An Object Oriented Approach to the Management of Models

R. Mlekus and S. Selberherr

Institut für Mikroelektronik, TU Wien
A-1040 Vienna, Austria

The object oriented **M**odel **D**efinition **L**anguage (MDL) for the selection and definition of algorithms or models is presented in this contribution. A number of basic algorithms and parameter types provided by the Algorithm Library can easily be extended by user defined algorithms and C++ classes describing new parameter types. MDL allows to add and/or modify new model algorithms of simulators by defining them on the input deck without recompilation. Model algorithms stored in object libraries or input deck libraries can easily be used by several independent programs.

1. Introduction

Almost any simulator has to provide the functionality to choose from different algorithms or models specialized for solving specific problems dependent on the actual simulation task or user commands. To support model developers in continuously improving these simulators by adding new model algorithms, it is essential to separate these models from the rest of the simulator code and to provide a simple modular mechanism to define, test, and archive them. Therefore the Algorithm Library was developed, which provides an object oriented approach to the programming and handling of model algorithms and the **M**odel **D**efinition **L**anguage (MDL). These algorithms are stored in libraries of object files or ASCII files containing input decks and can easily be reused by other programs or as parts of other algorithms. Automated extraction of log- files and debug information and a report containing information about available algorithms and their documentation is supported.

2. Model and Program Structure

Algorithms provided by the Algorithm Library are represented by C++ classes derived from the base class “`Model`” or its subclasses representing various specific model types [1]. The thereby defined inheritance tree is used to classify the various model algorithms and for run time type checking of expressions defined on the input deck. `Model` classes encapsulate the algorithm itself, private data values used to evaluate the algorithm, an interface containing the required input and output parameters, and the documentation.

New parameter types can be instantiated by specializing the template class “`Parameter`” with an arbitrary C++ class describing the parameter value. This parameter class contains a reference to its value, a default value, the name and the documentation of the parameter. Methods to link several parameters together manage their value references to point to a value shared between them including a run time type check. For each of these parameters a set of operators and functions can be specified which can be used in calculations defined on the input deck as well as in algorithms defined in C++.

A different approach to support the development of models is presented in [5], where the modeling language PMDL is introduced. The PMDL compiler provides a subset of the C language extended by data types and expressions dealing with mesh data and the automated generation of the Jacobian matrix. In distinction to this the Algorithm Library is designed to support libraries of arbitrary user defined data types and algorithms without any specialization.

A basic set of predefined algorithms and parameter types providing all standard C++ data types and basic operations on them, is already provided by the Algorithm Library. These can be extended at any time by additional user defined libraries of further algorithms and parameter types or by using the Model Definition Language.

An instance of a specific algorithm can be generated by forwarding the model type name to the Algorithm Library or by giving an instance name for the algorithm. In this case the actual class type is determined at run time by parsing the input deck. To evaluate the algorithm, its class instance is connected to an interface providing the necessary parameter values.

A minimal program using the Algorithm Library to evaluate a single algorithm may be structured as following:

1. Initialize the Algorithm Library by parsing the input deck.
2. Create the interface containing all parameters and the required model type.
3. Request a model instance from the Algorithm Library and link it with a specified parameter interface.
4. Repeat as necessary: Compute the values of the input parameters.
   Evaluate the model.
   Use the resulting parameter values for further computations.
5. Delete the interface and the model instance.

Steps 1–3 should take place during the initialization phase of the program because they require the rather time consuming parsing and interpretation of the input deck. Once the internal data structures of the Algorithm Library are assembled, the additional time consumption caused by the usage of the Algorithm Library are typically between 5 – 50 % depending on the complexity of the models.

3. Model Definition Language

The Algorithm Library contains a parser for the Model Definition Language which allows to:

- Define the actual algorithms to be used for a specific task.
- Define the default parameter values of a certain model instance.
- Define global parameters.

- Define new algorithms consisting of previously defined models and calculations with global and local parameters. Arbitrary loop and condition expressions can be expressed by using special model classes.
- Request a report describing all available algorithms, their interfaces and the thereby defined model hierarchy.
- Request a report describing the actually used algorithms and/or parameter values for a specific model instance.

