

Strategy for Successful Enterprise-Wide Modeling and Simulation with COTS Software

Rob Aberg* and Stacey Gage†
The MathWorks, Inc., Natick, MA, 01760

This paper discusses a strategy enabling successful and productive enterprise-wide modeling and simulation using commercial off-the-shelf (COTS) software packages. This paper will focus on the *definition of enterprise-wide modeling, a summary of enterprise-wide modeling challenges, the enterprise-wide deployment of COTS block diagram software package based modeling, and the efficient simulation of enterprise-sized models, including speed tips and tricks. The reasons for the development of this strategy will also be discussed. Successful enterprise-wide modeling and simulation implementation with COTS software is based on experience, extensive customer feedback, and consulting project debriefs.*

I. Introduction

THIS paper will define enterprise-wide modeling and discuss some of the signs of enterprise-wide modeling, such as using a large or enterprise-sized model which one individual does not know all the details of that model. These signs allow individuals to determine if they are working within an enterprise-wide modeling situation. Within an enterprise-wide modeling situation, there are a number of challenges to productivity and efficiency. One essential aspect to resolving these challenges is the careful deployment of COTS software and internal tools within a company. Additionally other key processes must be managed and integrated with COTS software to surmount the challenges. Some of these processes include the use of validated design configurations, the traceability of requirements, and the use of modeling standards. This paper will discuss the processes and actions that are necessary for a company to overcome challenges from enterprise-wide modeling.

To provide a specific example of a COTS software package for enterprise-wide modeling and simulation, this paper will examine deploying MATLAB® and Simulink® in an enterprise-wide environment. The example provides suggestions for creating a strong foundation for an enterprise-wide environment, as well as examples of how the key processes can be managed and integrated.

As a final topic, this paper will discuss efficient simulation of enterprise-sized models, including speed tips and tricks. The focal points for efficient simulation include the model structure, the model execution settings and model environment, and the prospective use of multiple CPUs for model simulation. The goal in enterprise-sized models is to optimize the model structure while retaining the clarity of the design. Some optimized modeling techniques for faster simulations will be discussed, including turning off unnecessary logging, disabling or decimating plotting, using variable step solvers, and finding and removing costly calculations. The use of code generation to accelerate execution and its trade-offs in improving simulation speed will be discussed, as well as general topics, such as batch strategies, multiple CPUs, and distributed systems.

II. Enterprise-wide Modeling Defined

To successfully and productively use COTS software packages for enterprise-wide modeling and simulation, one should be able to determine if they are in an enterprise-wide modeling situation. Signs of enterprise-wide modeling within an organization include a large model that is too big for one person to know all the details, team members sharing a top-level model that includes multiple components, and many teams in the organization sharing reusable components. Details that quantify these three signs follow.

* Simulink and Real-Time Workshop Engine Development Manager, 3 Apple Hill Drive.

† Simulink Applications Developer, 3 Apple Hill Drive, and AIAA Member.

A. A large model that is too big for one person to know all the details

One way of determining how large models are is by counting the blocks in a block diagram model. How does one define a large model? A large model can be defined as one model referencing more than approximately 10,000 blocks or as one model linking to more than approximately 10 custom libraries.

In Simulink, blocks within a model can be counted with the command *sldiagnostics* at the MATLAB command line. In the example below, a model named 'intercept.mdl' is used in the *sldiagnostics* command with the 'countblocks' option to determine the number of blocks in the model.

```
>>[txt,dst] = sldiagnostics('intercept','countblocks')
```

```
intercept Total blocks: 1166
          Abs :    11
          ActionPort :    6
          Assertion :    1
          BusCreator :    5
          BusSelector :   13
          ...
```

Another *sldiagnostics* option that is useful for packaging models is the 'libs' option which lists all the libraries used in the model. For more information on the *sldiagnostics* command and additional options for the *sldiagnostics* command, at the MATLAB command line type:

```
>> help sldiagnostics
```

B. Team members sharing a top-level model which contains multiple components

Signs that team members are sharing a top-level model with multiple components are that more than about 5 engineers are collaborating on one design or that more than about 15 engineers are using the same model.

C. Many teams in the organization sharing reusable components

When a number of teams share a model, there are different problems to solve than those encountered in the "many individuals" or many team members' case. The following signs indicate that multiple teams in an organization are sharing components within a model: more than approximately 20 active configuration-controlled components or models, multiple active releases of the large design, or a dedicated "tools team" for supporting processes using COTS tool, such as MATLAB and Simulink.

Careful deployment of COTS tools is essential in an enterprise-wide modeling situation. Care should be taken to provide support, maintenance, and version compatibility for *your* internal support tools created to work with COTS tools. Having versioned internal releases will help to avoid chaos among teams and individuals within the teams, while having individual team members trained both in the COTS tool and internal support tools enhances process compliance, and tool effectiveness.

As a guide, enterprise-level challenges will probably be encountered if any of the above conditions apply. These challenges become more acute as the size of model or number of individuals and teams increase.

