

# Domain-Level Debugging for Compiled DSLs with the GEMOC Studio

(Tool Demo)

Erwan Bousse   Tanja Mayerhofer   Manuel Wimmer

TU Wien (Austria)

September 17, 2017

1 Context

2 Domain-level feedback

3 Demo

4 Conclusion

# (Domain-level) interactive debugging

- Many existing approaches to define **Domain Specific Languages (DSLs)** with execution semantics.
- Enable the use of **interactive debugging** (ie. breakpoints, step into, etc.) to understand and investigate defects.
- Two main kinds of DSLs to consider:
  - interpreted DSLs (with operational semantics)
  - compiled DSLs (with translational semantics)

The screenshot displays the Gemoc IDE interface for an Arduino Blinker project. The top-left pane shows the 'Debug' view with a tree structure of the execution model, including 'Gemoc debug target', 'Model debugging', and 'Global context: Project'. The top-right pane shows the 'Variables' table:

Name	Value
level (Arduino Board.0 :DigitalPin)	1
level (Arduino Board.1 :DigitalPin)	1
level (Arduino Board.2 :DigitalPin)	0
value (newSketch.1 :IntegerVariable)	3

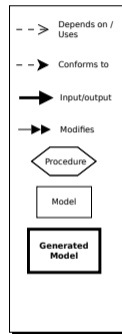
The middle-left pane shows a hardware diagram of an Arduino Board connected to three LEDs: RED LED, BLUE LED, and WHITE LED. The middle-right pane shows a block diagram of the sketch, featuring a 'Repeat 3' loop containing blocks for 'RED LED : (pin2)', 'WHITE LED : (pin4)', and 'Set = (set)'. The bottom pane shows the 'Multidimensional Timeline' with 'All execution states (23)' and a 'Timeline for dynamic information' for variables like 'level (Arduino Board.1 :DigitalPin)' and 'value (newSketch.1 :IntegerVariable)'.

# (Domain-level) interactive debugging

- Many existing approaches to define **Domain Specific Languages (DSLs)** with execution semantics.
- Enable the use of **interactive debugging** (ie. breakpoints, step into, etc.) to understand and investigate defects.
- Two main kinds of DSLs to consider:
  - interpreted DSLs (with operational semantics)
  - compiled DSLs (with translational semantics)

The screenshot displays the Gemoc IDE interface for debugging an Arduino sketch. The 'Debug' window shows the execution model tree, including 'Gemoc debug target', 'Model debugging', and 'Global context: Project'. The 'Variables' window shows the state of variables: 'level (Arduino Board.0 :DigitalPin)' at 1, 'level (Arduino Board.1 :DigitalPin)' at 1, 'level (Arduino Board.2 :DigitalPin)' at 0, and 'value (newSketch.1 :IntegerVariable)' at 3. The 'Hardware' window shows a physical Arduino board connected to three LEDs (RED, BLUE, WHITE). The 'Sketch' window shows a block diagram of the code, including a 'Repeat' loop with 'RED LED : (pin2)', 'WHITE LED : (pin0)', and 'Set += (int)' blocks. The 'Multidimensional Timeline' window shows the execution flow across different levels, with 'level (Arduino Board.1 :DigitalPin)' and 'level (Arduino Board.0 :DigitalPin)' highlighted.

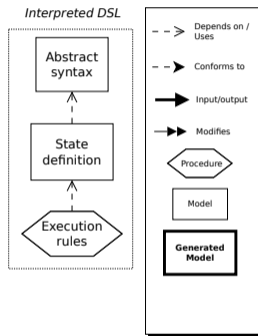
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

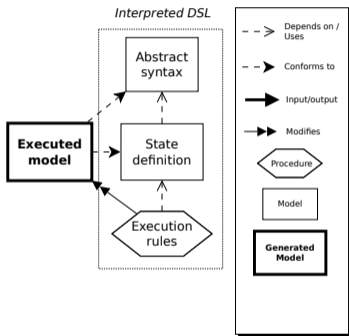
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

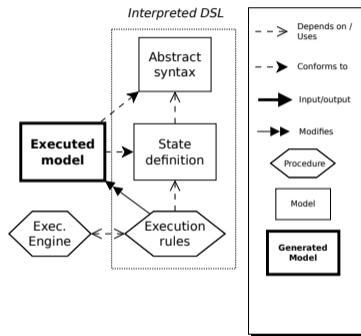
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

# Interpreted VS compiled DSLs

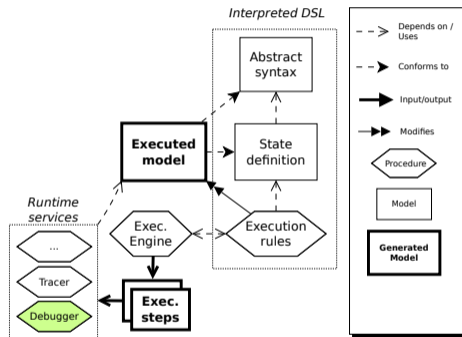


## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*



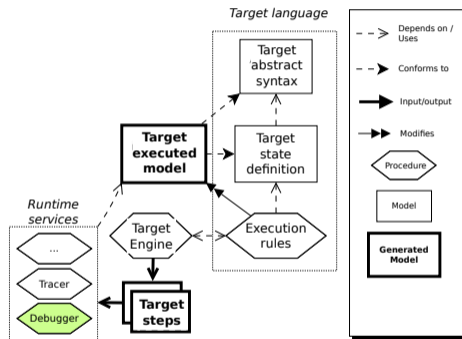
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

