

Ford Motor Company
General Motors Corporation
Motorola Automotive and Industrial Electronics Group

Smart Vehicle Challenge Problems

Model Composition and Analysis Challenge
Problems

In support of the University California Open
Experimental Platform for DARPA – MoBIES

Model Composition and Analysis Challenge Problems

This document presents the challenge problems in the area of model composition and analysis for the *SmartVehicle* Open Experimental Platform (OEP) provided by The University of California at Berkeley. The challenge problems are not specified to the greatest level of detail to allow the MoBIES Phase I developer more flexibility to apply their own frameworks and methodologies where possible. The challenge problems represent Ford's immediate needs in the area of automatic model composition. All Phase I developers are welcomed to propose and solve related problems in addition to these challenge problems. Although the problems are stated within the context of MATLAB/Simulink/Stateflow modeling environment, Ford remains open to exploring the solutions in other modeling domains as well.

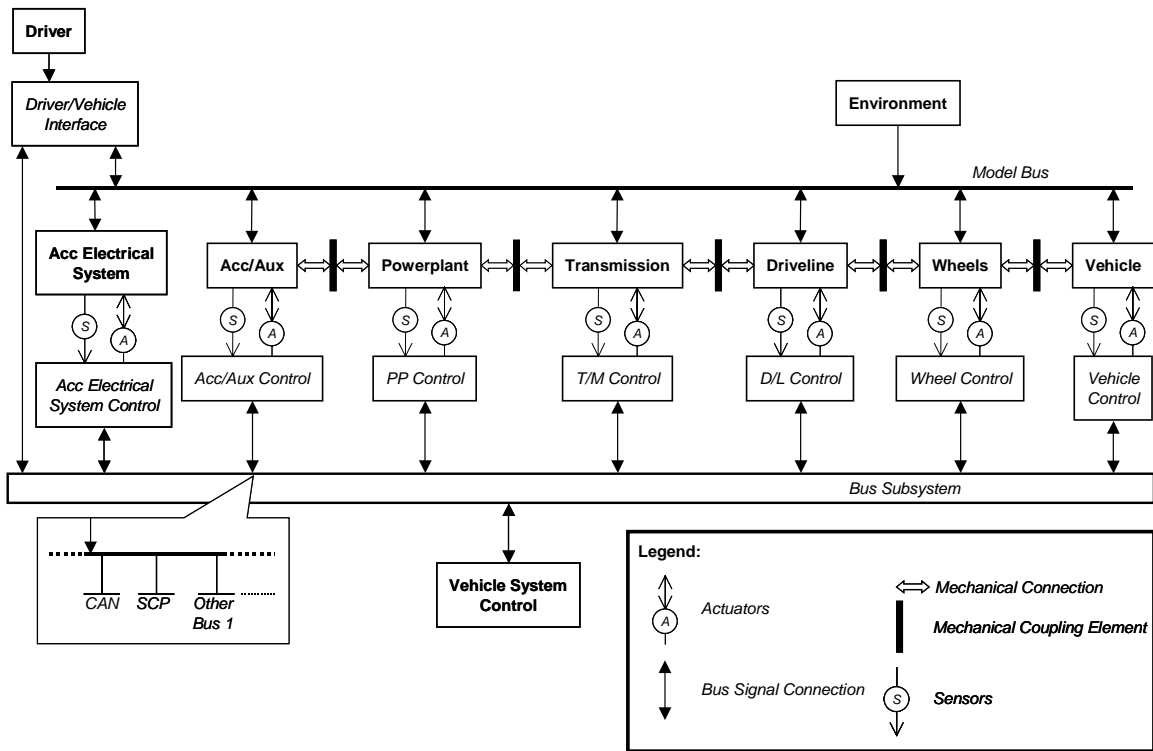
1 Motivation

The complexity of the embedded systems being developed for powertrain control is increasing. At the same time the amount of time allowed for development is shrinking. Current trends in the automotive industry are to accelerate the integration of new features and technologies with fewer prototypes, thus increasing the importance of getting it right the first time. The best way to achieve this is through the extensive use of modeling, simulation, and analysis.

