Equation-Based Model Data Structure for High Level Physical Modelling, Model Simplification and Modelica-Export

Hisahiro Ito[1,*] Akira Ohata[1] Ken Butts[2,**]
Jürgen Gerhard[3,***] Masoud Abbaszadeh[3] David Linder[3] Erik Postma[3] Elena Shmoylova[3]

[1]Toyota Motor Corporation, 1200 Mishuku, Susono, Shizuoka, 410-1193, Japan
[*]ito@hisahiro.tec.toyota.co.jp
[2]Toyota Technical Centre, 1555 Woodridge Avenue, Ann Arbor, Michigan, 48105-9748, USA
[**]ken.butts@tema.toyota.com
[3]Maplesoft, 615 Kumpf Drive, Waterloo, Ontario, N2V 1K8, Canada
[***]jgerhard@maplesoft.com

Abstract

This paper proposes a novel data structure for equation-based plant models. The data structure facilitates the plant modelling process starting from physics-based component creation through model simplification to Modelica model export.

Keywords DAE, HLMD, HLMT, Maple, Modelica, Model Simplification, MSModel, Symbolic Manipulation

1. Introduction

The importance of plant models has been increasing in the automotive industry where control systems must address more stringent requirements for emissions, fuel economy, and functional safety than ever before. Moreover, competitive business pressures dictate that development time and effort are effectively managed. One key strategy to meet these challenges is to embrace Model-Based Development (MBD), thereby enabling concurrent development of the plant and controller subsystems as opposed to the conventional serial process where the plant is developed first and the controller second.

Concurrent plant–controller development implies that there is no hardware (i.e. target plant) available for experiment. Thus, in order to service controller development, it is critical that the plant model can be developed in a timely manner. Although there are many plant model libraries available, they are not (and will never be) sufficient to meet the new and more challenging requirements that arise from new and more sophisticated control system development projects. As a consequence, we conclude there is always a need to create new plant models.

4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. September, 2011, ETH Zürich, Switzerland.
 Proceedings available at:
http://www.ep.liu.se/ecp/056/
EOOLT 2011 website:
http://www.eoolt.org/2011/

The traditional approach to build a plant model is to first construct a set of equations to describe the dynamics of the system. However, in this approach, it is the author's responsibility to ensure that the model adheres to physics-based principles such as conservation and it is very difficult for other people to check the quality of the model (i.e. conformance to physics-based principles), especially when the model is large and complex.

In order to overcome this problem, we developed the High Level Modelling (HLM) framework [1, 5] wherein a formalized and physics-based plant-model-development process is defined. If one follows this process, a proper description of the system in question can be created. We call this description the High Level Model Description (HLMD). With HLMD, those who are not the author of the model can readily review and critique the design of the model. In addition, HLMD-based models are easier to reuse and/or modify than traditional equation-only models due to the clear exposition of design intent.

To verify the feasibility of our HLM framework, we also developed a software package called the High Level Modelling Tool (HLMT). With HLMT, one can create an HLMD of a system by following the formalized modelling process and successively run a simulation. Due to the nature of HLMD where the system is represented in highly redundant, non-linear differential algebraic equations (DAE), Maple's [3] symbolic manipulation technology is used to derive simulatable equations. To date, several application models of various physical domains with different levels of complexity have been successfully created and simulated in HLMT. These models range from simple electric circuits to an internal combustion engine with crankshaft angle resolved gas flow and mechanical dynamics.

With the aim of streamlining the plant modelling process even further, we have developed a generalized data structure called `MSModel` to accommodate information about an equation-based plant model. While this data structure can store information generated in HLMT, it is designed to be flexible enough so that one can efficiently ap-

ply a variety of model simplification methods to the model. Furthermore, `MSModel` contains sufficient information to export the model to the Modelica language, which is becoming one of the most widely accepted plant modelling languages [4].

This paper is organized as follows. In Section 2, the HLM framework is briefly introduced. In Section 3, the `MSModel` data structure is explained with some examples. The full specification of the `MSModel` data structure is described in Appendix A.