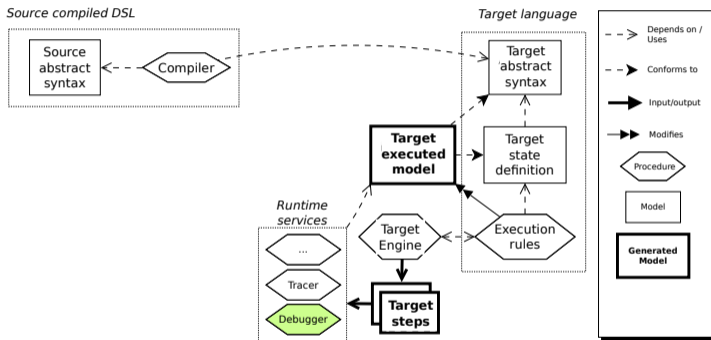
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

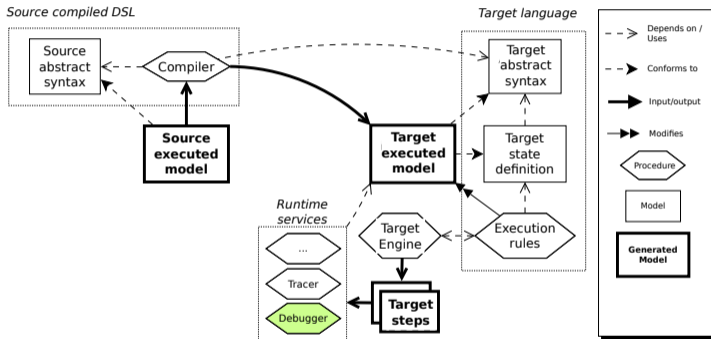
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

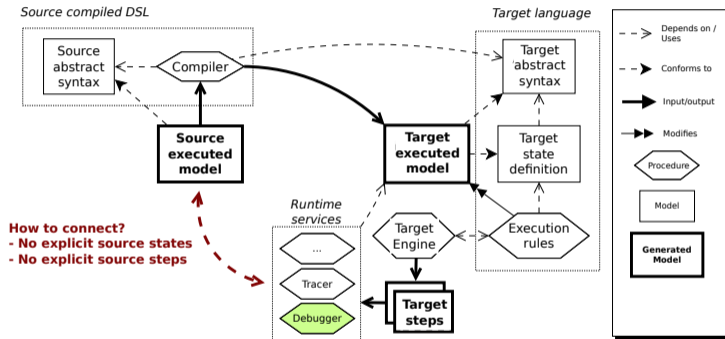
# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

# Interpreted VS compiled DSLs



## Runtime services

Provides services at runtime by observing occurring *steps* and the changing *state*

# Problem

How to provide domain-level interactive debugging when executing models conforming to compiled DLSs?

## Idea

At runtime, automatically translate on-the-fly *target* states and steps into *source* states and steps, which can then be observed by domain-level tools.

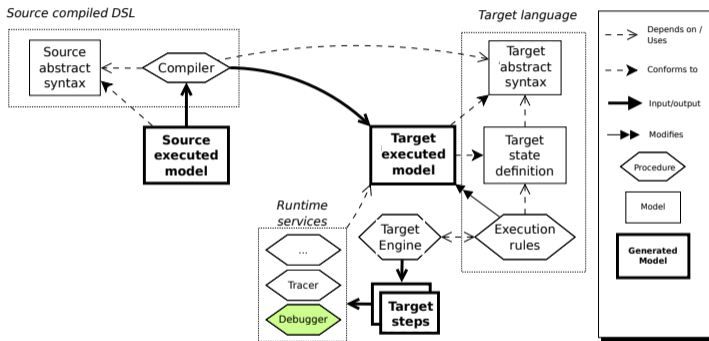
# Problem

How to provide domain-level interactive debugging when executing models conforming to compiled DLSs?

## Idea

At runtime, automatically translate on-the-fly *target* states and steps into *source* states and steps, which can then be observed by domain-level tools.

# Overview

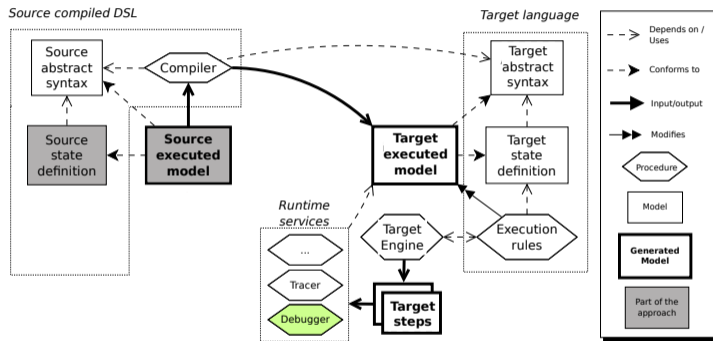


## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!



