

# MDebugger:

A model-based debugger for real-time and embedded systems

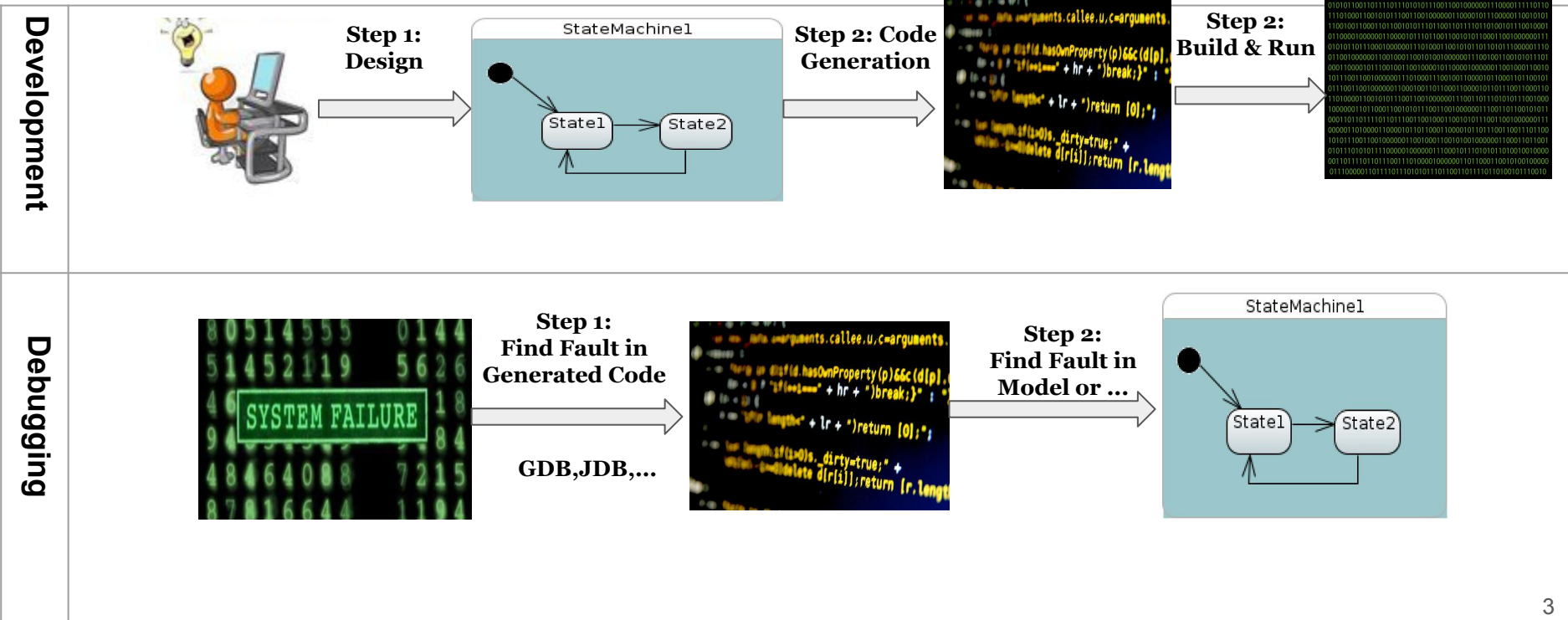
Mojtaba Bagherzadeh, Nicolas Hili, Juergen Dingel

