

Large Scale Modeling and Simulation of Propulsion Systems

Siva Nadarajah
The MathWorks

Copyright © 2007 The MathWorks, Inc.

Abstract

Current trends in the automotive, aerospace and other industries are resulting in the development of exceptionally large system-level models of control law or physical system behaviors. This is especially true in the propulsion systems algorithm and software development areas. The MathWorks has been actively engaged with a number of engineering groups faced with industry demands of increased software feature content with higher complexity and a shorter turn-around time.

This paper discusses a roadblock encountered on a typical propulsion system software simulation due to its sheer size. It also discusses challenges, results, and lessons learned in solving the above problem from both a tool technology and engineering perspective.

Introduction

Using a traditional, hand-coded software process caused long, time-consuming, iteration cycles and imposed severe limits on the number of iterations that could be performed to develop a system. This in turn required control system designers to make final design decisions without adequate information. A newer, more efficient process is to use Model-Based Design with Simulink® for system simulation and embedded target automatic code generation.

With Model-Based Design, control models are graphically defined and tested against plant models using simulation.

This paper describes the inherent difficulties in establishing large scale vehicle propulsion system simulation capability and how they can be solved. The MathWorks Simulink based propulsion system under development (plant and controller) model grew too large (about 650,000 Simulink blocks) and exceeded the usable memory on a desktop PC with 2 GB of RAM for normal simulation. The desktop PC system simulation capability which had been lost is crucial in the software development cycle to identify system defects early at the desk rather than later on during vehicle tests.

This paper using a generic real-life example describes the key steps used to achieve the solution:

1. Initial investigation
2. Solution implementation
3. Results
4. Future plans
5. Conclusion

Initial Investigation

Using Simulink for a normal simulation of the entire propulsion system became impossible due to the system's sheer size. There were approximately 650,000 Simulink blocks for the propulsion system, which not only exhausted all the available memory on a Windows 2000, 3 GHz Pentium desktop PC with 2 GB of RAM, but it brought it to a screeching halt. The following options were evaluated for a scalable, flexible system simulation solution:

1. Using a UNIX or Linux system for simulation

2. Using 64-bit Windows for simulation (only Beta version was available)
3. Simulation using the new Model Reference capability in Simulink
4. Other third-party distributed simulation solutions

Ultimately, due to existing software industry development infrastructure constraints, the MathWorks new Model Reference capability using the existing desktop PC based development environment was selected. The version of The MathWorks tools used for this endeavor was R14sp2.

System Model Architectural Overview

The typical top-level propulsion system model contains major *subsystems* such as engine controller, transmission controller, etc. Each major subsystem is broken down into *features* at the top (first) level. Examples of *features* are spark control, fuel control, airflow estimation, etc. on the engine subsystem as shown in Figure 1. At the *feature* level, strict interface rules and naming conventions are followed.

Overview of Simulink Model Reference

The Model Reference capability in Simulink

enables you to include models of other models as blocks. You create references to other models by creating instances of model blocks in a parent model. Each instance of a model block in a parent model represents a reference to another model called a referenced model. A model block displays inputs and outputs corresponding to the root-level inputs and outputs of the model it references, enabling you to incorporate the referenced model into the block diagram of the parent model.

During simulation, Simulink invokes an automatically generated S-function DLL, called the referenced model's simulation target, to compute the model block's outputs as needed. If the simulation target does not exist at the beginning of a simulation, Simulink generates it from the referenced model. If the simulation target does exist, Simulink checks whether the referenced model has changed since the target was last generated. If so, Simulink regenerates the target to stay current with the referenced model, based on user-specified settings.

A referenced model can itself reference other models. The top-most model in a hierarchy of model references is called the root model.