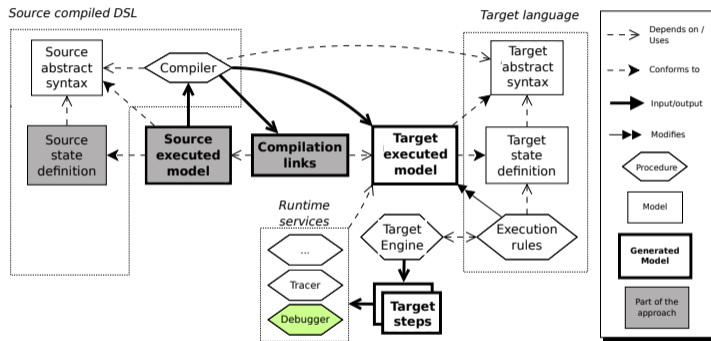
# Overview



## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!

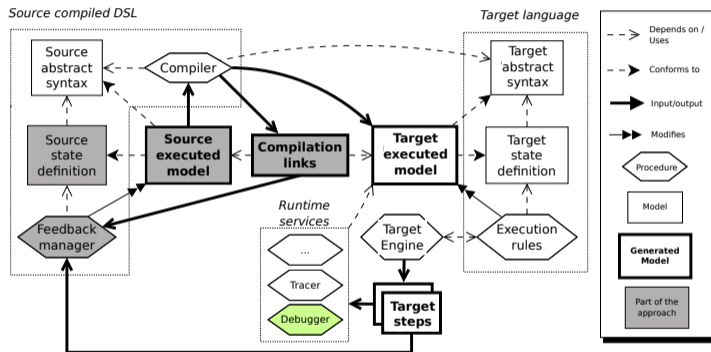
# Overview



## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!

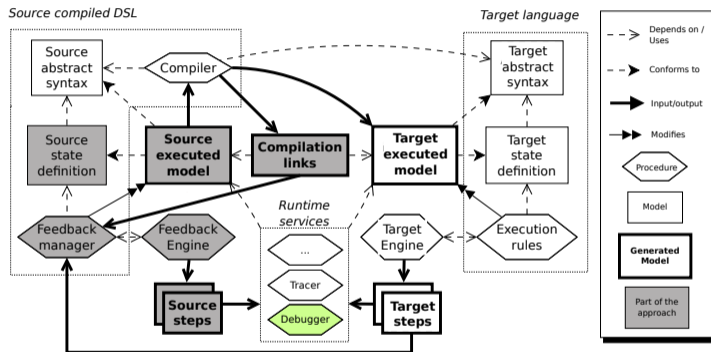
# Overview



## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!

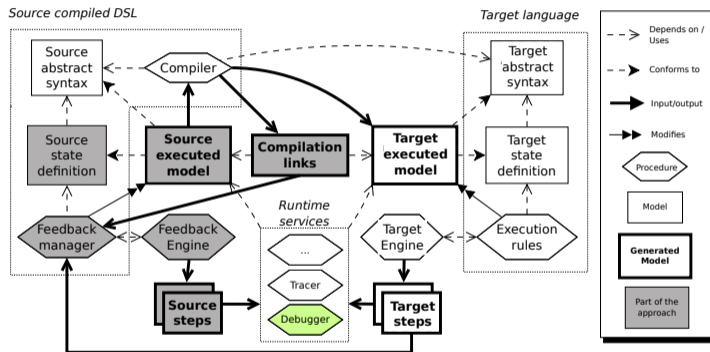
# Overview



## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!

# Overview



## Result

The same runtime services can be (re)used for both interpreted and compiled DSLs!

# Demo

Demo...

# Conclusion

- Providing interactive debugging for compiled DSLs is not trivial
- **Idea:** a *feedback manager* to translate target steps back to the source domain
- **Evaluation:** median execution slowdown of 1,8 times due to feedback
- **Future work:** manage non GEMOC-ready compilers (eg. *code generators*)

- Eclipse Research Consortium GEMOC: sustains the GEMOC studio as a research platform to experiment on the globalization of, possibly executable and heterogeneous, modeling languages
- Contributors are welcome!



<http://gemoc.org/>

<https://github.com/eclipse/gemoc-studio-modeldebugging>

# Conclusion

- Providing interactive debugging for compiled DSLs is not trivial
  - **Idea:** a *feedback manager* to translate target steps back to the source domain
  - **Evaluation:** median execution slowdown of 1,8 times due to feedback
  - **Future work:** manage non GEMOC-ready compilers (eg. *code generators*)
- 
- **Eclipse Research Consortium GEMOC:** sustains the GEMOC studio as a research platform to experiment on the globalization of, possibly executable and heterogeneous, modeling languages
  - Contributors are welcome!



<http://gemoc.org/>

<https://github.com/eclipse/gemoc-studio-modeldebugging>