# Outline

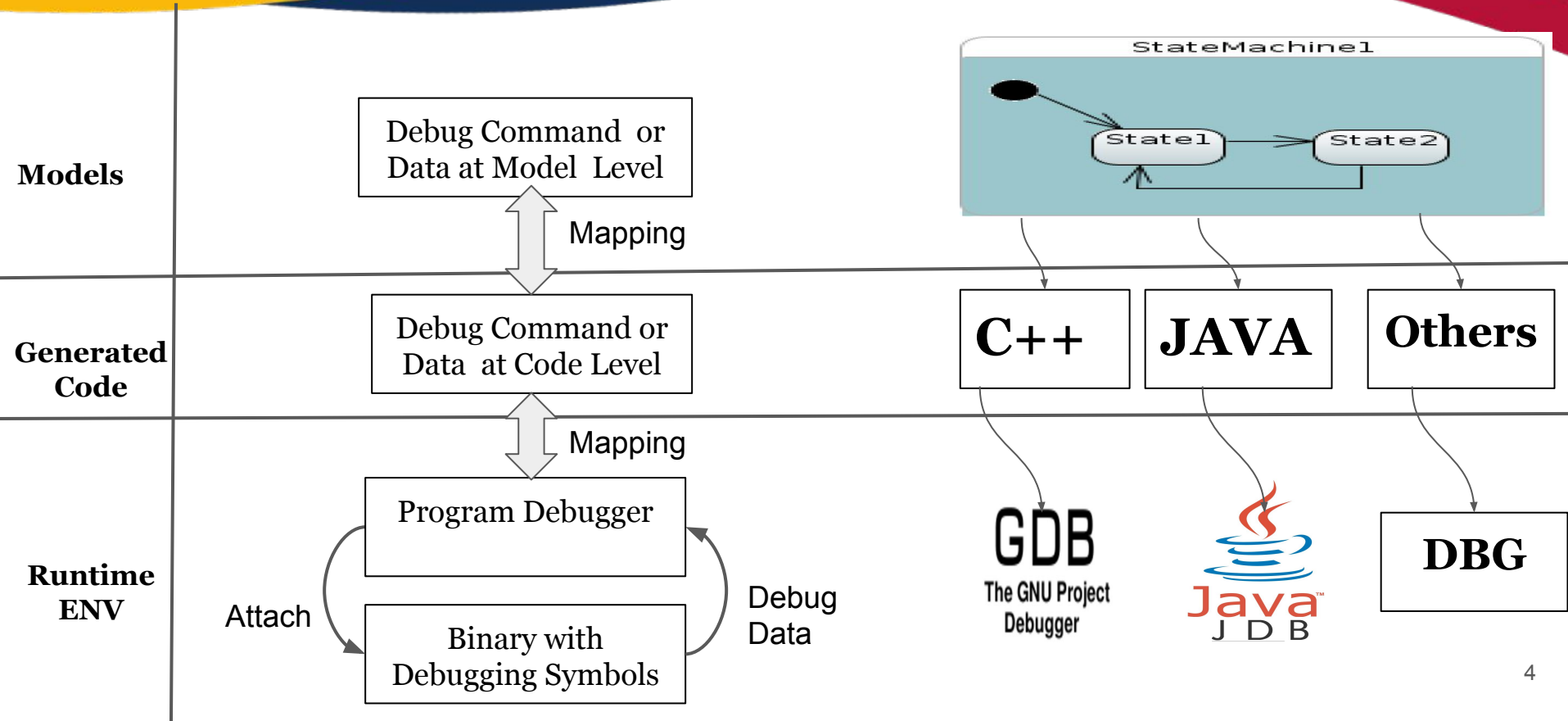


- Problem Statement
- Solution
- Concepts and Techniques
- MDebugger features
- Future Work
- Conclusion

