第2章调用OMC-OpenModelica编译器/解释器子系统

可以通过两种方式调用OpenM Odelica编译器/解释器子系统（OMC）：

- 作为整个程序，在操作系统级调用，例如作为命令。作为服务器，通过CORBA客户端-服务器接口从客户端应用程序调用。

在下文中，我们将更详细地描述这些选项。

2.1 编译器/解释器的命令行调用

OpenModelica编译子系统称为OMC（OpenModelica编译器）。可以为编译器提供下面指定的文件参数，以及后续部分中描述的标志。

OMC文件.Mo OMC文件.MOF	通过代码实例化file.Mo文件中的最后一个类返回平面Modelica，将file.Mo中最后一个类的代码实例化生成的平面Modelica放在名为file.MOF的文件中。运行名为file.MOS的Modelica脚本文件。不带参数调用OMC将显示帮助:

OMC file.MOS OMC$./OMC OpenModelica Compiler 1.8.1（r11525）Copyright Linkoping University 1997-2012在OMSC-PL和GPL下分发，请参阅www.openmodelica.org用法：OMC[-runtimeOptions+omcoptions]（model.Mo|script.MOS）[libraries|.Mo-files]*libraries：要在处理模型或脚本之前加载的库的完全限定名称。*库应由空间分隔：lib1 lib2..。Libn.*RuntimeOptions：调用OMC-帮助查看运行时选项*OMCOptions：+D，+DEBUG设置调试标志。使用+HELP=DEBUG查看可用标志。+HELP显示帮助文本。有效选项：DEBUG，OPTMODULES+RUNNING-运行TestSuite时使用的TestSuite。++V，+VERSION打印版本并退出。+target设置要使用的目标编译器。有效选项：GCC，MSVC+G，+语法设置要接受的语法和语义。有效选项：modelica，metamodelica，parmodelica+annotationversion设置应使用的注释版本。1.有效选项:X，2.X，3.X+STD设置应使用的语言标准。1.有效选项:X，2.X，3.1，3.2，3.3+ShowErrormesSages在发生错误时立即显示错误消息。+ShowAnnotati ons显示平面化代码中的注释。+NoSimplify如果设置，则不简化表达式。+PreOptModule设置要在后端使用的预优化模块。有关详细信息，请参阅+help=o ptmodules。有效选项：*RemoveSimpleEquat Ions Fast*RemoveSimpleEquat Ions*InlinearRayEqn*RemoveFinalParameters

*RemoveEqualFuncti OnCalls*RemoveProtectedPA参数*RemoveUnusedParam Eter*RemoveUnusedVaria Bles*PartitionIndepend EntBlocks*CollapseIndepende NtBlocks*ExpanderOperator*ResidualForm+IndexReducti OnMethod设置要使用的索引缩减方法。有效选项：*DummyDerivative*DynamicStateSelec tion+PostOptModul es设置要在后端使用的后优化模块。有关详细信息，请参阅+help=o ptmodules。有效选项：*lateinline*RemoveSimpleEquat Ions Fast*RemoveSimpleEquat Ions*RemoveEqualFuncti OnCalls*InlinearRayEqn*RemoveUnusedParam Eter*ConstantLinearSys TEM*DumpComponentsGra PhStr+SimCodeTarge T设置代码生成的目标语言有效选项：CSharp，CPP，ADEVS，QSS，C，C，Dump+OrderConnect Ions Orders Connect Equatio NS如果设置，则+modelicaoutp ut+p，+paramsstruct+Q，+silent打开静默模式。+C，+corbasession name如果使用+d=interactivecorba，则设置orba会话的名称。+N，+numprocs设置要使用的处理器数量.+l，+latency设置并行执行的延迟.+B，+bandwidth设置并行执行的带宽.+I，+instClass实例化完全限定路径给定的类.+V，+vectorizatio nlimit设置矢量化限制，大于此值的数组和矩阵将不会被矢量化.+s，+SimulationCG打开模拟代码生成。+EvalAnnotati OnParams设置是否评估注释中的参数。+GenerateLabe LEDSimCode打开归约算法的标记SIMC代码生成。+ReduceTerms打开归约算法的归约项。+ReductionMet Hod设置要使用的还原方法。有效选项：删除，代换，Linearization+PlotSilent定义绘图命令是否应打开OMPLOT或仅输出结果。*示例：OMC Model.Mo将在标准输出OMC+S Model.Mo上生成展平模型。模型将生成模型的模拟代码：*Model.C模型C代码*模型_函数。模型函数C代码*Model.Makefile编译模型的生成文件。*模型_init.XML初始值OMC Script.MOS将从Script.MOS运行命令。OMC Model.Mo Modelica将首先加载1.Modelica库，然后在标准输出OMC ModelMo Model2.Mo上生成平展模型将加载Model1.Mo和Model2.Mo，并在标准输出*.Mo（Modelica文件）*.MOS（Modelica脚本文件）

2.1.1 通用编译器标志

以下是与调试或跟踪不特别相关的用途的常规标志:

OMC+S文件.Mo/.MOF	为file.Mo或file.MOF中的最后一个模型生成模拟代码。将生成以下文件: modelname.CPP、modelname.H、modelname_init.txt、modelname.Makefile。
OMC+Q OMC+D=BLT OMC+D=交互式	安静地运行编译器，不输出到stdout。对方程进行BLT变换。使用套接字通信在交互模式下运行编译器。此功能已过时，并被较新的CORBA通信模块所取代，但在某些情况下仍可用于调试通信。此标志仅适用于Linux和Cygwin。使用CORBA通信在交互模式下运行编译器。这是用于交互模式的标准通信。实例化由完全限定路径类路径给定的类，而不是默认情况下文件中的最后一个类。返回OMC编译器的版本号。
OMC+D=InterActiveCorba	
OMC+I=类路径	
OMC++V	

2.1.1.1生成独立模拟代码示例要从命令行运行OMC并生成模拟代码，请使用以下标志:

OMC+S型号.Mo

目前，类加载器不会自动从Modelicapath加载包，因此.Mo文件必须包含所有使用的类，即必须创建“总模型”。一旦生成了C代码（和Makefile等），就可以使用

make-f modelname.makefile

2.1.2编译器调试跟踪标志

使用逗号分隔的不带空格的标志列表运行OMC，

“ OMC+D=FLG1，FLG2，..”

这里是FLG1，FLG2，..是下面标志描述最左边一列中的标志名称之一。名为ALL的特殊标志打开所有标志。

通过为打印函数提供标志名称来打开调试跟踪打印，如:

debug.fprint（“ Li ”，“查找信息：..”）

如果OMC使用以下命令运行:

OMC+D=foo，Li，bar，..

