

OpenModelica Algorithmic Code Debugger for Modelica/MetaModelica

Adeel Asghar

Motivation

- Old Debugger
 - Slow Performance
 - High compilation time due to extra trace code
 - Slow execution due to tracing

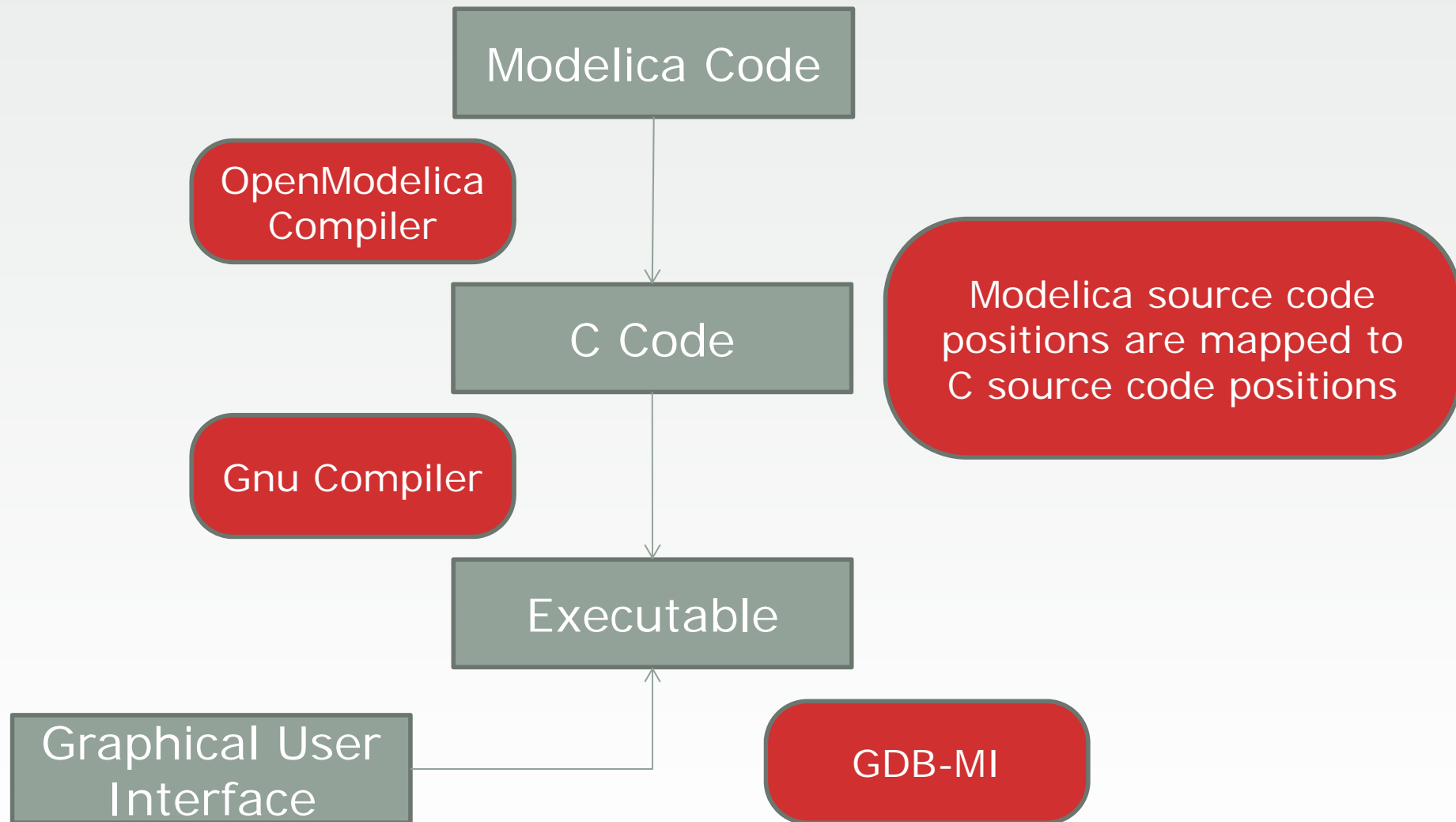
Outline

- Section I - Architecture
 - How does it work?
 - Code Generation
- Section II – MDT-Debug & GDB-MI
 - Breakpoints Support
 - Program Execution
 - Events
- Section III – MDT-Debug Screens
- Section IV – Conclusion & Future Work

Section I

Architecture

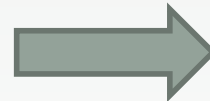
How does it work?



