Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

P. Bengtsson, H. Jansson, N. Pettersson, T. Sandberg
Scania CV AB, Sweden

Development of a Modelica Heavy Vehicle Modeling Library
pp. 467-476

Development of a Modelica Heavy Vehicle Modeling Library

Per Bengtsson Henrik Jansson Niklas Pettersson Tony Sandberg
Scania CV AB
151 32 Södertälje, SWEDEN

Abstract

Physical modeling for simulation of fuel consumption and other dynamic behavior in heavy vehicles can be useful in many areas from concept design to sales support. Similar models of vehicle subsystems are needed in many applications, it would thus be beneficial to have access to a library of reusable vehicle subsystem and component models to avoid repeated implementation. A solution based on a model architecture and a supporting Modelica library for structured storage of models and components is proposed. The work has been focused on promoting modeling practices enabling reuse, but we have also tried to maintain as much freedom as possible for the modeler.

1 Introduction

There exist a number of different proposals for vehicle modeling architectures in Modelica (for example [5] and [2]. The aim of this project has been to create a complete system with both a hierarchical model structure defining the interfaces between subsystems on several levels, and a model library. The library is used to store sub-system interfaces along with available implementations and required supporting components such as connector definitions. The system is intended to be used for various research and development efforts within Scania CV AB. Since development projects may have very different aims, and be focused on different subsystems, it is unlikely that the library will provide a final model for the task. Hopefully, existing versions of most sub-systems can be used together with new models specific to the current problem. A key consideration in the work has been to build a system which is suitable for use by both experienced and novice modelers. The project is rather applied in its nature, and the article is intended to describe our experiences.

2 Architecture Concerns

2.1 Multi-domain Library

One much hailed property of the Modelica language is its multi-domain modeling capability. Components from model libraries describing different domains can be used together in the same model. However, the majority of available libraries are focused on one domain. In most cases this is a natural partitioning. In this project the common denominator has not been the engineering domain, but rather the system to be described. The purpose of the library is to store component models, defined through the partitioning of the described system into physical sub-systems.

A design goal has been to keep all available models in one central location, easily accessible to everyone. Existing models use an in-house media library to represent air- and coolant flows. This domain specific library is thus also needed by users of the new model library. It was decided to place it within the new library. This issue is further discussed in section 4.2.3

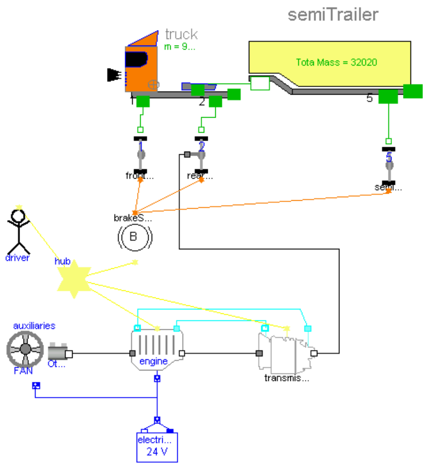


Figure 1: Heavy truck model in Dymola^{TM}.

2.2 Heavy Vehicles

Most existing vehicle model libraries are designed primarily for cars. Heavy vehicles have a number of subsystems which are not present in passenger cars. Particularly the engine/transmission system includes devices like an exhaust brake and possibly a retarder. Further, the cooling system also has a more prominent role than in cars, and coolant is often used both by the engine and the transmission.

2.3 Model Variants

Since this library is designed to be used in many different projects, there is a need to handle different variants of component models. Supporting different model variants, while attempting to preserve compatibility and avoid hidden interdependences has been one of the greatest challenges in this work. In some cases it will be necessary to create an entirely new model to be used instead of one already in the library, but substantial changes in behavior can be achieved without going that far. The library supports three ways of changing model behavior, listed in the order they should be considered.

Some models depend on data found in external parameter files or lookup tables. Theses can easily be changed at run-time without any need to modify or recompile the actual model. When models are added, this approach should be considered for any data that needs to be changed frequently. Recompilation between simulation runs is not only time consuming, it also assumes the presence of a licensed installation of the compiler.