Today we develop many models of various components and sub-systems in the vehicle. These are utilized by various groups to deal with specific problems. Whenever a different group wishes to use a model to analyze a vehicle characteristic, they often develop their own model. This results in duplication of modeling effort that adversely affects productivity when new applications are being developed. There are two reasons for this. First is the lack of communication between groups. A group is often unaware of the existing model. Second is that the model was created in a different language or tool that is inappropriate for the new problem.

To mitigate the above problems and facilitate model reuse, Ford has created a central model archive or *bookshelf* and adopted MATLAB/Simulink/Stateflow as the modeling tool of choice. Ford has invested a great deal of effort to make use of these tools in our product engineering environment. It is desirable to have a tool that can automatically compose a fully executable model vehicle model like the one shown in Figure 1 from the specified list of components in the archive based on some architectural description of the overall model [9].

Vehicle Model Architecture



• Figure 1 Proposed vehicle model architecture in Simulink.

2 Component Overview

2.1 Defining a component

For the purposes of this document a component shall be defined as follows:

1. A component shall be a Simulink model. It may or may not contain a Stateflow block.
2. The component shall be contained in an .mdl format file. The file will have a unique name.
3. There shall also be a *.m file, sharing the same unique name, which shall contain all the workspace parameters called the *calibration parameters* associated with the component.
4. There shall be a *.mat file, sharing the same unique name, which shall contain all the Simulink data objects for the signals and parameters.
5. Each component shall have the following simulation properties:
 - a. ODE solver (variable or fixed step) or *any* (any solver is allowed).
 - b. Time step properties

For variable-step solvers:

- i. Max step size (step size value or *auto*)
- ii. Initial step size (step size value or *auto*)
- iii. Relative tolerance (tolerance value or *auto*)
- iv. Absolute tolerance (tolerance value or *auto*)

For fixed-step solvers

- i. Step size (step size value or *auto*)
- ii. Mode (*single-tasking*, *multi-tasking*, or *auto*)

Given the above list of properties, each time two or more components are composed together to form another component these properties must be checked to confirm that the resulting component is well formed and complete. Completeness implies that the component is complete within the limits implied by the architecture description. So if we ask to compose two components to generate a third component, completion is achieved when the new third component is generated. While the result of a composition might be a true model, as defined in [9], it can also result in a larger component that, in turn, could be used to create a model or yet another larger component.

2.2 Major Categories of Composability Properties

For each component conforming to the description in the previous section, there are three major interconnection avenues that the component may use to communicate or interoperate with other components. Consequently, they must be considered when determining composability of the given components to form a larger component or model.

1. *Signals*. These are typically represented graphically as lines interconnecting the inputs and outputs of the component. There are two types of signals, *data* and *control* signal. A data signal simply relays the data from an output port of one component to an input port of another component. A control signal, in contrast, does not communicate data as such. Rather, it communicates trigger, enable, or function call information between the controlling and the controlled components.

To facilitate the composability analysis, each signal in the model is augmented with additional information, which ranges from simple information such as name, data type, units, and dimension of the signal to more sophisticated information such as the communication protocol for the signal, e.g. CAN interface, to describe the implementation of the signal in the real hardware. The information augmentation is done using Simulink Data Objects in MATLAB 6 (R12) as described in Item 4 in Section 2.1.

2. *Execution criteria*. These are the properties that described how the simulation is to be executed for the component. They correspond to Item 5 in Section 2.1.
3. *Workspace parameters*. These are typically calibration parameters of the component. However they may multiple components that shared the same parameter names. Therefore each *.m file will contain a separate copy of the parameter assignment which may, or may not, contain the same calibration data.