# Problem Statement



# Problem Statement (Existing approaches)

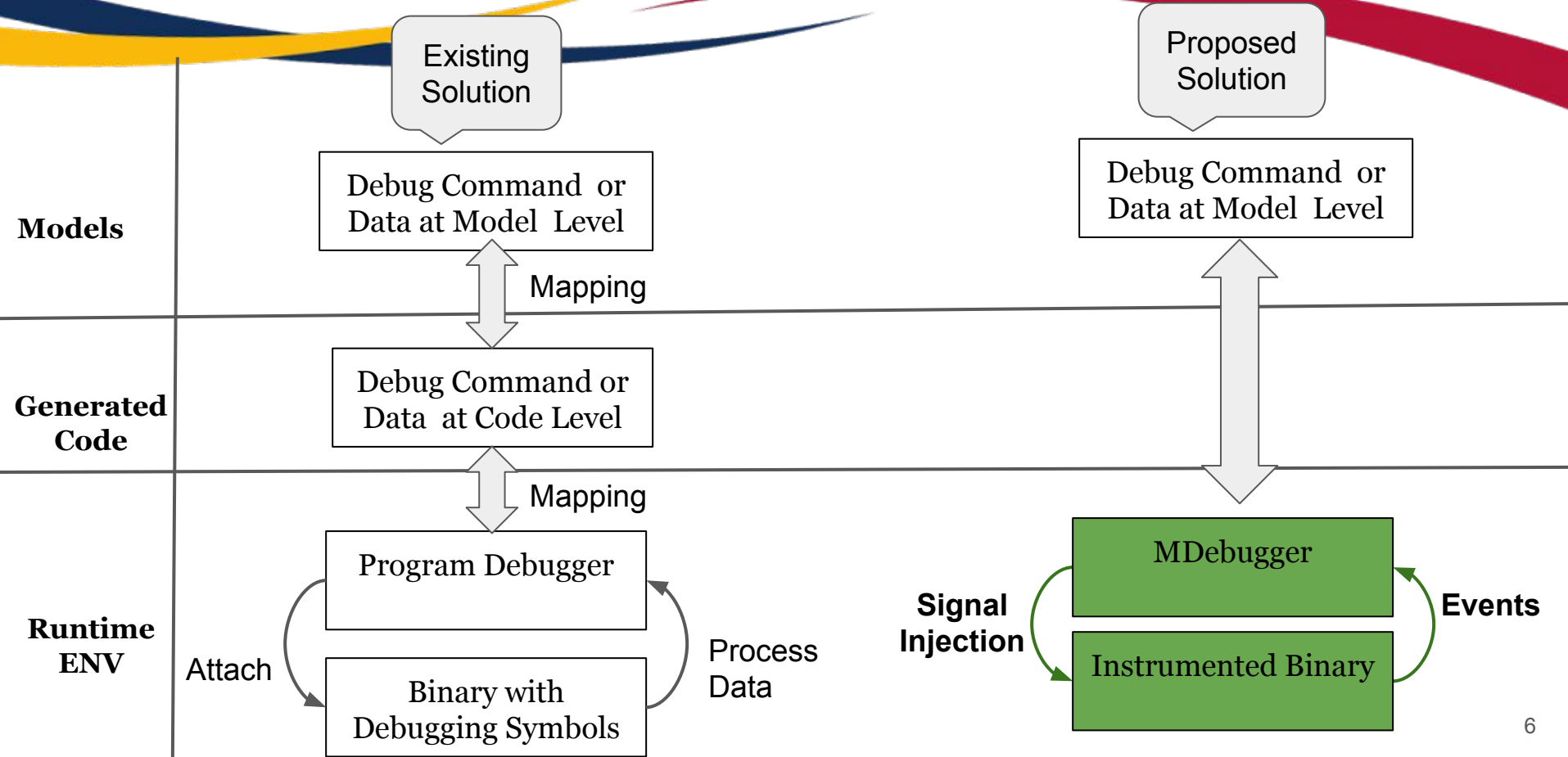


# Problem Statement (Summary)

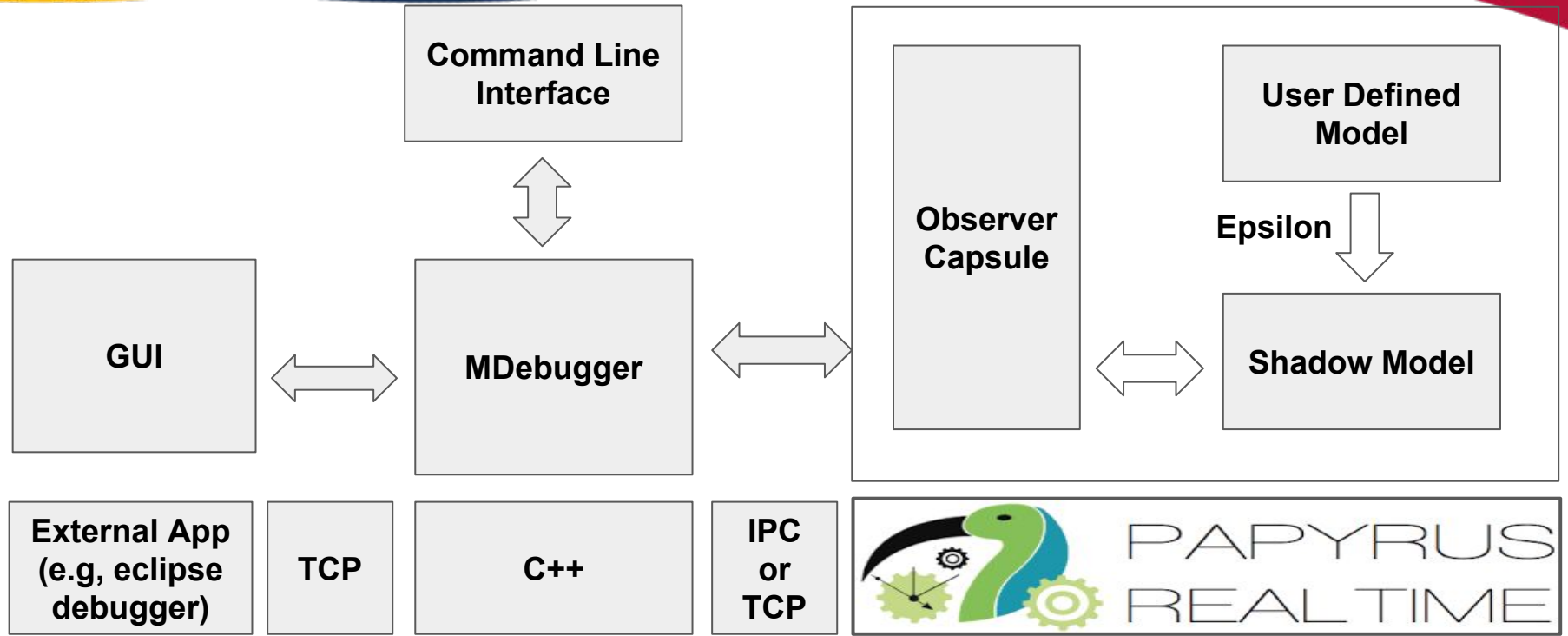


- Existing work has tried to solve the issue by creating wrappers around the program debuggers and mapping the features.
- Extra metadata generation are required for mapping.
- Using program debugging, causes dependency and compatibility and integration issues.
- The process is time consuming and challenging process.
- Debugging issue is one of the main barriers to adoption of MDD.

# Proposed Solution (Platform Independent Debugger)



# Overall Architecture



# MDebugger (Command Line Interface)

```
mdebugger#help
```

```
Available Options
```

```
"help|h"                (Show the commands and their options)
"breakpoint|b" -c capsuleName -t name -b -i traceNo(Set breakpoint at start of a transition)
"breakpoint|b" -c capsuleName -t name -e -i traceNo(Set breakpoint at end of a transition)
"breakpoint|b" -c capsuleName -t name -s -r -i traceNo(Remove breakpoint at end of a transition)
"breakpoint|b" -c capsuleName -t name -e -r -i traceNo(Remove breakpoint at end of a transition)
"next|n"          -c capsuleName -i traceNo      (Execute until next step)
"continue|c"     -c capsuleName -i traceNo      (Continue execution until next breakpoint)
"run|r"          -c capsuleName -i traceNo      (Run capsule without interrupt)
"modify|m"       -c capsuleName -n name -v value -i traceNo(Modify a attribute of capsule)
"view|v"         -c capsuleName -v -i traceNo   (View the capsule's attributes)
"view|v"         -c capsuleName -n count -e -i traceNo(View n last action of capsule's action chain)
"list|l"         -i traceNo                    (List running capsules and their current state)
"list|l"         -c capsuleName -i traceNo      (List capsule's configuration including breakpoints and etc)
"list|l"         -c capsuleName -b -i traceNo   (List exiting breakpoint)
"save|s"         -c capsuleName -i traceNo      (Save existing events)
"connect|con"    -h host -p port -i traceNo     (Connect to eclipse debugger)
```