该行将出现在stdout上，否则不会出现。对于尚未整理的调试打印的向后兼容性，旧的调试打印调用:

调试.打印

已更改为如下所示的呼叫:

调试.fprint（“ OldDebug ”，..）

因此，如果OMC使用调试标志OLDDEBUG（或ALL）运行，则将显示这些消息。对debug.print的调用最终应更改为适当标记的调用。

此外，将“-”放在旗子前面会关闭该旗子，即：

OMC+D=全部，-转储

这将打开除转储以外的所有标志。使用Graphviz对抽象语法树进行可视化，可以通过给出一个Graphviz标志并将输出重定向到文件来完成。然后运行“ dot-TPS filename-o filename.PS ”或" dotty

文件名。

以下是所有可用调试跟踪标志的简短说明。现在，当最近开发的带有数据结构查看器的交互式调试器可用时，不再需要使用其中的某些标志。•All Debug Tracing All打开所有调试跟踪。

如果未给出任何标志，则此标志的默认值为True。•常规

信息	一般信息。

打印发送到旧调试的消息。打印•转储

解析转储	转储解析树。
转储	转储Absyn树。
DumpGraphviz	以Graphviz格式转储Absyn树。
Daedump	以打印形式转储DAE。
DaeDumpGraphviz	以Graphviz格式转储DAE。daedumpdebug以表达式形式转储DAE。
转储	转储跟踪。BeforeFixModOut在将修改公式移动到var声明中之前，以表达式形式转储PDAE。

类型TF类型和功能。TYTR类型跟踪。•查找Li查找信息。LOTR查找轨迹。LOCOM查找比较。•静态SEI信息setr跟踪•eLab_ClassDef的SCODE ECD跟踪。实例化Insttr代码实例化跟踪。•环境

•

环境打印环境图形扩展打印	在每个类实例化时转储环境。与EnvPrint相同，但使用Graphviz。公式精化时的转储环境。

ExpendGraph 公式精化时的转储环境。

2.1.2.1日志信息生成示例

$OMC+S+SIMCODETARGET=转储mymodel.Mo Modelica+I=my.model.name

打印日志，如:
当（#2）：时间>0.5时，I=整数
（R）[a.Mo:7:5-7:19]部分：在
m内；实例OptLst:
ConnectEquationOptLst：类
型Lst：操作（0）：

2.1.3编译器优化标志

后端优化阶段的简化示意图如图2-1所示。

图2-1:后端优化阶段的简化方案在优化前阶段和优化后阶段，几个优化模块被相继应用。因此，该顺序对于该过程是高度基本的。

2.1.3.1预优化模块使用逗号分隔的无空格标志列表运行OMC，以指定自定义预优化阶段:

“ OMC+前置模块=FLG1，FLG2，...”

这里是FLG1，FLG2，..是标志描述最左边一列中的标志名称之一。可以使用“ OMC+HELP=PreOptModules ”生成标志描述。以下是默认情况下使用的模块列表:

“ OMC+PreOptModules=Evaluatere PlaceFinalEvaluateParameters，SimplifyIfEquations，RemoveEqua lFunctionCalls，PartitionIndependentBlocks，ExpanderOperator，FindStateOrder，ReplaceEdgeChange，InlinearRayEQN，RemoveSimpleEquations ”

有效选项包括:

*RemoveSimpleEquations*RemoveAllSimpleEquations*InlinearRayEqn*EvaluateFinalParameters*EvaluateEvaluateParameters*EvaluateReplaceFinalParameters*EvaluateReplaceEvaluateParameters

*EvaluateReplaceFinalEvaluatePA参数
*RemoveEqualFunctionCalls*RemoveProtectedParameters*RemoveUnusedParameter*RemoveUnusedVariables*PartitionIn

2.1.3.2 转型阶段

此阶段使用索引缩减方法组合匹配和排序。因此，可以使用以下标志：CheapMatchingAlgorithm

“ OMC+CheapMatchingAlgorithm=0 ”

设置要使用的廉价匹配算法。一个廉价的匹配算法通过启发式给出了一个跳跃开始匹配。有效选项包括：

* 0 *
1 * 3

匹配算法

“ OMC+匹配算法=PFplus ”

设置要使用的匹配算法。有关详细信息，请参阅+HELP=OptModules。有效选项包括：

*BFSB*DFSB*MC21A*PF*PFPLUS*HK*HKDW*ABMP*PR*DFSBEXT*BFSBEXT*MC21AEXT*PFEXT*PFPLUSEXT*HKEXT*HKDWEXT*

匹配算法

“ OMC+IndexReductionMethod=UOD E ”

设置要使用的索引缩减方法。有关详细信息，请参阅+HELP=OptModules。有效选项包括：
*uode*dynamicStateSelection

2.1.3.3 后优化模块

使用逗号分隔的不带空格的标志列表运行OMC，以指定自定义优化后阶段：

“ OMC+PostOptModules=FLG1，FLG2，...”

这里是FLG1，FLG2，..是标志描述最左边一列中的标志名称之一。可以使用“ OMC+HELP=PostOpTM odules ”生成标志描述。以下是默认情况下使用的模块列表：

“ OMC+PostOptModules=RelaxSyst EM，InlinearRayEQN，ConstantLinearSystem，SimplifySemilinear，RemoveSimpleEquations，EncapsulateWhenConditions，TearingSystem，CountOperations，RemoveUnused Functions，InputDerivatives Used，DetectJacobi anSparsePattern，RemoveConstants ”

有效选项包括：

*EncapsulateWhenConditions*LateInlineFunction*RemoveSimpleEquationsFast*RemoveSimpleEquations*EvaluateFinalPara
s*EvaluateReplaceFinalParameters*EvaluateReplaceEvaluateParameters*EvaluateReplaceFinalEvaluateEPA
参数
*RemoveEqualFunctionCalls*InlinearRayEQn*Remoun

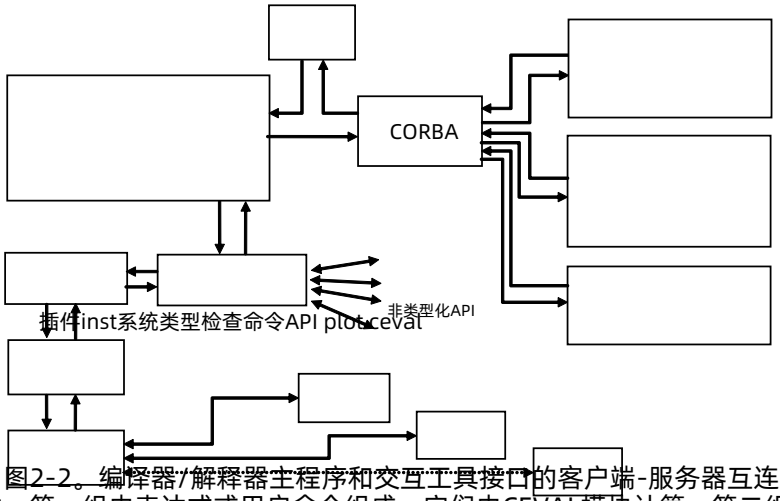
2.1.4模拟初始化标志

您可以通过以下模拟标志使用不同的初始化配置：“-IIM State ”： [default]将执行正常初始化“-IIM None ”： 将不执行初始化->使用起始值“-IOM Nelder_Mead_Ex ”： [default]将使用新的优化方法->全局同伦“-IOM Nelder_Mead_Ex2 ”： 将使用新的优化方法->不使用全局同伦

simplex -IOM “：将使用来自OM1.8.0的旧的单纯形初始化，仅与新的事件处理等一起使用。” -IIF<matfile>“：从给定的MATLAB文件（时间0.0或-iit<time>）导入所有变量的起始值” -iit<time>": 以上IIF标志的时间点

2.2 OpenModelica客户端-服务器架构

OpenModelica客户端-服务器架构如图2-2所示，显示了两个典型的客户端：图形模型编辑器和用于命令解释的交互式会话处理程序。解析客户端：图形模型编辑器服务器：主程序包括编译器、解释器等。客户端：omshell交互式会话处理程序代码交互式客户端：eclipse



等等。图2-2。编译器/解释器主程序和交互工具接口的客户端-服务器互连结构。来自CORBA接口的消息有两种。第一组由表达式或用户命令组成，它们由CEVAL模块计算。第二组是经由交互模块处理的类、变量等的声明、赋值和客户端-服务器API调用，该交互模块还在环境结构的顶层存储关于交互声明/赋值的项的信息。SCODE模块简化了ABSYN表示，公共组件收集在一起，受保护的组件收集在一起，等等。交互式模块提供非类型化API、更新、搜索和保持抽象语法表示。环境结构不会保留/缓存，而是由inst在每次调用时构建。在某些情况下，请调用Inst以进行更精确的实例查找。当编译某些东西时，整个Absyn AST被转换为SCODE，例如，转换整个标准库，如果有的话。命令或Modelica表达式通过CORBA接口以文本形式从客户端发送，由主程序解析并分为两组：•各类类、类型、函数、常量等的声明，以及方程和赋值语句。此外，对非类型化API的函数调用也属于这个组