III. Enterprise-wide Modeling Challenges

Additional tooling is needed to more fully automate enterprise-wide modeling when using COTS tools. The major challenges to productivity and efficiency within an enterprise-wide modeling situation are large model handling, team-based modeling, and intricate, changeable configurations with a model.

A. Large model handling

The main concerns with large model handling within a COTS environment are componentized modeling, speed, and memory requirements. Componentized modeling aids in the sharing of models across teams and team members for collaboration on designs. Speed and memory requirements apply to loading models, saving models, editing models, updating diagram of models, and code generation from models.

Features to look for in a COTS tool are speed, stability, efficient use of memory, together with component-based editing, simulation, and code generation for scalability of models.

B. Team-based modeling

The main concerns with team-based modeling within a COTS environment are integration of components at nested levels within the model, manipulation of models at a high level to support different activities, such as requirements, architecture, control design, software design, documentation, deployment of customized tooling, deployment of a modeling standard, and automated compliance checking for a modeling standard. Integration of components at nested levels within the model is necessary for collaboration among team members and across teams. Being able to manipulate models at a high level to support different activities reduces rework and errors that can occur from having multiple task-specific models. Deployment of customized tooling allows the COTS tool to be extended or manipulated to meet the specific needs of the enterprise. The deployment of a modeling standard provides many benefits. The most obvious is that anyone in an organization can understand the model design. Lastly, having an automated compliance checking for a modeling standard confirms that models conform to a modeling standard and therefore that the models can be inspected or manipulated by automatic scripts.

Additional items to look for in a COTS tool are IP-protected or "black box" model deployment; automated testing for verification and validation with results archiving; incremental code generation, enabling code to be generated only for changed components; and production-quality document generation.

C. Intricate, changeable configurations

The main concerns with having intricate, changeable configurations within a COTS environment are managing all design versions of components and custom utilities, maintaining model consistency amid design changes, ensuring that a valid set of components is deployed and loaded with the models, managing design data (e.g., parameters) across teams, avoiding model corruption, and ensuring the availability of component test suites. Most of these concerns focus on the desire to maintain control over model configurations to know what is being simulated with the model. The last concern, component test suites availability, implies that the individual components can be validated at any time to find potential problems earlier in the design process.

Additional items to look for in a COTS tool are interfaces to the "best of breed" of tools for configuration management, requirements, testing, and other tasks.

IV. Enterprise-wide Deployment Strategies

To deploy enterprise-wide models successfully, a strong foundation is required. A strong foundation can be established by controlling the software environment, using the model to communicate, architecting models using a hierarchy of stand alone design components, automating the documentation of the design validation and release procedure, and developing a modeling standard for the COTS tools.

A. Environment control is key to efficient collaboration

Controlling the software environment not only makes it easy for a new team member to understand the project by providing obvious entry points, such as a master GUI, an open model, or a message displayed with the first command needed; it also reduces configuration problems. Two items which will support environmental control are path and setup, and network shared areas.

To create a consistent working environment, a standardized path and initialization should occur on the startup of the COTS tool. Using MATLAB, MATLABPATH can be used to generate a standard path from a standardized startup.m file. Additional initialization can also be handled in the startup.m file. When you are working on multiple versions, different startup.m files for each deployed version of your environment can be created.

Use the shared network area to deploy custom utilities and tools based on the COTS tool, including the enterprise-wide model and documentation. As a best practice, "locked down" disk areas should be used for released versions. From the network area, tool builders can copy down a reference master, and then check in changes. Tool users can then refer to reference master from the network area, causing fewer configuration problems. Configuration management must be easy to perform to ensure buy-in from the tool builders and tool users.

As always, when chaos occurs, adapt the process.

B. Keep "one truth": Use models to communicate

Using models to communicate, also known as Model-Based Design, alleviates many pain points that engineers usually experience under the traditional engineering process. These include ambiguous documentation, unavailable physical prototypes, time-consuming and error-prone hand-written code, and testing the systems at the end of the process, which can result in problems being discovered late in the engineering process, when they are most expensive to fix. Typically, the traditional development process uses text documents to communicate among groups. This can lead to separation between tasks and walls being built up between groups.