2. High Level Modelling Framework

As mentioned in Section 1, the HLM framework was developed to enable peer review of the design of a plant model from the standpoint of adherence to physical principles and to realize rapid modelling. In this section, HLM is briefly introduced.

2.1 Formalized Modelling Process and HLMD

In the HLM framework, the process to create a plant model is formalized as follows:

1. Partition the system in question into components
2. Define conserved quantities (CQ) in each component
3. Connect CQs to specify how they flow from one component to another
4. Set physical constraints as needed

By following this process, a physical description called HLMD of the system can be created.

As an example, let us consider a chemical reaction process in a closed chamber where the mixture of hydrogen gas and oxygen gas is burned, and water and heat are generated (Figure 1).

$$H_2 + O_2 \rightarrow H_2O + \frac{1}{2}O_2 + c\,[J]$$

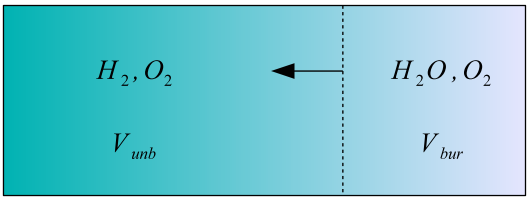


Figure 1. Combustion inside a chamber.

In this case, homogeneous mixing is assumed, and also the combustion is assumed to occur at a thin layer as indicated in Figure 1 by the dashed line traveling inside the chamber from right to left. When modelling of this system is considered, the following constraints apply:

$$p_{\text{unb}}(t) = p_{\text{bur}}(t)$$
$$V_{\text{unb}}(t) + V_{\text{bur}}(t) = V_0 \qquad \text{(const)}$$

where $p_{\text{unb}}(t)$ and $p_{\text{bur}}(t)$ are the pressures of the unburned and burned gas, respectively. $V_{\text{unb}}(t)$, $V_{\text{bur}}(t)$ and V_0 are the volumes of the unburned gas, burned gas, and combustion chamber, respectively.

The HLMD of this system is shown in Figure 2 where a component (e.g. “unburned”, “flame front” and “burned”) is represented as a square with rounded corners, a CQ is represented as a circle, a flow of CQ is represented as an arrow whose direction specifies the sign convention, and constraints are represented as a square with dashed lines connected to components.

Equations appearing in the HLMD are assembled to form the system equations. (Note that some of the equations are omitted in Figure 2 for the sake of simplicity.) For example, we have the number of moles of generated water molecule when the conservation law is applied:

$$\frac{d}{dt}N_{H_2O,\text{ff}}(t) = n_{H_2\text{to}H_2O,\text{ff}}(t) + n_{O_2\text{to}H_2O,\text{ff}}(t) - n_{H_2O,\text{ff}}(t) - e_{\text{ff}}(t)$$

The HLM framework allows the use of intermediate variables in addition to CQs and flows. For example, the average degrees of freedom of the unburned gas $f_{\text{unb}}(t)$, and burned gas $f_{\text{bur}}(t)$, the gas pressure $p(t)$ which is used instead of either $p_{\text{unb}}(t)$ or $p_{\text{bur}}(t)$, as well as $V_{\text{unb}}(t)$ and $V_{\text{bur}}(t)$, are defined as intermediate variables, from which the energies of the unburned gas $E_{\text{unb}}(t)$, and burned gas $E_{\text{bur}}(t)$, are computed as CQs:

$$E_i(t) = \frac{f_i(t)}{2} \cdot p(t) \cdot V_i(t)$$

where i is “unb” or “bur”.

Since the system equations assembled from HLMD are highly redundant and nonlinear, some pre-processing is necessary in order to perform a simulation (i.e. numerical integration). Redundancy removal is obviously needed, but we also need to deal with non-linearities which may lead to multiple solutions. Consequently, a more sophisticated symbolic manipulation algorithm must be used to derive the physically meaningful single solution. Although we have a working algorithm, we consider that this is one of the most challenging aspects of the HLM approach, and, thus, it is still a research topic to improve the robustness and scalability of the multiple solution algorithm.

