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**

**Development**

1. **Design**
   - Step 1: Design
   - Step 2: Code Generation

2. **Build & Run**
   - Step 2: Build & Run

**Debugging**

1. **Find Fault in Generated Code**
   - Step 1: Find Fault in Generated Code
   - GDB, JDB, ...

2. **Find Fault in Model or ...**
   - Step 2: Find Fault in Model or ...
Problem Statement (Existing approaches)

<table>
<thead>
<tr>
<th>Models</th>
<th>Generated Code</th>
<th>Runtime ENV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Command or Data at Model Level</td>
<td>Debug Command or Data at Code Level</td>
<td>Attach</td>
</tr>
<tr>
<td>Mapping</td>
<td>Mapping</td>
<td>Binary with Debugging Symbols</td>
</tr>
</tbody>
</table>

- Models
  - C++
  - JAVA
  - Others

- Generated Code
  - C++
  - JAVA
  - Others

- Runtime ENV
  - GDB
  - DBG
Problem Statement (Summary)

● Exiting 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)

Existing Solution

Debug Command or Data at Model Level

Mapping

Debug Command or Data at Code Level

Program Debugger

Generated Code

Runtime ENV

Attach

Binary with Debugging Symbols

Mapping

Debug Command or Data at Model Level

MDebugger

Instrumented Binary

Process Data

Signal Injection

Events
Overall Architecture

- **GUI**
- **MDebugger**
- **Observer Capsule**
- **User Defined Model**
- **Shadow Model**
- **Command Line Interface**
- **External App (e.g., eclipse debugger)**
- **TCP**
- **C++**
- **IPC or TCP**

- **Epsilon**
MDebugger (Command Line Interface)

Available Options

```
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 start 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
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.
Create a DSL to enable users to define customized instrumentation at model level.
The Big Picture

Use defined models

Instrumentation Rule and configuration
- Genare Model Transformation

Refined models with trace points

Model Transformation

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.
Architecture

Instrumentation DSL

[Logos for Epsilon and Papyrus Realtime]
Progress till now:

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