Interactive Debugging for Extensible Languages in Multi-Stage Transformation Environments

2nd International Workshop on Executable Modeling

Domenik Pavletic and Kim Haßlbauer

2016
Agenda

* Extensible Languages
* Bugs in Extensible Language Programs
* The MuLDer Framework
* mbeddr multi-level Debugger
* Conclusion
Extensible Languages

the context
Extensible Languages

enable syntactical and semantic extensions

Base Language

E0

E1

E2

E4

E5

E6

En

General Purpose Language

other Artifacts

(incrementally reduced)
Extensible Languages
mixed-language programs (mbeddr example)
Extensible Languages
high-level code gets incrementally reduced
Who can introduce Bugs that manifest at Runtime?
Who can introduce Bugs?

Language user - source level

Language user manually introduces source level bug
Who can introduce Bugs?

language engineer - intermediate/base level

Language engineer introduces bug through faulty transformation
Multi-Level Debuggers can analyze both bug categories
Multi-Level Debuggers

two types of users

Language User

Language Engineer

{ }

Source Level

n Intermediate Levels

Base Level

Target Level (text)

(later demonstrated)
The MuLDer Framework
(The Multi-Level Debugger Framework)
MuLDer Framework overview

- Based on mbeddr Platform + JetBrains MPS (customized version)
- For languages supporting „call stack / stepping“-based debug approach (e.g., imperative)
- Debugging support is built per language construct
- Implementation is restricted to MPS, approach is workbench independent
MuLDer Framework
initial source-level debugger framework

- No support for alternative transformations
- Hard to establish a mapping between both levels
- Changing in low-level generators require updates in debugger definitions
MuLDer Framework
incremental approach with MuLDer

- Multi-Level debugging support
- Support for alternative transformations
- Easy to establish a mapping between both levels
- Debugger definitions are independent from changing low-level generators
MuLDer Framework approach

1. Specify debug semantics of language constructs

2. Annotate transformations (M2M/M2T) to map between high-level and generated code
MuLDer Framework

**specify debug semantics**: implement interfaces
MuLDer Framework

**Specfify debug semantics**: provide specification
MuLDer Framework
annotate transformation rules (M2M)

```javascript
exported int32 main() {
    return test[tc];
} main (function)

testcase tc {
    boolean trueValue = true;
    assert(true && trueValue);
}

@StackTrace2StackTrace
main → main
exported int32 main() {
    return blockexpr {
        int32 failures = 0;
        @OutlineStackFrame: blockStatement2Testcase {
            boolean trueValue = true;
            if (!(true && trueValue)) {
                failures++;
            } if
        }
    yield failures;
};
} main (function)
```

```
[concept TestcaseRef]
[inheritors false]
[condition <always>]

---> content node:
<TF>@OutlineStackFrame: blockStatement2Testcase
{[COPY_SRCL$[int32 bla = 123; ]
}
```

[409x935]MuLDer Framework
annotate transformation rules (M2M)
MuLDer Framework
annotate transformation rules (M2T)

```java
@StackFrameProvider
text gen component for concept Function {
  (context, buffer, node)->void {
    ...
    append { };
    @IdentifierProvider
    append ${node.mangledName()} ;
    append {();
    ...
  }
}
```

→ Associates target level Functions with base level
mbeddr
multi-level Debugger
mbeddr multi-level debugger

different languages supported

• mbeddr C
• State machines
• Components
• Decision Tables
• ...
mbeddr multi-level debugger

source-level debugging

➡ Debugging source-level code (high level)
mbeddr multi-level debugger

base-level debugging

Debugging base-level code (C)
Conclusion
Conclusion

identified advantages

✓ Alternative transformations supported
✓ Building debuggers requires little effort due to the incremental approach
✓ Debugger implementation does not depend on low-level languages/generators
✓ Debuggers can be used by language users and language engineers
Conclusion

identified limitations

- Implementation restricted to MPS
- Tracing across all levels required (MPS modified)
- Debug performance decreases with each additional abstraction level
- Language must support the „stepping / call stack“ based debugging approach
return "any Questions?";

https://github.com/DomenikP/MuLDer