Apart from the mathematical aspects of the HLMD, the purpose of creating an HLMD is to enable peer review on the design of the model at the physics level. For example, by looking at the HLMD in Figure 2, one can notice that no interaction between gas and chamber is considered. Although it is possible to make such an analysis by only examining the equations, what is done in such a case is to construct a kind of HLMD in the mind. Also, the acceptability of the current model design depends on the purpose of the model. The HLMD allows a physics level examination in an efficient manner.

2.2 Tool Support for HLMD

To evaluate the feasibility of the HLMD and its mathematical framework, a software package called the HLMT was developed by Maplesoft and Toyota. The tool is still a prototype and Toyota owns the intellectual property. With HLMT, one can author an HLMD and perform simulation.

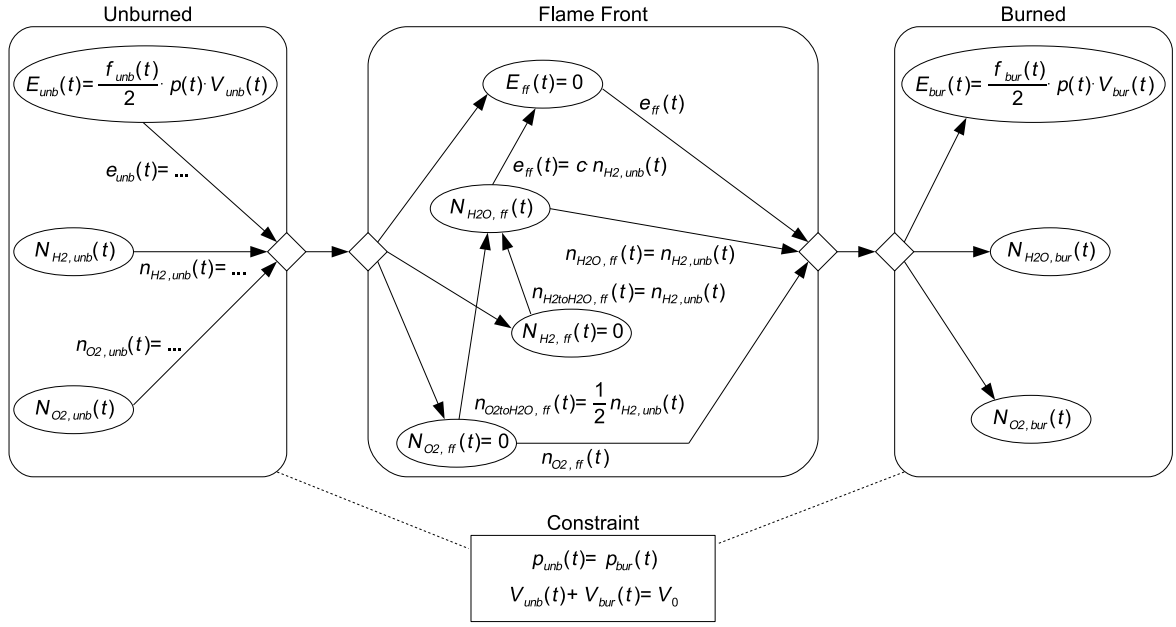


Figure 2. High Level Model Description (HLMD) example - hydrogen-oxygen combustion in a closed chamber.

HLMT also allows export of the model at the equation level to Maple where symbolic model manipulation and numerical simulation services can be applied.

A simulation result of the combustion model explained above is shown in Figure 3. It can be confirmed that the two algebraic constraints, one for pressure and the other for volume, are met.

By exporting the model from HLMT to Maple, some equation-level information about the model can be obtained. For example, it can be known that, after our simplification algorithm derived a single simulatable solution, this model has 11 differential equations. Seven of those are explicit for 7 differential variables $\frac{d}{dt}p(t)$, $\frac{d}{dt}N_{H_2,\mathrm{unb}}(t)$, $\frac{d}{dt}N_{O_2,\mathrm{unb}}(t)$, $\frac{d}{dt}N_{H_2O,\mathrm{ff}}(t)$, $\frac{d}{dt}N_{H_2O,\mathrm{bur}}(t)$, $\frac{d}{dt}N_{O_2,\mathrm{bur}}(t)$ and $\frac{d}{dt}f_{\mathrm{bur}}(t)$. Four other differential equations are implicit interms of 1 differential variable $\frac{d}{dt}V_{\mathrm{unb}}(t)$, and 3 algebraic variables $e_{\mathrm{unb}}(t)$, $n_{O_2,\mathrm{unb}}(t)$ and $n_{O_2,\mathrm{bur}}(t)$. It can also be confirmed that there is 1 DAE constraint, and that $V_{\mathrm{bur}}(t)$ which does not appear in the aforementioned variables is calculated from $V_{\mathrm{unb}}(t)$.