Like Simulink subsystems, model referencing allows you to organize large models hierarchically, with model blocks representing major subsystems. However, model referencing has significant advantages over subsystems in many applications:

- **Modular development:** You can develop the referenced model independently from the models in which it is used.
- **Inclusion by reference:** You can reference a model multiple times in another model without having to make redundant copies, and multiple models can reference the same model.
- **Incremental loading:** The referenced model is not loaded until it is needed, speeding up model loading.
- **Incremental code generation**

For referenced models, Simulink creates C code and binaries (DLL) to be used in simulations for computing the outputs of model blocks. If the binaries are up-to-date, that is, the binaries are not older than the

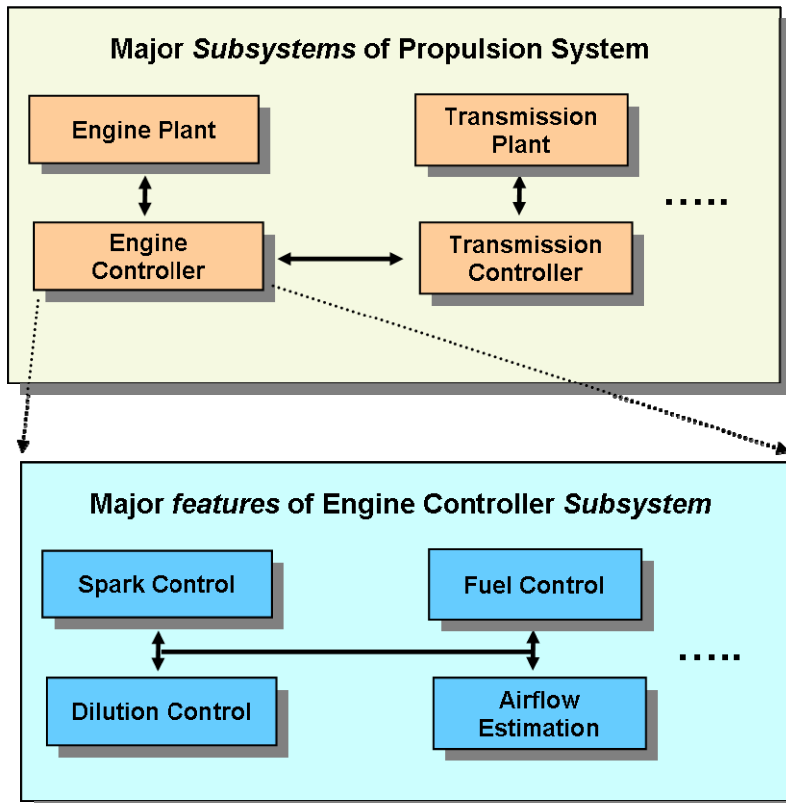


Figure 1: Propulsion System Model Architecture

models from which they were generated, no code generation occurs when models that reference them are simulated or compiled.

Solution Implementation

It became evident that on each propulsion subsystem, *features* which readily follow strict interface and naming rules are prime candidates for model reference components.

Model Component	Approximate Number of blocks
Complete System	650,000
Top (root level)	115,000
Dilution	8,000
Spark	90,000
Diagnostics	35,000
Fuel Control	250,000
Airflow Estimation	175,000

Figure 2: Major features of PTSystem

Empirical results show that these *features* as Simulink model reference blocks with simulation by execution of the generated code (DLL) of the referenced models, dramatically reduced memory footprint during execution. Figure 2 shows five major *features* of the Engine Controller subsystem of the propulsion system, (PTSystem) and their sizes in blocks.

Challenges Faced — Reference Model Constraints

1. Only one function-call trigger can cross the *feature* (or Model Reference) boundary in R14sp2, indirectly by encapsulating a model block within a function-call subsystem. To facilitate passing multiple function-call triggers, custom `function_encoder` and `function_decoder` blocks were developed and used on the referenced model periphery. The example in Figures 3, 4, and 5 illustrates the details.

In Figure 3, the two input function calls `Trig_12p5ms` and `Trig_100ms` to the

feature are passed into a custom Function Call Encoder block, which produces a single function-call output and a corresponding number that indicates which of the two triggers will be invoked.

