MetaModelica – A Symbolic-Numeric Modelica Language and Comparison to Julia

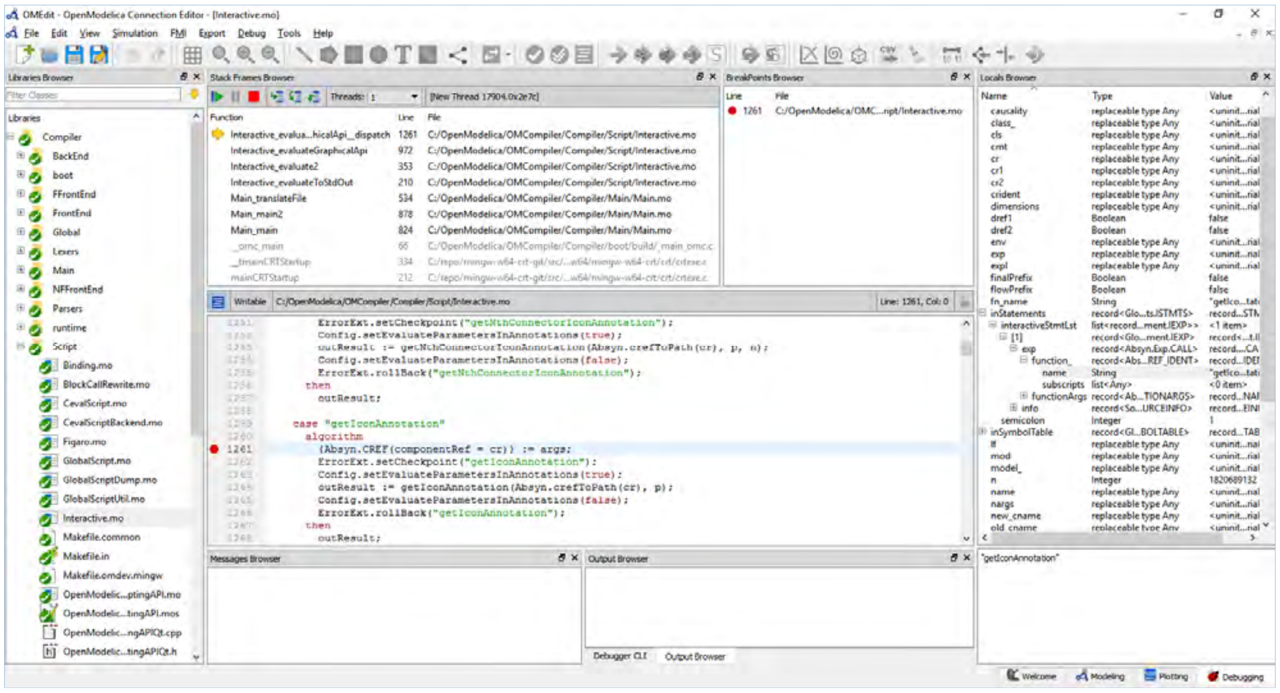
Peter Fritzson Adrian Pop Martin Sjölund Adeel Asghar

PELAB – Programming Environment Lab, Dept. of Computer and Information Science
Linköping University, SE-581 83 Linköping, Sweden
{peter.fritzson,adrian.pop,martin.sjolund,adeel.asghar}@liu.se

Abstract

The need for integrating system modeling with advanced tool capabilities is becoming increasingly pronounced. For example, a set of simulation experiments may give rise to new data that are used to systematically construct a series of new models, e.g. for further simulation and design optimization. Such combined symbolic-numeric capabilities have been pioneered by dynamically typed interpreted languages such as Lisp and Mathematica. Such capabilities are also relevant for advanced modeling and simulation applications but lacking in the standard Modelica language. Therefore, this is a topic of long-running design discussions in the Modelica Design group. One contribution in this direction is MetaModelica, that has been developed to extend Modelica with symbolic operations and advanced data structures, while preserving safe engineering practices through static type checking and a compilation-based efficient implementation. Another recent effort is Modia, implemented using the Julia macro mechanism, making it dynamically typed but also adding new capabilities. The Julia language has appeared rather recently and has expanded into a large and fast-growing ecosystem. It is dynamically typed, provides both symbolic and numeric operations, advanced data structures, and has a just-in-time compilation-based efficient implementation. Despite independent developments there are surprisingly many similarities between Julia and MetaModelica. This paper presents MetaModelica and its environment as a large case study, together with a short comparison to Julia. Since Julia may be important for the future Modelica, some integration options between Modelica tools and Julia are also discussed, including a possible approach for implementing MetaModelica (and OpenModelica) in Julia.

