The LIME Interface Specification Language and Runtime Monitoring Tool (Tool Paper)

Kari Kähkönen, Jani Lampinen, Keijo Heljanko, Ilkka Niemelä

Department of Information and Computer Science
Helsinki University of Technology

June 26, 2009

Outline

- Introduction
- LIME Interface Specification Language
- LIME Interface Monitoring Tool
- Conclusions
- Demo

Introduction

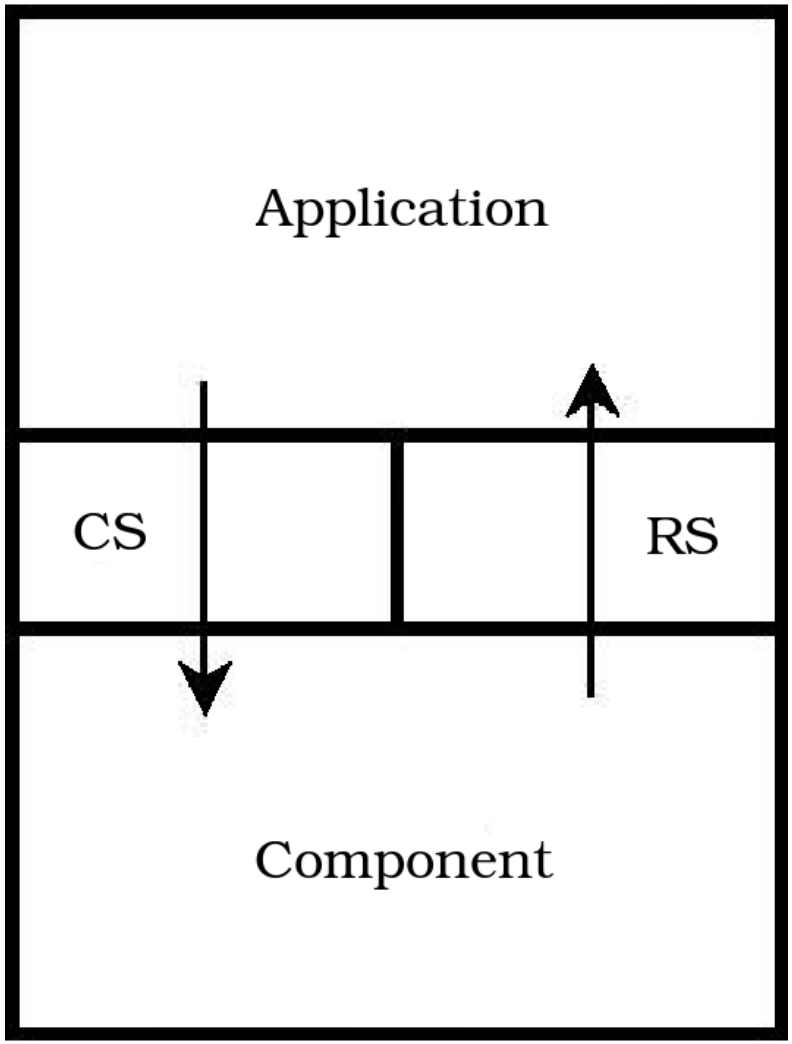
- The LIME specification language allows defining how components can interact through interfaces in Java
  - How the methods in Java interfaces can be called and how they should respond
- Specifications can be expressed as past time LTL formulas, (safety) future LTL formulas, regular expressions and NFAs
- The specification language is supported by LIME Interface Monitoring Tool
- Tools such as MOP, JML, Java PathExplorer and the tool of Stolz and Bodden have been sources of inspiration

Goals

- Extend the design by contract approach to behavioral aspects of interfaces
- Allow the user to target the critical parts of the system by allowing partial and incremental specifications
- Provide a structured and modular way of writing and testing specifications