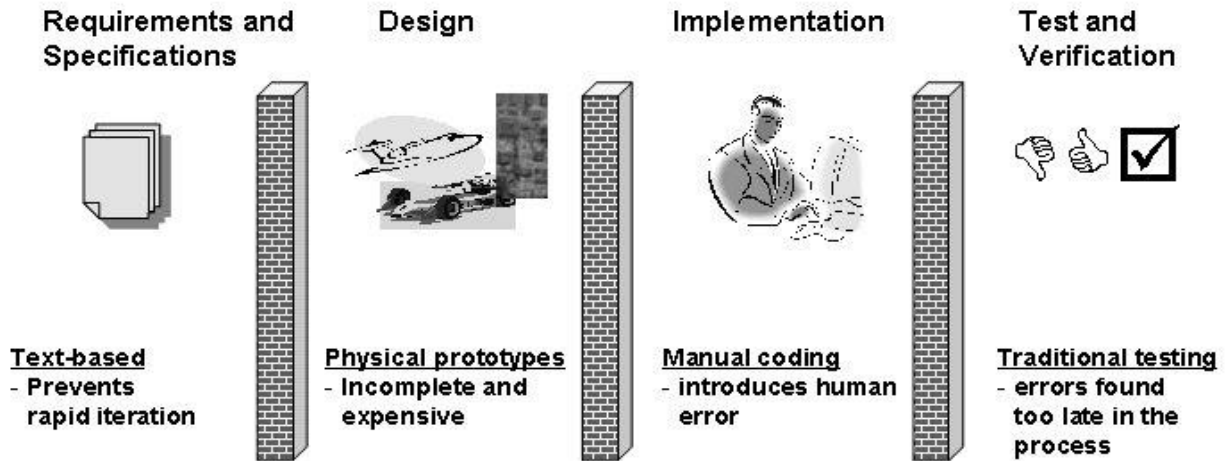


Figure 1. Problems with Traditional Development

Model-Based Design works excellently within COTS tools. The models are easy to share among team members and teams. The models are unambiguous and allow you to maintain one “true” representation of your design and implementation. The model increases in complexity as it moves from research and development requirements, through design, implementation and testing, into production with automatically generated code, and into logistics support, where it is used for diagnostics.

In Model-Based Design, models serve as executable specifications, replacing ambiguous text documents. Dependence on physical prototypes can be significantly reduced by using the model to simulate the system behavior. The model becomes a software prototype, and systematic “what-if” analysis can be performed to identify design flaws, or even to suggest design improvements. Using the model with automatic code generation can minimize coding errors, since the generated code has strong dependence and relation to the design, simplifying debugging.

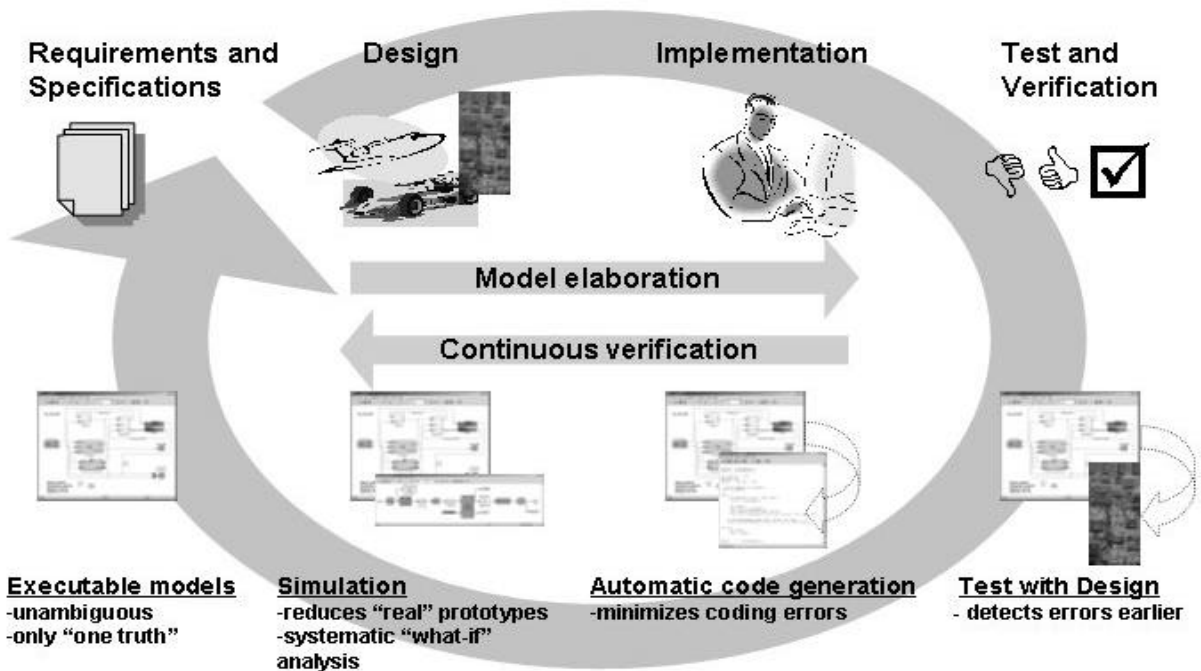


Figure 2. Advantages of Model-Based Design.

Throughout the engineering process, quality is ensured by integrating tests into the model. Each model has a test suite of check cases with baseline results for each released version of a model. This continuous verification and simulation helps identify errors early, when they are easier and less expensive to fix. The result is faster development and better designs.

The model is not only used to communicate; it is also used to generate reports from data extracted from the model. There is no need to integrate multiple task-specific models for a given project or to create separate models for linearizing, debugging, logging, and so on.