Generate C Code

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

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



```
 HelloWorld.conv.c X
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"
```

Section II

MDT-Debug & GDB-MI

MDT-Debug & GDB-MI

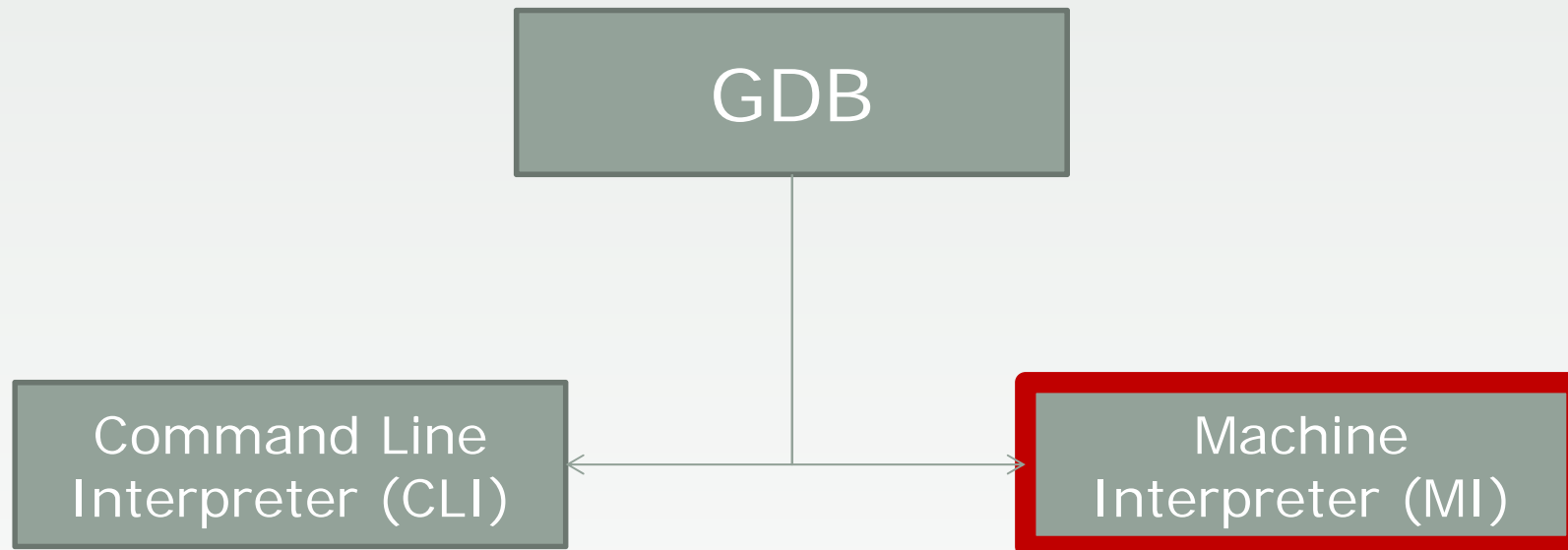
- MDT (Modelica Development Tooling) is the integrated development environment.
- The new debugger is implemented as a debug plugin within MDT.

The screenshot displays the Eclipse IDE with the MDT-Debug plugin. The interface is divided into several panes:

- List of Stack Frames:** A tree view on the left showing the execution stack. The current frame is 'Main Thread (stepping)' at line 2034 of 'Interactive.mo', with a red box highlighting it.
- Variables View:** A table on the right showing the current state of variables. A red box highlights this view.