3 Model Composition Challenge Problem

3.1 Overall Challenge Problem

Given the archived component models conforming to the format described in Section 1:

Design a software tool that automatically composes a fully executable model from a list of components and an architectural description of the final model.

The tool must be able to handle all three aspects of composability outlined in Section 2.2. These requirements are described in the following sections.

3.2 Signal Composability

3.2.1 Connectivity

The tool must provide complete connectivity for all signals in the final model.

3.2.2 Type Resolution

The type information for each signal stored in the associated Simulink data object may describe both *static* and *dynamic* [7][8] properties of the signals. Examples of static properties of a signal are data type (real, integer, boolean, trigger, enable, function call, etc.), units, and dimension. Dynamic properties describe how the signal evolves with time or how it is updated possibly both in simulation and real hardware. Examples of these properties are.

- Continuous-time vs. discrete-time (with sampling rate)
- Data transfer protocols such as CAN, TTP, or direct connection (signal values are always identical at both ends of the connection at any time).

The tool must resolve the type for all partially specified signals and ensure compatibilities of both static and dynamic properties of the signals being connected. The tool must notify the user when type resolution conflict is detected.

The type resolution may be done through the use of *lossless convertibility* where the output signal can be converted into the format of the connected input signal without loss of information. For example, an integer output can be converted into a real input. A discrete-time signal with can be up-sampled to a higher sampling rate provided that the new sampling rate is a multiple of the original sampling rate. Similar idea based on partial ordering on the simulation relation of the dynamic signal types has been proposed in [7][8].

3.2.3 Collision Resolution

It is possible that the given architectural description for the model is ambiguous. As a result of such description, there may be multiple candidate output signals that can be interpreted as the source for an input signal. For example, the architectural description may state only that the signals with the same name should be connected within a particular subsystem but there are multiple output signals and only one input signal with the same name. Since only one output signal is allowed to be connected to the input signal, the composition tool must provide a judicious signal collision resolution scheme to select the appropriate output signal. It is preferable that the selection of the collision resolution scheme be automatic. However, the tool is allowed to prompt for user intervention when necessary. In any case, the mechanism for resolving the collision, i.e. the appropriate signal selector, must be provided automatically.

A question may be raised as to why such ambiguity in the architectural description would be allowed for model composition at all. To see this, consider a large model with a large number of interconnection. It would be extremely tedious for the user to explicitly specify all signal interconnections in the model when in many cases the connections can be inferred easily. For example, when only one pair of input and output signals with the same

name and signal properties exist in the subsystem, it is obvious that the two should be connected. The user should be able to specify the smallest amount of information possible for the final model to be composed correctly to keep the architectural description small and concise.

The signal collision resolution mechanism may be view as analogous to the mechanism for handling *polymorphism* in object-oriented languages. The concept of component *classes* and wiring *methods* may need to be introduced to implement this mechanism. For example, for a component of class *fuel-cell*, the *series* method connects n components in series automatically. Even though there are n output signals and n input signals with the same name, the method knows to connect the output of each cell to the input of the next cell.

3.3 Execution Criteria Composability

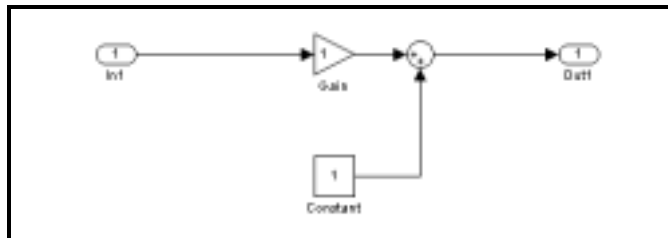
The tool must ensure that the simulation properties described in Item 5 in Section 2.1 for all components in the model are compatible. This maybe done through the concept of lossless convertibility similar to the discussion in Section 3.2.2. For example, a model component that uses a fixed step ODE solver can be simulated with another component that uses a fixed step solver with a smaller time step, provided that the larger time step is a multiple of the smaller time step. The aggregate component now uses the smaller time step for the solver, i.e. the most general simulation property among all components is used. If no common simulation property can be found that all components can be converted into, the tool must notify the user of the conflict.