Numerical parameters which are not set through data files can still be influenced at run-time. The simulation reads an initial state file, where values different from the default ones can be specified for real, integer and Boolean model parameters. This solution requires less complex source code than the data file approach, and is advantageous when only a few parameters need to be accessible. This approach would mainly be useful in creating an end-user application where the user should for example be allowed to choose between different tire models in an external GUI.

When a new model structure is needed, and even redeclaration of submodels is not enough, an entirely new model should be created. To make the new model usable in other projects, the existing base classes should be used to define the interfaces. If it is necessary, additional base classes can be created to supply extra connectors. The new model should of course be documented and made as flexible as possible with parameters and replaceable components used appropriately.

We have designed the library with fundamental base classes as blueprints for the physical subsystems and their major components. Only the interfaces required for simple implementations of the models are included. Additional base classes can be used to add more connectors if required by more advanced models. We have opted not to define a completely fixed architecture where all connections are always identical, but rather a supporting framework for developers intending to create reusable models. See also section 3.2.

2.4 Signaling Bus

A key issue in an architecture which contains both physical plant and controller models is the handling of electrical signals. The controllers need to exchange data among themselves and they need to exchange signals with sensors and actuators. For our applications the actual signaling behavior is not that important, an ideal communications model is sufficient. For the communication between a plant and its controller, standard library inports and outports are used. The communication between the controllers was a tougher case. Two implementations of the same controller may not have the same signaling needs, thus it must be possible to change the set of signals sent between control units.

Separate input and output ports for all links between control units in the vehicle would create an undecipherable graphical mess. Some type of signaling bus is needed. Both the standard library bus connectors and the type of bus used in the vehicle modeling architecture proposal by Tiller et. al [5] were evaluated. We did not find enough information about the inter-controller communication in the Tiller paper to implement that system. Our main problem was to find a way of having compatible connectors in all controllers, without modifying the code of every controller when a signal was added to the bus. The Modelica standard library bus does not solve that problem, since it requires all signals to be declared in the connector. Eventually we chose a simpler solution based on a common connector called ”CAN” with a replaceable variable, called ”protocol”, which contains all the signals. The protocol variable can easily be redeclared into a type which contains exactly the signals broadcast on the bus in a particular model. Different implementations of the CAN connector are used for different signal buses in the vehicle.

Most of our control units are implemented through external function calls, thus the drawback of having no convenient graphical way of converting a signal from inport/outport to bus format is minor. See listing 4 for an example of how the electronic control unit models use the CAN bus. The connected control unit model is shown in figure 2.

Listing 1: The CAN connector base class.

```
partial connector CANBase
  "Basic connector for modelling CAN
  comunincation"
  replaceable Protocols.Interfaces
    .ProtocolBase protocol
    "Protocol to be used";
end CANBase;
```

Listing 2: Implementation of the general CAN bus connector.

```
connector CAN_s
  "General control system
    communication bus connector"
  extends CANBase;
  annotation (...);
end CAN_s;
```

Listing 3: A part of the definition of a CAN protocol.

```
record ProtocolStd
  extends Interfaces.ProtocolBase;
  Real EngineSpeed
    "Speed of engine in rpm";
  Real EngineTemp
    "Engine temperature in deg C";
    ...
end ProtocolStd;
```

Listing 4: Sample usage of the CAN bus in the engine control unit.

```
...
/* I/O mapping (sensors/actuators)*/
engineSpeed = inport[ENGINE_SPEED];
outport[FUELING] = fueling;
outport[EXHAUST_BRAKE_ON] =
  CAN.protocol.ReqExhaustBrake;
...
/* Write CAN values */
CAN.protocol.EngineSpeed = engineSpeed;
CAN.protocol.EngineTemp = engineTemp;
CAN.protocol.ActualEngineTorque =
  inport[ACTUAL_ENGINE_TORQUE];
CAN.protocol.ActualExhaustBrakeTorque =
  inport[EXHAUST_BRAKE_TORQUE];
...
```