4. Example

To give a short example for the usage of the Model Definition Language, a simple carrier mobility model is defined by combining a lattice scattering model with a carrier-carrier scattering model using the Mathiessen rule. The interface for all carrier mobility models (class name `MobilityModel`) contains among others the parameters `temp` (lattice temperature in [K]), `mu` (the resulting carrier mobility in [$cm^2 V^{-1} s^{-1}$]) and `np` (the product of the electron and hole densities in [cm^{-6}]). It is assumed that an abstract class `MobilityModel` which defines the interface for all carrier mobility models and the following simple lattice scattering and carrier-carrier scattering models are already contained in a Model Library or defined in a previously scanned input deck file:

Lattice Scattering [2]:

$$mu = mu0 \cdot \left(\frac{temp}{300} \right)^{-alpha}$$

Carrier-Carrier Scattering [3]:

$$mu = \frac{1.428 \cdot 10^{20}}{\sqrt{np} \cdot \ln\left(1 + 4.54 \cdot 10^{11} \cdot (np)^{-1/3}\right)}$$

Listing: Input deck defining a new carrier mobility model for electrons

```
CombinedModel LC_Mobility : MobilityModel {
   Model LatticeScatteringMobility "LSMob";
   Model CC_ScatteringMobility    "CCSMob";

   // new default values for some sub models
   Parameter " LSMob"."mu0" = {{1448}};
   Parameter " CCSMob"."alpha"  = {{2.33}};

   link Interface."temp" to "LSMob"."temp";
   link Interface."np" to "CCSMob"."np";

   calc "MatthiesenRule" {
      Interface."mu" = 1/("LSMob"."mu" + 1/ "CCSMob"."mu");
   }

   EvaluationOrder "LSMob", "CCSMob", "MatthiesenRule";
}

// Specify the actual Mobility Model Type to use
Model "MobilityModel" = LC_Mobility;
```

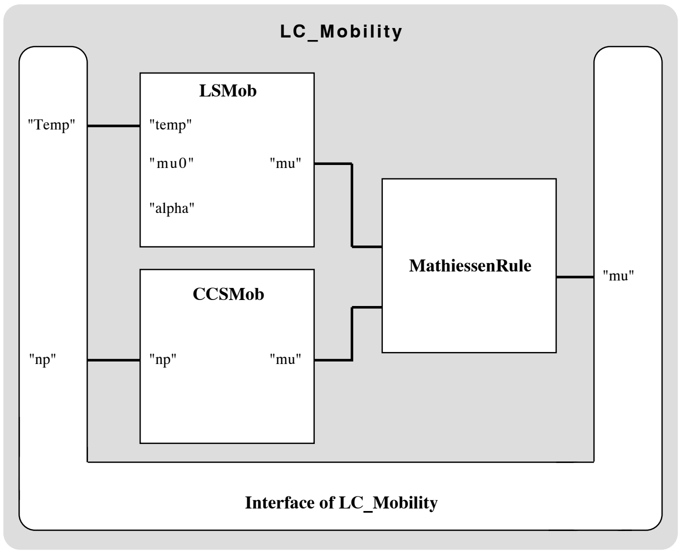

Fig. 1: Block diagram of the new carrier mobility model

5. Conclusion and future aspects

By using the model library a clean interface is introduced between modularized algorithms and the rest of the program. These algorithms can easily be replaced by newly defined ones during run time without any additional coding efforts within the simulator. Due to the relatively low run time performance losses and the great simplifications in introducing new algorithms into existing simulators, the Algorithm Library is a valuable tool for simulator and model development and their daily usage.

Acknowledgment

The authors would like to thank Siemens AG, Munich, Germany, for the support.

References

[1] B. Stroustrup, “The C++ Programming Language”, Addison-Wesley 1986, ISBN 0-201-12078-X

[2] N.D. Arora, J.R. Hauser, D.J. Rouston, “Electron and Hole Mobilities in Silicon as a Function of Concentration and Temperature”, *IEEE Trans.Electron Devices,* ED-29 (1982), p. 292 – 295

[3] M.S. Adler, “Accurate Numerical Models for Transistors and Thyristors”, in Miller [4], p 5 – 8

[4] J.J. Miller, “An Introduction to the Numerical Analysis of Semiconductor Devices and Integrated Circuits”, (Dublin, 1981), Boole Press

[5] J. Litsius, “A modeling language for mixed circuit and semiconductor device simulation”, Hartung-Gorre 1996, ISBN 3-89649-034-6