Since there are not only statistics but also full access to all of the equations once the model is exported to Maple, it is also possible to perform other types of manipulation such as applying additional model simplification methods including approximation, or exporting to yet another modelling and simulation environment.

Eventually, after successful modelling and simulation in HLMT for models with different levels of complexity, together with research progress in Maple-based model simplification for HLMT-generated models, we decided to develop a flexible data structure by which the modelling process including simplification can be facilitated.

3. Data Structure for Model Simplification

Requirements for the model simplification data structure include 1) it can store information generated in HLMT, 2) it can store a simplified set of equations, 3) it can provide convenience for model simplification methods, and 4) it can generate a Modelica representation of the stored model.

The third requirement may need more breakdown as we develop our model simplification methods, but we now have a realization of this data structure called `MSModel`. An `MSModel` data structure is captured as a Maple Record and it serves as a modelling research artefact with which more a thorough design of the data structure can be made.

In this section, the current implementation of the `MSModel` is explained with some examples. For the complete coverage of the specification of `MSModel`, please see Appendix A.

3.1 Overview of `MSModel` Data Structure

The `MSModel` data structure consists of the following pieces of information.

- An independent variable (e.g. time t)
- Differential equations (DE) and variables (DV)
- Algebraic equations (AE) and variables (AV)
- Intermediate equations (IE) and variables (IV)
- Dependent equations and variables
- Parameters
- Inputs and outputs
- Name, type and value of variables
- Blackbox functions (e.g. lookup tables and user-defined functions)

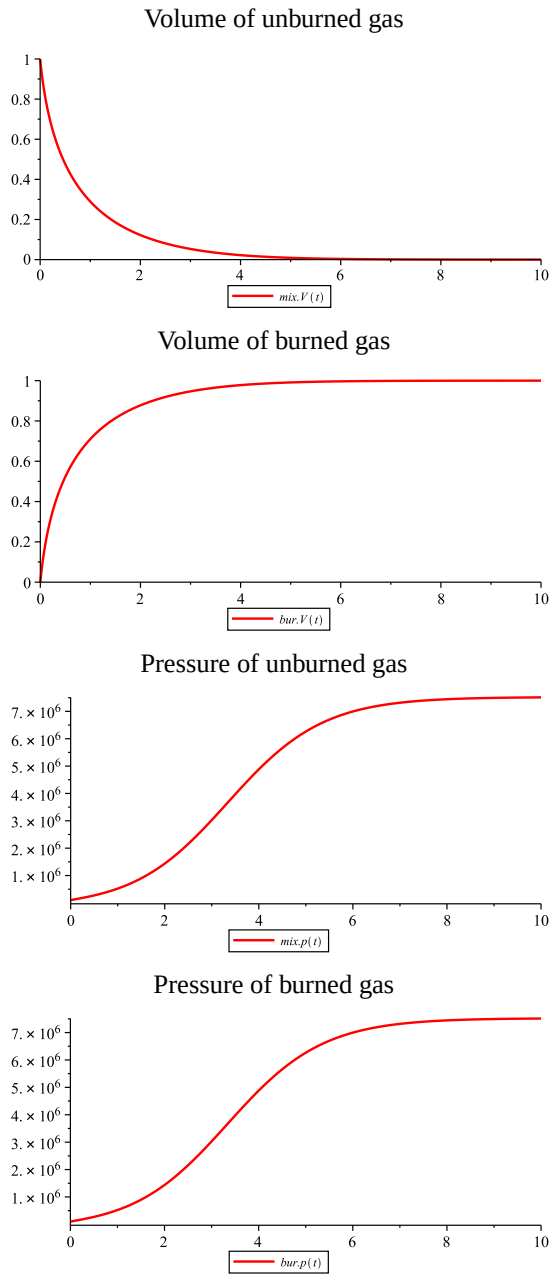


Figure 3. Simulation result of the combustion model, produced in HLMT.