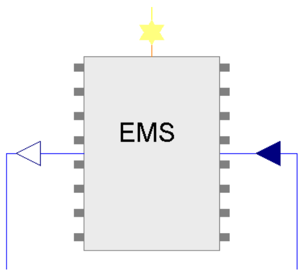
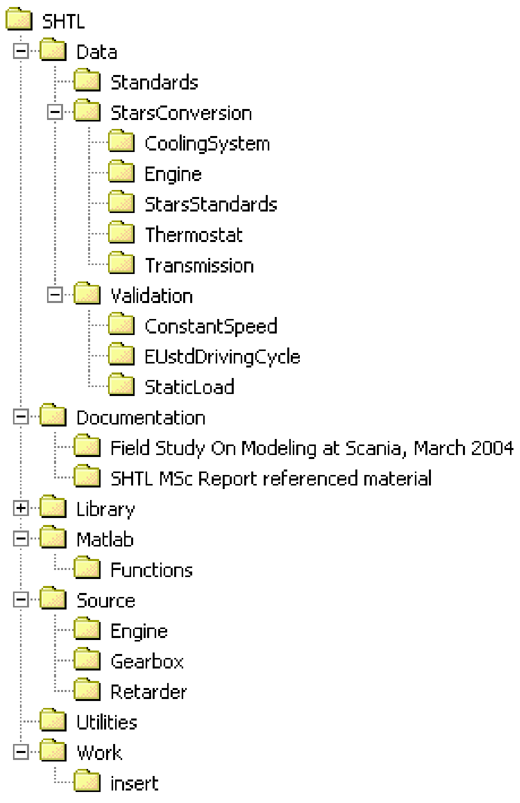


Figure 2: Engine management system electronic control unit.



```
SHTL
├ Data
│ ├ Standards
│ ├ StarsConversion
│ │ ├ CoolingSystem
│ │ ├ Engine
│ │ ├ StarsStandards
│ │ ├ Thermostat
│ │ └ Transmission
│ └ Validation
│   ├ ConstantSpeed
│   ├ EUstdDrivingCycle
│   └ StaticLoad
├ Documentation
│ ├ Field Study On Modeling at Scania, March 2004
│ └ SHTL MSc Report referenced material
├ Library
├ Matlab
│ └ Functions
├ Source
│ ├ Engine
│ ├ Gearbox
│ └ Retarder
├ Utilities
└ Work
  └ insert
```

Figure 3: Directory structure for non-model files.

3 Usability Concerns

3.1 Non-model Files

The model library itself is relatively easy to distribute to the users. It is sufficient to copy the directory structure containing all the models to an appropriate place in the file system and load the library into Dymola. To get a nice working environment, where all external files are found by the system and the library is included in the Dymola GUI, takes a little more effort. In addition to the library directory tree our completed system consists of a directory structure on a higher level in the file system. This tree contains the model library itself and all external function source code and model data files for the. A work directory for the user is also provided. Data file paths and external source code links (through include annotations in functions) in the model library are given with relative paths, making sure that the files are found if the work directory is used.

If the user starts Dymola with the script provided with the library, an included configuration file is used to make sure that the heavy vehicle library automatically is included in the model browser.

3.2 Top Level Model

It is anticipated that a significant part of the work done with the library will be carried out in project form by people with little or no previous Modelica and/or modeling experience. As a result overly complicated and abstract language constructs have been avoided. The suggested method of putting together a vehicle for a particular task is to select those sub-system models which are most suited and add them to a new model. A ”master” model with most components declared as replaceable would enable new versions to be created with fewer lines of code, but the added abstraction has been deemed to be more difficult to handle than the extra coding. Future enhanced modeling tools may reverse this decision, but today we think that the ability for the novice modeler to fully understand his or her source code is warrants some code duplication.

The master model approach is best suited when it is anticipated that all implementations of a sub-system will be absolutely compatible. While this is a nice assumption we don't think that it will be valid in our case. The range of intended applications for the vehicle library is so broad that some modifications also to the structure and interfaces certain components will be necessary in many projects. Real-time simulation of components as parts in simulink models is one example where non-standard shortcuts have been very efficient. Strict adherence to the interfaces is certainly optimal from a reusability perspective, but we have not yet found a set of interfaces which have been practical to use in every application. Further work may bring us ever closer to that goal.