Noting all the benefits, it is clear to see that Model-Based Design is closely aligned with the Six Sigma goal of defect reduction and elimination. Using the model “saves time in product development by eliminating the need for a physical mock-up, and allows for early detection of interferences between components, and a number of trade-off or optimization studies.”¹ With Model-Based Design one can pay “attention to the early detection, preferably prevention of unnecessary design changes will have dramatic effect on cycle time and cost.”²

C. Architect models using a hierarchy of stand alone design components

Design components should be architected in the modeling environment to stand alone during development through the use of a component test harness driving the component through its system-level interface. The interfaces can be hierarchical in that base components are integrated into a subassembly, and so forth, up to the full system level model. This design decomposition and test harnessing permits automated testing of the components against requirements and collection of test coverage data at the component level. Later, when the components are integrated at the next highest level, problems are more likely to be limited to interface and system level issues. In contrast, building a model out of components that cannot be individually tested means that a component-level problem occurs is much more difficult to isolate since a system with feedback will propagate the behavior throughout the model. Additionally, working with a large model can be more difficult than working with a single component due to the extra complexity involved.

To support this concept, some modeling environments offer a hierarchical component that stands alone. In Simulink 6.0, this is the Model block. The Model block allows one model to reference another model that contains a complete execution specification and is able to run independent of any other model. This is different from a Simulink Subsystem block in that the Subsystem block does not specify a complete execution environment, so when it is used in a model it acts as a template where each instance of the Subsystem in a model inherits attributes such as data type and sample time from the parent model. Because the Model block does not inherit any attributes that change its behavior, it provides a “strong” boundary to ensure that a design component will function in parent model references exactly as it does in the test harness.

D. Create engineering reports directly from models

COTS tools can be used to eliminate manual creation of design documents by directly creating the documents from the model. Common report types include interface description documents, compliance reports, violations reports, simulation results summaries, model coverage during testing, and design study results, e.g., sensor/actuator tolerances via Monte Carlo testing.

From Simulink, the Simulink Report Generator can be used to generate these multiple reports. Additionally, written documentation can be attached to a model via a doc block.

E. Automate repetitive tasks with scripts

Whenever possible, scripts should be used to lock in best practices. With scripts, there is no need to keep reinventing a best practice, and team members have access to each others' M-files. Automation via scripts gives consistent results, increases efficiency, and ensures that every team member uses the same process consistently. Candidates for automation include common tasks, such run analysis; tool validation test suites; code generation; and a "Drawing Room Manual," a checker for models that enables engineers to review standards-based diagrams quickly.

F. Develop your own modeling standard

To ensure that model diagrams are easily understood, a modeling standard for COTS tools should be developed to meet the needs of the organization and the project. The modeling standard should parallel how engineering drawings follow a "Drawing Room Standard." Using a modeling standard allows anyone in the organization to easily understand a design. It also helps ensure that designs meet organizational requirements, enables the use of scripts to automatically check diagrams, and ensures compliance with additional constraints. Being able to

understand model diagrams from other teams is important in enterprise-wide modeling, since each component is usually part of a larger design. Usually, one team's "top-level" model is another team's component.

One of the most important benefits of using modeling standards is the improved likelihood that diagrams can be inspected or manipulated by automatic scripts.

An example modeling standard is usually a good starting point for creating a customized modeling standard. One existing modeling standard or style guide is the MathWorks Automotive Advisory Board (MAAB) Style Guide³. The goal of the style guide was to make clear, readable designs that generate production-ready code using MathWorks product Release 12, circa 2000. Most customized modeling standard have similar goals but not necessarily identical goals.

V. Efficient Simulation of Large Models

There are several ways to increase the simulation efficiency of large models. Areas to focus on to enhance simulation include model structure, model management and data design, the optimization of modeling techniques, and the use of code generation to accelerate simulation.

A. Structuring models

One of the key ideas in structuring the models is to retain the clarity of the design. In order to retain clarity, the model must be created with some things in mind. These are using the correct type of diagram for each task within the large model, planning the architecture of the large models to last, specifying interfaces within the large model, and defining limits graphically.

Within COTS tools, several types of diagrams can be used to construct a model. The types of COTS diagrams that are usually available are block diagrams, state machines, truth tables, scripting blocks, and special block diagrams. The block diagrams, such as Simulink, are particularly good for modeling feedback paths and dependencies, while state machines, like those in Stateflow[®], are excellent for control transitions, mode logic and fault detection⁴. The truth tables, like those in Stateflow, are often used to express complete stateless systems. Scripting blocks, such as the Embedded MATLAB block in Simulink, are useful for formulas that are more easily and clearly described with code than with a block diagram. Special block diagrams are available in COTS tools like Aerospace Blockset, which provides pre-built aerospace components, and SimMechanics, which includes tools for modeling mechanical systems.

After deciding what type of diagram works best to construct portions of the large model, one should consider how the model is architected, since models with planned architecture last through future designs. The model architecture should make it easy to update and fit within a configuration management system and the model should be able to easily support future design changes.

