



Universiteit
Leiden
The Netherlands

Combining Monitoring with Run-time Assertion Checking

Gouw, C.P.T. de

Citation

Gouw, C. P. T. de. (2013, December 18). *Combining Monitoring with Run-time Assertion Checking*. Retrieved from <https://hdl.handle.net/1887/22891>

Version: Corrected Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/22891>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/22891> holds various files of this Leiden University dissertation

Author: Gouw, Stijn de

Title: Combining monitoring with run-time assertion checking

Issue Date: 2013-12-18

We show here an example of the input to SAGA (a communication view), and the corresponding generated Java code which SAGA generates for the History class. Figure A.1 shows the communication view of the Stack, which will serve as the example input.

The next pages show the corresponding automatically generated Java output.

```
local view StackHistory grammar Stack.g
specifies StackInterface {
    call void push(int item) PUSH,
    return int pop() POP
}
```

Figure A.1: Input to the tool: communication view of the Stack

A. INPUT AND OUTPUT OF SAGA

```
import java.util.IdentityHashMap; // stores local
    histories and objToId
import org.antlr.runtime.*; // for use in
    StackHistory
import java.util.ArrayList; // for use in
    StackHistory
import java.util.Iterator; // for use in
    StackHistory
import java.util.HashSet; // for use in StackHistory
import java.util.HashMap; // for use in StackHistory

aspect StackHistoryAspect {
    private StackHistory h = new StackHistory();

    //
    ////////////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////////////////////// Event classes
    ////////////////////////////////////////////////////////////////////
    //
    ////////////////////////////////////////////////////////////////////

    public class call_push extends org.antlr.runtime
        .CommonToken {
        private static final long serialVersionUID = 3
            L;

        private final Object caller;
        public Object caller() {
            return this.caller;
        }
        }

        private final StackInterface callee;
        public StackInterface callee() {
            return this.callee;
        }
        }

        private final int item;
        public int item() {
            return this.item;
        }
        }

        public String toString() {
            return "o" + StackHistory.objToId.get(caller
                ) + ":o" + StackHistory.objToId.get(
                    callee) + ".push(" + item + ")";
        }
        }
```

```
public call_push(Object caller, StackInterface
    callee, int item) {
    super(-1);

    this.caller = caller;

    this.callee = callee;

    this.item = item;
}
}

public class return_pop extends org.antlr.
    runtime.CommonToken {
    private static final long serialVersionUID = 3
        L;

    private final Object caller;
    public Object caller() {
        return this.caller;
    }

    private final StackInterface callee;
    public StackInterface callee() {
        return this.callee;
    }

    private final int result;
    public int result() {
        return this.result;
    }

    public String toString() {
        return "o" + StackHistory.objToId.get(caller
            ) + ":o" + StackHistory.objToId.get(
                callee) + ".pop(" + result + ")";
    }

    public return_pop(Object caller,
        StackInterface callee, int result) {
        super(-1);

        this.caller = caller;
```

A. INPUT AND OUTPUT OF SAGA

```
        this.callee = callee;

        this.result = result;
    }
}

//
// //////////////////////////////////////
// ////////////////////////////////////// History class
// //////////////////////////////////////
//
// //////////////////////////////////////

public static class StackHistory implements
    TokenSource {
    public static IdentityHashMap<Object, Integer>
        objToId = new IdentityHashMap<Object,
        Integer>();
    public static HashMap<Integer, Object>
        idToObj = new HashMap<Integer, Object>();
    private HashSet<Integer> actors = new HashSet<
        Integer>();

    private ArrayList<CommonToken> _L = new
        ArrayList<CommonToken>();
    private Integer _currentToken;
    private StackParser.start_return _start; //
        Synthesized attributes of start non-
        terminal

    public StackHistory() {
        _L.add(new CommonToken(Token.EOF));
        _L.add(new CommonToken(Token.EOF));

        parse(); // the empty history
    }

    // Implemented for TokenSource interface
    public String getSourceName() {
        return null;
    }

    public void print() {

        System.err.println("=== □ERROR! □Local □history
```

```

        of view StackHistory (events: " +
        Integer.toString(_L.size()-2) + ") of
        StackInterface object violates protocol
        structure specified in Stack.g==="\n");

// Print actors of the sequence diagram
Iterator<Integer> it = actors.iterator();
while(it.hasNext()) {
    Integer objId = it.next();
    if(idToObj.get(objId) != null) {
        System.out.println("o" + objId + "
        :" + idToObj.get(objId).
        getClass().getName());
    } else {
        System.out.println("o" + objId + "
        :Object");
    }
}
// Print messages between actors
System.out.println("");
for(int i=0; i<_L.size()-2; i++) {
    System.out.println(_L.get(i).toString());
}
}

public CommonToken nextToken() {
    return _L.get(_currentToken++);
}

// Parse the history in antlr and set
// attribute values
private void parse() {
    _currentToken = 0;
    CommonTokenStream tokens = new
        CommonTokenStream(this);
    StackParser parser = new StackParser(tokens)
        ;

    try {
        _start = parser.start();

    } catch(RecognitionException r) {
        print();
        assert false; // Assertion Failure
    } catch(AssertionError r) {
        print();
        assert false; // Assertion Failure
    }
}