27检查函数名称是否属于API名称。交互式模块处理这组声明和非类型化API命令。•表达式和类型检查API命令，由CEVAL模块处理。非类型化API调用不通过SCODE和INST传递给CEVAL的原因是CEVAL只能处理类型化调用−类型始终被计算和检查，而非类型化API优先考虑性能和类型灵活性。主模块检查被调用函数名的名称，以确定它是否属于非类型化API，并应路由到Interactive。此外，交互模块在顶层维护所有交互给定的声明和赋值的环境，这是这些项目需要由交互模块处理的原因。

2.3 用于脚本的客户端-服务器类型检查命令API

以下是用于OpenM Odelica环境的类型检查脚本命令/交互式用户命令的简短摘要。重点是用户命令的安全性和类型检查，而不是第2.4节中描述的非类型化命令接口中的高性能运行时命令解释。这些命令对于加载和保存类、读取和存储数据、绘制结果以及各种其他任务非常有用。传递给脚本函数的参数应遵循Modelica和相关脚本函数的语法和类型规则。在下面的表格中，我们通过下面的符号简要地指出函数的形式参数的类型或特征:

- 字符串类型参数，例如“ Hello ”、“ myfile.Mo ”。TypeName-类、包或函数名称，如MyClass、Modelica.Math。VariableName-变量名称，如v1、v2、vars1[2].X等。整型或实型参数，例如35、3.14、XintVariable。
- OPTIONS-带有命名形参传递的可选参数。

以下是OpenModelica环境中可用的最常用脚本命令的简要说明。在文件animate（类名，选项）（notyetimplemented）CD（dir）中也有一些示例调用。

	显示最新模拟的三维可视化。输入：TypeName类名；输出：布尔RES；
	更改目录。输入：字符串dir；输出：布尔RES；
CD（）检查型号（类名）（未实现）清除（）。	返回当前工作目录。输出：字符串res；实例化模型，优化方程，并报告错误。输入：类名类名；输出：布尔RES；
	清除所有内容：SymbolTable和变量。输出：布尔RES；
ClearClasses（）（NotYetImplemented）ClearLog（）（NotYetImplemented）	从SYMBOLTABLE中清除所有类别定义。输出：布尔RES；
ClearVariables（）ClosePlots（）（NotYetImplemented）GetLog（）（NotYetImplemented）	清除日志。输出：布尔RES；清除所有用户定义的变量。输出：布尔RES；关闭所有绘图窗口。输出：布尔RES；以字符串形式返回日志。输出：字符串日志；实例化模型，生成展平Modelica的.MOF文件。
InstantiateModel（类名）	
	输入：类名类名；输出：布尔RES；
列表（类名）	打印类别定义。输入：类名类名；输出：字符串ClassDef；
LIST（）LISTVARIABLES（）LOADFILE（filename）	打印所有加载的类定义。输出：字符串ClassDefs；打印用户定义的变量。输出：variableName Res；从文件加载模型。
	输入：字符串文件名输出：布尔值res；
loadModel（类名）	加载与类对应的文件，使用Modelica类名到文件名的映射来定位文件。输入：类名类名输出：布尔res；
绘图（变量，选项）	绘制变量，它是变量名称的向量。

E.{xmin，xmax}；实Yrange[2]I。E.{Ymin，Ymax}；输出：布尔RES；绘图（变量，选项）	输入：variableName变量；字符串标题；布尔图例；布尔网格线；实Xrange[2]I。
输入：variableName var；字符串标题；E.{xmin，xmax}；实Yrange[2]I。E.{Ymin，Ymax}；输出：布尔RES；绘图参数（Vars1，Vars2，选项）	绘制名为var的变量。 布尔图例；布尔网格线；实Xrange[2]I。
输入：variableName vars1[：]；variableName vars2[size（variables1，1）]；字符串标题；布尔图例；布尔网格线；实数范围[2，2]；输出：布尔RES；	从的向量中画出每对相应的变量。 变量vars1，vars2作为参数图。
打印参数（var1，var2，选项） 输入：variableName var1；变量名称var2；字符串标题；布尔图例；布尔网格线；实数范围[2，2]；输出：布尔RES；绘图矢量（V1，V2，选项）	将变量var2与var1绘制为参数图。
将矢量V1和V2绘制为X-Y图。输入：variableName（？notyetimplemented）V1；变量名称v2；输出：布尔RES；ReadMatrix（文件名，MatrixName）输入：String filename；字符串MatrixName；（？notyetimplemented）输出：布尔矩阵[：，:]；ReadMatrix（文件名，MatrixName，nRows，nColumns）和#列。输入：字符串文件名；（？NotYetImplemented）字符串MatrixName；整数nrows；整数列；输出：实数res[nrows，ncolumns]；	从给定filename和matrixname的文件中读取矩阵。
ReadMatrixSize（文件名，矩阵名称）（？未实现）输入：字符串文件名；字符串MatrixName；输出：整数大小[2]；读取模拟结果（文件名，变量，大小）值的矩阵（每一列作为一个向量或变量的值）。结果的大小也作为输入给出。输入：字符串文件名；variableName变量[：]；整数大小；输出：实数res[size（variables，1），size）]；	从文件中读取矩阵，给定文件名，矩阵名称，#行 从给定矩阵名称的文件中读取矩阵维度。 读取变量列表的模拟结果，并返回
ReadSimulationResultSize（文件名）（？未实现）RunScript（文件名）	从文件中读取轨迹矢量的大小。输入：字符串文件名；输出：整数大小；
	执行作为参数给定的脚本文件。 输入：字符串文件名；输出：布尔RES；
SaveLog（文件名）（？NotYetImplemented）SaveModel（文件名，类名）	将日志保存到文件。 输入：字符串文件名；输出：布尔RES；
（NotYetImplemented）Save（类名）	将类定义保存在文件中。输入：字符串文件名； TypeName类名输出：Boolean res；
	将模型（A1）保存到从中加载模型的文件中。

	输入：类名类名
SaveTotalModel（文件名，类名）（？NotYetImplemented）Simulate（类名，选项）	将总类定义保存到类的文件中。输入：字符串文件名；TypeName类名输出：Boolean res；
	模拟模型，可选择设置模拟值。输入：TypeName类名；实际开始时间；实际停止时间；整数区间数；实输出区间；字符串方法；真正的宽容；实固定步长；字符串OutputFormat；输出：SimulationResult SimRes；
系统（文件名）	执行系统命令。输入：字符串文件名；输出：整数RES；
TranslateModel（类名）（？NotYetImplemented）	实例化模型，优化方程，生成代码。输入：TypeName类名；输出：SimulationObject res；
WriteMatrix（文件名，矩阵名，矩阵）（？NotYetImplemented）	在给定矩阵名称和矩阵的情况下，将矩阵写入文件。输入：字符串文件名；字符串MatrixName；实矩阵[：，:]；输出：布尔RES；

2.3.1 例子

OpenModelica中的以下会话说明了上述几个函数的使用。

```
>>模型测试真实X；结束测试；确定>>S：=列表（测试）；>>S模型测试真实X；方程der（X）=X；结束测试；“>>InstantiateModel（Test）”
FClass Test Real X；方程der（X）=X；结束测试；“>Simulate（Test）
Record ResultFile=”C：\OpenModelica1.6.0\Test_Res.PLT“ End Record>A：=1:10{1，2，3，4，5，6，7，8，9，10}>A*2{2，4，6，8，10，12，14，16，18，20}>ClearVariables（）True>List（Test）”
Model Test Real X；方程der（X）=X；
```

