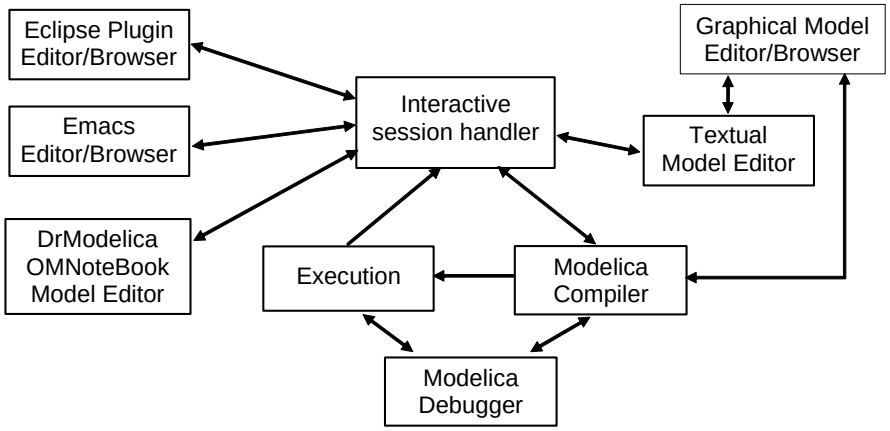
OpenModelica Compiler (OMC) Overview

Peter Fritzson, Adrian Pop, Peter Aronsson

OpenModelica Course at INRIA, 2006 06 08

OpenModelica Environment Architecture



OpenModelica Compiler/Interpreter

- New version (1.4.0) released May 15, 2006
- Currently implemented in 100 000 lines of MetaModelica
- Includes code generation, BLT-transformation, index reduction, connection to DASSL, etc.
- Most of the Modelica language including classes, functions, inheritance, modifications, import, etc.
- Hybrid/Discrete event support

Invoking OMC – two Methods

- Calling OMC from the command line
- Calling OMC as a server via the Corba interface

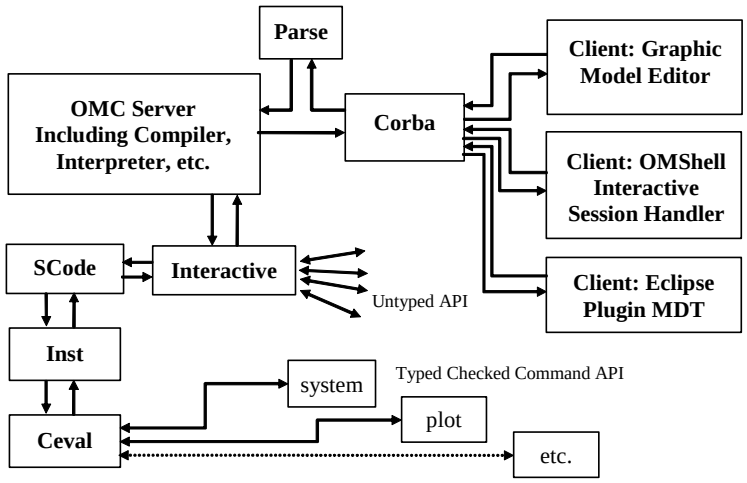
Command Line Invokation of OMC

- omc file.mo
  - Return flat Modelica by code flattening of the class in the file file.mo which is a the top of the instance hierarchy (toplevel class)
- omc file.mof
  - Put the flat Modelica produced by flattening of the toplevel class within file.mo in the file named file.mof.
- omc file.mos
  - Run the Modelica script file called file.mos.

Some General OMC Flags

- omc +s file.mo/.mof
  - Generate simulation code for the model last in file.mo or file.mof. The following files are generated: modelname.cpp, modelname.h, modelname_init.txt, modelname.makefile.
- omc +q
  - Quietly run the compiler, no output to stdout.
- omc +d=blt
  - Perform BLT transformation of the equations.
- omc +d=interactive
  - Run the compiler in interactive mode with Socket communication. This functionality is depreciated and is replaced by the newer Corba communication module, but still useful in some cases for debugging communication. This flag only works under Linux and Cygwin.
- omc +d=interactiveCorba
  - Run the compiler in interactive mode with Corba communication. This is the standard communication that is used for the interactive mode.