Name	Declared Type	Value	Actual Type
b2	Boolean	false	signed char
count	Integer	13	long int
inStatements	record<Interactive.Statements.IS...	record<Interactive.State...	void *
interactiveStmtList	list<record<Interactive.Statement...	<1 item>	void *
[1]	record<Interactive.Statement.IEX...	record<Interactive.State...	void *
exp	record<Absyn.Exp.CALL>	record<Absyn.Exp.CALL>	void *
function_	record<Absyn.ComponentRef.C...	record<Absyn.Compon...	void *
name	String	"loadModel"	void *
subscri	list<Any>	<0 item>	void *
functionAr	record<Absyn.FunctionArgs.FUN...	record<Absyn.Function...	void *
semicolon	Integer	1	void *
inSymbolTable	record<Interactive.SymbolTable	record<Interactive.Sym...	void *
LoadModel			
- Code Editor:** Shows the source code for 'Interactive.mo'. The current line is highlighted, showing a match statement for 'getIconAnnotation'.
- Output View:** A console window at the bottom showing the output of the debugger, with a red box highlighting it.

MDT-Debug & GDB-MI (contd.)



- CLI interpreter is more user friendly.
- MI interpreter is more convenient for detailed control of the execution at the low level.

Breakpoints Support

- `-break-insert -f <filename:linenumber>`
 - `-f` create a pending breakpoint
- `-break-enable <breakpoint-number>`
- `-break-disable <breakpoint-number>`
- `-break-delete <breakpoint-number>`
- Other alternatives
 - `-break-insert -f function` (not supported)
 - `-break-insert -f filename:function` (not supported)

Breakpoints Support (contd.)

```

protected function getNthAlgorithmItemInClassParts
+ "function: getNthAlgorithmItemInClassParts
  input list<Absyn.ClassPart> inAbsynClassPartLst;
  input Integer inInteger;
  output String outString;
algorithm
  outString := matchcontinue (inAbsynClassPartLst,inInteger)
    local
      String str;
      list<Absyn.AlgorithmItem> algs;
      list<Absyn.ClassPart> xs;
      Integer n,c1,newn;
    case ((Absyn.ALGORITHMS(contents = algs) :: xs),n)
      equation
        str = getNthAlgorithmItemInAlgorithms(algs, n);
  contents = algs) :: xs),n) /* The rule above failed, subtract
  countInAlgorithmItems(algs);
  emInClassParts(xs, newn);
  amInClassParts(xs, n);
  
```

The screenshot shows a context menu over the code. The menu items are:

- Toggle Breakpoint
- Add Bookmark...
- Add Task...
- Show Quick Diff Ctrl+Shift+Q
- Show Annotation
- Show Line Numbers
- Folding ▶
- Preferences...