Figure 4 illustrates how, using R14sp2 in this project, a function-call trigger can cross the model reference boundary. The function-call subsystem submodel encapsulates the *feature* model block. Note that as a reference, R14sp3 released in September 2005, introduced a direct function call model capability which eliminated the need to wrap the model block in a function-call subsystem.

The model block in turn references the *feature* model, which contains a custom function-call decoder block that translates the function code input number to individual function-call output, as shown in Figure 5.

2. Since the PTSystem architecture calls for *features* to be entirely function-call driven, referenced *feature* models have to inherit their sample time. A referenced model inherits its sample time if all the following conditions are true:
 - a. None of its blocks specify sample times (other than inherited and constant).
 - b. It does not have any continuous states.
 - c. It does not contain any blocks that use absolute time.

The *features* have to be sample time independent. This can be verified by selecting the Ensure sample time independent option of the Periodic sample time constraint on the Solver configuration parameters dialog. By doing so, the only sample times allowed in the referenced model are Constant Sample Time (inf) and Inherited Sample Time (-1).

3. Global Goto/From blocks cannot cross the *feature* (Model Reference) boundary. According to the example PTSystem architecture, all Goto/From blocks have the visibility of local, which eliminated the need for special handling.

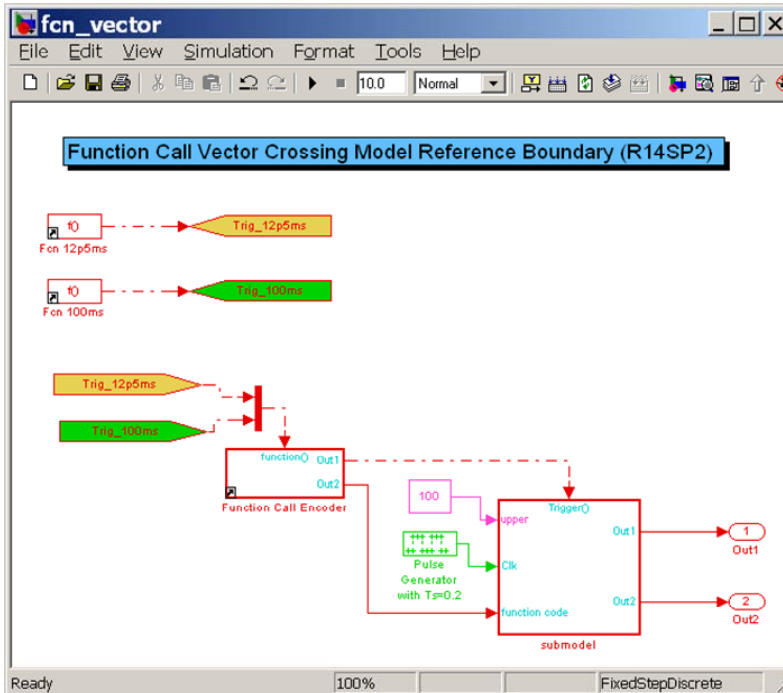


Figure 3: Passing a Function-call vector to a Model Reference component

4. In R14sp2 global data stores can be used across the *feature* (Model Reference) boundary using a corresponding Simulink.Signal object in the base workspace. Example PTSystem architecture does not allow data stores crossing *feature* boundary, and the data stores used are local to each *feature*, eliminating the need to use Simulink.Signal objects.
5. The number of model instances of the referenced model is set to 1 because there were machine-parented global data and exported graphical functions in Stateflow® charts of *features*. The above mentioned items had to be local-scoped to a *feature* and prohibited from crossing the model reference boundary, even with this single instance setting.

System Model Simulation Workflow

1. The user opens the propulsion system model.
2. Then the user opens the Simulation Assistant GUI that lists all the *features* of the chosen subsystem, as shown Figure 6. The right pane lists all *features* of the selected PTSystem/Engine Controls subsystem that are model reference blocks. The left pane lists all *features* of

the selected subsystem that are library blocks. The Make Model Ref button allows the user to replace a *feature* that is a library block with an equivalent model reference block. On the other hand, the Make Library button replaces a model reference block with an equivalent library block.

