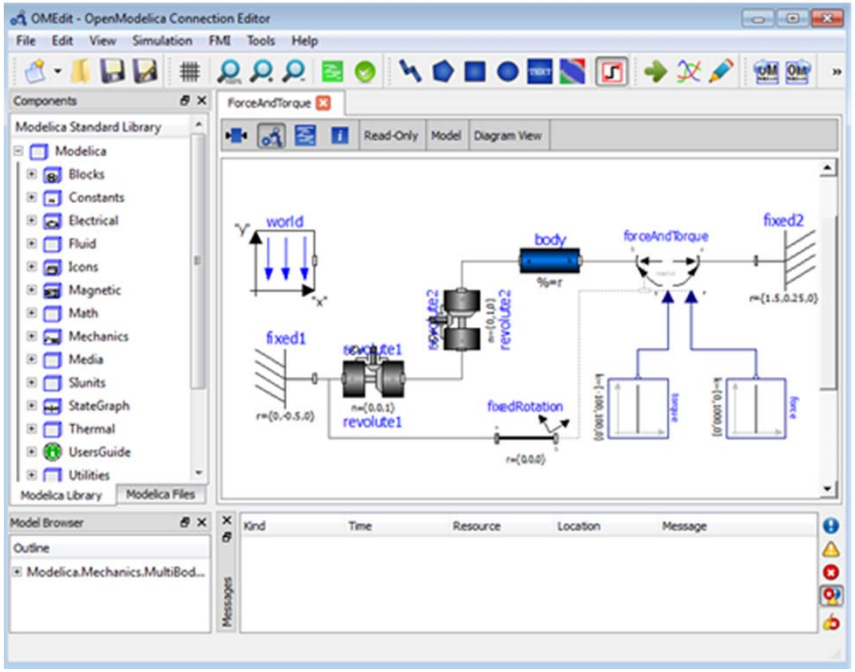
Tool Demonstration: OpenModelica Graphical Editor and Debugger