A natural mechanism for utilities such as custom filters are libraries. Another common COTS mechanism for storing utilities or components is based on models such as the Model Reference in Simulink. With Model Reference, each component is an actual model which simulates, contains model specific parameters, defines input and output interfaces, and provides incremental code generation. Since libraries and Model Reference are file-based and contain only one utility or component per file, they are well suited to configuration management tools, which are also file-based.

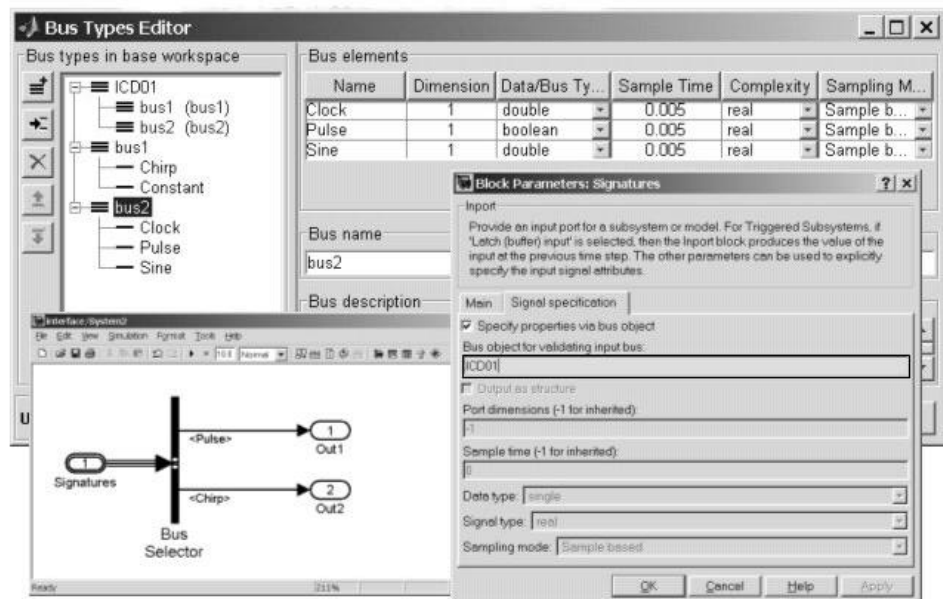


Figure 3. Interface Specification with Bus Objects.

To create a model that lasts, one should use "roll-forward" model architecture to ensure that the model will work for future design modifications. An example where future design modifications need to be considered would be in the area of observables. Today observables include radar, IR, and acoustic, but what about tomorrow's observables? How will those new observables be integrated into the model? One way to achieve this flexibility for the future is to use buses and vectors to pass signals within the model.

Specifying interfaces within the model not only includes defining what signals should be passed to and from components, but also signal data types and sample times. Input and output interface specification reduces headaches. Locking down the interface reduces inadvertent misuse. Input and output signal properties should be specified. In Simulink, this is done either by Bus Objects for bus signals, by setting properties on the individual input ports, or by using the signal specification block. Using Bus Objects to define signal, signal data types and signal properties is shown in Fig. 3. In this figure, the input port, signatures, has a Bus Object, ICD01, defined as a bus containing two other buses that contain signals with data types and sample times specified. Another way to specify the interfaces is to use a data-driven interface allowing design "layering." Again, the interface is locked down, but in this case we are considering data type and sample times. Parameterized design layers provide instant switching. For example, using an number of variables for sample times and data types within the blocks of the model and the model would allow switching between continuous-time simulation in double-precision, multi-rate discrete time simulation in double-precision, and multi-rate discrete time simulation in target specific data types, all by changing a few parameter values. Figure 4 provides an example of this. Variable sample time, T_s , and variable output data type, `profFloatType`, allow the sample time and data type for the input port, fuel, to be easily changed from the command line.

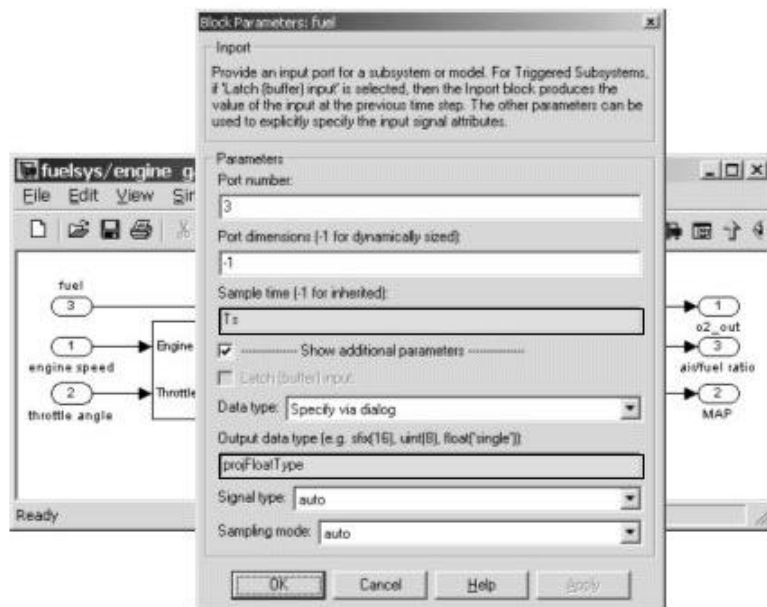


Figure 4. Data-Driven Interface

Another way to make large models clear is to graphically define limits with assertions. Simulink provides a number of assertion blocks. In addition to showing the model's design intent graphically, assertion blocks check run-time limits and can be set on and off locally, globally, or for code generation.

