SmartVehicle Evaluation Report

DESERT/AMC Evaluation

In support of the University of California Open Experimental Platform for DARPA – MoBIES, Contract F33615-01-C-1841.

Author:
Kenneth R. Butts
Ford Motor Company
Dearborn, MI
kbutts1@ford.com

Contact:
William P. Milam
Ford Research Laboratory
Dearborn, MI
wmilam@ford.com

1. INTRODUCTION

Popular embedded control system modeling and simulation tools use signal-flow diagrams to capture system behavior. These tools are effective when applied to systems of low-to-moderate complexity. However, difficulties arise in their application to more complex systems. One difficulty is modeling scalability. The tools require modelers to manually assert signal-flow connections from component to component. This modeling effort is not sustainable for the production of systems such as the automotive engine controller wherein more than 2000 signal-flow connections must be made between 100 or so components. In practice, while control system components are efficiently developed with model-based methods, the assembly and analysis of large systems must revert to traditional build and test software integration methods. Thus there is a need for automated 'model compilers' that create large assembly models from model components to realize the benefits of model-based development for complex systems.

This need for a model compiler was introduced in the Model-based Integration of Embedded Software (MoBIES) challenge problems [1], [2] and further elaborated in [3]. In this report, we review one such model compiler developed by Sandeep Neema, Institute for Software Integrated Systems (ISIS) at Vanderbilt University [4]. The work model compiler is derived from Dr. Neema's tool for design space exploration (DESERT) [5], [6] and instantiated in ISIS' Generic Modeling Environment [7]. In our review, the components to be assembled are modeled in The MathWorks' Simulink® control system modeling tool. Simulink®[1] is also the target-modeling environment for the compiled system model.

2. PROBLEM DESCRIPTION

This review of DESERT is motivated by our need to assemble Simulink® models of reusable control components into application specific system models. The problem under consideration is structurally equivalent to a major subsystem component of our embedded powertrain controller. (Note that reusable components are often constructed from smaller components.) However, the signal-flow names, performance requirements, and performance attributes are fictitious to respect intellectual property. Our goal is to 1) specify the structural architecture of the desired assembly subject to constraints, 2) choose a consistent set of elements from our component library that satisfy the constraints, and 3) compile the components into a Simulink® assembly model.

2.1. Component Characteristics

Components are grouped by function and, in our problem, each function has three alternative component designs. Components are Simulink® models constructed with the components' input / output context at the root of the model (see Appendix A). Each component is characterized by its list of inputs, list of outputs, Central Processing Unit usage, Read Only Memory usage, Random Access Memory usage, and its application membership. Name, unit, and type characterize each input and output signal. Inputs are further characterized by their sampling (read) period. Likewise, outputs are further characterized by their output (write) period. The component characteristic information is captured in Matlab® files in the same format used in the MoBIES code generation challenge problems [8]. An example is included in appendix A of this report. Table 1 summarizes the characteristics of the example library's18 functional component models and its single scheduling component model.

[1] Simulink and Matlab are registered trademarks of The MathWorks. Inc., Natick, MA

Name	# inputs (scheduling)	# outputs (scheduling)	CPU usage %	ROM usage	RAM usage bytes	membership
b2p	22 (3)	14	3	1000	30	'speed_density' 'mass_flow'
b2p_1	22 (3)	14	38	1200	50	'speed_density' 'mass_flow'
b2p_2	23 (3)	14	45	1400	60	'speed_density'
e6212	15 (1)	2	3	2500	120	'speed_density' 'mass_flow'
e6212_1	15 (1)	3	64	3000	180	'speed_density' 'mass_flow'
e6212_2	15 (1)	4	80	3500	200	'speed_density'
h1813	6 (1)	2	3	700	60	'speed_density' 'mass_flow'
h1813_1	7 (1)	2	6	1200	250	'speed_density' 'mass_flow'
h1813_2	7 (1)	2	17	2000	300	'speed_density'
h32p	12 (1)	5	3	10000	300	'speed_density' 'mass_flow'
h32p_1	12 (1)	5	12	15000	400	'speed_density' 'mass_flow'
h32p_2	12 (1)	5	22	20000	600	'speed_density'
h620	8 (1)	6	3	15000	1000	'speed_density' 'mass_flow'
h620_1	8 (1)	6	17	20000	2000	'speed_density' 'mass_flow'
h620_2	8 (1)	6	23	21000	3000	'speed_density' 'mass_flow'
v6212	12 (3)	7	24	13000	1200	'speed_density' 'mass_flow'
v6212_1	12 (3)	7	28	16000	1300	'speed_density' 'mass_flow'
v6212_2	12 (3)	7	19	19000	1500	'speed_density'
scheduler	0	10 (10)	3	1500	100	'speed_density' 'mass_flow'

Table 1 Component Characteristics

2.2. Desired Architecture

We wish to create a Simulink® model component with the following hierarchical structure:

Controller
- scheduler (base component)
- ford_test (intermediate container)
  - b2p (base component)
  - h1813 (base component)
  - h32p (base component)
  - h620 (base component)
  - v6212 (base component)
  - e6212 (base component)

2.3. Connection Constraints

The model compiler should check that all necessary input connections between components, intermediate containers, and external interfaces are consistent. Connection consistency means that the signals are produced and that signal names, types, and units match. In our case, the controller (the compiled component) external input and output signals are defined by Matlab® component characterization files similar to those found in Appendix A.