The user typically selects all the *features* from the left library pane except the ones the user is currently working on, and presses the Make Model Ref button to convert to model references. This facilitates “white box” access and visibility to the *features* the user is currently working on and “gray box” access to the rest of the model reference *features*.

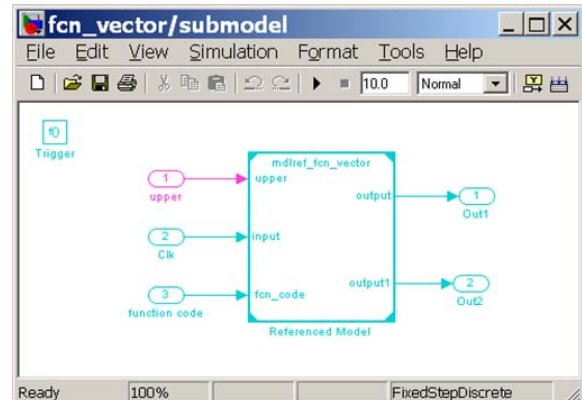


Figure 4: Model block encapsulated in function-call subsystem

3. The user presses the Start button on the PTSystem model to simulate it. Simulink will determine which model reference component needs to be rebuilt to produce the simulation executable image (that is, the DLL) and builds it automatically.

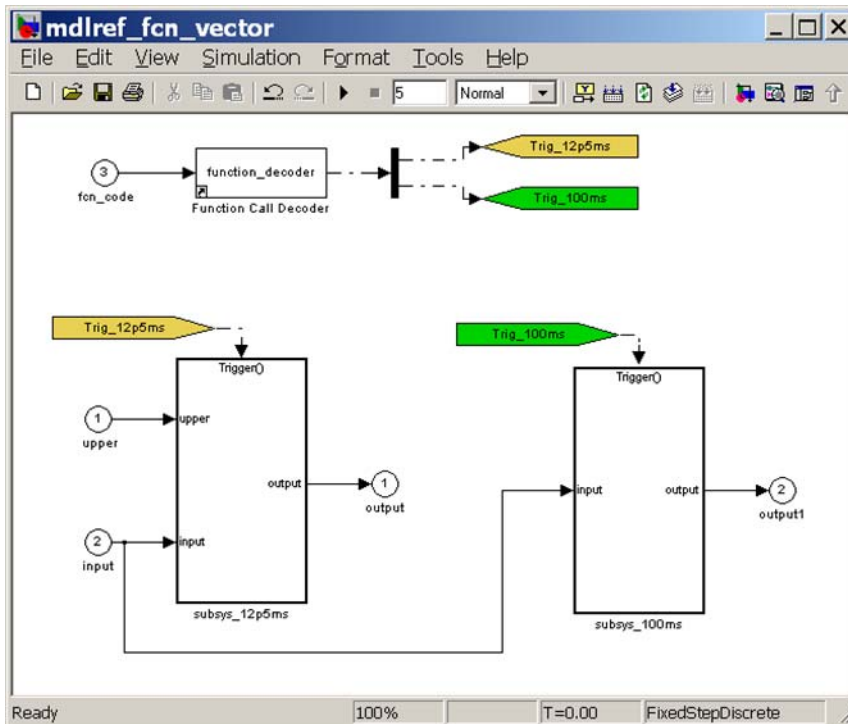


Figure 5: Referenced model - passing a function-call vector

The user also has the option of performing simulation target build on the Simulation Assistant dialog by selecting *features* on the Model Reference Features pane and pressing the Build Model Ref. Components button.

Results

Using the Model Reference approach, the total code generation and executable build time for the example PTSystem model of about 650,000 Simulink blocks with major five *features* shown above in Figure 2 as model references is approximately 40 minutes on a 3 GHz Pentium desktop PC with 2 GB of memory with Windows2000. Once the build is complete, the simulation startup time is less than 2 minutes, meaning the time elapsed from when the user presses the Simulation Start button in Simulink, to simulation starting is less than 2 minutes. This start up time can be reduced significantly by converting the remaining 30 *features* of the

Engine Controller subsystem to model references in addition to the five major *features* stated above.