Adeel Asghar and Peter Fritzson

EOOLT Workshop
April 19, 2013, Nottingham



Linköpings universitet

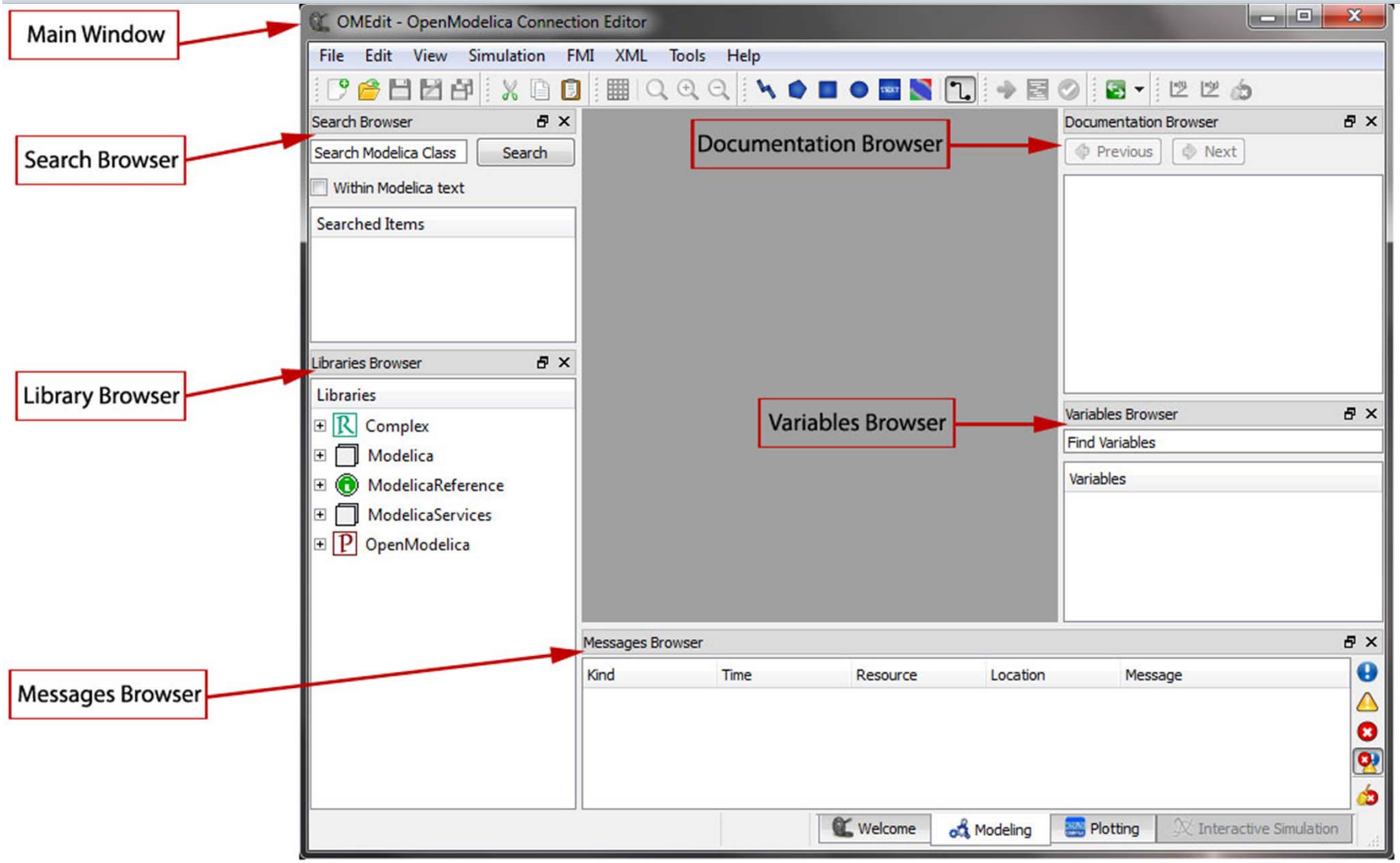
MODELICA

OpenModelica OMEdit Graphic Editor

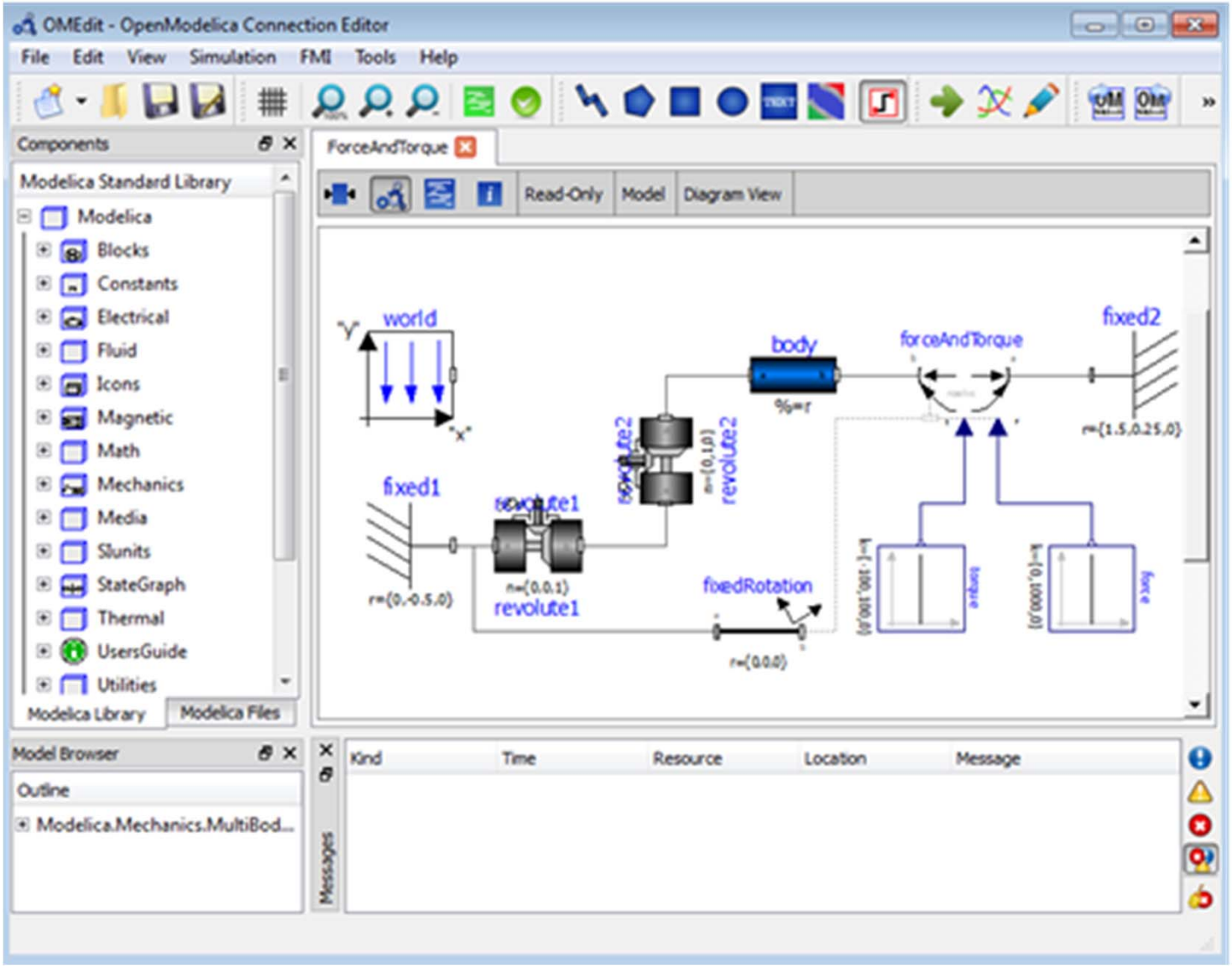
GUI Improvements in coming 1.9.0 final release (now available in nightly builds)