OpenModelica Client-Server Architecture



OMC Corba Client-Server API

- Simple text-based (string) communication in Modelica Syntax
- API supporting model structure query and update

Example Calls:
Calls fulfill the normal Modelica function call syntax.:

```
saveModel("MyResistorFile.mo",MyResistor)
```

will save the model MyResistor into the file “MyResistorFile.mo”.

For creating new models it is most practical to send a model, e.g.:

```
model Foo   end Foo;
```

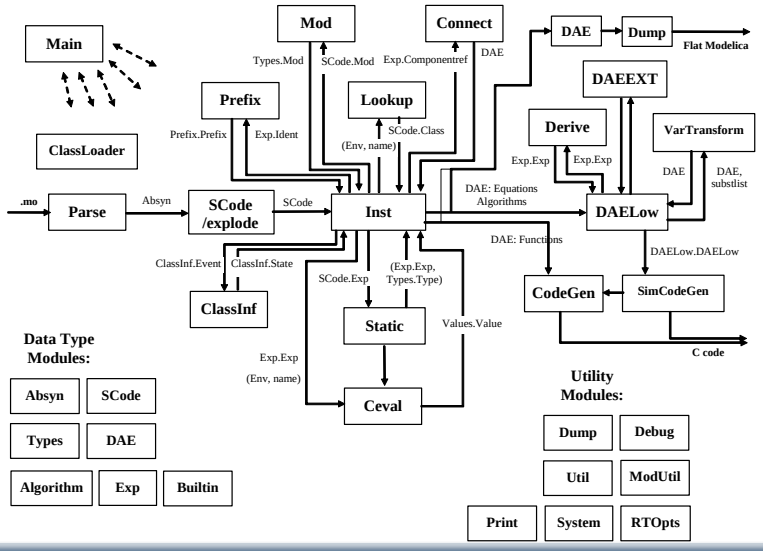
or, e.g.,

```
connector Port   end Port;
```

Some of the Corba API functions

`saveModel(A1<string>,A2<cref>)`	Saves the model (A2) in a file given by a string (A1). This call is also in typed API.
`loadFile(A1<string>)`	Loads all models in the file. Also in typed API. Returns list of names of top level classes in the loaded files.
`loadModel(A1<cref>)`	Loads the model (A1) by looking up the correct file to load in `$MODELICAPATH`. Loads all models in that file into the symbol table.
`deleteClass(A1<cref>)`	Deletes the class from the symbol table.
`addComponent(A1<ident>,A2<cref>, A3<cref>,annotate=<expr>)`	Adds a component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
`deleteComponent(A1<ident>, A2<cref>)`	Deletes a component (A1) within a class (A2).
`updateComponent(A1<ident>, A2<cref>, A3<cref>,annotate=<expr>)`	Updates an already existing component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
`addClassAnnotation(A1<cref>, annotate=<expr>)`	Adds annotation given by A2(in the form `annotate= classmod(...))` to the model definition referenced by A1. Should be used to add Icon Diagram and Documentation annotations.
`getComponents(A1<cref>)`	Returns a list of the component declarations within class A1: `{{Atype,varidA,"commentA"},{Btype,varidB,"commentB"}, {...}}`
`getComponentAnnotations(A1<cref>)`	Returns a list `{...}` of all annotations of all components in A1, in the same order as the components, one annotation per component.
`getComponentCount(A1<cref>)`	Returns the number (as a string) of components in a class, e.g return `"2"` if there are 2 components.
`getNthComponent(A1<cref>,A2<int>)`	Returns the belonging class, component name and type name of the nth component of a class, e.g. `"A.B.C,R2,Resistor"`, where the first component is numbered 1.
`getNthComponentAnnotation( A1<cref>,A2<int>)`	Returns the flattened annotation record of the nth component (A2) (the first is has no 1) within class/component A1. Consists of a comma separated string of 15 values, see Annotations in Section 2.4.4 below, e.g `"false,10,30,..."`
`getNthComponentModification( A1<cref>,A2<int>)??`	Returns the modification of the nth component (A2) where the first has no 1) of class/component A1.
`getInheritanceCount(A1<cref>)`	Returns the number (as a string) of inherited classes
`getNthInheritedClass(A1<cref>,`	Returns the type name of the nth inherited class of a class. The first class has number 1.