结束测试；>>
清除（）
true>>列出
（）{}

模拟后接绘图的常见组合：

>模拟（我的电路，停止时间=10。0);1.>绘图
（{RV}）；

有几种可能的输出格式。默认值为“ PLT ”，PLT是当前唯一能够使用Val（）或PLOT（）功能的格式。CSV（逗号分隔值）在数据密集的模拟中大约快两倍，并且在模拟期间不需要在RAM中分配所有输出数据。Empty根本没有输出，应该是目前为止最快的。

模拟（.，输出格式=“ CSV ”）模拟（.，输出格式=“ PLT ”）模拟（.，输出格式=“ EMPTY ”）

2.4用于模型查询的客户端-服务器非类型化高性能API

以下API主要设计用于客户端通过CORBA（或套接字）接口调用OpenModelica编译器/解释器，以获取有关模型结构的信息并操作模型结构，但这些函数也可以作为用户命令和/或脚本命令直接调用。API具有以下常规属性：•无类型，不执行类型检查。原因是高性能，每次调用的开销低。•所有命令都作为Modelica语法中的字符串发送；所有结果都以字符串形式返回。•多态类型命令。命令在内部被解析为Modelica抽象语法，但在某种程度上并不强制统一类型（类似于注释所允许的类型）。例如，像{true，3.14，“ hello ”}这样的向量可以被传递，即使这些元素具有混合的元素类型，这里是（Boolean，real，string），这在M odelica类型系统中是不允许的。交互式/的API增量开发由交互模块中的一组Modelica函数组成。对这些函数的调用可以作为纯文本从客户端发送到交互式环境，并使用Modelica的表达式解析器进行解析。如果被调用的函数名在此API的名称集中，则对此API的调用将被解析并从主模块路由到交互模块。所有API函数都返回字符串，例如，如果返回值true，则文本“ true ”将被发送回调用方，但不带字符串引号。•当函数无法执行其操作时，将返回字符串“-1 ”。•这些函数的所有结果都作为字符串返回（不带字符串引号）。该API可由人类用户在交互构建模型时直接使用，或通过使用脚本间接使用，但也可由例如想要与符号表交互以添加/更改/删除模型和组件等的模型编辑器使用。（？ 未来扩展:还描述了OpenModelica内部相应的内部调用）

2.4.1 定义

一个<ident><string><符	参数编号例如，A1是第一个参数，A2是第二个参数，等等。标识符，例如A或Modelica。Modelica字符串，例如“ Nisse ”或“ Foo ”。任意Modelica表达式..类引用，即类的名称，例如电阻器。

2.4.2 呼叫示例

调用满足正常的Modelica函数调用语法。例如:

保存模型（“ myresistorfile.Mo ”，myre sistor）

将模型MyResistor保存到文件“ MyResistorFile.Mo ”中。对于创建新模型，最实用的方法是向API发送模型声明，因为API还接受Modelica声明和Modelica表达式。例如，发送:

型号FOO END FOO;

将创建一个名为foo的空模型，而发送:

连接器端口端口;

将创建一个名为Port的新的空连接器类。在OpenModelica TestModels目录中的OMNotebook文件ModelQueryAPIExamples.onb中可以找到更多的API示例调用。

2.4.3 用于模型查询和操作的非类型化API函数

以下是OpenModelica环境中可用的非类型化API函数的简要说明，这些函数用于获取有关模型和/或操作模型的信息。API调用由交互式包中的EvaluateGraphicalAPI和EvaluateGraphicalAPI2解码。调用的结果以文本字符串的形式返回（没有字符串分隔符“ ”）。类型化API（第2.3节）中的函数由CEVAL包处理。OpenM Odelica TestModels目录中的文件ModelQueryAPIExample.onb中提供了对这些函数的可执行示例调用。在单独的文件OMC_API-HOWTO.PDF中提供了带有示例（包括下面未提及的一些函数）的附加的、更广泛的文档。

-源文件-getSourceFile（A1<字符串>）setSourceFile（A1<字符串>，A2<字符串>）	
	获取作为参数（A1）给出的类的源文件。将作为第一个参数（A1）给出的类与作为第二个参数（A2）给出的源文件关
	联，检索具有指定名称的环境变量。将具有指定名称的环境变量（A1）设置为给定值（A2）。加载文件中的所有模型。也在类型化API中。返回加载文件中顶级类的名称列表。加
-环境变量-GetEnvironmentVar（A1<String>）SetEnvironmentVar（A1<String>，A2<String>）-类和模型-LoadFile（A1<String>）	载文件中的所有模型。也在类型化API中。返回的列表
LoadFileInteractiveQualified	

（A1<字符串>）LoadFileInteractive（A1<字符串>）	加载的文件中顶级类的限定名。将给定的文件作为参数加载到编译器符号表中。与LoadFile的区别是什么？
备荷模型（A1<cref>）保存模型（A1<字符串>，A2<cref>）	通过在OpenModelicaLibrary中查找要加载的正确文件来加载模型（A1）。将该文件中的所有模型加载到符号表中。将模型（A2）保存在由字符串（A1）指定的文件中。此调用也在类型化API中。注意：？尚未完全实施。将模型（A1）保存到先前从中加载模型的文件中。此调用也在类型化API中。从符号表中删除类，将名称为A1的现有类重命名为名称（A2）。在引用类A1的所有已加载模型中递归执行重命名。注意：？该实现目前有很多错误/非常慢。检索给定类（A1）中所有元素的info属性。这包含元素类型、文件名、IsReadOnly、行信息、名称等的信息，以包含记录构造函数形式rec（.）上的元素描述符的向量的形式，例如：{rec（attr1=value1，
保存（A1<cref>）	
删除类（A1<cref>）重命名类（A1<cref>，A2<cref>）	
-类属性-getElementsInfo（A1<cref>）	属性2=值2.），.，记录（属性1=值1，属性2=值2.）}
setClassComment（A1<cref>，A2<string>）addClassAnnotation（A1<cref>，annotate=<expr>）	设置类（A1）字符串注释（A2）。 将A2给出的注释（格式为Annotate=ClassMod（..））添加到A1引用的模型定义中。应用于添加图标图和文档注释。返回由A1命名的类的图标注释。返回由A1命名的类的关系图注释。注1:由于图注释可以在基类中找到，因此执行部分代码实例化，使继承层次结构扁平化，以便找到所有注释。注2:由于部分展平，返回的格式不符合图表注释的Modelica标准。返回由命名的类/包中所有包的名称
GetIconAnnotation（A1<cref>）GetDiagramAnnotation（A1<cref>）	
获取软件包（A1<cref>）GetPackages（）。	A1作为列表，例如：{电气，块，力学，常数，数学，SIUNITS} 返回全局范围中所有包定义的名称。返回类/包中所有类定义的名称。返回全局范围中所有类定义的名称。返回客户端中作为模拟候选对象的所有“打开模型”的列表。
获取类别名称（A1<cref>）获取类别名称（）获取类别名称模拟（）	