3.3 Concurrent development

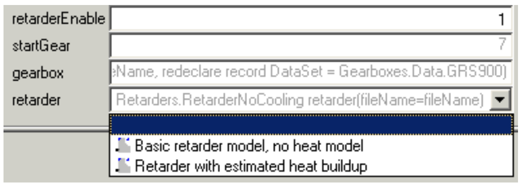
Traditionally one of the main obstacles to reuse of Modelica models created in previous projects has been that new functionality has been spread through many subsystems, rather than contained in one. When developers in two projects have enhanced different subsystems they also have modified many others in the process, making it difficult to incorporate enhanced components from different projects in the same model. Hopefully, the new library will promote solutions where components to a higher degree are created as self contained units.

A version control repository is used to make sure that two different groups do not accidentally make simultaneous conflicting changes. If only a particular sub-tree is checked out with write privileges the developer is encouraged to find solutions within that subsystem. The version control repository also supports named versions of the library to be created, which is useful to make sure that the exact model versions used in a project or application will always be available. The model library code is managed and provided to users in the same way as other source code in the organization. Users across various departments and groups can thus use a code management system which they are already accustomed to.

3.4 Choice Annotations

A number of Modelica entities can have choice annotations, which allow the model user to select appropriate parameter values easily in a modeling GUI. This feature has been used in many places where the parameter is not a physical quantity. For examples file name parameters used to specify data files are declared to be of a certain filetype. Each filetype has an associated list of suggested file names. In figure 4 the dropdown box for retarder model selection is shown.

The signal bus protocol used throughout the vehicle is determined by the type of a replaceable variable. The final setting of the type is propagated to the vehicle level, making it easy to change the type of every bus connector. Available protocols are presented in a



retarderEnable	1
startGear	7
gearbox	eName, redeclare record DataSet = Gearboxes.Data.GRS900)
retarder	Retarders.RetarderNoCooling retarder(fileName=fileName)

Basic retarder model, no heat model
Retarder with estimated heat buildup

Figure 4: The result of a choice annotation to aid in the selection of a retarder model

list, but it is still up to the user to choose one which contains exactly the set of signals that is being broadcast by the currently used set of control units.

4 Model and Library Structure

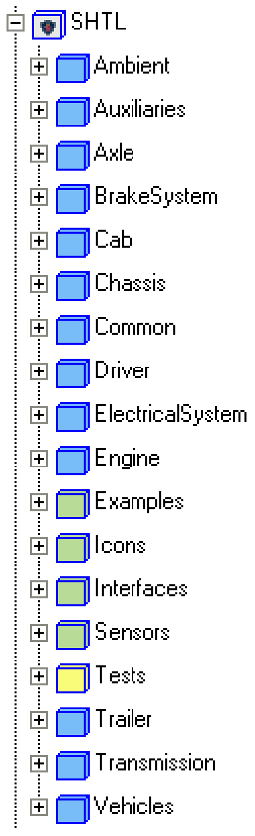


Figure 5: Top level packages in the vehicle library

4.1 Hierarchical levels

The created vehicle model structure defines multiple levels. The vehicle is built from physical components which also have a defined substructure. The supporting library mimics that same substructure. For example control units for subsystems are included in the subsystems modules themselves, to reduce the required number of components and connections in the top level model. In most cases one controller model and one or more interconnected plant models make up a subsystem. The top level subsystems cover the same areas as corresponding groups in the research and development organization. Local development of models by experts in various fields is thus simplified.

4.2 Library Structure

4.2.1 Color Coding

The vehicle library has been created with the same basic structure and package naming conventions as the Modelica standard library. Additionally the various package types have been color coded to make navigation in the package tree easier. Packages containing interfaces, sensors, icons and examples are made green. Data records are kept in red subpackages, and test models in yellow. Ordinary packages are a slightly darker shade of blue than the standard package icon.

4.2.2 Packages

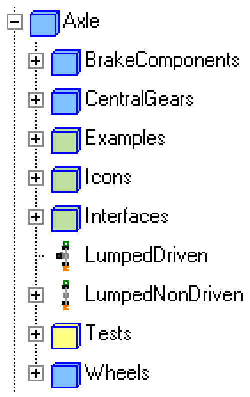


Figure 6: The ”axle” package with subpackages in the library browser.