3.4 Workspace Parameters Composability

3.4.1 Introduction

Simulink blocks, such as constants, gains, and table look-ups, can be parameterized by referencing the values from the GUI or from variables in the MATLAB workspace. Many times the modeler chooses the workspace option. The following example demonstrates the reason for such choice.

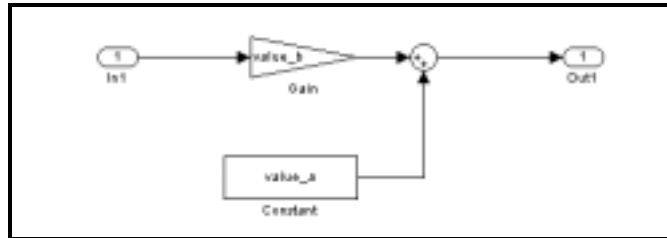
Consider the Simulink diagram in Figure 2. We see that there are 2 parameters, namely the scalar values for the gain and constant blocks. To change the parameter values via the GUI, the user must double click on each block and then change the value. For such a small example, this may not seem to be a problem. For larger and more complex models, however, managing various sets of parameters via the GUI can become a serious problem.



• Figure 2

To make reparameterization manageable, we can modify the model to reference workspace variables instead as shown in Figure 3. Here we use the variables `value_a`

and `value_b` instead of the numeric values. These workspace variables are defined in the calibration `.m` file associated with each component or subsystem model (Item 3 in Section 2.1).



• Figure 3

The workspace parameterization makes it easier to manage large sets of parameters, as multiple parameters can be edited and stored in the calibration `.m` file.

Despite the above advantage, the approach also has its drawback. The approach amounts to the use of *global variables* in programming language, one of the great taboos in the computer science discipline. The use of global variables can produce the so called *side effects*, the situation where subroutine can change the behavior of another subroutine by modifying its internal variables and produce undesirable effects. Side effects, if not documented, are difficult to discover and debug and are to be avoided unless absolutely necessary.

Workspace parameterization can clearly produce side effects when two components share common calibration parameter names. The behavior of the overall model may be very different from the expected behavior depending on which calibration file is executed last. Other `.m` scripts that are run during the simulation that happens to have the variables with the same names as the calibration parameters in the workspace may further contribute to the problem. It is also difficult to distinguish between calibration parameters and other MATLAB variables in the main workspace. This gives rise to the challenge problem for the workspace parameters described in the next section.

3.4.2 Parameter Collision Detection

The tool must automatically detect all conflicting workspace data definitions by examining the calibration `.m` files for all components in the model. For example if there are two components that both have a workspace variable named `value_b`, check whether or not the definitions are compatible and the actual data values are the same. The tool must be able to handle the following cases.

1. Unique workspace data items.

When a workspace data item has a unique identifier, no other instance of the name occurs in any other subsystem, then the workspace data item is used as is. No further action is required.

2. Workspace data items with same name.

When workspace data from different components shares a common name there are several actions that may be appropriate.

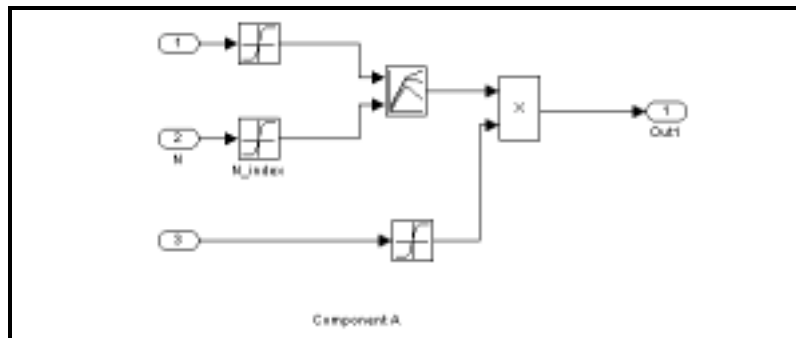
a) Name matches and data definition differs.