LIME Interface Specification Language

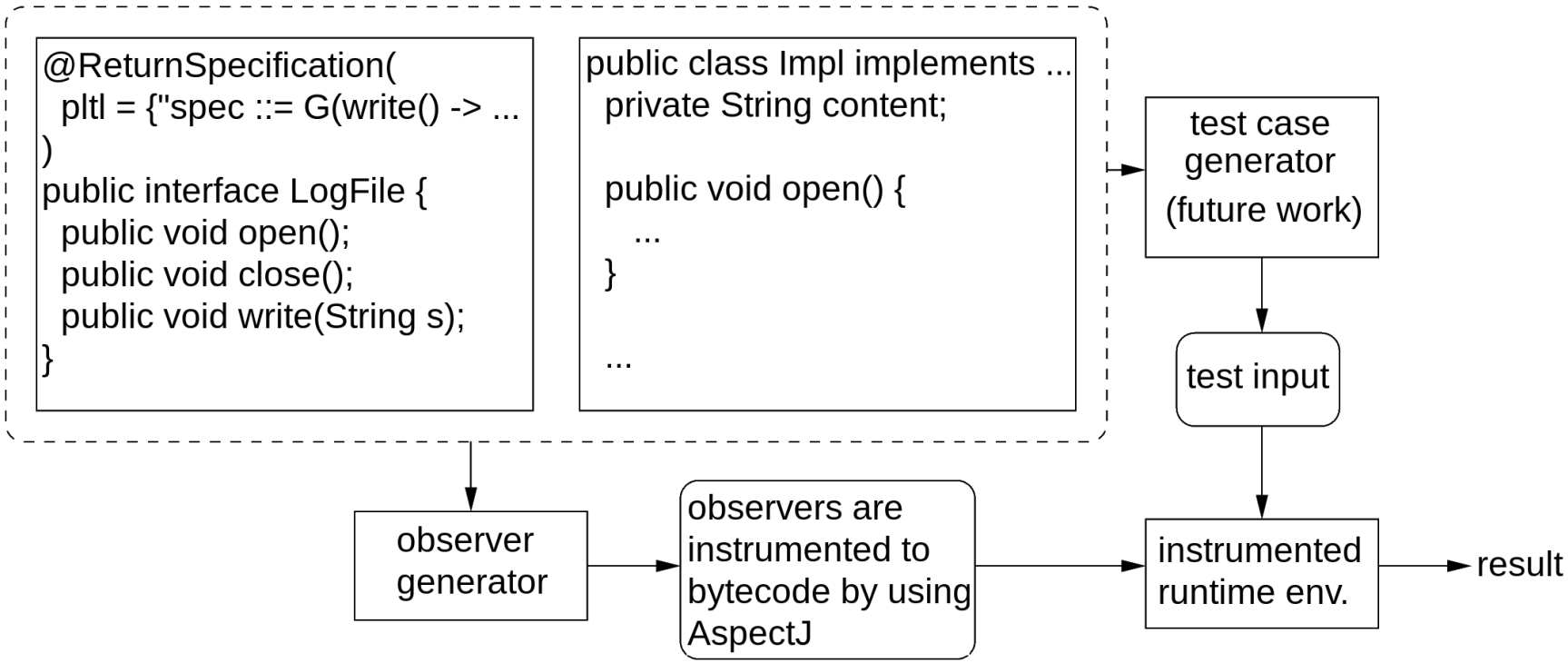
- LIME ISL divides a component interface to two parts
  - Call specifications (CS)
  - Return specifications (RS)
- The component that violates a specification can be identified



Example - LIME ISL

```
1  @CallSpecifications(
2   regexp = {"FileUsage ::= " +
3             "(open() ; (read() | write())* ; close())*"},
4   valuePropositions = {"validString ::= (#entry != null)"},
5   pltl = {"ProperData ::= G (write() -> validString)"}
6  )
7  @ReturnSpecifations(
8   valuePropositions = {
9   "okLength ::= #this.length() == " +
10                "#pre(#this.length() + #entry.length())"
11  }
12  pltl = { "ProperWrites ::= G (write() -> okLength)" }
13 )
14 public interface LogFile {
15   public void open();
16   public void close();
17   public String read();
18   public void write(String entry);
19   public long length();
20 }
```