为模拟设置类名（A1<ST Ring>）	在客户端中设置作为模拟候选的所有“打开模型”的列表。字符串必须符合以下格式：“{model1，model2，model3}”以以下格式返回所有可能的类信息：
GetClassAttributes（A1<cref>）	记录（属性1=值1，属性2=值2..）
GetClassRestriction（A1<cref>）	返回<cref>的限制类的种类，例如“模型”，“连接器”、“功能”、“封装”等
GetClassInformation（A1<cref>）	返回有关类A1的以下信息的列表：{“ restriction ”，“ comment ”，“ filename.Mo ”，{bool，bool，bool}，{“ re adonly|writable ”，int}}如果类是基元类型，则返回“ true ”，否则返回
---受限类谓词IsPrimitive（A1<cref>）	
	“假的”。
IsConnector（A1<cref>） IsModel（A1<cref>） IsRecord（A1<cref>） IsBlock（A1<cref>）IsType（A1<cref>）IsFunction（A1<cref>）IsPackage（A1<cref>）IsClass（A1<cref>）IsParameter（A1<cref>）	如果类是连接器，则返回“ true ”，否则返回“ false ”。如果类是模型，则返回“ true ”，否则返回“ false ”。如果类是记录，则返回“ true ”，否则返回“ false ”。如果类是块，则返回“ true ”，否则返回“ false ”。如果类是类型，则返回“ true ”，否则返回“ false ”。如果类是函数，则返回“ true ”，否则返回“ false ”。如果类是包，则返回“ true ”，否则返回“ false ”。如果A1是类，则返回“ true ”，否则返回“ false ”。如果A1是参数，则返回“ true ”，否则返回“ false ”。注：？尚未实施。如果A1为常量，则返回“ true ”，否则返回“ false ”。注：？尚未实施。如果A1受保护，则返回“ true ”，否则返回“ false ”。注：？尚未实施。如果类存在于SymbolTable中，则返回“ true ”，否则返回
是常数（A1<cref>）	
IsProtected（A1<cref>）	
ExistClass（A1<CREF）	“假的”。
-组件-获取组件（A1<cref>）	返回类A1中的组件声明列表：
	“{atype，varida，” commenta “}，{btype，varidb，” commentb “}，{.}”
setComponentProperties（A1<cref>，A2<cref>，A3<Boolean>，A4<Boolean>，A5<Boolean>，A6<Boolean>，A7<string>，A8<{Boolean，Boolean}>，A9<string>）	设置类（A1）中组件（A2）的以下属性。 -A3最终（对/错） -A4流（对/错） -A5受保护（正确）或公开（错误） -A6可替换（对/错） -A7变量：“恒定”或“离散”或“参数”或“ ” -A8动态_ref：{inner，outer}-两个布尔值。

	-A9因果关系：“输入”或“输出”或"返回A1中所有组件的
GetComponentAnnotations（A1<cref>）	所有注释的列表{.}，顺序与组件相同，每个组件一个注释。获取组件引用文件和位置信息。
GetCrefInfo（A1<cref>） }添加组件（A1<ident>，A2<cref>，	返回一个列表：{文件，只读|可写，开始行，开始列，结束行，结束列} >>getcrefinfo（bouncingball）{C：/openmodelica1.4.1/testmodels/bouncingball。Mo，可写，1，1，20，17
A3<cref>，注释=<expr>）	添加具有名称（A1）、类型（A2）和类别（A3）的组件作为论据。可选的注释由命名参数Annotate提供。删除类（A2）中的组件（A1）。
deleteComponent（A1<ident>，A2<cref>）updateComponent	
（A1<ident>，A2<cref>，a3<cref>，annotate=<expr>）	以名称（A1）、类型（A2）和类（A3）为参数更新现有组件。可选的注释由命名参数Annotate提供。将名称为（A1）的类中定义的名称为A2的现有组件重命名为新名称（A3）。
重命名组件（A1<cref>，A2<ident>，A3<ident>）	在引用类A2中声明的组件的所有已加载模型中递归执行重命名。注意：？该实现目前有很多错误/非常慢。返回类/组件A1中第n个组件（A2）（第一个没有1）的展平注释记录。由15个值的逗号分隔字符串组成，参见下面第2.4.4节中的注释，例如“ FALSE，10，30，..”返回第n个组件（A2）的修改，其中第一个没有类/组件A1的1）。返回类（A1）中组件（如变量、参数、常量等）（A2）的值。将类（A1）中的组
GetNthComponentAnnotation（A1<cref>，A2<int>）	件（例如变量、参数、常量等）（A2）的modfier值设置为表达式（未求值）在A3中。检索？的名称类中的所有组件。
GetNthComponentModification（A1<cref>，A2<int>）GetComponentModifierValue（A1<cref>，A2<cref）SetComponentModifierValue（A1<cref>，A2<cref>，A3<exp>）	
getComponentModifierNames（A1<cref>，A2<cref>）-继承-getInheritanceCount（A1<cref>）getNthinHeritedClass（A1<cref>，A2<int>）getExtendsModifierNames（A1<cref>）	返回类的继承类的数量（以字符串形式）。返回类的第n个继承类的类型名称。第一个类的编号为1。返回对EXTENDS子句进行修改的修饰符名称。例如：
	模型试验扩展A（P1=3，P2（Z=3））；结束测试； getExtendsModifierNames（test，a）=>{P1，P2}
GetExtendsModifierValue（A1<cref>）	的扩展子句的子修饰符值。

	实例，“模型测试扩展A（P1=3，P2（Z=3））；结束测试；” getExtendsModifierValue（Test，A，P1）=>=3
-连接-getConnectionCount （A1<cref>）getNthConnection （A1<cref>，A2<int>）	返回模型中的连接数（以字符串形式）。
	返回第n个连接，以逗号分隔的连接器对的形式，例如1.“ RN，R2.P ”。第一个有数字1。
GetNthConnectionAnnotation（ A1<cref>，A2<int>）	以逗号分隔列表的形式返回第n个连接批注 对于展平记录的值，请参见下面第2.4.4节中的注释。
添加连接（A1<cref>，A2<cref>， A3<cref>，注释=<expr>）	将连接CONNECT（A1，A2）添加到模型A3，其中由命名参数Annotate提供的注释。更新已存在的连接。
UpdateConnection（A1<cref>， A2<cref>，A3<cref>， Annotate=<expr>）DeleteConnection （A1<cref>，A2<cref>，A3<cref>）- Equations-Addequation（A1<cref>， A2<expr>，A3<expr>）（？ Notyetimplemented）	删除A3给出的类中的连接CONNECT（A1，A2）。
GetQuationCount（A1<cref>）	将等式A2=A3添加到由A1命名的模型中。
返回名为A1的模型（？ NotYetImplemented）中的方程数（以字符串形式）。（包括连接）	
getNthEquation（A1<cref>，A2<int>）	返回由A1命名的模型的第n个（A2）方程式。例如（？ notyetimplemented）“ der（X）=-1 ”或“ connect （a.B，C.a）”。第一个有 1号。
删除方程（A1<cref>，A2<int>） （？ 未实现）	删除由A1命名的模型中的第n个（A2）方程。第一个有数字1。
-杂项-检查设置 （）。	
	GetSettings（）的改进版本。用于调试用户的设置。它检查编译器是否已安装并正常工作，是否设置了环境变量，使用了哪个操作系统，等等。
GetVersion（）GetASTACorbastrin G （[FileName=<String>]）	返回OMC版本，例如“ 1.4.2 ”此命令使用UnionTypes的Java CORBA格式将所有加载文件的内部AST解析为文本。如果给出了文件名，则将文本转储到该文件，而不是通过CORBA发送。这很有用，因为您可以将内部AST保存在文件中以供将来使用。如果您在通过CORBA发送大文本时遇到问题，您还可以使用该文件作为中间输出，以克服32位平台上的CORBA或RML实现中的错误和限制。
dumpXMLDAE（模型名[，asInSimula tionCode=<Boolean>][， FilePrefix=<String>][， StoreIntemp=<Boolean>][， AddMathMLCode=<Boolean>]）	此命令使用XML表示转储模型的数学表示，并带有可选参数输入：类型名称类名；布尔型ASInSimulationCode；字符串FilePrefix；布尔存储EMP；布尔型AddMathMlCode； 输出：字符串XmlFile 特别是，ASInSimulationCode定义了停止的位置