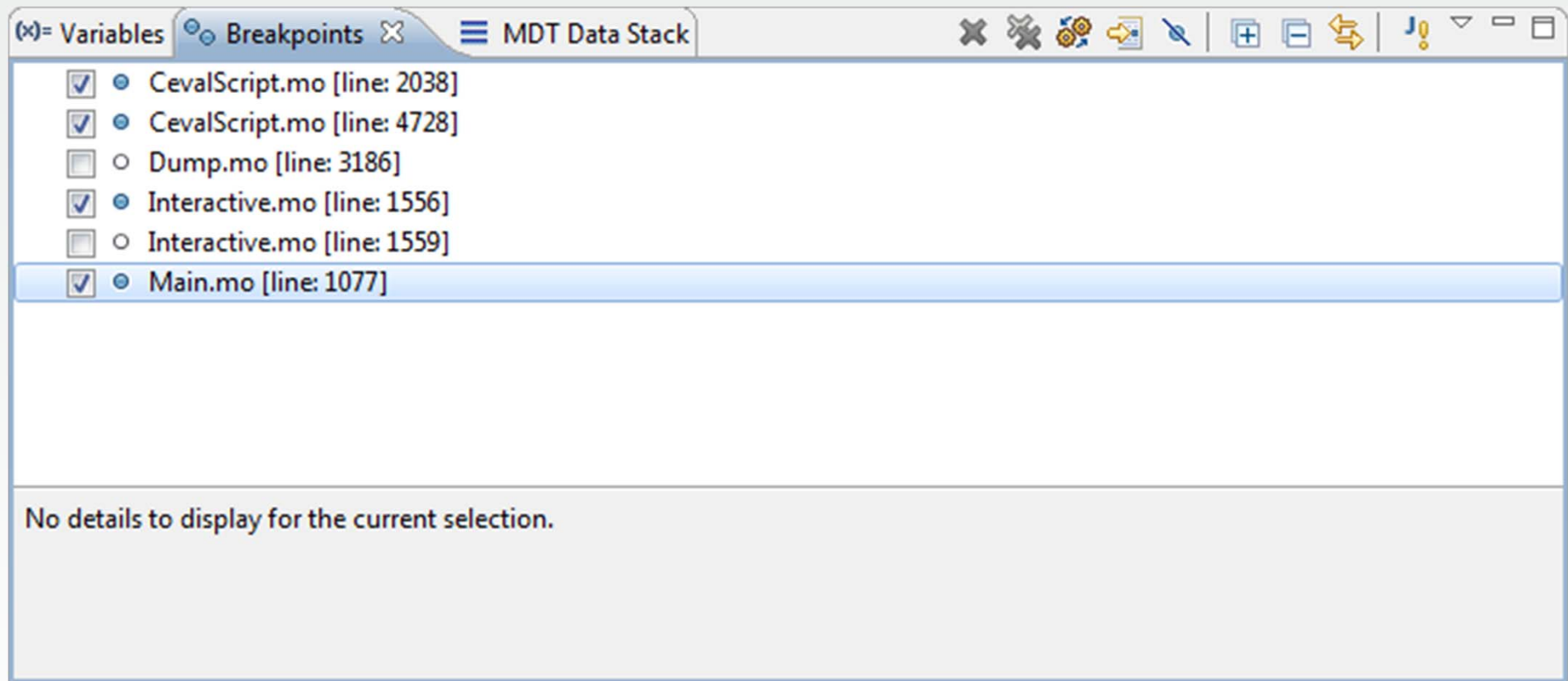
Section I

Section II

Section III

Section IV

Breakpoints Support (contd.)



The screenshot shows the 'Breakpoints' window in the OpenModelica debugger. The window has a title bar with tabs for 'Variables', 'Breakpoints', and 'MDT Data Stack'. The 'Breakpoints' tab is active, displaying a list of breakpoints. Each entry consists of a checkbox, a radio button, and a text label indicating the file and line number. The 'Main.mo [line: 1077]' entry is selected and highlighted in blue. Below the list, a message states 'No details to display for the current selection.' The toolbar on the right includes icons for deleting, adding, and managing breakpoints.

Checked	Selected	File and Line
<input checked="" type="checkbox"/>	<input checked="" type="radio"/>	CevalScript.mo [line: 2038]
<input checked="" type="checkbox"/>	<input checked="" type="radio"/>	CevalScript.mo [line: 4728]
<input type="checkbox"/>	<input type="radio"/>	Dump.mo [line: 3186]
<input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Interactive.mo [line: 1556]
<input type="checkbox"/>	<input type="radio"/>	Interactive.mo [line: 1559]
<input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Main.mo [line: 1077]

No details to display for the current selection.