Keywords: Modelica, MetaModelica, symbolic, Julia, meta-programming, language, compilation



```
Function                               Line  File
Interactive_evalua...hicalApi__dispatch 1261 C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateGraphicalApi        972  C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluate2                   353  C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Interactive_evaluateToStdOut            210  C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo
Main_translateFile                      534  C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main2                              878  C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
Main_main                               824  C:/OpenModelica/OMCompiler/Compiler/Main/Main.mo
__omc_main                              66   C:/OpenModelica/OMCompiler/Compiler/boot/build/_main_omc.c
__tmainCRTStartup                       334  C:/repo/mingw-w64-crt-git/src/...w64/mingw-w64-crt/crt/crtexe.c
mainCRTStartup                          212  C:/repo/mingw-w64-crt-git/src/...w64/mingw-w64-crt/crt/crtexe.c
```

```
Line  File
1261  C:/OpenModelica/OMC...ript/Interactive.mo
```

```
Writable  C:/OpenModelica/OMCompiler/Compiler/Script/Interactive.mo    Line: 1261, Col: 0
```

```
        ErrorExt.setCheckpoint("getNthConnectorIconAnnotation");
        Config.setEvaluateParametersInAnnotations(true);
        outResult := getNthConnectorIconAnnotation(Absyn.crefToPath(cr), p, n);
        Config.setEvaluateParametersInAnnotations(false);
        ErrorExt.rollBack("getNthConnectorIconAnnotation");
      then
        outResult;

    case "getIconAnnotation"
      algorithm
1261    (Absyn.CREF(componentRef = cr)) := args;
        ErrorExt.setCheckpoint("getIconAnnotation");
        Config.setEvaluateParametersInAnnotations(true);
        outResult := getIconAnnotation(Absyn.crefToPath(cr), p);
        Config.setEvaluateParametersInAnnotations(false);
        ErrorExt.rollBack("getIconAnnotation");
      then
        outResult;
```

Name	Type	Value
caucality	replaceable type Any	<uninit...rial
class_	replaceable type Any	<uninit...rial
cls	replaceable type Any	<uninit...rial
cmt	replaceable type Any	<uninit...rial
cr	replaceable type Any	<uninit...rial
cr1	replaceable type Any	<uninit...rial
cr2	replaceable type Any	<uninit...rial
crident	replaceable type Any	<uninit...rial
dimensions	replaceable type Any	<uninit...rial
dref1	Boolean	false
dref2	Boolean	false
env	replaceable type Any	<uninit...rial
exp	replaceable type Any	<uninit...rial
expl	replaceable type Any	<uninit...rial
finalPrefix	Boolean	false
flowPrefix	Boolean	false
fn_name	String	"getIco...tat
inStatements	record<Glo...ts.ISTMTS>	record...STM
interactiveStmtLst	list<record...ment.IEXP>>	<1 item>
[1]	record<Glo...ment.IEXP>	record<...t.I
exp	record<Absyn.Exp.CALL>	record....CA
function_	record<Abs...REF_IDENT>	record...IDEI
name	String	"getIco...tat
subscripts	list<Any>	<0 item>
functionArgs	record<Ab...TIONARGS>	record...NAI
info	record<So...URCEINFO>	record...EINI
semicolon	Integer	1
inSymbolTable	record<Gl...BOLTABLE>	record...TAB
lt	replaceable type Any	<uninit...rial
mod	replaceable type Any	<uninit...rial
model_	replaceable type Any	<uninit...rial
n	Integer	1820689132
name	replaceable type Any	<uninit...rial
nargs	replaceable type Any	<uninit...rial
new_cname	replaceable type Any	<uninit...rial
old_cname	replaceable type Any	<uninit...rial

"getIconAnnotation"

Figure 1. The integrated MetaModelica OMEdit-based development environment in debugging mode. *Left*: the package browser. *Top*: the active stack frames (including C routines) and breakpoints. *Middle*: text editing and breakpoint setting. *Right*: the local variables browser. The user can switch to modeling mode which has both textual and graphical editing.