- Better Package support
- Better and simpler access to parameters
- Better model documentation display

OpenModelica OMEDit Graphic Editor & GUI

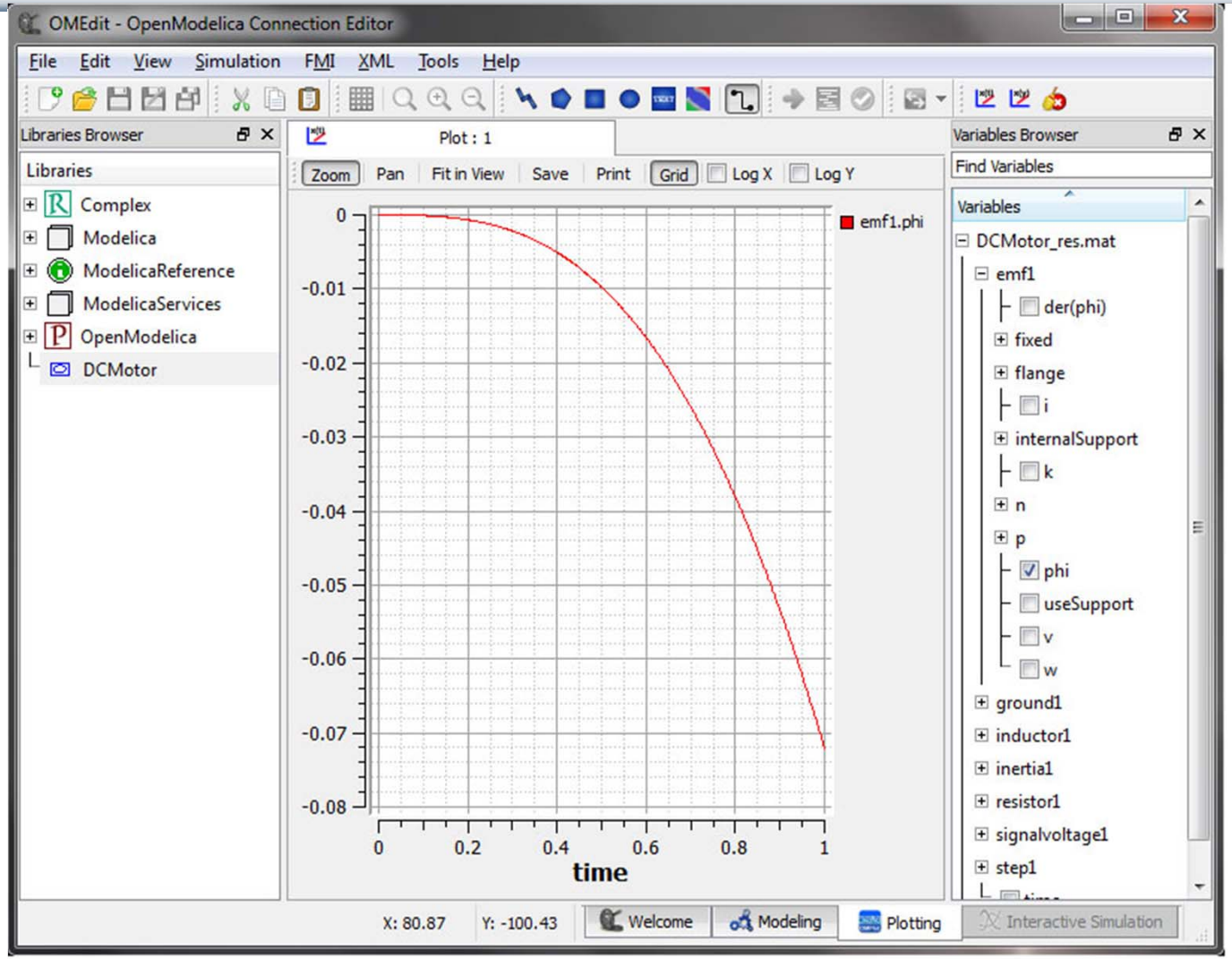


MODELICA


OMEdit - OpenModelica Connection Editor
File Edit View Simulation FMI Tools Help
Components
Modelica Standard Library
Modelica
Blocks
Constants
Electrical
Fluid
Icons
Magnetic
Math
Mechanics
Media
SIunits
StateGraph
Thermal
UsersGuide
Utilities
Modelica Library
Modelica Files
ForceAndTorque
Read-Only
Model
Diagram View
world
body
forceAndTorque
fixed2
fixed1
revolute1
revolute2
fixedRotation
Model Browser
Outline
Modelica.Mechanics.MultiBod...
Messages
Kind
Time
Resource
Location
Message

