructure changes that must be made.

Future Plans

Based on the successful Pilot programs, we are deploying the model-based tools and process across the department for Machine and Engine Monitoring and Control Development. Additional improvements to the tools, the development process, and to the infrastructure are planned to further reduce the time and effort required for application development.

References

1. www.cat.com
2. www.mathworks.com