2.4. System Constraints

In addition to checking the connection consistency, the DESERT-based model compiler uses a constraint solver to ensure that the compiled model satisfies additional constraints. We leverage this capability to specify that the compiled component's:

CPU usage is less than 75%,
ROM usage is less than 200000 bytes,
RAM usage is less than 25000 bytes.

We also define component relationship constraints as follows:

component b2p version 2 requires e6212 version 2
component h32p version 0 requires h1813 version 2

Finally, we wish to specify that all the components used in an assembly have a common membership attribute. These membership attributes indicate component applicability to various product classification schemes. In our example, all components shall have 'speed_density' membership.

3. DESERT-based Model Compiler

The DESERT-based model compiler offers significant capabilities in addition to those envisioned in the original concept [1], [2]. DESERT introduces a design space wherein design alternatives are automatically evaluated for feasibility. Thus automated support is provided to help the designer choose the model-components in the model-compiler specification. This is important when thousands of components reside in the model repository.

A special extension to the data-flow modeling paradigm allows the designer to a) specify the desired structure of the compiled model and b) identify the points where design alternatives should to be considered. The design space exploration tool then prunes the set of potential component choices to those that satisfy a set of specified system constraints. The tool provides progressive (iterative) and symbolic constraint satisfaction methods to deal with computational complexity.

DESERT leverages ISIS modeling infrastructure called the Multi-Graph
Architecture (MGA) [9]. The MGA is comprised of the Graphical Model Editor for the creation of domain-specific model constructs, the Model Database for managing the models, and the Model Interpreter for translating the models into executable and analyzable form.

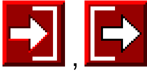
3.1. DESERT Architecture Modeling

The designer creates an architecture specification model to specify the desired structure of the compiled model and to indicate where design alternatives should be considered. The modeling paradigm is comprised of several modeling objects that include:

Compound: a hierarchical container for further model structure.

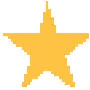
Alternative: a place to define a design choice that contains the Primitive objects corresponding to those choices.

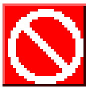
Primitive: a leaf in the model structure that maps to a specific model component. In our case, the map contains the Simulink® model name, the Simulink® model file name, and the Matlab® component characterization file name.

 Input, Output Port: places to connect signal-flow.

Multiplexer: a place to aggregate signal-flows.

De-multiplexer: a place to de-aggregate signal-flows.

 Star connecter: a place where all signal-flow input is made available as signal-flow output.

 Constraint: a place to specify constraints on the compiled model.

The following diagrams illustrate how these modeling objects were used to specify the desired model architecture for our example problem.