Each physical subsystem has its own top level package for all its components, interfaces, etc. Additionally there are packages for examples, interfaces, icons, examples and tests on the top level. For an example of a top-level package see figure 6. Complete vehicle models, which can be used as components together with environment models from the ambient package, have a category of their own. To avoid a very deep tree structure these various types have been put at the same level in the hierarchy.

4.2.3 The Common Sub-tree

There are a number of models required for the modeling of a complete vehicle which do not clearly belong in any particular subsystem. Base classes for electronic control units, the signal bus connectors and the media library used for coolant and air modeling are a few examples. These functions are kept under a sub-tree called ”Common”. While it may seem more natural to create a separate library at least for the media components we prioritized keeping the entire model library self contained. The only external dependencies are the Modelica Standard Library and accompanying ModelicaAdditions library.

4.3 Physical Subsystems

The physical subsystems with the interfaces described are used in the current models. As it is impossible to foresee exactly what applications the vehicle library will be used for in the future, unused connectors are not included on speculation. More connectors are likely to be added in the future. To preserve current base classes this can be done through additional base classes as described in section 2.3.

4.3.1 Ambient

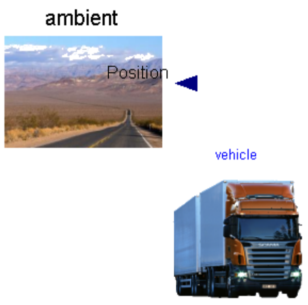


Figure 7: A vehicle model with the ambient component.

The ambient category is used to represents the environment around the vehicle. Models of this type supply data about surrounding temperature, air pressure and other environment constants. These models are also used to keep track of the road parameters such as slope and speed limit. Through the Modelica inner/outer construct one ambient component is accessible to all the other components in a model, which makes this the ideal place for any data that needs to be globally shared. Each model should have exactly one ambient component. The ambient interface has one input connector which is used to communicate the position of the vehicle. A vehicle and ambient combination can be seen in figure 7.

4.3.2 Auxiliaries

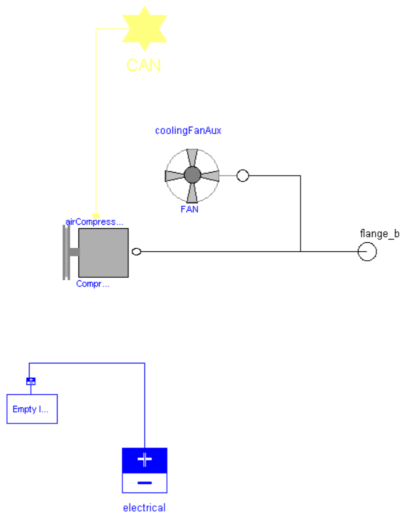


Figure 8: A model representing the auxiliary units.

Auxiliaries are components like the cooling fan, AC compressor, electric generator etc. These are generally mounted somewhere on the front side of the engine, and traditionally obtain their operating power through a mechanical link. It is recommended that any generator model is placed among the auxiliaries, and not in the electrical system. Separate models for each auxiliary unit are placed within this container. Subtrees in the library contain models related to the various units. The auxiliaries are connected to the electrical system and the engine, an implementation can be seen in figure 8.

4.3.3 Axle

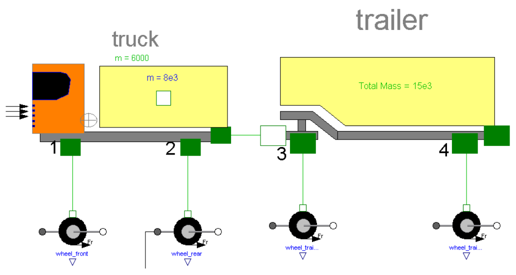


Figure 9: Truck model with full trailer and four axles.