LIME Interface Monitoring Tool

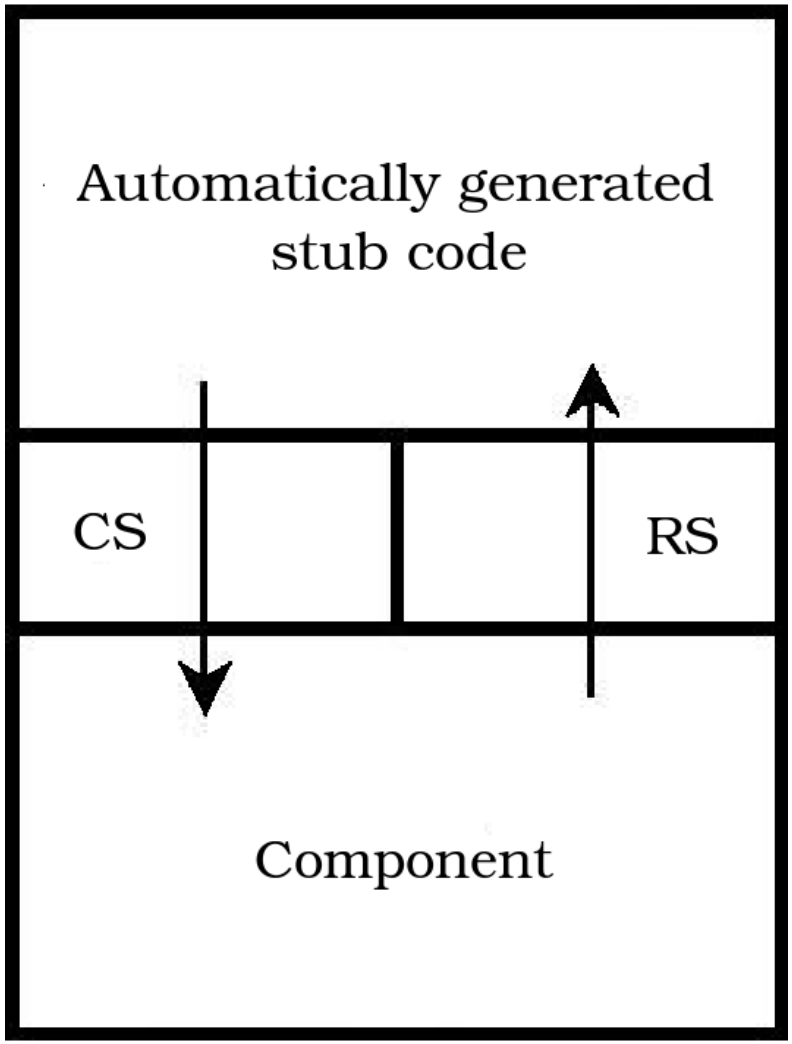


Implementation

- Runtime observers are created by translating the specifications into minimal deterministic finite state automata
- DFAs are translated into Java code that is instrumented with AspectJ to the monitored program
- As an optimization, past time LTL formulas are translated directly to Java code using the technique by Havelund and Roşu
- Past time subformulas are allowed inside future time formulas

Closing Partially Implemented Systems

- The monitoring tool can generate stub code that simulates the calling environment
- Nondeterministic environment where call specifications describe the allowed call sequences
- The method is intended to be used with a directed randomized testing tool (similar to, e.g., jCUTE and Pex)



Example - Log File

```
1  public class TestDriver {
2    public static void main( String[] args ) {
3      /* ... variable declarations ... */
4
5      ExceptionOverride.setCallException(obj,
6                          InconclusiveException.class);
7
8      while (testDepth++ < 5) {
9        int i = r.nextInt(5);
10
11       switch (i) {
12         case 0: obj.length(); break;
13         case 1: javalangString1 = RandString.getString(r);
14                 obj.write(javalangString1); break;
15         case 2: obj.read(); break;
16         case 3: obj.close(); break;
17         case 4: obj.open(); break;
18       }
19     }
20   }
21 }
```

Conclusions

- LIME ISL allows specifying component interactions
- Runtime monitoring tool supporting LIME ISL has been implemented

Future research directions:

- Extending LIME ISL to C programming language
- Implementing the test generator tool
- Adding support for multi-threaded programs
- Investigating the notion of interface composition

LIME Interface Monitoring Toolkit is available for download at `http://www.tcs.hut.fi/~ktkahkon/LIMT/`