Program Execution

- GDB command types
 - Synchronous commands
 - Asynchronous commands
- The program execution commands are asynchronous
 - `-exec-run` runs the program
 - `-exec-continue` continue the program
 - `-exec-next` performs the step over
 - `-exec-step` performs the step into
 - `-exec-finish` executes the function and return to the function call

Section I

Section II

Section III

Section IV

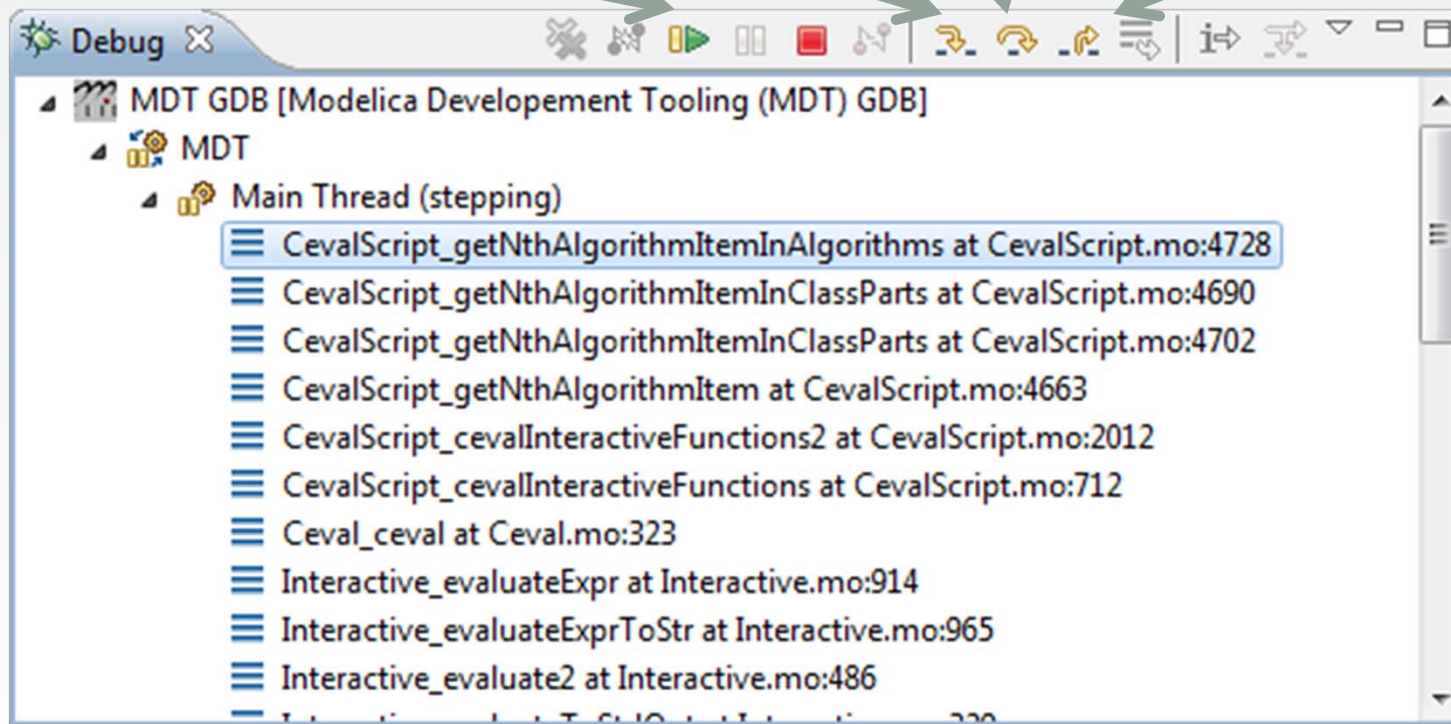
Program Execution (contd.)

-exec-continue

-exec-step

-exec-next

-exec-finish



Events

- GDB raise events
 - For asynchronous commands
 - For notifying program state
- For example
 - breakpoint-hit when a breakpoint is hit
 - end-stepping-range when a step into or step over operations are finished
 - function-finished when a step return operation is finished
 - signal-received e.g SIGSEGV