# MDebugger Integration with PapyrusRT

The screenshot displays the Eclipse IDE interface for debugging a PapyrusRT project. The main window is titled "runtime-Epsilon - Debug - platform:/resource/CounterCaseStudy2/Counter.di - Eclipse Platform".

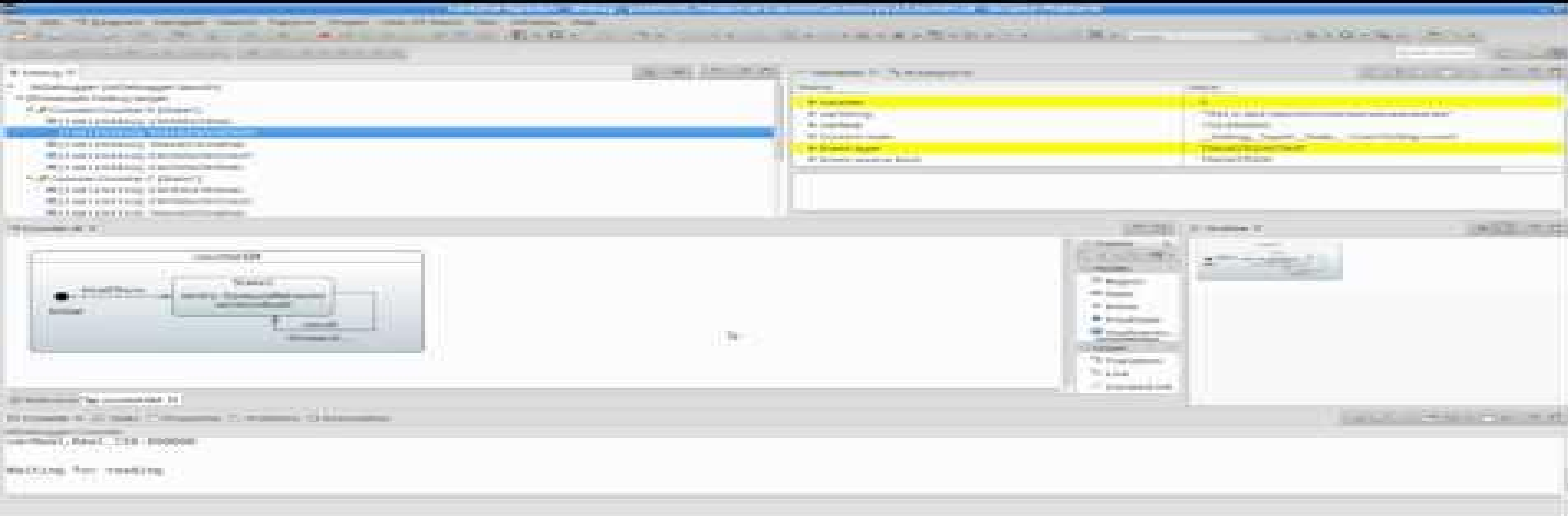
**Debug Console:** Shows the MDebugger launch process and the state of the debug target. The state machine is currently in the "Current state" of "\_\_\_Debug\_\_Super\_\_State\_\_\_:ConnToDbg::IntalTrans".