B. Managing models and data design

In addition to structuring the models to retain the clarity of the design, large models and their data need to be well managed so that team members can consistently use the correct configuration of the model. Componentized models and data dictionaries are important in the management of large models.

The recommended strategy for managing models is to componentize designs at the file level. The design components are then stored in separate files, with one subsystem per file in a library, or in model file such as model reference. This strategy allows configuration management tools to work naturally, since those tools are file- and folder-based. The strategy also allows configuration management projects to contain *any* type of file. Once the model has been componentized, model configuration control ensures consistency in the composition of the large model, since the model configuration is created from a set of versions of files. A model configuration can be saved in separate file under version management. To ensure consistency, a utility can be created to check model versions prior to code generation, simulation, etc. In Simulink, the versions of library files within the large model can be obtained with the following commands at the MATLAB command line. In the example below, a model named 'intercept_components' is used in the `sldiagnostics` command with the 'libs' option to determine a list of libraries in the model. The tenth library in the list is then examined. First, the name and file location is determined, and then its version number. These commands could be part of the utility script for verifying all file versions within the model configuration.

```

>> [t,s] = sldiagnostics('intercept_components','libs');
>> s(10).libs
    iv_airframe
>> which(s(10).libs)
    D:\work\demos\iv_airframe.mdl
>> get_param(s(10).libs,'ModelVersion')
    ans = 1.4

```

A common approach for sharing interface information for COTS models is a data dictionary. A data dictionary stores names, attributes, scoping, etc. for signals and parameters. It can also store to, from, and in data for components vs. signals and parameters. Another name for a data dictionary is an interface control document (ICD). Just as for configuration management utilities, custom automated tools for interface control need to be practical and easy to use. The utility should automate the task of moving data between model and the ICD. This automation can ensure that there is one copy of "the truth" for the model data. There are a number of ways to store the data. It can be put data in a database such as Access, MySQL, DB2, or Oracle. The data can also be stored in Excel files, MATLAB structures, and other formats. Interface control data can be used optionally to auto-create empty components for models and to auto-check components for compliance by pushing back data to the ICD.

To aid in model management and data design, COTS tools interface with information management software. Third-party tools interface with MathWorks products are in the areas of version management (Synergy, Visual Source Safe, PVCS, Clear Case, CVS, RCS, etc.), testing and test vector generation (Reactis, T-VEC Tester for Simulink, Rational Test RealTime), requirements management (DOORS), configuration management (Synergy CM, Clear Case, CVS), and interface control, the generic capability via Database Toolbox (Access, MySQL, Oracle, DB2, Excel, etc. or customized database access through JDBC and ODBC).

C. Optimizing modeling techniques

The main goal of optimizing model structure is to achieve a faster simulation while retaining design clarity. Common techniques for faster simulations include objectively measuring simulation time (use tic, toc, and Simulink Profiler), saving parameters in binary files instead of ascii text files (.mat versus .m in MATLAB) for fast loading, to enable COTS tools simulation optimizations (inline parameters ON enables many Simulink optimizations), to halt simulations with Stop and Assertion blocks, to turn off unnecessary logging, disable or decimate plotting, to choose a variable step solver with zero crossings to optimize speed versus detail and to choose refined or additional output to get "low cost" data at specified rate. Other common techniques include finding and removing costly calculations, vectorizing to reduce block count, and lastly using discrete multi-rate modeling i.e., refactored dynamics.

When trying to find and remove costly calculations, the Simulink profiler objectively measures speed and hot spots. This profiler helps locate expensive calculations like pow (u,0.4) so they can be replaced by more efficient table lookups. Additionally, the profiler can locate subsystems that can be triggered, or enabled subsystems, to end unnecessary execution.

Whenever possible, calculations should be vectorized to reduce the number of blocks, increasing simulation speed. You can also reduce the number of blocks with the use of iterator blocks for scalar algorithms. In conjunction with vectorization, parameterizing the number of active elements in the vectorization allows flexibility when the model needs to increase or decrease the number of elements. A good example of a vectorized model, multi-UAV, is discussed in reference 5.

Lastly, simulations can be accelerated by discretizing 'slow' continuous components. These slow continuous components greatly increase the order of the solver matrix, resulting in a reduced execution rate. Discretizing these components isolates the slower dynamics from solver, increasing simulation speed. An example of COTS tools automating discretization is the model discretizer. The model discretizer works with Simulink to automate discretizing task. The model discretizer automates block-by-block conversion of continuous blocks to discrete. The option of using a parameter for the sample times is available. There are also options to discretize current selection or system, to discretize all selected blocks, and to discretize preset blocks. These discretization options also let you select the method of block replacement. Either the block can be directly replaced by the discrete version of the block or a configurable subsystem can be used where the continuous and discrete version of the blocks can be toggled from the block menu.