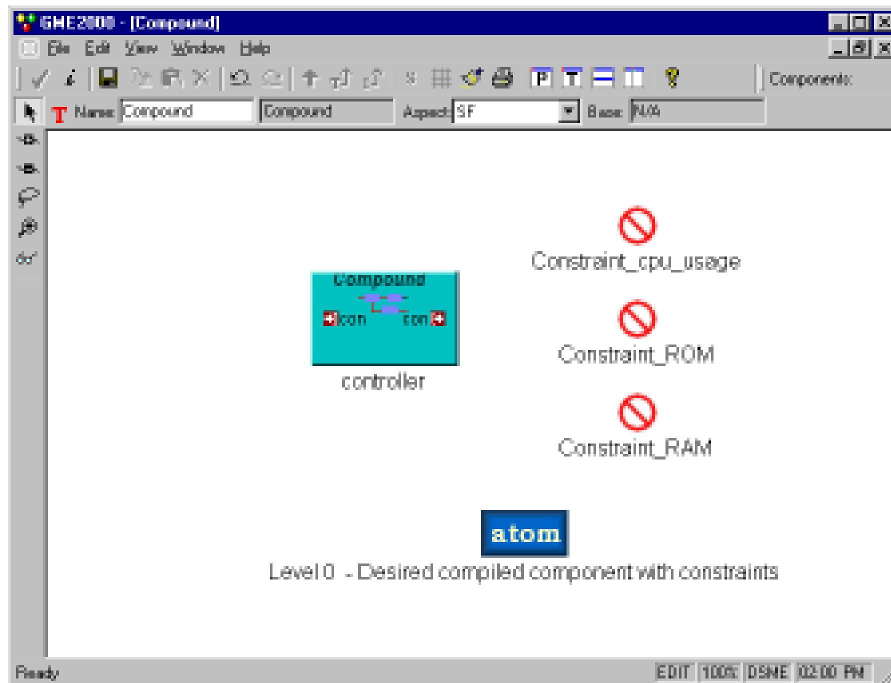


Figure 1 – Top level architecture specification in DESERT

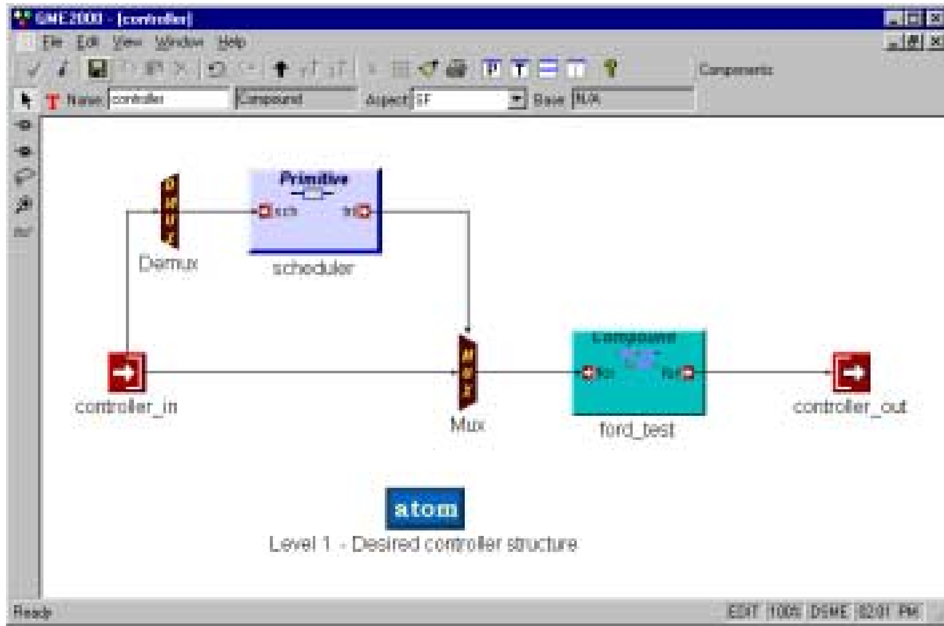


Figure 2 - Second level architecture specification in DESERT

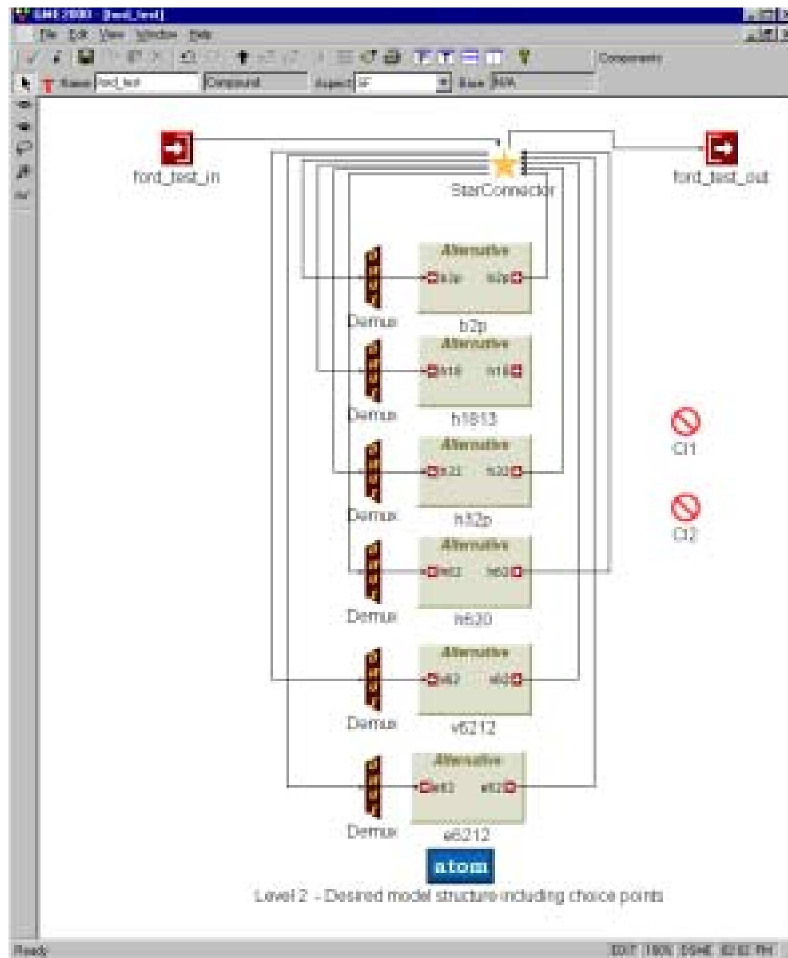


Figure 3 - Third level architecture specification in DESERT

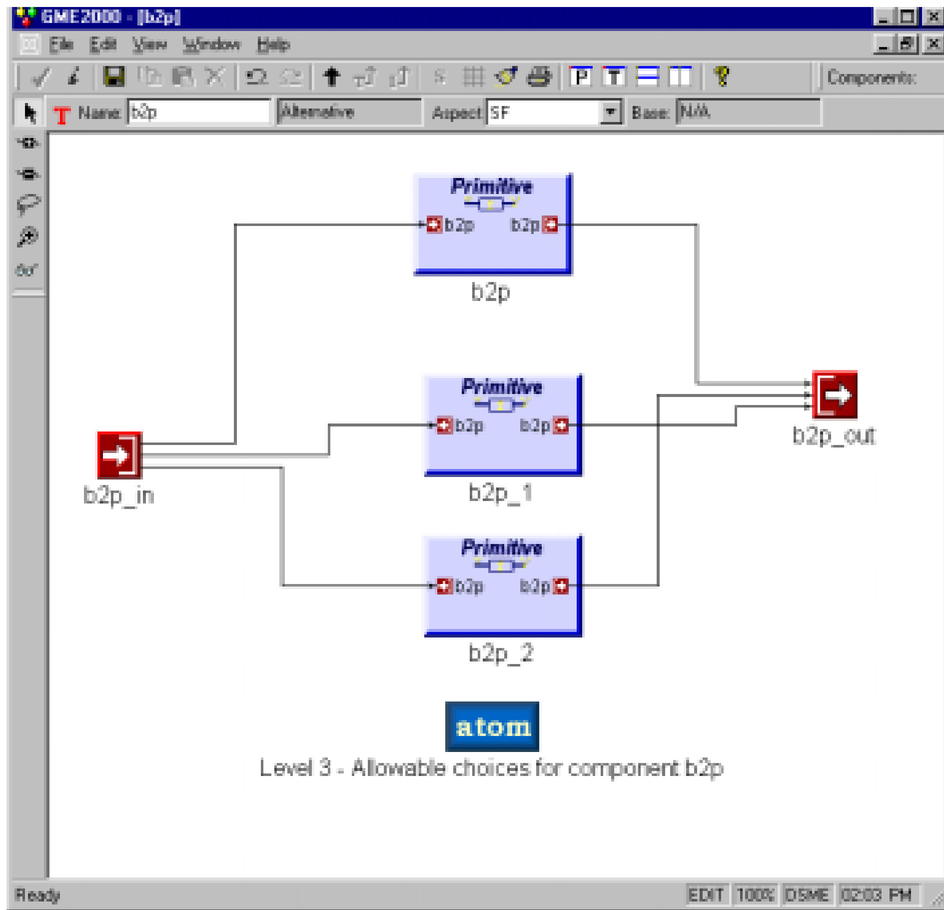


Figure 4 - Fourth level architecture specification in DESERT

3.2. DESERT Constraint Modeling

DESERT provides a textual constraint language that allows the user to express desired performance and dependency properties of the compiled model. The constraint language is derived from the Object Constraint Language [10]. The following constraints were used in our example.

Performance:
(1) cpu_usage() < 75
(2) ROM() < 200000
(3) RAM() < 25000

Dependency:
(4) children("b2p").implementedBy() =
children("b2p").children("b2p_2") implies
children("e6212").implementedBy() =
children("e6212").children("e6212_2")

(5) children("h32p").implementedBy() =
children("h32p").children("h32p") implies
children("h1813").implementedBy() =
children("h1813").children("h1813_2")

Note that DESERT does not currently allow for the 'membership' constraint discussed in the Problem Description section.

3.3. Tool Operation Sequence

The following steps comprise the DESERT automated model compilation process.

1) Characterize Simulink model components. This is currently a manual process since The MathWorks do not provide a data-dictionary facility for Simulink. A representative component characterization file may be found in Appendix A.
2) Specify compiled component interface. This specification allows the model-compiler to a) ensure that all input signal flows are either provided externally or generated internally and b) output any internally generated signalflows at the compiled component interface. Again, this is a manual process captured in a component characterization file.
3) Specify the desired compiled model structure and constraints. This results in a DESERT model similar to that described in the previous sections. The model contains mapping information that ties the DESERT Primitive objects to the appropriate Simulink component models.
4) Generate a complete model of the design space. The executable DSME2HSFA.exe generates a concrete model of the design space with all i/o definitions, property definitions, modeled constraints, automatically generated i/o consistency constraints. (*_hsfa.xml)