Figure 7 shows the memory usage[†], simulation target executable image (DLL) build time, and the number of blocks of five major *features* of PTSystem. For example the largest *feature*, Fuel_Control with approximately 250,000 blocks, consumed 850 seconds of build time and 800 MB of RAM for the simulation target executable image build.

Figure 8 shows memory usage[†] for opening the PTSystem model and performing a simulation. Note that the memory consumed for PTSystem simulation shown in the figure is when there are no changes to the referenced models since the last build and therefore there was no need to rebuild simulation executable targets of referenced

changes to the referenced models since the last build and therefore there was no need to rebuild simulation executable targets of referenced

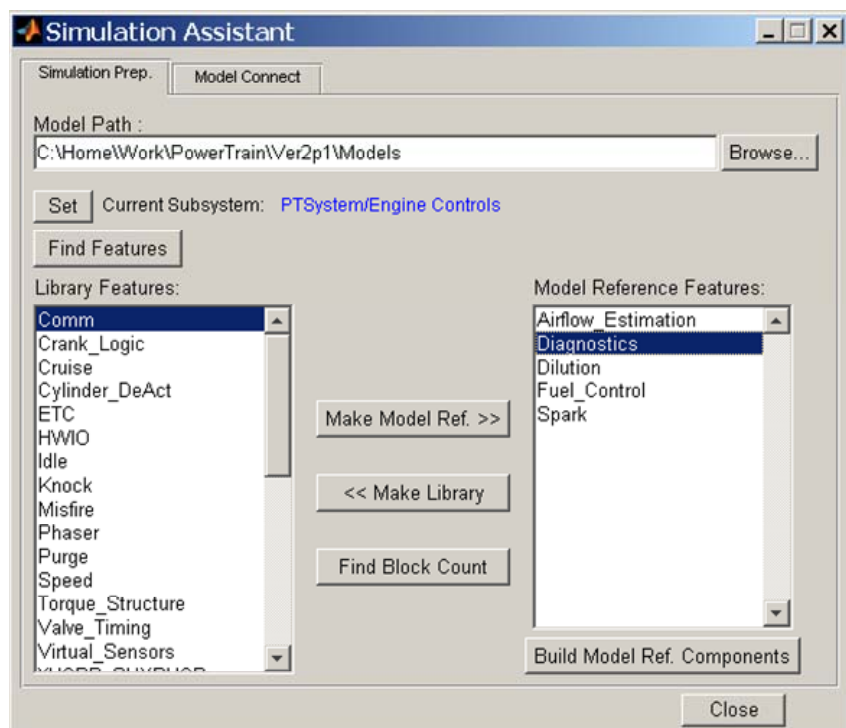


Figure 6: GUI that enables Library to Model Reference block conversion and vice versa

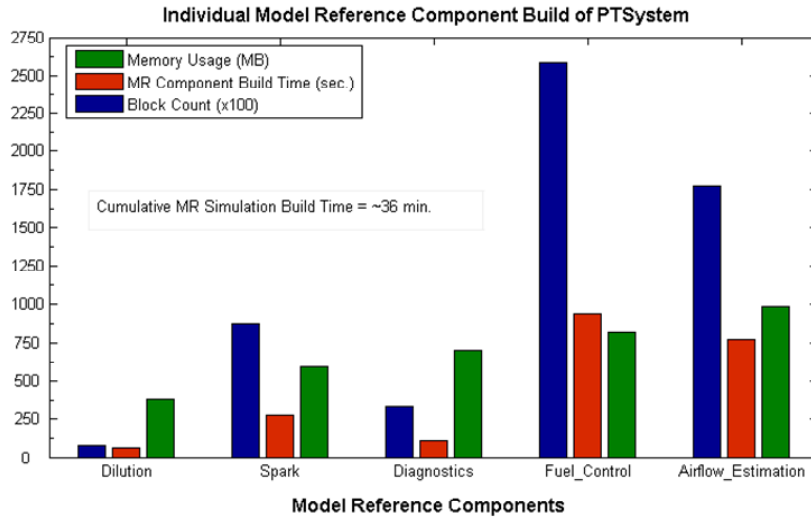


Figure 7: Memory usage, executable build time of referenced model features and their block count

model features. In such a situation, the entire system model simulation was performed on a desktop PC consuming less than 1 GB of RAM.