Detailed Architecture of OMC (OpenModelica Compiler)



Three Kinds of Modules in OMC

- *Function modules* that perform a specified function, e.g. Lookup, code instantiation, etc.
- *Data type modules* that contain declarations of certain data types, e.g. Absyn that declares the abstract syntax.
- *Utility modules* that contain certain utility functions that can be called from any module, e.g. the Util module with list processing funtions.

- Note: Some modules perform more than one kind of function

Approximate Description

- The **Main** program calls a number of modules, including the parser (Parse), SCode, etc.
- The parser generates abstract syntax (**Absyn**) which is converted to the simplified (**SCode**) intermediate form.
- The model flattering module (**Inst**) is the most complex module, and calls many other modules. It calls **Lookup** to find a name in an environment, calls **Prefix** for analyzing prefixes in qualified variable designators (components), calls Mod for modifier analysis and Connect for connect equation analys. It also generates the DAE equation representation which is simplified by **DAELow** and fed to the **SimCodeGen** and **CodeGen** code generators
- The **Ceval** module performs compile-time or interactive expression evaluation and returns values. The Static module performs static semantics and type checking.
- The **DAELow** module performs BLT sorting and index reduction. The DAE module internally uses Exp.Exp, Types.Type and Algorithm.Algorithm; the SCode module internally uses Absyn
- The **Vartransform** module called from **DAELow** performs variable substitution during the symbolic transformation phase (BLT and index reduction).

Short Overview of OMC Modules (A-D)

- Absyn – Abstract Syntax
- Algorithm – Data Types and Functions for Algorithm Sections
- Builtin – Builtin Types and Variables
- Ceval – Evaluation/interpretation of Expressions.
- ClassInf – Inference and check of class restrictions for restricted classes.
- ClassLoader – Loading of Classes from $MODELICAPATH
- Codegen – Generate C Code from functions in DAE representation.
- Connect – Connection Set Management
- Corba – Modelica Compiler Corba Communication Module

Short Overview of OMC Modules (D-G)

- DAE – DAE Equation Management and Output
- DAEEXT – External Utility Functions for DAE Management
- DAELow – Lower Level DAE Using Sparse Matrises for BLT
- Debug – Trace Printing Used for Debugging
- Derive – Differentiation of Equations from DAELow
- Dump – Abstract Syntax Unparsing/Printing
- DumpGraphviz – Dump Info for Graph visualization of AST
- Env – Environment Management
- Exp – Typed Expressions after Static Analysis /*updated)
- Graphviz – Graph Visualization from Textual Representation

Short Overview of OMC Modules (I-P)

- Inst – Flattening of Modelica Models
- Interactive – Model management and expression evaluation Keeps interactive symbol tables. Contains High performance API, etc.
- Lookup – Lookup of Classes, Variables, etc.
- Main – The Main Program. Calls Interactive, the Parser, the Compiler, etc.
- Mod – Modification Handling
- ModSim – /*Depreciated, not used). Previously communication for Simulation, Plotting, etc.
- ModUtil – Modelica Related Utility Functions
- Parse – Parse Modelica or Commands into Abstract Syntax
- Prefix – Handling Prefixes in Variable Names
- Print – Buffered Printing to Files and Error Message Printing

Short Overview of OMC Modules (R-V)

- SCode – Simple Lower Level Intermediate Code Representation.
- SimCodegen – Generate simulation code for solver from equations and algorithm sections in DAE.
- Socket – (Partly Depreciated) OpenModelica Socket Communication
- Static – Static Semantic Analysis of Expressions
- System – System Calls and Utility Functions
- TaskGraph – Building Task Graphs from Expressions and Systems of Equations. Optional module.
- TaskGraphExt – External Representation of Task Graphs. Optional module.
- Types – Representation of Types and Type System Info
- Util – General Utility Functions
- Values – Representation of Evaluated Expression Values
- VarTransform – Binary Tree Representation of Variable Transformations