The axle models contain tires, brake actuators and a final gear for driven axles. They are always connected to the chassis and the brake CAN bus. The axles generate a retardation force due to the rolling resistance in the wheels, and possibly due to the wheel brakes. Driven axles have an additional connector which allows the power train to transfer torque to the axle, and propel the vehicle. The number of axles in a vehicle configuration varies; it has to match the number of axle connectors on the chassis and all trailer models used. The vehicle shown in figure 9 thus requires four axles. An air interface could be added to simulate air-powered brake actuators.

4.3.4 Brake System

The brake system is a container for the brake management system and any related plants except for the brake actuator, which are represented at their physical location in the axle modules. The brake system is attached to the vehicle signaling bus and the brake signaling bus. Additional connections are likely in future more developed brake system models.

4.3.5 Chassis

The chassis models represent the frame of the truck. Cargo, axles and trailers attach to the chassis. The base class has the connectors for wheel axles and a draw bar. Derived classes add either a fifth-wheel (where the semi-trailer is attached) for tractor configurations, or a cargo attachment point for rigid trucks.

4.3.6 Driver

The driver model is responsible for overall control of the vehicle. Decisions to accelerate or decelerate depending on the surroundings are made by the driver. Depending on the vehicle, different control signals may be required. A manual transmission requires the driver to select an appropriate gear, while the GMS handles that duty for an automatic or automated manual transmission. In cruise control mode, the EMS controls the fueling, in driver demand mode the throttle is controlled directly by the driver. The driver logic depends on the control units used in other parts of the vehicle. The driver interface is very simple with only one connector, which is used to attach it to the CAN bus.

To control a vehicle from an external model (e.g. in Simulink) input and output ports are needed. In such a case input and output ports could be added to the driver interface. The driver model would then take the commands received from the external input and convert them to appropriate CAN signal values. The resulting action of the truck would be sent back through the output port.

4.3.7 Electrical System

The electrical system is included to enable studies of electrical energy flows. It could for instance be used to study battery operating conditions or effects of using electrically powered accessories instead of mechanically powered ones. The electrical system package is primarily intended for components without a simulated direct mechanical connection to the vehicle. The interface specifies a single special connector, see section 4.4

4.3.8 Engine

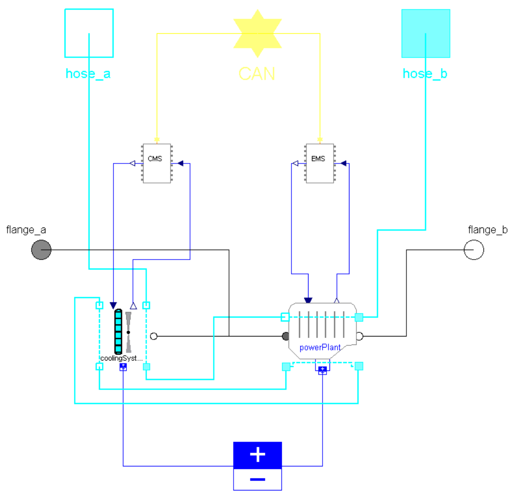


Figure 10: Engine with cooling system.

The engine (figure 10) is one of the larger subsystems, particularly when cooling system behavior is taken into account. The engine model contains submodels for both the power plant itself and any radiators and other cooling system components found in close proximity to it. The power plant model often includes an exhaust brake. The engine connectors required differ depending on the aspects considered. The base interface defines rotational mechanical connections to the auxiliaries and the transmission. The engine is connected to the vehicle signaling bus and the electrical system. An additional base class provides coolant hoses which allow coolant to flow in a circuit through other vehicle systems.

4.3.9 Trailer

The trailer is in many ways rather similar to the chassis. They are both rigid bodies represented as point masses with forces acting on them through translational connectors. The base model is a semi-trailer which can be attached to the back of a tractor. Through the use of a dolly, the semi-trailer gets a second axle and can be used as an independent full-trailer, this configuration is shown in figure 9. A trailer connects to two axles, a towing vehicle and possibly another trailer. A semi-trailer connects to one axle, a tractor or dolly, and can be used to pull a trailer.