	在转换过程中（在转储模型之前），其他选项与文件存储有关：FilePrefix用于指定不同的名称，StoreIntemp用于使用临时目录。可选参数 AddMathMLCode 提供了不在XML文件中打印MathML代码的可能性，以使其更具可读性。用法很简单，只需：AddMathMLCode=true/false（默认值为false）。有关示例，请参见第2.5.5节。
导出DaetomatLab（模型名）	将模型的关联矩阵转储为MATLAB格式。见第2.5.6节。
SetDebugFlags（A1=<String>） 在交互式会话中启用调试标志。即使启动OMC时未设置标志（大多数交互式客户端启动OMC时未设置任何标志），也可用于启用故障跟踪。	

2.4.3.1 错误处理

当上述任何函数发生错误时，将返回字符串“-1 ”。

2.4.4 注释

注释可以出现在几种Modelica构造中。

2.4.4.1 变量注释

变量注释（即元件注释）是对以下（展平的）Modelica记录的修改:

记录放置布尔值visible=true；实变换.X=0；实变换.y=0；实变换.scale=1；实变换.aspectratio=1；布尔变换.fliphoriz ontal=false；布尔变换.flipverti cal=false；实变换.旋转=0；实数 IconTransformation.X=0；实数IconTransformation.y=0；实数 IconTransformation.Scale=1；实数 IconTransformation.aspectratio=1；布尔型 icontransformation.fliph orizontal=false；布尔型 icontransformation.flipv ertical=false；实数 iconTransformation.Rotation=0；末端放置；

2.4.4.2 连接注释

连接注释是对以下（展平的）Modelica记录的修改:

记录线实点[2][：]；整数颜色[3]={0，0，0}；枚举（无，实线，虚线，点，点划线，点划线）模式=实线；实际厚度=0.25；枚举（无，开放，填充，半F）箭头[2]={无，无}；实际箭头大小=3.0；

布尔平滑=假；端线；

这是平面记录图标，用于图标图层注释

记录图标实坐标系统.extent[2，2]={{-10，-10}，{10，10}}）；GraphicItem[：]图形；结束图标；

这种平面记录的文本表示在某种程度上更为复杂，因为图形向量在概念上可以包含不同的子类，如线条、文本、矩形等。为了解决这个问题，我们将使用记录构造函数作为这些函数的表达式。例如，以下注释:

注释（图标（坐标系统={-10，-10}，{10，10}，图形={矩形（范围={-10，-10}，{10，10}），文本（{-10，-10}，{10，10}，文本字符串=“图标”）}））；

将生成平面记录图标的以下字符串表示形式:

{-10，10}，{10，10}，{Rectangle（Tru e，{0，0，0}，{0，0，0}，LinePattern.Solid，FillPattern.None，0.25，BorderPattern.None，{-10，-10}，{10，10}，0），Text（{-10，-10}，{10，10}，TextString=“ icon ”）}

以下是图表注释的平面记录:

记录图实坐标系统.extent[2，2]={{-10，-10}，{10，10}}）；GraphicItem[：]图形；端面图；

平面记录字符串表示形式与图标注释的平面记录相同。

2.4.4.3图元平面记录

记录行布尔可见=真；实点[2，：]；整数颜色[3]={0，0，0}；线阵图案=线阵n.solid；实际厚度=0.25；箭头箭头[2]={arrow.none，ar row.none}；实际箭头大小=3.0；布尔平滑=假；端线；记录多边形布尔值visible=true；整数LineColor[3]={0，0，0}；整数FillColor[3]={0，0，0}；线阵图案=线阵n.solid；fillpattern fillpattern=fillpattern.none；真实线度=0.25；实点[2，：]；布尔平滑=假；结束多边形；记录矩形布尔可见=真；整数LineColor[3]={0，0，0}；整数FillColor[3]={0，0，0}；线阵图案=线阵n.solid；fillpattern fillpattern=fillpa ttern.none；真实线度=0.25；

border pattern border pattern=Bo rderpattern.none；真实范围
[2，2]；真实半径；端矩形；记录椭圆布尔可见=真；整数
LineColor[3]={0，0，0}；整数FillColor[3]={0，0，0}；线阵图案=线
阵n.solid；fillpattern fillpattern=fillpa ttern.none；真实线度
=0.25；真实范围[2，2]；端椭圆；记录文本布尔值visible=true；整数
LineColor[3]={0，0，0}；整数FillColor[3]={0，0，0}；线阵图案=线
阵n.solid；fillpattern fillpattern=fillpa ttern.none；真实线度
=0.25；真实范围[2，2]；字符串TextString；实际字体大小；字符串
字体名称；文本样式文本样式[：]；结束文本；记录位图布尔值
visible=true；真实范围[2，2]；字符串文件名；字符串
ImageSource；结束位图；

2.5 类型化命令API的Modelica标准化探讨

交互功能接口可以是Modelica规范或基本原理的一部分。为了添加这一点，不同的实现（OpenModelica、Dymola和其他）需要在通用API上达成一致。本节介绍了在决定标准API时需要考虑的一些命名约定和其他API设计问题。

2.5.1 命名约定

建议：函数名称应该以非大写字母开头，并且名称中的每个新单词都有一个大写字母，例如

加载模型打开模
型文件

2.5.2 返回类型

当前的实现之间存在差异。OpenModelica非类型化API返回字符串“ OK ”、“-1 ”、“ false ”、“ true ”等，而类型化OpenM Odelica命令API和Dymola返回布尔值，例如true或false。

建议：所有不返回信息的函数，例如getModelName，应该返回一个布尔值。（？注意：这不是最终的解决方案，因为我们还需要处理返回信息的函数的失败指示，当异常处理可用时，这可以做得更好）。

2.5.3参数类型

关于某些函数的参数类型，实现之间也存在差异。例如，Dymola使用字符串来表示模型和变量引用，而OpenModelica直接使用模型/变量引用。例如，Dymola中的LoadModel（“ Resistor ”），而OpenModelica中的LoadModel（Resistor）。由于M odelica在不久的将来可能会出现功能过载，因此也可以同时支持这两种替代方案。

2.5.4 API函数集

当然，主要的问题是要包括哪些功能子集，以及它们应该做什么。下面是合并在一起的Dymola和OpenModelica函数表。该表还包含对可能标准的建议。

<s>=字符串<Cr>=组件引用[]=列表构造函数，例如[<s>]=字符串的向量Dymola OpenModelica描述建议

列表（）列表变量（）		列出所有用户定义的ListVariables（）变量。列出内置函数ListFunctions（）名称和说明。列出所有加载的类列表（）定义。列表（<cref>）或<cref>的列表模型定义。	
ListFunctions（）-			
-	列出（）。		
-	列表（<cref>）	LIST（<String>）返回当前	CurrentDirectory（）。
ClassDirectory（）CD（）。		人名地址录删除模型。	
Eraseclasses（）清除（）。	清除类（）清除（）。		ClearClasses（）。
		删除所有（包括清除所有）（）模型和变量。	
-	ClearVariables（）删除所有用户定义的变量。	ClearVariables（）。	
-	ClearClasses（）删除所有类	ClearClasses（）。定义。从文件加载所有定义。包含LoadModel（<cref>）的加载文件，	
OpenModel（<字符串>）LoadFile（<字符串>）			加载文件（<字符串>）
OpenModelFile（<字符串>）SaveTotalModel（-<字符串>，<字符串>）	负载模型（<cref>）	模型。LoadModel（<String>）保存总模型SaveTotalModel（<ST	
		Ring>、<cref>中模型的	定义）或