These elements are stored in a Maple Record in a structured manner. A fictitious example of `MSModel` is shown in Figure 4.

In this example, each Record entry is accessible by, for instance, `msm:-DE[1]` for a list / array element or `msm:-variables[x1]` for a table element. The name `msm` is just an example and any Maple variable name is fine.

The combination of differential equations, `msm:-DE`, algebraic equations, `msm:-AE`, and intermediate equations, `msm:-intermediate`, comprise the minimum set of equations (i.e. *core* equations) with which the time evolution of the system can be computed.

```
msm := Record(MSMODEL,
 DE=[
  ( diff(x1(t),t)=-a*x1(t)+u1(t) ),
  ...],
 DV=[ 'x1' , ...],
 AE=[],
 AV=[],
 t='t',
 intermediate=(Array(1..0,[])),
 intermediateVariables=[],
 dependent=(Array(1..3,{
  1=[{ e1(t)=-1/2*sin(x1(t)) },{e1(t)}],
  2=[{ e2(t)=u1(t)*e1(t) },{e2(t)}],
  3=[{ y(t)=e1(t)+e2(t) },{y(t)}] })),
 dependentVariables=[ 'e1' , ...],
 parameters=[ 'a' , ...],
 inputs=[ 'u1' , ...],
 outputs=[ 'e1' , ...],
 variables=(table([
  (x1)=Record(MSVARIABLE,
   name=x1,
   type="differential",
   value=.9,
   unit=(NULL)),
  (a)=Record(MSVARIABLE,
   name=a,
   type="parameter",
   value=2,
   unit=(NULL)),
  ...])),
 blackboxes=[]
);
```

Figure 4. `MSModel` data structure example.

While `msm:-DE` and `msm:-AE` are a list of equations in no particular order, `msm:-intermediate` is an array ordered in a straight-line causal arrangement. The example in Figure 4 has no element in the `msm:-intermediate` field, but the example `msm:-dependent` field contains an array of such an arrangement with 3 elements. In both `msm:-intermediate` and `msm:-dependent`, each element has two sub-elements. The first sub-element is the equation, the second is the variable to be determined by that equation. There are two significant differences between `msm:-intermediate` and `msm:-dependent`. 1) In `msm:-intermediate`, there are no derivatives contained whereas in `msm:-dependent`, equations can be implicit and/or differential. 2) equations in `msm:-intermediate` must be computed at every integration step time while those in `msm:-dependent` have to be computed only when necessary.

Differential, algebraic, intermediate and dependent variables are stored in `msm:-DV`, `msm:-AV`, `msm:-intermediateVariables`, and `msm:-dependentVariables` as a list of variable names, respectively. Variables in these lists are stored, not as a function of the independent variable, but just as a name.

A single independent variable by which all other variables may be parameterized is stored at `msm:-t`.

Parameters, inputs and outputs are stored in `msm:-parameters`, `msm:-inputs` and `msm:-outputs` as a list of their names, respectively.

The `MSModel` Record has an entry called `variables` which is a table of all the variables including parameters described above with their type, initial value and unit information. In the example Record in Figure 4, the unit has not been assigned to any variable.

To deal with special types of functions, specifically lookup table functions and user-defined functions, the `MSModel` Record has an entry called `blackboxes`. It can store information such as the dimension and the data of a lookup table, or the definition of a user-defined Maple procedure.