4.3.10 Transmission

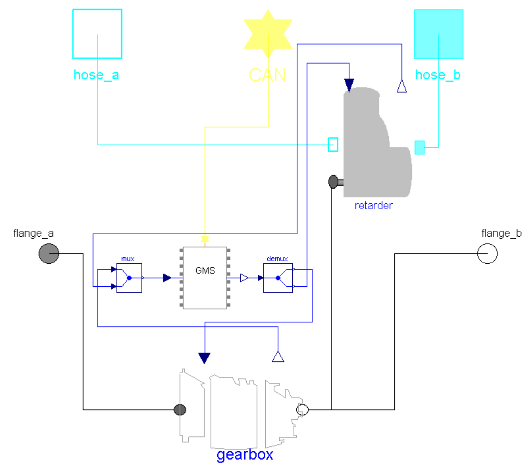


Figure 11: Transmission model with coolant flow.

The transmission includes a gearbox, which can be of any type. There is also a gear management system which has to be compatible with the gearbox used. A retarder, a type of hydraulic brake which acts on the drive shaft, is often included as well. The retarder generates a lot of heat, and a complete cooling system representation needs a connection to it. A transmission model with coolant flow is shown in figure 11. The Transmission has rotational mechanical connections to the engine and any driven axles. It is also connected to the vehicle signaling bus. An optional base class provides coolant hoses.

4.4 Special Connectors

Whenever it has been possible standard library connectors have been used in the models. In some places however we have seen the need to use special connectors which can carry all the information of a certain type between two components with only one connect statement. This has been the case for the signal bus and the electrical connectors. The electrical connector contains two electrical pins. All components connected to the electrical bus are connected in parallel, and adapters are used between the standard electrical pin connectors and our electrical connector. In the future it is envisioned that the electrical system model may contain multiple circuits and voltage levels in the electrical connector, making the advantages of using it higher. The single voltage electrical bus connector is shown in listing 5.

Listing 5: Electrical bus connector with single voltage level.

```
connector ElectricalSingleVoltage
  "Connector class for a single voltage
    electrical system"
  extends Interfaces.ElectricalBase;

  Modelica.Electrical.Analog.Interfaces
    .PositivePin p;
  Modelica.Electrical.Analog.Interfaces
    .negativePin n;
end ElectricalSingleVoltage;
```

The signal bus connector (also described in section 2.4) allows for relatively simple transfer of many (currently around 20) control signals between the various electronic control units in the vehicle. There is a second signal bus, using another identical connector except for the color and protocol, used to connect the brake system to the actuators on the axles. The media library used for the cooling system contains general hose connectors which can carry the simulated media.

5 Sample Vehicle Models

To validate the new architecture two slightly different vehicle models were used in the new framework. Both models share the same overall structure, but one version contains a thermodynamic cooling system representation (this model is seen in figure 1), while the other has no cooling system model at all (figure 12). The two models require different controllers and plant models for the engine and transmission. These are the systems which are affected by the inclusion of coolant flow. All other controllers and plant models are identical in the two versions.

The two sample models illustrate the idea of this vehicle model architecture very well. It isn't possible to generate the two versions from the same ready-

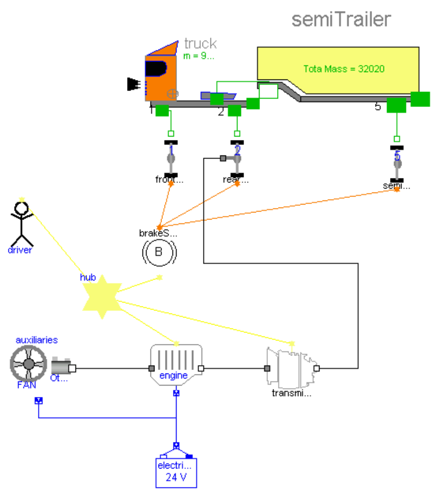


Figure 12: Heavy truck model without any cooling system.

made template only through redeclare statements. On the other hand, the same base classes and connectors are used for all common interconnects. In this way vastly differing projects can still use some of the same components. We hope that this middle ground between a completely rigid architecture and a collection of independently created models with some similarities but many differences will prove useful.