D. Faster simulation via code generation

One of the easiest ways to accelerate execution is to use code generation on the large model. As the model moves farther away from the COTS tool there is a gradual trade-off in flexibility for improved speed.

There are a number of forms of ‘code generation’ available for Simulink. These include the use of Simulink Model Reference, Simulink Accelerator, S-function target, and RSIM target. Model Reference has inherent simulation acceleration through incremental code generation. The clarity of the design is maintained since the Simulink environment is still in use. Simulink Accelerator increases simulation speed while maintaining the ease of use of the Simulink environment. Although generated code is essentially behind the block diagram, it supports full debug and all parameters evaluated at the start of simulation. The next form that gets closer to actual generated code is Real-Time Workshop® S-function target, a ‘black box’ model deployment. The S-function target is used to replace components of a large model with an S-function block accessing generated code for that component. S-function target can be used to ‘accelerate’ individual subsystems, and can reduce the number of active parameters within the simulation. Another way to use S-function blocks is to balance the need for speed and debugging by swapping out individual S-function blocks for systems being debugged. The last form of code generation is Real-Time Workshop RSIM target. RSIM target with inline parameters turned on allows quick parameterization and gives the fastest host-based execution.

Figure 5 compares results for the different forms of code generation and different solver types for an example model, intercept.mdl. The scenario modeled in intercept.mdl helps designers minimize the susceptibility of an air vehicle to a radar-guided surface-to-air missile. Significant speed improvements can be seen as the model approaches generated code. Additionally, major benefits can be seen in when a variable step solver rather than a fixed step solver is used within the Simulink environment.

Description: intercept.mdl w/70+ continuous states with 0.01 sec digital control		
Environment: Simulink 5.0 on AMD Athlon XP 1.2 GHz SuSE linux 8.0, 3 run avg.		
	Fixed-Step @ 0.002 sec	Variable-Step w/ 0.010 sec dig.
Normal Simulation	77.10	44.30
Normal / Inline=ON	75.50	43.30
Accelerator / ON	3.80	3.00
RSIM / ON / gcc -O	1.40	1.20
RSIM / ON / gcc -O3	1.15	1.00

Figure 5. Code Generation Can Significantly Improve Speed

Batch strategies enable even greater performance improvements. Two types of batch clusters to be considered are an executable batch cluster and a Simulink batch cluster. Executable batch cluster, which uses compiled generated code, the most flexible configuration, is compatible with batch tools or can be used custom batch scripting. For Simulink models, RSIM-generated executables should be used in an executable batch cluster. Simulink batch cluster use normal or accelerator modes for simulation on any MathWorks supported platform.

RSIM-generated executables used in an executable batch cluster on Linux systems are useful if conditions are met. These include using rsh, fork(), or equivalent on multiple machines, one CPU can run one simulation in a reasonable amount of time, and many simulation runs need to be performed. Some of the benefits of using RSIM distributed on Linux are it is fast with low execution overhead, it supports variable- and fixed-step solvers, and it works with dual and multi CPU machines where the OS dispatches to free CPUs. Drawbacks of using RSIM distributed on Linux are that the model structure cannot be altered without regenerating code, there can be no algebraic loops, and there are no progress graphics.

An example of running RSIM distributed on Linux follows. To easily run distributed RSIM executables, write a short M-file script for creating input sets together with a ‘run script’. The M-file script would be used in this example to generate 8 test case files using the rsimgetrtp command. Parameters in the model, intercept.mdl should be updated for each test case. The following commands would be used save the parameters for each test case.

```
>> rTP = rsimgetrtp('intercept')
>> save params001.mat rTP
```

From the M-file script the time series inputs for the ‘run script’ are generated. For this example the ‘run script’ is for 4 dual CPU machines farm1, farm2, farm3, farm4. In the script, each CPU receives a parameter set as an input to the executable and a file name for the output data. A portion of the ‘run script’ is in the following text.

```
!/bin/tcsh
rsh -n farm1 ~/intercept -p parms001.mat -o intercept001.mat &
rsh -n farm1 ~/intercept -p parms002.mat -o intercept002.mat &
rsh -n farm2 ~/intercept -p parms003.mat -o intercept003.mat &
rsh -n farm2 ~/intercept -p parms004.mat -o intercept004.mat &
...
rsh -n farm4 ~/intercept -p parms008.mat -o intercept008.mat &
```

This simple ‘run script’ would need enhancements to be tool-quality. Enhancements could include the addition of a multi-user setup for sharing machines, i.e. a queued file area, a two-tier output capture either to local disk, or upload to central area, and a job sequencer to handle timeouts, and cleanly killing processes. An alternative to creating a custom ‘run script’ is to consider COTS or free batch environments for extensive use.