5) Generate a file suitable for DESERT analysis. The executable HSFA2DESERT.exe generates an input file for analysis in DESERT (*_ds.xml).
6) Prune the design space. The designer uses DESERT to generate a set of feasible design solutions. This set is catalogued in a configuration listing (*_ds_back.xml).
7) Select the solution to generate. The executable DBHSFA2HSF.exe presents a user interface to the designer to mark the feasible solutions(s) for model generation. This selection is captured in (*_cfg_*.xml).
8) Generate the solution. The executable HSF2M.exe geneates a Matlab model construction script (*_cfg_*.m) that may, in turn, be executed in the Matlab environment.

4. RESULTS

4.1. Interoperability

This work is an example of interoperability. Matlab® and Simulink® are used for component construction and characterization. The DESERT-based model compiler environment provides automated component abstraction, design space exploration, pruning, and Matlab® model scripting. Thus, our desire to start and end with Simulink® models (components to compiled assemblies) is satisfied.

The DESERT-based model compiler architecture is layered such that much of the machinery is reused from previous and related work. GME, DESERT, Hierarchical Signal-Flow with Alternatives (HFSA) modeling paradigm, and MoBIES Matlab® Data Models [11] are principal examples. Moreover, the layered architecture suggests that the DESERT-based model compiler could be readily applied to additional modeling environments.

4.2. Analysis

The DESERT-based model compiler provides signal type and unit consistency checking and design space exploration based on constraint satisfaction. The design space exploration capability is a very powerful and useful addition to the original model compiler concept. Application of the iterative pruning process to our example resulted in a reduction of the design space from 729 possible to 36 feasible solutions. The impact of

successive constraint application is shown in Figure 5.Our requirements to 1) check that all input signals have a source and 2) generate a specified external interface to the compiled model are not yet supported. These are essential requirements that must be addressed.

In addition, we would like to prune the design space with component membership characterization. This mechanism would allow us to automatically leverage the compatibility relationship database that we currently maintain.