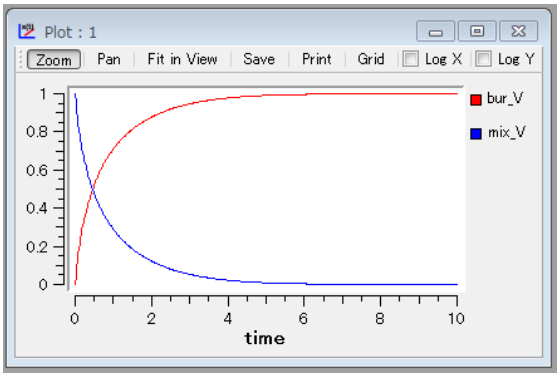
3.2 Workflow Example for HLMT, `MSModel`, Simplification and Modelica-Export

To illustrate the use of an `MSModel` data structure as part of the plant modeling process, here the hydrogen-oxygen combustion model mentioned in the previous section will be exported from HLMT to Maple, and stored in `MSModel`. Then simplification methods will be applied to it, and the reduced model will be exported to Modelica, and finally run in a Modelica-based simulation tool.

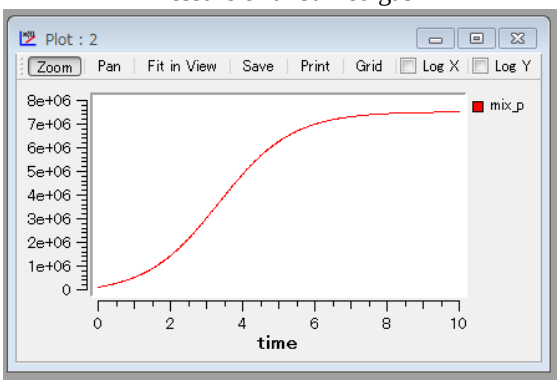
After the first equation-based simplification algorithm was applied to the HLMD of the system, the model consisted of 11 DEs for 8 DVs and 3 AVs (see Section 2.2) with 0 IEs. It also has 32 dependent equations and the same number of dependent variables. Once this model is stored in `MSModel`, other symbolic simplification algorithms which Maplesoft and Toyota developed were applied to it. As a result, 10 DEs for 7 DVs and 3 AVs with 4 new IEs for 4 new IVs consisted of the core equations while 1 DE for $\frac{d}{dt}N_{H_2O,\mathrm{ff}}(t)$ was categorized as one of the 8 dependent equations.

From this `MSModel`, a Modelica language file was generated, and it was simulated in OpenModelica [2] with the dassl solver as shown in Figure 5.

Volumes of unburned gas and burned gas



Pressure of unburned gas



Pressure of burned gas

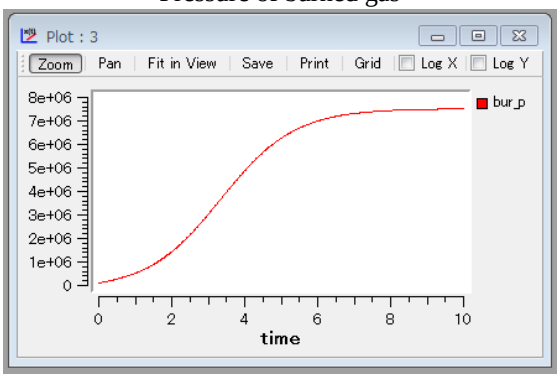


Figure 5. Simulation result of the combustion model, produced in OpenModelica.

4. Conclusion

Part of a plant modelling process for Model Based Development was introduced. A physics-based modelling framework as well as a novel data structure for model simplification and Modelica-export were developed. It was shown that these tools and data structure can greatly streamline the plant modelling process.

References

[1] Bakus J. Bernardin L. Gerhard J. Kowalska K. Léger M. Wittkopf A. High-Level Physical Modeling Description and Symbolic Computing. *IFAC Proceedings of the 17th World Congress*, pages 1054–1055, 2008.

[2] Fritzson P. *et al.* http://www.openmodelica.org/. *The OpenModelica Project.*

[3] Maplesoft. Maple 11 User Guide. 2007.

[4] Modelica Tools. http://www.modelica.org/tools. *Modelica Association*, 2011.

[5] Ohata A. Ito H. Gopalswamy S. Furuta K. Plant Modeling Environment Based on Conservation Laws and Projection Method for Automotive Control Systems. *SICE Journal of Control, Measurement and System Integration*, 1(3):227–234, 2008.