Section III

MDT-Debug Screens

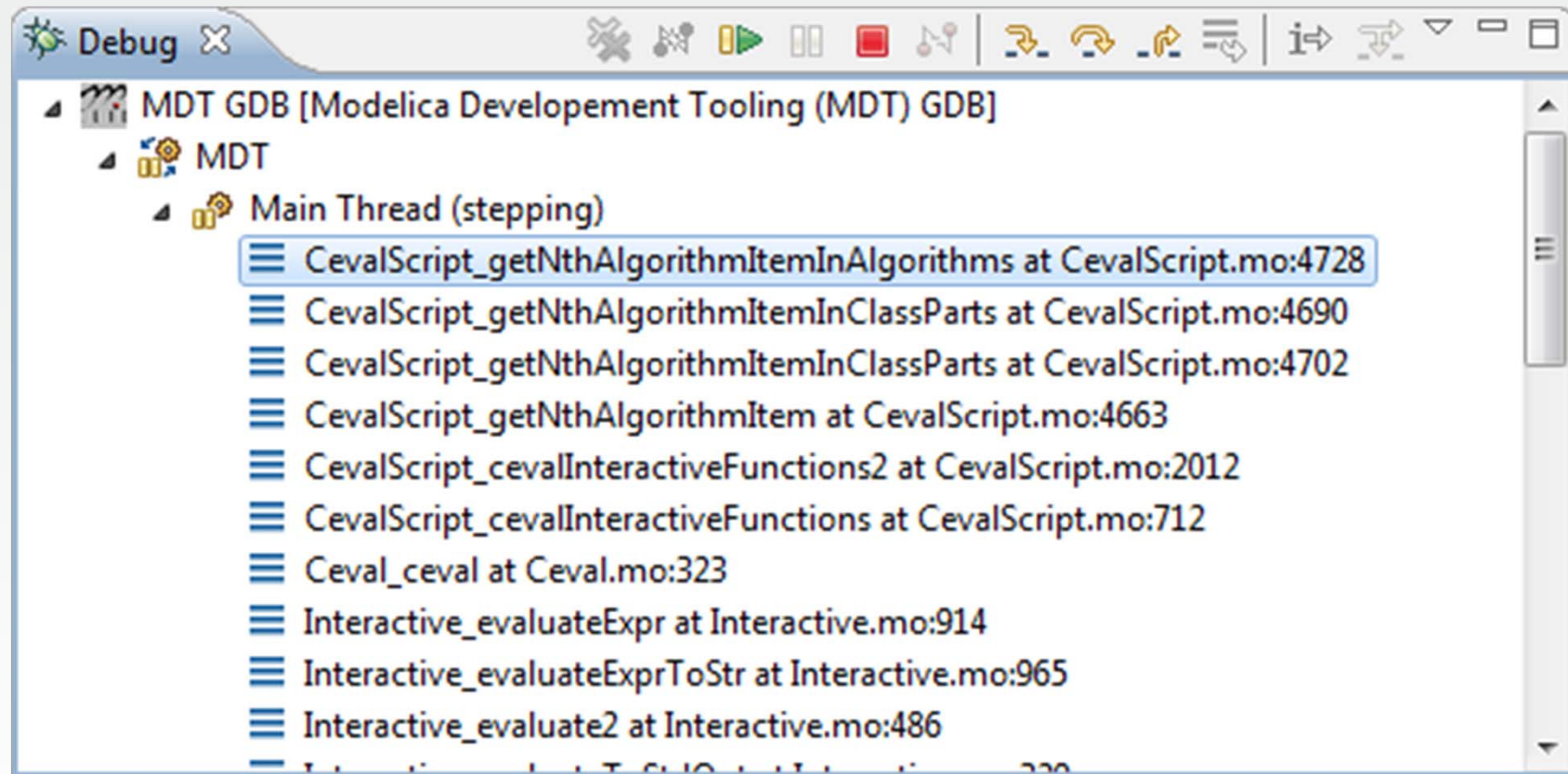
Section I

Section II

Section III

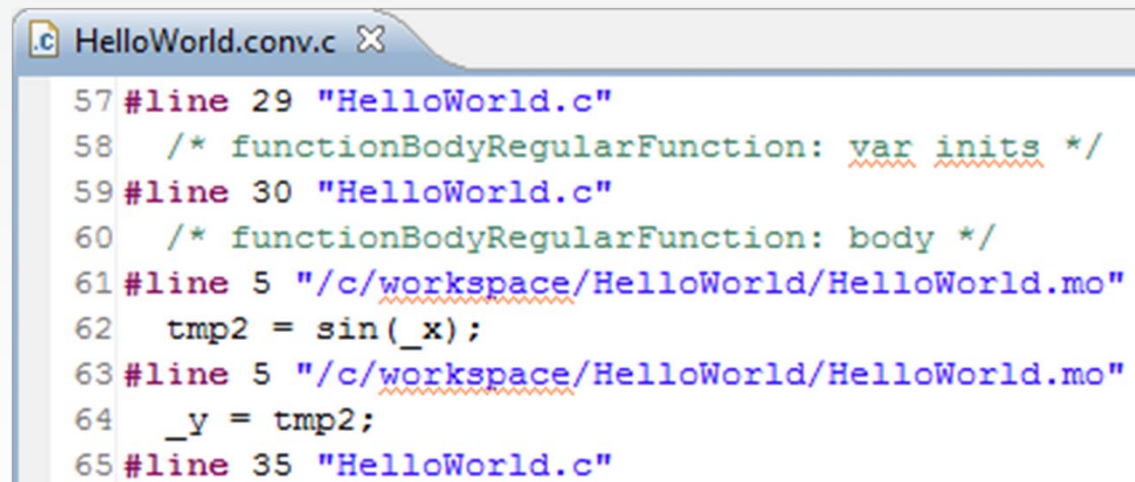
Section IV

Stack Frames



Stack Frames (contd.)

- Whenever a program execution is stopped e.g because of any event.
- A stack of frames is created.
- `-stack-list-frames` (returns a list of frames).
- Filter C files.



```
.c HelloWorld.conv.c X
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"
```

Variables View

Name	Declared Type	Value	Actual Type
▲ ◆ p	record<Absyn.Program.PROGRAM>	record<Absyn.Program.PROGRAM>	void *
▲ ◆ classes	list<record<Absyn.Class.CLASS>>	<1 item>	void *
▲ ◆ [1]	record<Absyn.Class.CLASS>	record<Absyn.Class.CLASS>	void *
◆ name	String	"m1"	void *
◆ restriction	record<Absyn.Restriction.R_MODEL>	record<Absyn.Restriction.R_MODEL>	void *
▲ ◆ body	record<Absyn.ClassDef.PARTS>	record<Absyn.ClassDef.PARTS>	void *
◆ typeVars	list<Any>	<0 item>	void *
▶ ◆ classParts	list<record<Absyn.ClassPart.PUBLIC>>	<1 item>	void *
◆ comment	Option<Any>	NONE()	void *