Name	Value
varInt	0
varBool	false
counter	0
varString	"My new value"
varReal	0
Current state	___Debug__Super__State___:ConnToDbg::IntalTrans

**State Machine Diagram:** The diagram shows a state machine named "counterSM" with an initial state "Initial" and a transition "IntalTrans" leading to "State1". State1 has the label "/entry OpaqueBehavior serializetself" and a self-loop labeled "count timeout/...".

**Console:** The MDebugger Console shows the following message: "sending message: '0058m -c Counter:Counter:0 -n varString -v 'My new value' -i 6' 6|ACK". The console is currently "Waiting for reading".

# MDebugger Integration with PapyrusRT



# Future Work



- Model-based instrumentation framework.
- Complete the current implementation
- Add debugging facility for action codes
- Automatic root cause analysis using program slicing on action codes
- Generate sequence diagram to present the runtime behaviour

# Conclusion



- We presented a new way of providing debugging at model-level.
- Our solution is implemented at model-level using modeling concept and is not dependent on program debugger or generated code.
- Basic features such as setting breakpoints, watch and change variables are implemented.
- Graphical and command line user interface are presented.

# Model Instrumentation



Provide instrumentation by extending the code generation:

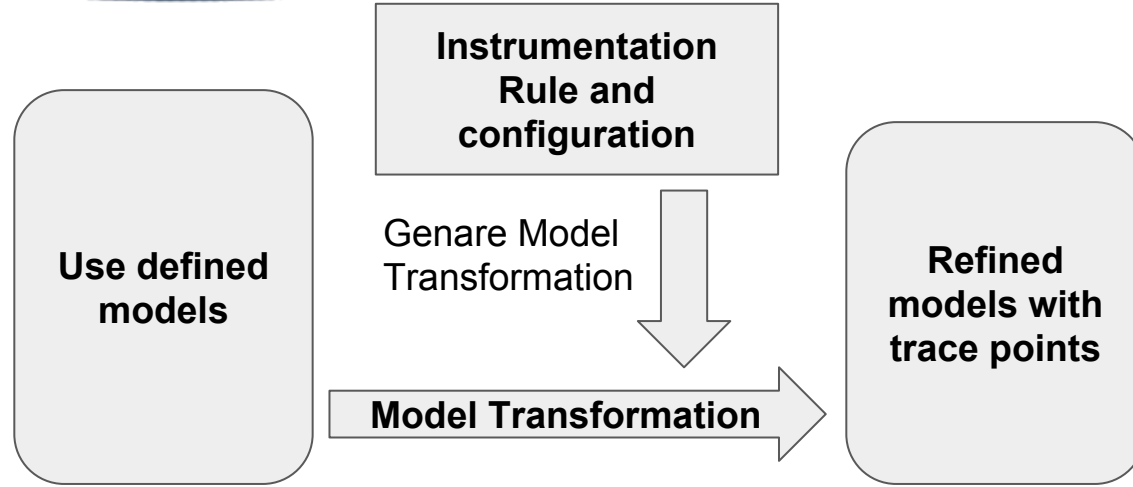
- Is a complex task.
- Causes maintenance and compatibility issues.
- Is platform and tool dependent.
- Is hard to validate and verify.
- Is not possible to capture all instrumentation requirements by pre-defined code generation.

# Vision

The top of the slide features three overlapping, wavy horizontal lines. From left to right, the colors are dark blue, yellow, and red. The lines are thick and have a soft, feathered edge, creating a modern, abstract header design.

Create a DSL to enable users to define customized instrumentation at model level.

# The Big Picture



Example of instrumentation rule:

- Trace all state changes.
- Trace all attribute changes that their type is Integer.
- Trace change of attribute  $x$  during entry of state 1.

Instrumentation DSL





# Progress till now:



- Integration between and Epsilon and PapyrusRT was done.
- Integration with LTTng and Observer as main tracing tools was done.



Thank You!