A. Appendix - The `MSModel` Data Structure

The `MSModel` data structure is implemented as a Maple Record, described in Section A.1. The Record structure is designed as a general container whose nested contents may be efficiently accessed for both reading and writing. A Record has a number of exports, or fields, which can be arbitrary Maple objects. A given field can itself be, for ex-

ample, a list of lists or a table of Records. In particular, two types of sub-Records occur inside the MSModel Record: the MSVariable Record used to give detailed information about variables and described in Section A.2, and the MSBlackBox Record, used to give detailed information about black box functions and described in Section A.3.

A.1 MSModel Record type

The MSModel data structure is implemented as a Maple Record. The particular structure and purpose of the individual fields of an MSModel Record are described below. The fields are references to some MSModel record named model.

model:-DE

A list of the core differential equations of the model. There is no additional structure or order to the appearance of the equations in this list. Together with the algebraic equations model:-AE and the intermediate equations model:-intermediate, these equations comprise the set of *core* equations which capture the characteristic dynamics of the system.

model:-DV

A list whose elements are of type name. This list designates the full set of differential variables present in the equations in either the DE, AE, or intermediate fields. In particular, the derivative of each variable in DV occurs in DE.

model:-AE

A list of the core algebraic equations of the model. Equations appear in this field by virtue of not being core differential or intermediate equations of the model. There is no additional structure or order to the appearance of the equations in this list.

model:-AV

A list whose elements are of type name. This list designates the full set of the algebraic variables present in the equations in either model:-DE, model:-AE, or model:-intermediate, but excludes the input and intermediate variables. These variables do not occur differentiated anywhere in the model.

model:-t

This field is a single name, which designates the independent variable (e.g., time) by which all other variables may be parameterized.

model:-intermediate

An Array specifying the intermediate equations. These equations are grouped into subgroups, each subgroup giving the equations to determine a certain subset of the intermediate variables. The elements of the Array are lists consisting of two sets: first a set of explicit equations, then the set of intermediate variables determined by those equations (represented as functions of model:-t); in other words, the second set forms the left hand sides of the equations making up the first set.

These lists are ordered in a straight-line causal arrangement; that is, the only variables occurring in right-hand sides of equations are either input variables, or differential variables, or they are intermediate variables that have been defined in earlier elements of the Array. Furthermore, model:-intermediate contains no derivatives.

The equations could be substituted into each other and then into model:-DE and model:-AE to obtain the core equations in a more explicit form.

model:-intermediateVariables

This is a list of the variable names appearing in all the second sets of the elements of model:-intermediate, or equivalently, on the left hand side of the equations in that field. These variables do not occur differentiated anywhere in the model.

model:-dependent

This is a list of differential or algebraic equations which are not part of the core dynamics of the model. Like model:-intermediate, the equations are given by an Array of lists consisting of two sets, where each first set specifies algebraic or differential equations or expressions determining the values of the dependent variables specified (as functions of model:-t) in the corresponding second set. Also, like model:-intermediate, the lists are ordered in a straight-line causal arrangement; in this case, that only means that dependent variables occurring in the equations are never defined in later elements of the Array (algebraic and differential core variables, intermediate variables, and input variables can occur throughout model:-dependent).

A difference between model:-dependent and model-:-intermediate is that equations in the former can be implicit or even differential.

Dependent equations can be solved after the core variables have been solved.

model:-dependentVariables

This is a list of all dependent variable names. That is, it is the list of variables that are not part of the core system or input or intermediate variables. They are to be solved in terms of core variables, and correspond to the entries of the second set of each list in model:-dependent.

model:-parameters

A list of parameter names. This list designates the full set of parameters of the model. A parameter is understood to be a quantity that, by design, does not vary over time.

model:-inputs

A list of input variable names. If the model is considered as a subsystem, then these variables are the input ports.

model:-outputs

A list of output variable names. If the model is considered as a subsystem, then these variables are the output ports.

Each output variable also occurs in exactly one of the fields DV, AV, intermediateVariables, dependentVariables, and inputs.

model:-variables

A table whose indices are of type name and whose corresponding entries are themselves Records. The indices are precisely the names of the variables, parameters, and blackbox functions occurring in the model. That is, the indices are precisely those names occurring in the fields DV, AV, intermediateVariables, dependentVariables, inputs, parameters, and blackboxes. Again in other words, the name of every function called in model:-DE, model:-AE, model:-intermediate, and model:-dependent that is not a function built-in to Maple is an index into the model:-variables table.