MODELICA

Plot Example



Show Demo Movie

MODELICA

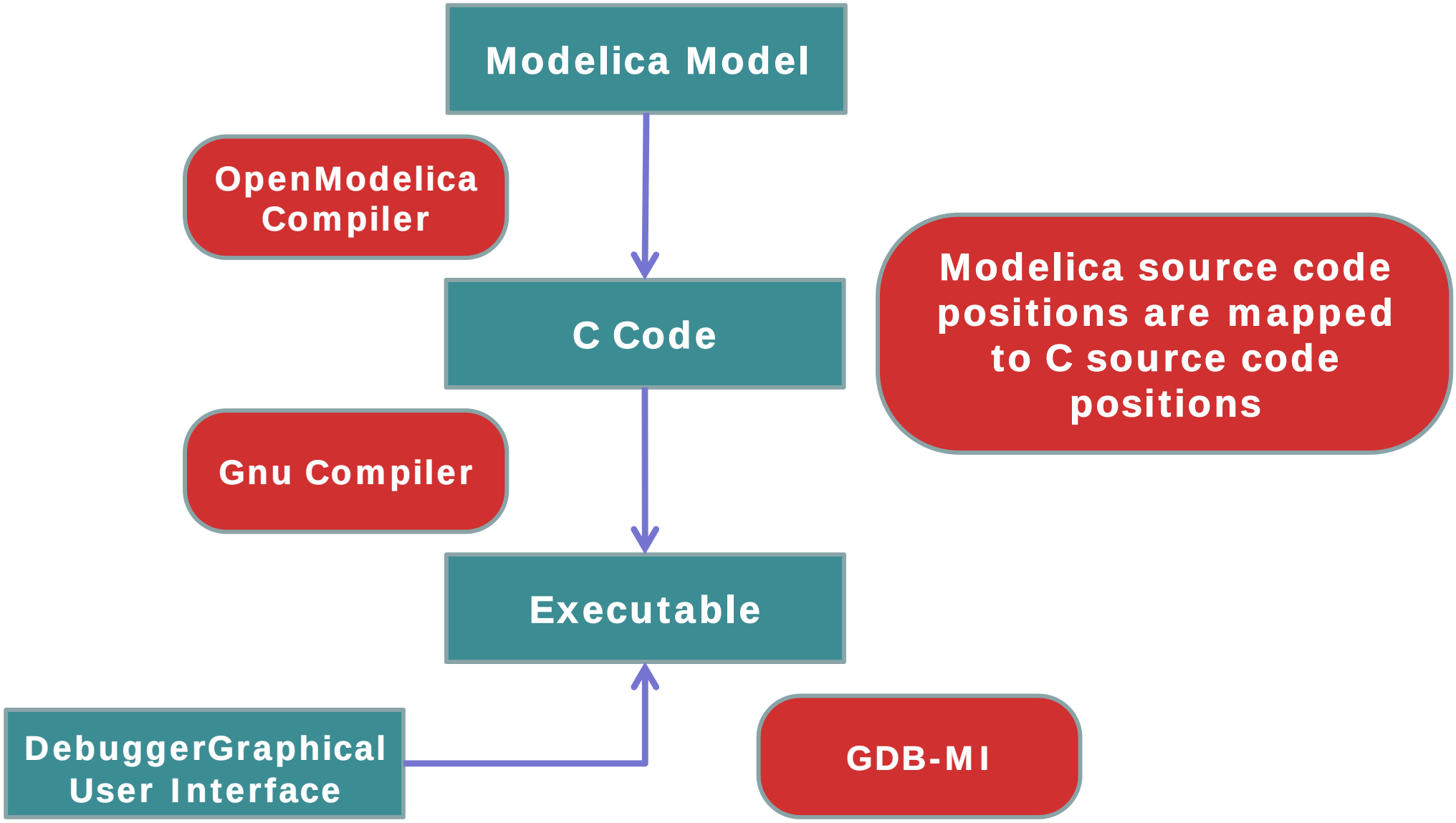
Static vs Dynamic Debugging

- Static Debugging
  - Analyze the model/program at compile-time
  - Explain inconsistencies and errors, trace error dependencies
  - Example: Underconstrained/overconstrained systems of equations
  - Example: errors in symbolic transformations of models
- Dynamic Debugging
  - Find sources of errors at run-time, for a particular execution
  - **Declarative dynamic debugging** – compare the execution with a specfication and semi- automatically find the location of the error