VI. Conclusion

Successful and productive enterprise-wide modeling and simulation using COTS software packages is sometimes a challenge to initiate. Without building a process around the use of Model-Based Design for large-scale models, the benefits possible from Model-Based Design can be overshadowed by process issues. By taking the time to design an engineering process that recognizes and handles the potential difficulties that can arise in working with large models used throughout an organization, the benefits of reduced risk and earlier detection of design cycle defects can be more readily realized.

References

¹Finn, G., “Six Sigma in the Engineering Design Process”, Prescient Technologies, Inc., April, 1999, pp 3, URL: http://web.utk.edu/~aspaqp/qpr/QPRC1999/papers/finn_gavin.pdf.

²Finn, G., “Six Sigma in the Engineering Design Process”, Prescient Technologies, Inc., April, 1999, pp 5, URL: http://web.utk.edu/~aspaqp/qpr/QPRC1999/papers/finn_gavin.pdf.

³MathWorks Automotive Advisory Board (MAAB), “Controller Style Guidelines for Production Intent Using MATLAB®, Simulink® and Stateflow®,” April, 2001, URL: <http://www.mathworks.com/industries/auto/maab.html>.

⁴Mosterman, P. and Ghidella, J., “Model Reuse for the Training of Fault Scenarios in Aerospace,” AIAA Modeling and Simulation Technologies Conference, AIAA-2004-4931, AIAA, Providence, RI, 2004.

⁵Lluch, D., “Building Multi-UAV Simulation Methods,” AIAA Modeling and Simulation Technologies Conference, AIAA-2002-4977, AIAA, Monterey, CA, 2002.

⁶Jackson E. B., Cruz C. L., “Preliminary Subsonic Aerodynamic Model for Simulation Studies of the HL-20 Lifting Body”, NASA TM4302, August 1992.

⁷MacConochie I. O., “A Study of Lifting Body as a Space Station Crew Exigency Return Vehicle (CERV)”, NASA CR-2000-210548, October 2000, pp 22.

⁸Banks, C., “A Discussion of Methods of Real-Time Aircraft Flight Simulation”, URL: <http://www.aeroflight.com/papers/meng/node1.html>.

⁹Baarspul. M., “A Review of Flight Simulation Techniques”, Progress in the Aerospace Sciences, Vol. 27, No 1, 1990, pp 75-80.

¹⁰Barnard, P. A., “Graphical Techniques for Aircraft Dynamic Model Development”, AIAA Modeling and Simulation Technologies Conference, AIAA-2004-4808, AIAA, Providence, RI, 2004.

¹¹MATLAB, Software Package, Ver. 7.0, The MathWorks, Inc., Natick, MA, 2004.

¹²Simulink, Software Package, Ver. 6.0, The MathWorks, Inc., Natick, MA, 2004.

- ¹³Simulink Accelerator, Software Package, Ver. 6.0, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁴Simulink Verification and Validation, Software Package, Ver. 1.0, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁵Simulink Report Generator, Software Package, Ver. 2.1, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁶SimMechanics, Software Package, Ver. 2.2, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁷Stateflow, Software Package, Ver 6.0, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁸Real-Time Workshop, Software Package, Ver 6.0, The MathWorks, Inc., Natick, MA, 2004.
- ¹⁹Aerospace Blockset, Software Package, Ver. 1.6, The MathWorks, Inc., Natick, MA, 2004.
- ²⁰Synergy , Software Package, Telelogic, Malmö, Sweden, 2004.
- ²¹Visual Source Safe, Software Package, Ver 6.0, Microsoft, Seattle, WA, 2004.
- ²²PVCS, Software Package, Synergex, Gold River, CA, 2004.
- ²³Rational Clear Case, Software Package, IBM, White Plains, NY, 2004.
- ²⁴CVS, Software Package, Ver 1.12.9, Freeware, 2004.
- ²⁵RCS, Software Package, Ver 5.7, Freeware, 2003.
- ²⁶Reactis, Software Package, Ver 2003.2, Reactive Systems, Inc., Falls Church, VA, 2004.
- ²⁷T-VEC Tester for Simulink, Software Package, Ver 1.7.2, T-VEC, Herndon, VA, 2004.
- ²⁸Rational Test RealTime, Software Package, IBM, White Plains, NY, 2004.
- ²⁹DOORS, Software Package, Ver 7.1, Telelogic, Malmö, Sweden, 2004.
- ³⁰Synergy CM, Software Package, Ver 6.3, Telelogic, Malmö, Sweden, 2004.
- ³¹Access, Software Package, Ver 2003, Microsoft, Seattle, WA, 2003.
- ³²MySQL, Software Package, Ver 5.0, MySQL AB, Uppsala, Sweden, 2004.
- ³³Oracle, Software Package, Ver 10d, Oracle, Redwood Shores, CA ,2004.
- ³⁴DB2, Software Package, IBM, White Plains, NY, 2004.
- ³⁵Excel, Software Package, Ver 2003, Microsoft, Seattle, WA, 2003.