PTSystem simulation is operational using Simulink Model Reference capability. As noted before, features that are model referenced provide only gray box visibility inside; therefore a user needing white box access to the two largest features simultaneously (for instance Airflow_Estimation and Fuel_Control in PTSystem) may not be possible. This is because these two largest features have to be converted to normal library blocks for white box visibility, which may result in out-of-usable-memory conditions during simulation.

Future Plans

1. The PTSystem with approximately 650,000 blocks took 40 minutes to build five major model referenced features, even though this happens only when all the model referenced features have changed since the last

build. The ability to save the executable images and other MathWorks folders pertaining to model reference simulation target build in a configuration management system is being explored. If this capability is deployable, a user will be able to check out the system model along with referenced model executable images and save the one-time build time of 40 minutes.

2. Large features such as Fuel_Control and Airflow_Estimation are being broken down to multiple smaller features to improve flexibility.
3. A process is being developed for propulsion programs, using this model referencing scheme of simulation for large scale Simulink models.

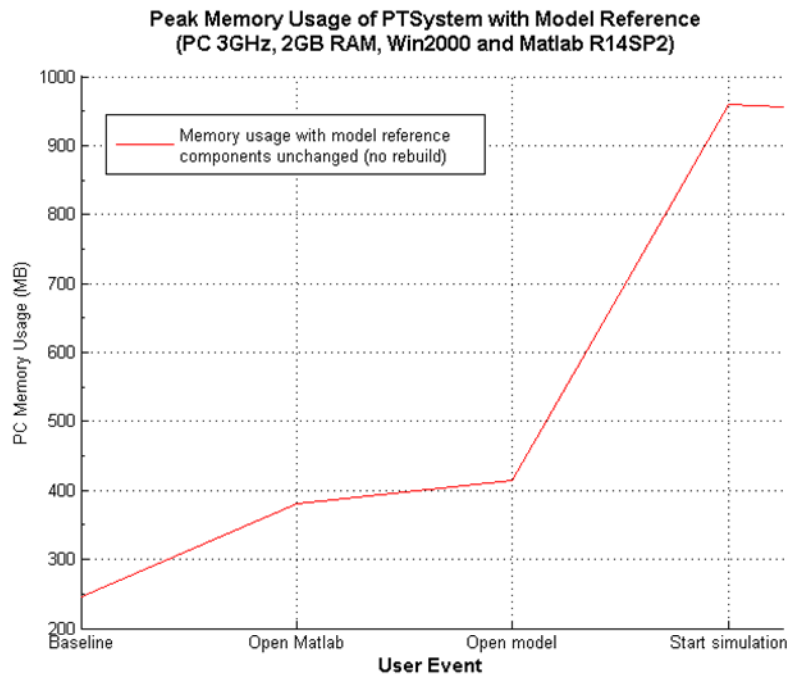


Figure 8: Memory usage of PTSysystem load and simulation

Conclusion

The Simulink Model Reference capability has revived the lost system model simulation capability for large models and is a great solution when the software development environment is a desktop PC with Microsoft 32-

bit Windows. In the long term, when faster PCs and a production 64-bit Windows environment are available, using a normal simulation (without referenced models) in Simulink will be desirable due to the fact it provides white box visibility to all *features* and simplicity of workflow.

[†] Memory usage on Windows is determined visually by tracking what is reported by the Performance dialog of Windows Task Manager.

*The MathWorks, Inc. retains all copyrights in the figures and excerpts of code provided in this article. These figures and excerpts of code are used with permission from The MathWorks, Inc. All rights reserved.

©1994-2007 by The MathWorks, Inc.

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.