	SaveTotalModel（<string>，<string>）SaveModel（<cref>，将模型保存在文件中。SaveModel（<string><string>），<cref>）或SaveModel（<string>，<string>）CreateModel（<cref>）创建新的空CreateModel（<cref>		
-			
-	）或创建模型（<字符串>）删除模型	模型。（<cref>）从删除模型	（<cref>）中删除模型
擦除类（{<字符串>}）		符号表）或	删除模型（<字符串>）
实例化模型（实例化类（<字	符串><cref>）	执行代码 类的实例化。cref>）或	实例化类（< 实例化类（<字符串>）

2.5.5 从模型导出XML的示例

下面是使用函数dumpXMLDAE导出模型的XML表示的示例。

模型电路1参数实数C（最小值=5e-07，最大值=2e-06）
=1e-06；参数实数R1=50；参数实数R2=50；参数实数R3（最小
值=500，最大值=2

000)=1000;输入实数I0；真实的I1；真正的I3；真实V1；真
实的V2；输出实数V3；方程C*der（V1）=I0-I1；V1-
V2=I1*R1；V2-V3=I1*R2；C*DER（V3）=I1-I3；
V3=R3*I3；结束电路1；

1.loadFile（'./path_Mo_文件的_/circu itMo '）；转储
XMLDAE（电路1）；

将产生以下结果：{“<? XML version=” 1.0 “ encoding=” UTF-8 “? ><DAE xmlns：P1=” HTTP：/
www.w3.org/1998/math/mathml “ xmlns：XLink=” HTTP：/www.w3.org/1999/XLink “ xmlns：xsi=”
HTTP：/www.w3.org/2001/xmlschema-instance “ xsi：noNamespaceSchemaLocation=” HTTP：/
home.dei.polimi.it/donida/projects/autoedit/images/DAE.XSD “><Variables Dimension=” 11
“><OrderedVariables Dimension=” 6 “><VariablesList><Variable ID=” 1 “ Name=” V3 “ Variability=”
ContinuousState “ Direction=” Output “ Type=” Real “ Index=” -1 “ Origname=” V3 “ Fixed=” True “
Flow=” NonConnector "><ClassesNames></ClassesNames></Variable><Variable ID= “ 2 ” Name= “ V2
” Variability= “ Continuous ” Direction= “ None ”

type= “ real ” index= “-1 ” origname= “ v2 ” fixed= “ false ” flow= “ nonconnector ” ><ClassesNames><Element>circuit 1</Element></ClassesNames></Variable><Variable ID= “ 3 ” name= “ v1 ” variability= “ continuousState ” direction= “ none ” type= “ real ” index= “-1 ” origname= “ v1 ” fixed=" true “ flow=” nonconnector “><ClassesNames><Element>circuit 1</Element></ClassesNames></Variable><Variable ID=” 4 “ name=” i3 “ variability=” continuous “ direction=” none “ type=” real “ index=” -1 “ origname=” i3 “ fixed=” false “ flow=” nonconnector "><ClassesNames><Element>circuit 1</Element></CLASSESNAMES></VARIABLE><VARIABLE ID= “ 5 ” NAME= “ I1 ” VARIABLY= “ CONTINUOUS ” DIRECTION= “ NONE ” TYPE= “ REAL ” INDEX= “-1 ” ORIGNAME= “ I1 ” FIXED= “ FALSE ” FLOW= “ NONCONNECTOR ” ><CLASSESNAMES><ELEMENT>电路1</variable><variable ID= “ 6 ” name= “$dummy ” variability= “ continuousState ” direction= “ none ” type= “ real ” index= “-1 ” origname= “$dummy ” fixed= “ true ” flow= “ nonconnector ” ><attributesValues><fixed string= “ true ” ><mathML><math xmlns= “ HTTP：//www.w3.org/1998/math/mathML ” ><apply><true/></apply></math></mathml></fixed></attributesvalues></variable></variableslist></orderedvariables><knownvariables dimension= “ 5 ” ><variableslist><variable ID= “ 1 ” name= “ I0 ” variability= “ continuous ” direction= “ input ” type= “ real ” index= “-1 ” origname= “ I0 ” fixed= “ false ” flow= “ nonconnector ” ><ClassesNames><Element>Circuit1</Element></ClassesNames></Variable><Variable ID= “ 2 ” name= “ R3 ” variability= “ parameter ” direction= “ none ” type= “ real ” index= “-1 ” origname= “ R3 ” fixed= “ true ” flow= “ nonconnector ” ><bindValueExpression><bindexpression string= “ 1000 ” ><mathML><math xmlns= “ HTTP：//www.w3.org/1998/math/mathML ” ><CN type= “ integer ” >1000</CN></math></mathML></bindexpression></bindValueExpression><ClassesNames><Element>Circuit1</Element></ClassesNames><AttributesValues><minValustring="500.3. “><mathml><math xmlns=” HTTP：//www.worg/1998/math/mathml “><CN type=” real “>500.0</CN></math></mathml></minvalue><maxvalue string=” 2000.0 “><mathml><math xmlns=” HTTP：//www.w3.org/1998/math/mathml “><CN type=” real ">2000.0</CN></math></mathml></maxvalue></attributesvalues></variable><variable ID= “ 3 ” name= “ R2 ” variability= “ parameter ” direction= “ none ” type= “ real ” index= “-1 ” origname= “ R2 ” fixed= “ true ” flow= “ nonconnector ” ><bindValueExpression><bindexpression string= “ 50 ” ><mathML><math xmlns= “ HTTP：//www.w3.org/1998/math/mathML ” ><CN type= “ integer ” >50</CN></math></mathML></bindexpression></bindValueExpression><ClassesNames><element>Circuit1</element></ClassesNames></variable><variable ID="4 “ name=” R1 “ variability=” parameter “ direction=” none “ type=” real “ index=” -1 “ origname=” R1 “ fixed=” true “ flow=” nonconnector “><BindValueExpression><BindexPression String=” 50 “><mathML><math xmlns=” HTTP：//www.w3.org/1998/math/mathML “><CN type=” integer ">50</CN></math></mathml></bindexpression></bindvalueexpression><ClassesNames><element>circuit1</element></ClassesNames></variable><variable ID= “ 5 ” name= “ C ” variability= “ parameter ” direction= “ none ” type= “ real ” index= “-1 ” origname= “ C ” fixed= “ true ” flow= “ nonconnector ” ><bindvalueexpression><bindexpression string= “ 1e-06 ” ><mathml><math xmlns= “ http：//www.w3.org/1998/Math/MathML ” ><CN type= “ real ” >1e-06</CN></math></mathml></bindexpression></bindvalueexpression>