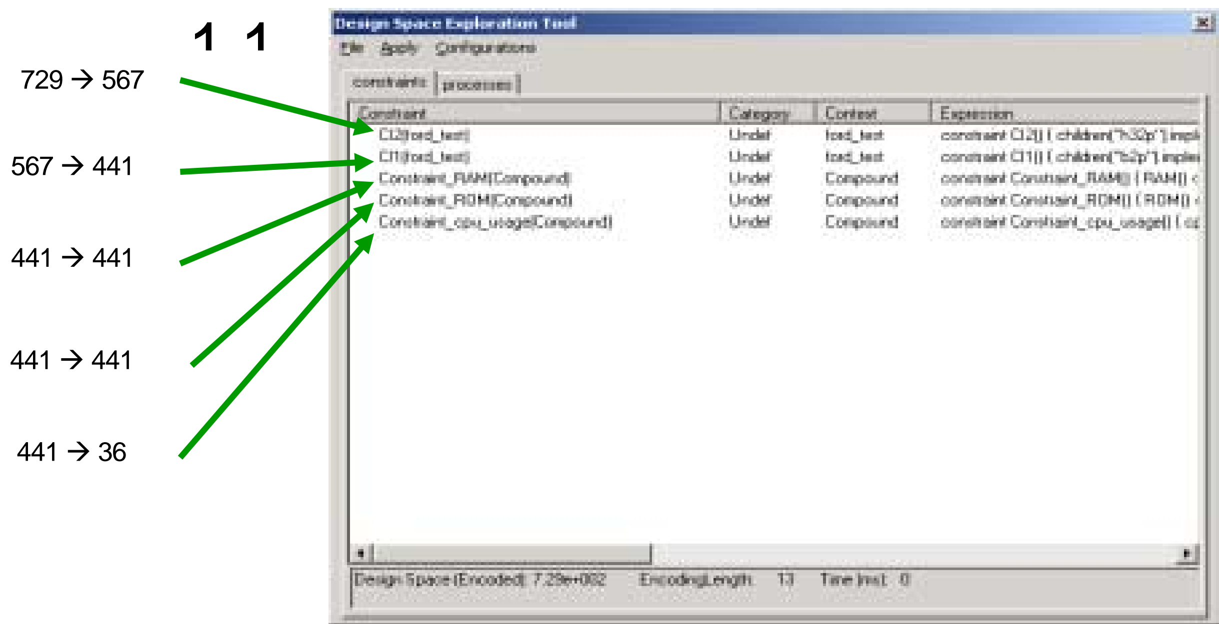


Figure 5 – Design space pruning via successive constraint application

4.3. Generation

The DESERT-based model compiler generates a Simulink® assembly model according to the structure specified in the architecture model. The benefits of automated construction are apparent upon inspection of the generated model shown in Figure 6 and Figure 7. Unfortunately, the analysis difficulties mentioned in the previous discussion are present in our example: unconnected output ports with no corresponding generator are marked in red and all (49) component-generated signals appear on the external interface rather than the 27 specified in the characterization file.

4.4. Ease of Use

The DESERT-based model compiler elements are easy to use and well documented. However, the user must issue a command for each data preparation step in the process (see Tool Operation Sequence.) We do not consider this manual intervention to be an issue since it can be easily remedied.

The GME-based modeling of the architecture description is quite pleasing. The graphical model is easy to construct and interpret. Most control-system domain users will find the syntax of the constraint language a bit arcane.

Of more concern, is the laborious and tedious work required to characterize the Simulink® components. This is not a criticism of the DESERT-based model compiler, rather a lament that such labor should be mitigated by base capabilities in Matlab® / Simulink®. Mechanisms will have to be developed to ease the Simulink® model component characterization process.