```

```
    }

    public EList<Object> stack() {
        return _start.stack;
    }

    public int bla() {
        return _start.bla;
    }

    public void update(call_push e) {
        e.setType(StackLexer.PUSH);
        _L.add(_L.size()-2, e);

        if(!objToId.containsKey(e.caller())) { //
            for printing
            objToId.put(e.caller(), objToId.size
                ());
            idToObj.put(idToObj.size(),e.caller()
                );
        }
        if(!objToId.containsKey(e.callee())) { //
            for printing
            objToId.put(e.callee(), objToId.size
                ());
            idToObj.put(idToObj.size(),e.callee()
                );
        }

        if(!objToId.containsKey(e.item())) { //
            for printing
            objToId.put(e.item(), objToId.size()
                );
            idToObj.put(idToObj.size(),e.item());
        }

        actors.add(objToId.get(e.caller()));
        actors.add(objToId.get(e.callee()));

        parse();
    }

    public void update(return_pop e) {
        e.setType(StackLexer.POP);
        _L.add(_L.size()-2, e);
    }
}
```



```

        if(!objToId.containsKey(e.caller())) { //
            for printing
            objToId.put(e.caller(), objToId.size
                ());
            idToObj.put(idToObj.size(),e.caller()
                );
        }
        if(!objToId.containsKey(e.callee())) { //
            for printing
            objToId.put(e.callee(), objToId.size
                ());
            idToObj.put(idToObj.size(),e.callee()
                );
        }

        if(!objToId.containsKey(e.result())) { //
            for printing
            objToId.put(e.result(), objToId.size
                ());
            idToObj.put(idToObj.size(),e.result()
                );
        }

        actors.add(objToId.get(e.caller()));
        actors.add(objToId.get(e.callee()));

        parse();
    }
}

```

```

//
// //////////////////////////////////////
// ////////////////////////////////////// Aspects
// //////////////////////////////////////
// //////////////////////////////////////

```

```

/* call void push(int item) */
before(Object clr, StackInterface cle, int item)
:
(call( void *.push(int)) && this(clr) &&
target(cle) && args(item)

```

A. INPUT AND OUTPUT OF SAGA

```
        && if(StackHistoryAspect.class.
            desiredAssertionStatus() )) {
            cle.h.update(new call_push(clr, cle, item));
        }

before(StackInterface cle, int item): // from
    static method
    (call( void *.push(int)) && !this(Object) &&
        target(cle) && args(item)
        && if(StackHistoryAspect.class.
            desiredAssertionStatus() )) {
            cle.h.update(new call_push(null, cle, item))
            ;
    }

/* return int pop() */
after(Object clr, StackInterface cle) returning(
    int ret):
    (call( int *.pop()) && this(clr) && target(cle)
        ) && args()
        && if(StackHistoryAspect.class.
            desiredAssertionStatus() )) {
            cle.h.update(new return_pop(clr, cle, ret));
    }

after(StackInterface cle) returning(int ret): //
    from static method
    (call( int *.pop()) && !this(Object) && target
        (cle) && args()
        && if(StackHistoryAspect.class.
            desiredAssertionStatus() )) {
            cle.h.update(new return_pop(null, cle, ret))
            ;
    }

}
```