In the case where the names do match, the next item to check is the data definition of the two items. Here the focus is on the data type, real, integer, scalar, vector, array, and matrix. If one is a scalar real and the other is an array of real, then an either an error has occurred or some type of data scoping would be needed to resolve the conflict. Simulink does support limited data scoping.

b) Name and Definition match. Data value differs.

They will be instances where a common workspace data item is used by more than one model component. As each component will have a copy of this data item, there will be instances where the data does differ. However this does not imply that only a complete one to one mapping is the only acceptable state.

If we take the instance where two components, named A and B, respectively, use a 1D table to map engineering units to a lookup index for a 2D table, as seen in Figure 4. Here the name and type information of N_index are the same for both components. However, the data lengths differ for component A and component B as shown in Figure 4. Yet the resulting look-up table using either one of the two parameters is identical. Therefore, this is an acceptable collision.



• Figure 4

- Component A: $N_index = \{(0,0), (6000,10)\}$
- Component B: $N_index = \{(0,0), (3000,5), (6000,10)\}$

However, if we take the case where the second pair of data in N_index for component B were (3200,5), then the actual behavior of component B would change if we adopted the component A version of the parameter. As is also the case if we adopt the component B version and apply it to component A. This is an error and the tool should notify the user of the conflict.

One can imagine that the parameter conflict detection tool is extremely desirable to manage a model where hundreds of components are used and many of them share the same workspace variable name.

4 Bus Composition Challenge Problem (Extra Credit)

4.1 Introduction

In contrast to the main challenge problem in Section 3, which addresses the general model composition problem in the MATLAB/Simulink/Stateflow environment, the challenge problem described in this section addresses a specific need for Ford Motor Company.

The architecture shown in Figure 1 has been proposed at Ford Motor Company for modeling complete vehicle dynamics and vehicle control systems in Simulink. To enhance readability, the architecture makes extensive use of *buses*, groups of signals that are multiplexed into single wires. The main bus, called the *model bus*, is the main source of signals for all subsystems in the vehicle. There are also input and output buses to and from each individual subsystem as depicted in Figure 5 for the *PP Control* subsystem.

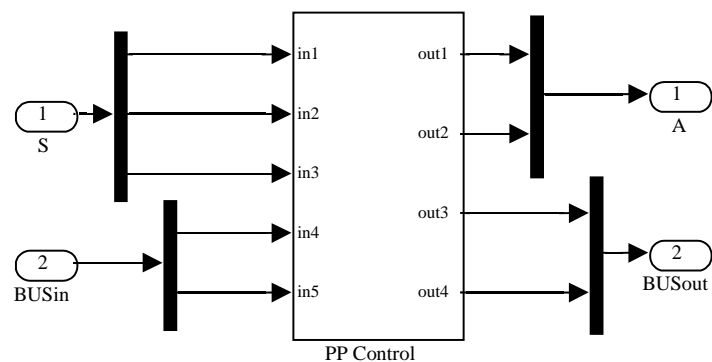


Figure 5 Subsystem Input and output buses

While the creation of certain buses is straightforward, involving only multiplexing the signals to create a larger bus or demultiplexing the signals to create a smaller bus, the situation may be more complicated in other cases, especially when there is a signal collision. This occurs when signals with the same name from multiple sources enter the same bus. In such case, the engineer must choose only one signal to be the output of the bus. The criterion for doing this may depend on the specific situation. For example, the output signal from the block with highest priority may override the same output signals from all other blocks. Similarly, the situation may warrant the use of a *merge* block in Simulink, outputting the most recently computed signal. Or alternatively, the output signal from the bus may represent the average of all colliding signals. Creating the buses manually is not desirable in either of the above two cases. In the straightforward case, the bus creations become too cumbersome and error-prone for the engineer. In the signal collision case, it is important for the engineer to select a signal collision resolution scheme that is appropriate and consistent based on some framework or criteria.