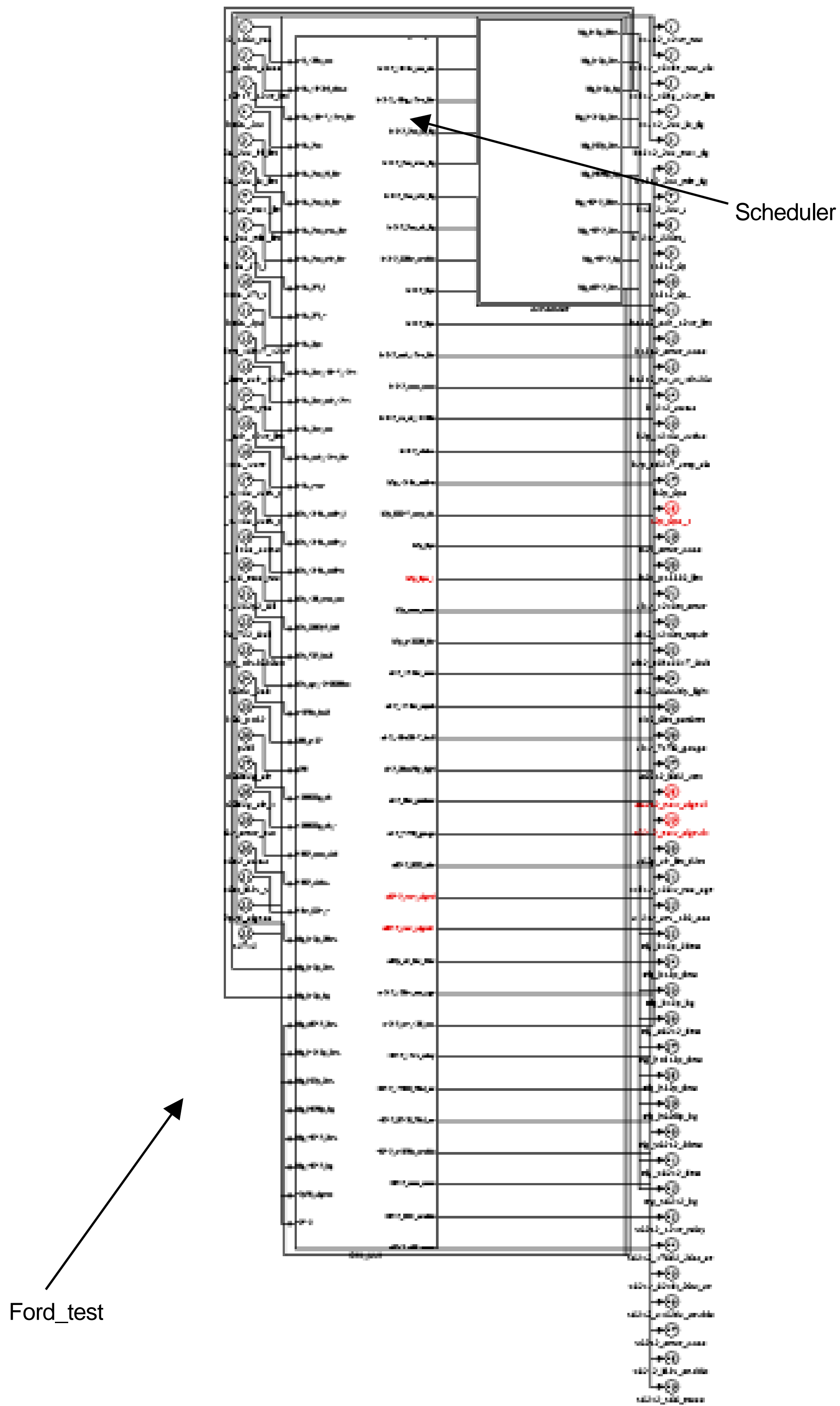


Figure 6 - Second level compiled model in Simulink®

Figure 7 – Third level compiled model in Simulink®

5. FUTURE WORK

5.1. Interface Specification and Signal producer / consumer consistency

The DESERT-based model compiler must provide a mechanism to specify the compiled-component's external interface. This will admit the required check for signal producer / consumer consistency. This check should be made at the first step of the design space pruning process. The designer should be presented with *Ignore, Resolve,* and *Prune* options before continuing with the pruning process.

5.2. Bus Based Connections

Simulink® provides a Bus connection mechanism that helps to manage large-model-signal-connection integrity and block-diagram clarity. By design, DESERT's Multiplexer and De-multiplexer objects map to Simulink®'s Bus Creator and Bus Selector blocks respectively. We suggest that DESERT's Simulink® model-generator should take advantage of these mappings to improve large-model scalability and utility.

5.3. Automated Component Definition

Simulink® now provides data objects that can store user definable signal and subsystem attributes suitable for model component characterization. To facilitate modeling tool independence, an external signals and component attribute database that can be used to import and export component characterization attributes to the appropriate Simulink® data objects should be provided. The DESERT tools could then interact with the external signals and attribute database to perform their analyses. Moreover, the external signals and attribute database could serve as the modeling team's centralized data dictionary or be replaced by the team's preferred database. We suggest the following architecture. Dashed lines delineate the new tool components.

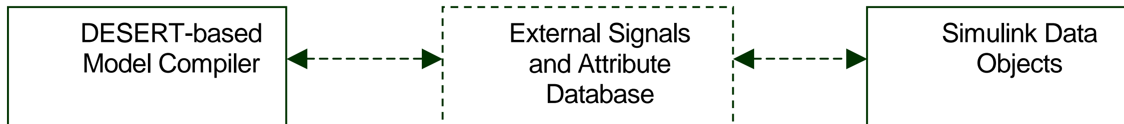


Figure 8 – Component Characterization Architecture

5.4. Connection to Model Repositories

A longer-term need is to provide an interface to model management systems. These model management systems serve as model-component repositories where versioned model components, architecture descriptions, and results of model-based engineering investigations reside. This interface would facilitate improved component and assembly architecture specification thereby enlarging the effectiveness of the design-space exploration concept.

6. Acknowledgements

We would like to thank Dr. Neema, Vanderbilt University, for his excellent support of the model compiler concept. His initial realization is a very impressive first step on the road to commercial application.

7. REFERENCES

[1] Chutinan, A, "Model Composition and Analysis Challenge Problems," MoBIES, http://vehicle.me.berkeley.edu/mobies/papers/model_composition_challenge.pdf, January, 2001.

[2] Milam, B., Chutinan, A., "A Proposal for a Model Compiler," MoBIES, http://vehicle.me.berkeley.edu/mobies/papers/model_compiler.pdf, 2001.

[3] Butts, K., Bostic, D., Chutinan, A., Cook, J., Milam, B., Wang, Y., "Usage Scenarios for an Automated Model Compiler," Proceedings of the First International Workshop on Embedded Software, EMSOFT 2001, Tahoe City, CA, Henzinger, T., Kirsch, C., Eds., LNCS221, Springer-Verlag, October 2001.

[4] Institute for Software Integrated Systems (ISIS), Vanderbilt University, http://www.isis.vanderbilt.edu/.

[5] Neema S., "Design Space Representation and Management for Model-Based Embedded System Synthesis," Technical Report ISIS-01-203, Institute for Software Integrated Systems, Vanderbilt University, February, 2001.

[6] Neema S., Sztipanovits J., Karsai G., "Design-Space Construction and Exploration in Platform-Based Design," Technical Report ISIS-02-301, Institute for Software Integrated Systems, Vanderbilt University, June 24, 2002.

[7] Generic Modeling Environment, Institute for Software Integrated Systems (ISIS), Vanderbilt University, http://www.isis.vanderbilt.edu/Projects/gme/default.html.