  - **Traditional dynamic debugging** – interactively step through the program, set breakpoints, display and modify data structures, trace, stack inspection
- Goal: Integrated Static and Dynamic Debugging

Dynamic Debugging

Large Modelica Algorithmic Code Models

Tool Architecture and Communication



Example Mapping Modelica Postions to C Code

Convert Modelica code to C source code by adding Modelica line number references.

HelloWorld.mo

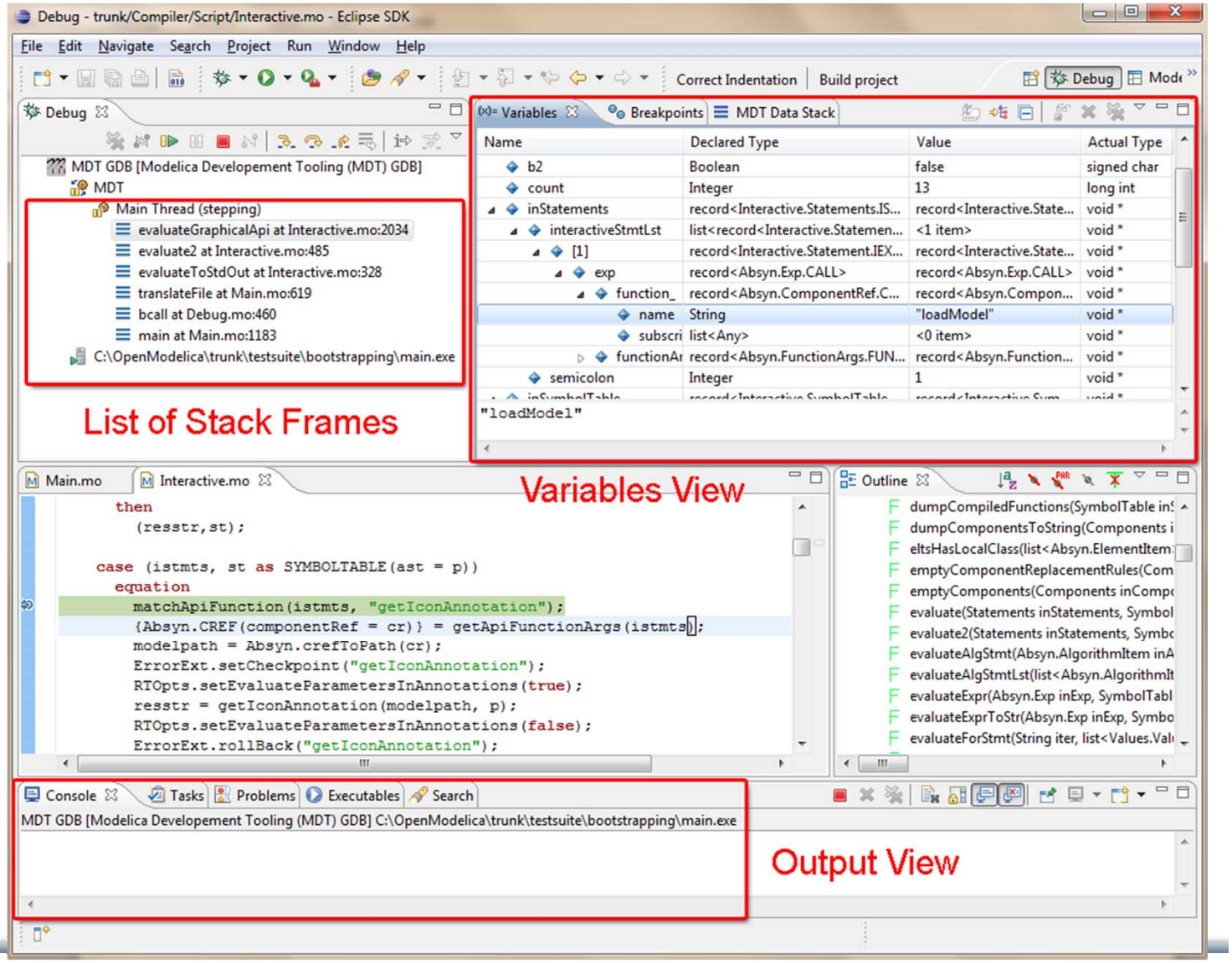
```
1 function HelloWorld
2   input Real x;
3   output Real y;
4 algorithm
5   y := sin(x);
6 end HelloWorld;
```