Each entry in the table indexed by the name of a variable or parameter is a Record of MSVariable type, which is described below. Each entry in this table indexed by the name of a blackbox function has as its entry a Record of MSBlackBox type, which is also described below.

model:-blackboxes

A list of undefined names representing lookup table functions and user defined functions, which may occur in any equation of the model. For example, as $z_{23}(t) = L1(z_{35}(t))$, where the right hand side is an unevaluated function call.

A.2 MSVariable Record type

This type of Record is used to store detailed information about any variable or parameter of the model. This includes core, input, dependent, and intermediate variables, and parameters. Such Records appear as entries of the variables field of the parent MSModel Record.

The fields of an MSVariable Record, named here as variable, are as follows:

variable:-name

The name of the variable described by the Record. This is the same as the index of the Record in the model:-variables table.

variable:-type

The role of the variable, given by one of the strings "algebraic", "differential", "intermediate", "dependent", "parameter", or "input". This corresponds to the field of the MSModel Record that variable:-name occurs in, as follows:

variable occurs in model:-	variable:-type
AV	"algebraic"
DV	"differential"
intermediateVariables	"intermediate"
dependentVariables	"dependent"
parameters	"parameter"
inputs	"input"

variable:-value

For parameters, this field holds the value for this parameter. For differential variables (those in model:-DV and the differential ones in model:-dependent), it contains the initial value, expressed as a numerical value or an expression in terms of the parameters. For algebraic variables, an initial value can also be given; this may contribute to determining the initial conditions. For input variables this entry will be a function of the independent variable (model:-t). This field may be empty.

variable:-unit

The unit in which the variable is measured. This field may be empty.

A.3 MSBlackBox Record type

This type of Record is used to store the particular information of any implicit lookup table function in the model, and for storing user-defined functions. The otherwise implicit blackbox functions are listed by name in the blackboxes field of the parent Record.

The MSBlackBox Record for a lookup table function contains the original numeric data, stored in tabular form. This allows for easy exporting of a lookup table to corresponding fields of a Modelica representation of the parent model.

The MSBlackBox Record for a user-defined function is similar, but it contains the Maple procedure that implements the user-defined function.

The fields of a MSBlackBox Record, named here as blackbox, are as follows:

blackbox:-name

The name of the blackbox function call described by the Record. This is the same as the index of the Record in the model:-variables table.

blackbox:-type

This string is the type of the black box function: either "lookup" for a lookup table, or "procedure" for a user-defined function.

blackbox:-dimension

The number of arguments of the blackbox function as occurring in the equations. For lookup table functions, this number corresponds to the dimension of the lookup table, and it must currently be either 1 or 2, which corresponds to the current support for lookup tables in MapleSim's implementation of Modelica.

blackbox:-data

For lookup table functions, this field is a list of lists of independent data points; each list contains all values for one of the arguments in strictly increasing order. The number of such lists is equal to blackbox:-dimension. The order of the lists corresponds to the order in which the arguments to the lookup table are given. The number of data points in each sublist must agree with the number of points

in the dependent `blackbox:-value` list for the given lookup table.

For user-defined functions, this field is not used; it may have any value.

`blackbox:-value`

For lookup table functions, this field contains the dependent values that the function attains at the respective data points; that is, the values corresponding to measurements of the dependent variable or quantity. These should either be given as one long list of numerical values, or a list of sublists each containing numerical values. If the lookup table is one-dimensional, then the list should simply contain the dependent values at the points in the same order as given in `blackbox:-data`. If the lookup table is two-dimensional, then the first entry of `blackbox:-value` should be a list of all dependent values where the first coordinate is the lowest value, ordered by increasing value of the second coordinate; the second entry should be a similar list for the second lowest value of the first coordinate, and so on. Alternatively, `blackbox:-value` may be the concatenation of these lists.

For user-defined functions, this field should contain the Maple language procedure that is to be used to evaluate the function.