<ClassesNames><Element>Circuit1</Element></ClassesNames><AttributesValues><MinValue String=“ 5e-07
” ><MathML><Math xmlns=“ HTTP：//www.w3.org/1998/Math/MathML ” ><CN type=“ real ” >5e-07</CN></Math></
MathML></MinValue><MaxValue String=“ 2e-06 ” ><MathML>3.HTTP：//www.worg/1998/math/mathml“><CN type=” real
“>2e-06</CN></math></mathml></maxvalue></attributesvalues></variable></variableslist></knownvariables></
variables><equations dimension=” 6“><equation ID=” 1 ">C*der（v1）=I0-I13.<mathm l><math xmlns=“ HTTP：//
www.worg/1998/math/mathml ” ><apply><equivalent/><apply><times/><ci>C</ci><apply><diff/><ci>v1</ci></apply></
apply><apply><minus/><ci>I0</ci><ci>I1</ci></apply></apply></math></mathml></equation><equation ID=“ 2 ” >v1-
v2=I1*R1<mathml><math xmlns=“ HTTP：//www.w3.org/1998/math/mathml ” ><apply><equivalent/><apply><minus/
><ci>v1</ci><ci>v2</ci></apply><apply><times/><ci>I1</ci><ci>R1</ci></apply></apply></math></mathml></
equation><equation ID=“ 3 ” >v2-v3=I1*R2<mathml><math xmlns=“ HTTP：//www.w3.org/1998/math/mathml
” ><apply><equivalent/><apply><minus/><ci>v2</ci><ci>v3</ci></apply><apply><times/><ci>I1</ci><ci>R2</ci></apply></
apply></math></mathml></equation><equation ID=“ 4 ” >C*der（v3）=I1-i3<mathml><math xmlns="3.www.worg/1998/
math/mathml><apply><equivalent/><apply><times/><ci>C</ci><apply><diff/><ci>v3</ci></apply></apply><apply><minus/
><ci>I1</ci><ci>i3</ci></apply></apply></math></mathml></equation><equation ID=“ 5 ” >v3=R3*i3<mathml><math
xmlns=“ HTTP：//www.w3.org/1998/math/mathml ” ><apply><equivalent/><ci>v3</ci><apply><times/><ci>R3</ci><ci>i3</
ci></apply></apply></math></mathml></equation><equation ID=“ 6 ” >der（$dummy）=sin（time*628.318530717）
<mathml><math xmlns=" HTTP：//www.w3.org/1998/math/mathml“><apply><equivalent/><apply><diff/><ci>$dummy</
ci></apply><apply><sin/><apply><times/><ci>time</ci><CN type=” real ">628.318530717</CN></apply></math></
1.MathML></Equation></Equations></DAE>“，” 模型已转储到XML文件：circuitXML "}

2.5.6从模型中导出MATLAB的示例

根据几个可选参数，命令export转储模型的XM L表示。

ExportDaetomatLab（模型名）；

此命令使用M atlab表示法转储模型的数学表示法。示例：

$cat daequery.MOS加载文件（“ bouncingball.Mo ”）；出口Daetomatlab（弹跳球）；readFile（“弹跳球_imatrix.m ”）；$OMC DAEQUERY.MOS TRUE“方程式系统被转储到MATLAB文件：BouncingBall_IMATRIX.M ”"%关联矩阵%=%行数:6 IM={[3，-6]，[1，{' if '，' true '，'='{3}，{}，}]，[2，{' if '，' edge（impact）'{3}，{5}，}]，[4，2]，[5，{' if '，' true '，'=='{4}，{}，}]，[6，-5]}；VL={' foo '，' V_new '，' impact '，' flyi ng '，' V '，' H '}；eqstr={' impact=H<=0.0；'，' foo=if impact then 1 else 2；'，' when{H<=0.0 and V<=0.0，impact}then V_new=if e dge（impact）then（-e）*pre（V）else 0.0；结束当；'，'当{H<=0.0和V<=0.0，影响}然后飞行=V_新>0.0；End when；'，' der（V）=if flying then-G else 0.0；'，' der（H）=V；'}；oldeqstr={' fClass BouncingBall '，'参数Real e=0.7“恢复系数”；'，'参数Real G=9.81“重力加速度”；'，' Real H（start=1.0）“球的高度”；'，' Real V“球的速度”；'，'Boolean flying（start=true）“ true，如果球在飞”；'，' boole an impact；'，' real V_new；'，' integer foo；'，' equation '，' impact=H<=0.0；'，' foo=if impact then 1 else 2；'，' der（V）=if flying then-G else 0.0；'，' der（H）=V；'，'当{H<=0.0且V<=0.0，影响}则'，' V_new=if ed ge（影响）then（-e）*pre（V）else 0.0；'，'飞行=V_new>0.0；'，' reinit（V，V_new）；'，'结束when；'，'结束弹跳球；'，' '};

第3章OpenModelica软件包的详细概述

本章概述了OpenModelica编译器/解释器和服务器功能中的所有软件包，以及模块之间的详细互连结构。

3.1编译程序包的详细互连结构

？中描述了互连结构的详细视图，即OpenModelica编译器中模块之间的主要数据流和依赖关系。下图3-1。

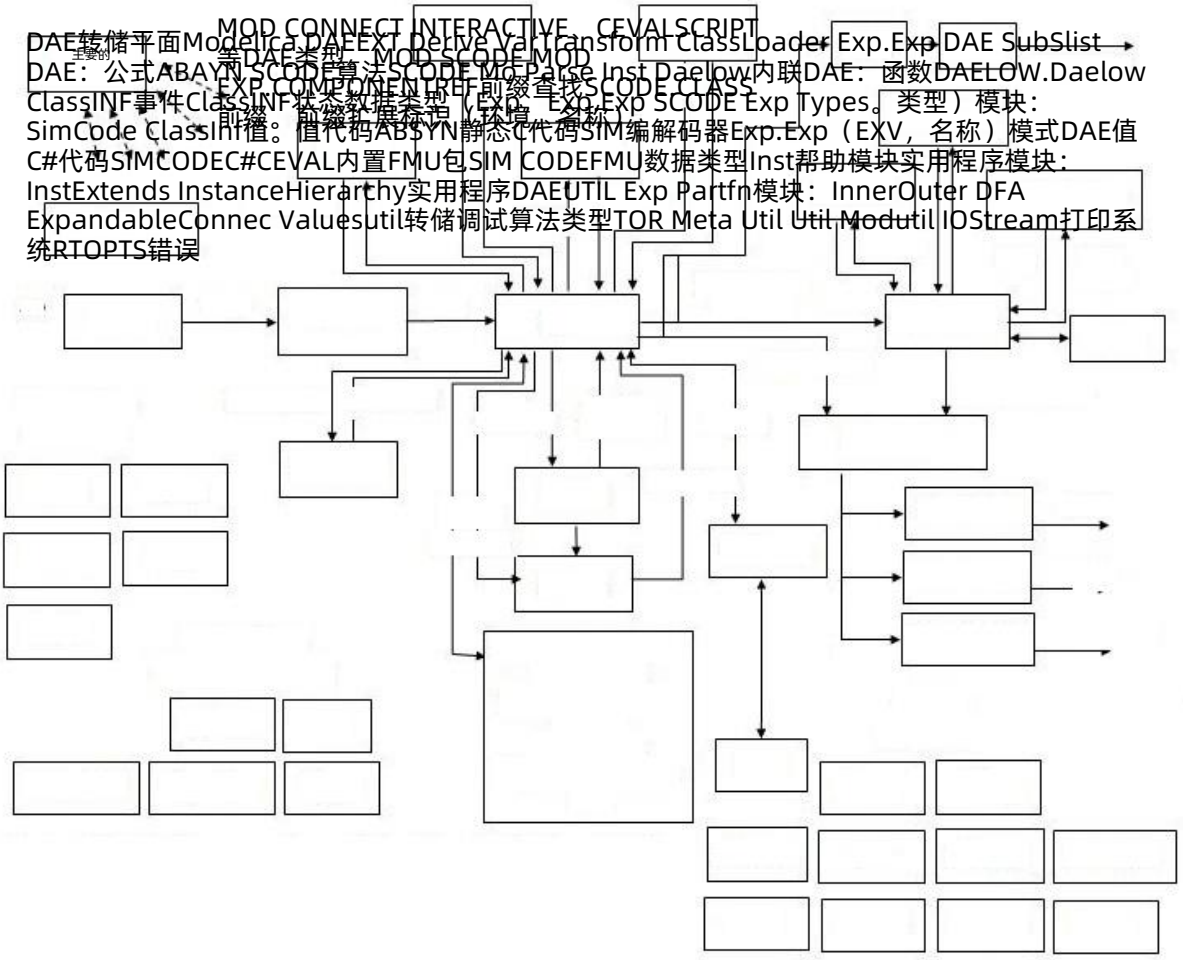


图3-1。OpenModelica编译器中的模块连接和数据流。注意：许多新模块尚未包括在此图片中。