In this particular case it would of course be rather easy to create a common vehicle model, which through replaceable subsystems could be used to generate either model. However, we do not know enough today about what will be required tomorrow to incorporate all possible variations. Thus we have chosen to let the modelers of the next project and the one after that decide how to best handle their particular challenges.

The vehicle model with cooling system representation give rise to a non-linear equation system which needs to be solved iteratively at every time step. This makes simulation very slow, a performance hit of about 100 times compared to the simpler model was observed. The vehicle without a cooling system can be simulated four hundred times faster than real time on a standard laptop (Pentium^{TM}M, 1.6 GHz, 1 Gb ram). The performance difference alone is ample justification to have two vehicle model variants.

6 Limitations

During our work we have faced many compatibility issues which cannot be directly attributed to model design. Phenomena may in themselves lead to varoius solver requirements (such as the cooling system case). Existing PID controllers in external code often have strict sample time needs. Each of these special considerations needs to be explicitly stated in documentation. Documenting every important aspect is a real challenge.

Novice users of Modelica and Dymola often have trouble deciphering the error messages that are output. A nearby expert who can guide through the minefield of rookie mistakes is an invaluable asset for anyone new to the field. Similarly it is anticipated that our library will require some previous knowledge of the tools, despite our intentions to make it simple to use.

7 Conclusion

Our proposed library imposes less rigid structural control than most other vehicle architectures. Generally it does not include connector which are not used. An effort has been made at creating something which works well in the local situation. The system is not primarily intended for exchange of models with external developers. Attention has also been given to a number of practical issues related to working (updating, installing etc.) with the system.

There are many similarities between this work and the vehicle model architecture (VMA) project [6]. With some modifications many of our components could be used in VMA utilising appropriate wrappers. One key difference is the localisation of control units. We have chosen to place them inside the subsystems they control, while the VMA places them at the top level. The future development of the VMA project will be observed with great interest.

8 Future Work

We foresee a continued study of the actual use of the new models to find areas where the architecture can be improved. Inclusion of a wider range of component models is also a likely continuation. More documentation and tutorials to aid users would be beneficial.

The work described was carried out with Dymola 5.1b, but the project has since been upgraded to version 5.3b and Modelica Standard Library 2.1. Certain components have also found use in real-time hardware-in-the loop simulations using Simulink.

The recently proposed relaxations of the connector equivalency requirements in Modelica opens up interesting options for improved handling of the CAN communication.

9 Acknowledgements

The described model and library architecture has been developed based on experiences from the usage of a previous Modelica complete vehicle modeling effort aimed at fuel consumption estimation. This effort is described in a licentiate Thesis by Tony Sandberg [4]. Experiences from the work on modeling auxiliary systems by Niklas Pettersson have also been valuable [3]. Many of the subsystem models in the reference vehicle model have been converted from versions used in those projects. The work on the new library and model has been carried out as a M.Sc. thesis project by Per Bengtsson under the supervision of Niklas Pettersson [1].

References

[1] Bengtsson P. Structuring of Models Intended for Complete Vehicle Simulation. Uppsala, Sweden: Master's thesis, Division of Systems and Control, Dept. of Information Technology, Uppsala University, Dec 2004.

[2] Laine L. and Andreasson J. Modelling of generic hybrid vehicles. *Proceedings of the 3rd International Modelica Conference*, pages 87–93. The Modelica Association, 2003.

[3] Pettersson N. Modelling and Control of Auxiliary Loads in Heavy Vehicles. Stockholm, Sweden: Licentiate thesis, Dept. of Signals, Sensors and Systems Royal Institute of Technology, 2004.

[4] Sandberg T. Heavy Truck Modeling for Fuel Consumption: Simulations and Measurements. Linköping, Sweden: Licentiate thesis, Dept. of Electrical Engineering, Linköping University, 2001.

[5] Tiller M., Bowles P., and Dempsey M. Development of a vehicle model architecture in Modelica. *Proceedings of the 3rd International Modelica Conference*, pages 75–85. The Modelica Association, 2003.

[6] Vehicle Model Architecture, developed at Ford Motor Company, http://www.modelica.org/projects/vma. 2005-01-31.