HelloWorld.conv.c

```
57 #line 29 "HelloWorld.c"
58   /* functionBodyRegularFunction: var inits */
59 #line 30 "HelloWorld.c"
60   /* functionBodyRegularFunction: body */
61 #line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
62   tmp2 = sin(_x);
63 #line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
64   _y = tmp2;
65 #line 35 "HelloWorld.c"
```

Debugger Integrated in Eclipse OpenModelica MDT Environment

- Eclipse plugin **MDT (Modelica Development Tooling**) is the integrated development environment
- Debugger is a debug plug-in within MDT



Static Debugging

Transformational Debugging of Equation-Based Models

Debugging Equation Systems

Modelica Compiler Backend

- Complex mathematical transformations
- Hidden to users
- Users want to access this information
- Not intuitive
  - No explicit control flow
  - Numerical solvers
  - Linear/Non-linear blocks
  - Optimization
  - Events

Tracing Symbolic Transformations

- Simple Idea
  - Store transformations as equation metadata
- Works best for operations on single equations
  - Alias Elimination (a = b)
  - Equation solving ($f_1(a,b) = f_2(a,b)$, solve for a)
- Multiple equations require special handling
  - Gaussian Elimination (linear systems, several equations)
  - ...

Tracing Overhead?

- OpenModelica compiler implementation is so fast that tracing is enabled by default
  - 1 extra comparison and/or cons operation per optimization
  - Not noticable during normal compilation
- No overhead unless you output the trace

Substitution Example, Storing the Trace

a = b

c = a + b

d = a - b

c = a + b (subst a=b) =>

c = b + b (simplify) =>

c = 2 * b

d = a - b (subst a=b) =>

d = b - b (simplify) =>

d = 0.0

- The alias relation a=b stored in variable a
- The equations are e.g. stored as (lhs,rhs,list<ops>)

Trace Example (1)

0 = y + der(x * time * z); **z = 1.0;**

(1) substitution:

y + der(x * (time * z))

=>

y + der(x * (time * 1.0))

(2) simplify:

y + der(x * (time * 1.0))

=>

y + der(x * time)

(3) expand derivative (symbolic diff):

y + der(x * time)

=>

y + (x + der(x) * time)

(4) solve:

0.0 = y + (x + der(x) * time)

=>

der(x) = ((-y) - x) / time