▶ ◆ info	record<Absyn.Info.INFO>	record<Absyn.Info.INFO>	void *
◆ within_	record<Absyn.Within.TOP>	record<Absyn.Within.TOP>	void *
▶ ◆ globalBuildTimes	record<Absyn.TimeStamp.TIMESTAMP>	record<Absyn.TimeStamp.TIMESTAMP>	void *
◆ b	Boolean	false	signed char
◆ dref1	Boolean	false	signed char
◆ protected_	Boolean	false	signed char
◆ dref2	Boolean	false	signed char
◆ flowPrefix	Boolean	false	signed char
◆ streamPrefix	Boolean	false	signed char
◆ b1	Boolean	false	signed char
◆ repl	Boolean	false	signed char

Section IV

Conclusion & Future Work

Conclusion and Future Work

- The debugger supports extended Modelica (MetaModelica) algorithmic code.
- Operates efficiently on large algorithmic code. (Tested on OpenModelica compiler with more than 100 000 lines of code).
- Only MetaModelica datatypes including primitive Modelica types Integer, Real, Boolean and String are supported.
- The debugger needs to be extended to support all Modelica datatypes (arrays, enumerations and records).

**Thank
You**

Mahalo

Kiitos

Tack

Toda

Grazie

Obrigado

Thanks

Takk

Gracias

Merci