With the above motivation, we now state the bus composition challenge problem.

4.2 Challenge Problem

Given all first-level subsystems in the vehicle model shown in Figure 1 composed from the model composition tool described in Section 3:

Design an add-on tool that automatically creates the main bus and the input and output buses to each subsystem.

The above challenge problem may be viewed roughly as the special case of Section 3.2. The specific requirement is that the tool must provide complete connectivity in form of buses that run between first-level subsystems in the model. The tool must satisfy the requirements described in the subsequent section. The first three requirements are essentially the special case of Section 3.2. The last one, in contrast, is an extra requirement.

4.2.1 Finding Appropriate Signals

The tool must be able to find the appropriate signals to form the subsystem input buses and the main bus.

4.2.2 Signal Type Resolution

The bus composition tool must resolve the type for all partially specified signals and ensure compatibilities of both static and dynamic properties of the signals being connected. The tool must notify the user when type resolution conflict is detected.

4.2.3 Signal Collision Resolution

The tool must provide a judicious signal collision resolution scheme when multiple signals with the same name and properties enter the same bus. It is preferable that the selection of the collision resolution scheme be automatic. However, the tool is allowed to prompt for user intervention when necessary. In any case, the mechanism for resolving the collision, i.e. the appropriate signal selector, must be provided automatically.

4.2.4 Physical Signal Modeling

In addition to resolving signal data types, the tool must automatically construct bus physical models that accurately implement the corresponding physical data transfer protocol associated with the signal type such as CAN or TTP for each signal.

5 References

- [1] Butts, K., Toeppe, S., Ranville, S., "Specification and Testing of Automotive Powertrain Control System Software using CACSD tools", Proceedings of the 17th AIAA/IEEE/SAE Digital Avionics System Conference 1998
- [2] Toeppe, S., Ranville, S., "Model Driven Automatic Unit Testing Technology: Tool Architecture Introduction and Overview", Proceedings of the 18th AIAA/IEEE/SAE Digital Avionics System Conference 1999
- [3] Toeppe, S., Ranville, S., Bostic, D., Rzeimen, K., "Automatic Code Generation Requirements For Production Automotive Powertrain Applications", IEEE International Symposium on Computer Aided Control System Design 1999

- [4] Toeppe, S., Ranville, S., Bostic, D., Wang, C., "Practical Validation of Model Based Code Generation for Automotive Applications", Proceedings of the 18th AIAA/IEEE/SAE Digital Avionics System Conference 1999
- [5] Toeppe, S., Ranville, S., Bostic, D., "Automating Software Specification, Design and Synthesis for Computer Aided Control System Design Tools", Proceedings of the 19th AIAA/IEEE/SAE Digital Avionics System Conference 2000
- [6] Toeppe, S., Ranville, S., "Powertrain Controller RTOS Evaluation: CPU and RAM Resource Usage Modeling and Benchmarks", Embedded Systems Conference - Europe 1999 and Embedded Systems Programming Magazine July 2000
- [7] E. A. Lee and Y. Xiong. System-Level Types for Component-Based Design. Technical Memorandum UCB/ERL M00/8, University of California at Berkeley, February 2000.
- [8] E.A. Lee. What's Ahead for Embedded Software? In *IEEE Computer*, September 2000, pages 18-26.
- [9] W. Milam and A. Chutinan. A Proposal for a Model Compiler. Report for DARPA MoBIES. http://vehicle.me.berkeley.edu/mobies/papers/model_compiler.pdf