[8] Ranville, S., Chutinan, A., "Embedded Software Analysis and Generation Challenge Problem," MoBIES, http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf, revised July, 2001.

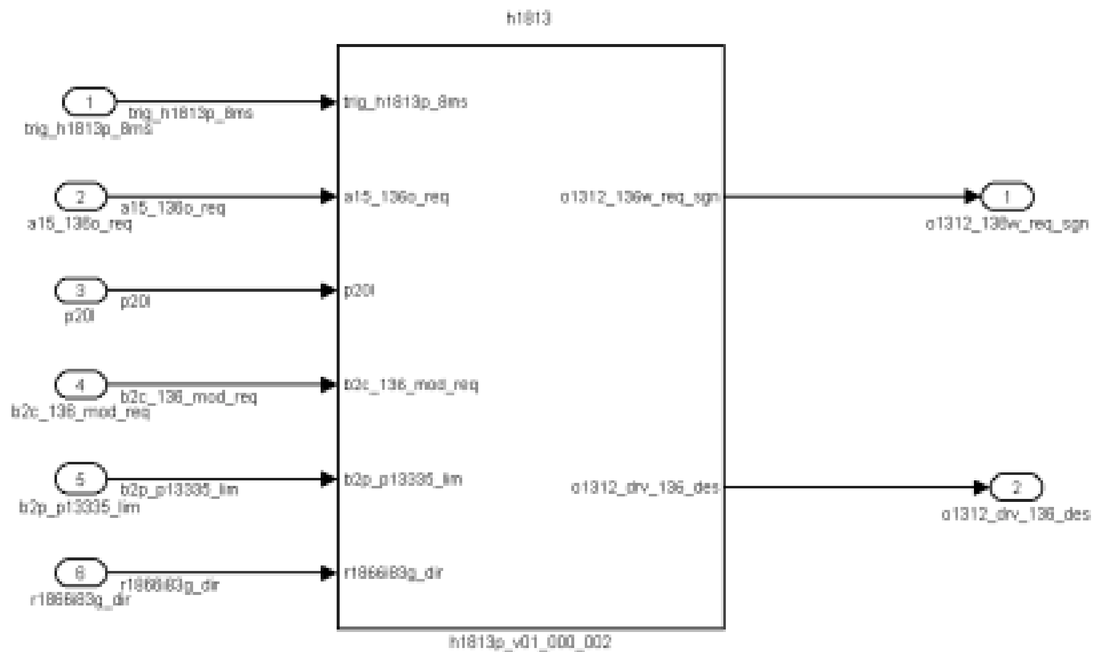
[9] Nordstrom G., Ledeczi A., "Formalizing the Specification of Graphical Modeling Languages," Technical Report ISIS-00-200, Institute for Software Integrated Systems, Vanderbilt University, 2000.

[10] Object Constraint Language, Object Management Group, http://www.omg.org/ .

[11] Vanderbilt University, "Matlab UDM," http://www.isis.vanderbilt.edu/Projects/mobies/filedownloads.asp , February, 2002.

8. APPENDIX A

Simulink® model component context:



Simulink® model component Characterization file:

```
% Definition for the ford test system

clear Definition

% Input definitions.

Definition.Input = struct([]);

Definition.Input(1).Name = 'trig_h1813p_8ms';
Definition.Input(1).Description = 'trigger signal';
Definition.Input(1).Unit = 'none';
Definition.Input(1).FloatingPointTargetType = 'function call';
Definition.Input(1).FixedPointTargetType.Signed = [];
Definition.Input(1).FixedPointTargetType.Bytes = [];
Definition.Input(1).FixedPointTargetType.BinPoint = [];
Definition.Input(1).Size = [1 1];
Definition.Input(1).Min = [];
Definition.Input(1).Max = [];
Definition.Input(1).Period = '8';

Definition.Input(2).Name = 'a15_136o_req';
Definition.Input(2).Description = [];
Definition.Input(2).Unit = 'metre';
Definition.Input(2).FloatingPointTargetType = 'double';
Definition.Input(2).FixedPointTargetType.Signed = 0;
Definition.Input(2).FixedPointTargetType.Bytes = 2;
Definition.Input(2).FixedPointTargetType.BinPoint = 2;
Definition.Input(2).Size = [1 1];
Definition.Input(2).Min = 0;
Definition.Input(2).Max = 1048;
Definition.Input(2).Period = '8';

Definition.Input(4).Name = 'b2c_136_mod_req';
```

```
Definition.Input(4).Description = [];
Definition.Input(4).Unit = 'enum';
Definition.Input(4).FloatingPointTargetType = 'boolean';
Definition.Input(4).FixedPointTargetType.Signed = 0;
Definition.Input(4).FixedPointTargetType.Bytes = 2;
Definition.Input(4).FixedPointTargetType.BinPoint = 2;
Definition.Input(4).Size = [1 1];
Definition.Input(4).Min = 0;
Definition.Input(4).Max = 1;
Definition.Input(4).Period = '8';

Definition.Input(5).Name = 'b2p_p13335_lim';
Definition.Input(5).Description = [];
Definition.Input(5).Unit = 'metre';
Definition.Input(5).FloatingPointTargetType = 'double';
Definition.Input(5).FixedPointTargetType.Signed = 0;
Definition.Input(5).FixedPointTargetType.Bytes = 2;
Definition.Input(5).FixedPointTargetType.BinPoint = 2;
Definition.Input(5).Size = [1 1];
Definition.Input(5).Min = 0;
Definition.Input(5).Max = 1048;
Definition.Input(5).Period = '8';

Definition.Input(3).Name = 'p20l';
Definition.Input(3).Description = [];
Definition.Input(3).Unit = 'Pascal';
Definition.Input(3).FloatingPointTargetType = 'double';
Definition.Input(3).FixedPointTargetType.Signed = 0;
Definition.Input(3).FixedPointTargetType.Bytes = 2;
Definition.Input(3).FixedPointTargetType.BinPoint = 2;
Definition.Input(3).Size = [1 1];
Definition.Input(3).Min = 0;
Definition.Input(3).Max = 1048;
Definition.Input(3).Period = '8';

Definition.Input(6).Name = 'r1866i83g_dir';
Definition.Input(6).Description = [];
Definition.Input(6).Unit = 'enum';
Definition.Input(6).FloatingPointTargetType = 'boolean';
Definition.Input(6).FixedPointTargetType.Signed = 0;
Definition.Input(6).FixedPointTargetType.Bytes = 2;
Definition.Input(6).FixedPointTargetType.BinPoint = 2;
Definition.Input(6).Size = [1 1];
Definition.Input(6).Min = 0;
Definition.Input(6).Max = 1;
Definition.Input(6).Period = '8';

% Output definitions.

Definition.Output = struct([]);

Definition.Output(1).Name = 'o1312_136w_req_sgn';
Definition.Output(1).Description = [];
Definition.Output(1).Unit = 'Pascal';
Definition.Output(1).FloatingPointTargetType = 'double';
Definition.Output(1).FixedPointTargetType.Signed = 0;
Definition.Output(1).FixedPointTargetType.Bytes = 2;
Definition.Output(1).FixedPointTargetType.BinPoint = 2;
Definition.Output(1).Size = [1 1];
Definition.Output(1).Min = 0;
Definition.Output(1).Max = 1048;
Definition.Output(1).Period = '8';

Definition.Output(2).Name = 'o1312_drv_136_des';
Definition.Output(2).Description = [];
Definition.Output(2).Unit = 'Pascal';
Definition.Output(2).FloatingPointTargetType = 'double';
Definition.Output(2).FixedPointTargetType.Signed = 0;
Definition.Output(2).FixedPointTargetType.Bytes = 2;
Definition.Output(2).FixedPointTargetType.BinPoint = 2;
Definition.Output(2).Size = [1 1];
Definition.Output(2).Min = 0;
Definition.Output(2).Max = 1048;
```

```
Definition.Output(2).Period = '8';

% Parameter definitions.


Definition.Parameter = struct([]);

Definition.Parameter(1).Name = 'h1813_cpu_usage';
Definition.Parameter(1).Description = 'percentage of CPU troughput used by h1813';
Definition.Parameter(1).Unit = '';
Definition.Parameter(1).FloatingPointTargetType = 'double';
Definition.Parameter(1).FixedPointTargetType.Signed = 0;
Definition.Parameter(1).FixedPointTargetType.Bytes = 2;
Definition.Parameter(1).FixedPointTargetType.BinPoint = 10;
Definition.Parameter(1).Size = [1 1];
Definition.Parameter(1).Min = 0;
Definition.Parameter(1).Max = 100;
Definition.Parameter(1).DefaultValue = 3;

Definition.Parameter(2).Name = 'h1813_ROM';
Definition.Parameter(2).Description = 'bytes of ROM used by h1813';
Definition.Parameter(2).Unit = '';
Definition.Parameter(2).FloatingPointTargetType = 'double';
Definition.Parameter(2).FixedPointTargetType.Signed = 0;
Definition.Parameter(2).FixedPointTargetType.Bytes = 2;
Definition.Parameter(2).FixedPointTargetType.BinPoint = 10;
Definition.Parameter(2).Size = [1 1];
Definition.Parameter(2).Min = 0;
Definition.Parameter(2).Max = 1000000;
Definition.Parameter(2).DefaultValue = 150000;

Definition.Parameter(3).Name = 'h1813_RAM';
Definition.Parameter(3).Description = 'bytes of RAM used by h1813, includes stack';
Definition.Parameter(3).Unit = '';
Definition.Parameter(3).FloatingPointTargetType = 'double';
Definition.Parameter(3).FixedPointTargetType.Signed = 0;
Definition.Parameter(3).FixedPointTargetType.Bytes = 2;
Definition.Parameter(3).FixedPointTargetType.BinPoint = 10;
Definition.Parameter(3).Size = [1 1];
Definition.Parameter(3).Min = 0;
Definition.Parameter(3).Max = 32000;
Definition.Parameter(3).DefaultValue = 300;

Definition.Parameter(4).Name = 'h1813_membership';
Definition.Parameter(4).Description = 'h1813 membership classifications';
Definition.Parameter(4).Unit = 'enum';
Definition.Parameter(4).FloatingPointTargetType = 'string';
Definition.Parameter(4).FixedPointTargetType.Signed = 0;
Definition.Parameter(4).FixedPointTargetType.Bytes = 2;
Definition.Parameter(4).FixedPointTargetType.BinPoint = 10;
Definition.Parameter(4).Size = [1 1];
Definition.Parameter(4).Min = 0;
Definition.Parameter(4).Max = 32000;
Definition.Parameter(4).DefaultValue = {'speed_density', 'mass_flow'};
```