Leibniz Universität Hannover

SOFTWARE ENGINEERING

Miniproject 2
<MBSE>

Marc Mogalle
Christian Reinbold
Daniel Schulz

28.06.2016

What is this about?

- Code generation with Xtext & Xtend
- A runtime implementation into the ecore model and an interpreter
- Henshin approach

1) Code Generation

- Start a new eclipse with our Xtext addon
- Create a new statemachine model
- A java model will automatically be compiled into the src-gen folder

```
Network Cafe{
    stateMachine{
        StateMachine Guest {
            initialState waiting
            state {
                State waiting,
                State drinking_coffee
            }
            transition {
                Transition {
                    sendReceive send
                    source waiting
                    target waiting
                    channel orderCoffee
                },
                Transition {
                    sendReceive send
                    source drinking_coffee
                    target waiting
                    channel payCoffee
                },
                Transition {
                    sendReceive receive
                    source waiting
                    target drinking_coffee
                    channel deliverCoffee
                }
            }
        },
        StateMachine Waiter {
            initialState waiting
```

1) Code Generator in Xtend

```
def toJavaCode(Network smn) '''
    import java.util.concurrent.ThreadLocalRandom;
    import java.io.BufferedReader;
    import java.io.IOException;
    import java.io.InputStreamReader;
    import java.util.ArrayList;
    import stateMachines.*;
    import misc.*;

    public class «smn.name» {


        private static ArrayList<StateMachine> stateMachines = new ArrayList<StateMachine>();
        private static ArrayList<Channel> channels = new ArrayList<Channel>();
        private static ArrayList<Pair> transitions = new ArrayList<Pair>();
        private static int steps = 20;

        public static void main(String[] args) {
            «FOR c : smn.channel»
                channels.add(new Channel("«c.name.toString»",«c.synchronous as Boolean»));
            «ENDFOR»
            «FOR sm : smn.stateMachine»
                stateMachines.add(new «sm.name»("«sm.name»" ,channels));
            «ENDFOR»
            for(StateMachine sm : stateMachines) {
                for(Transition t : sm.getTransitions()) {
                    transitions.add(new Pair(sm, t));
                }
            }
```

1) Generated Java Code

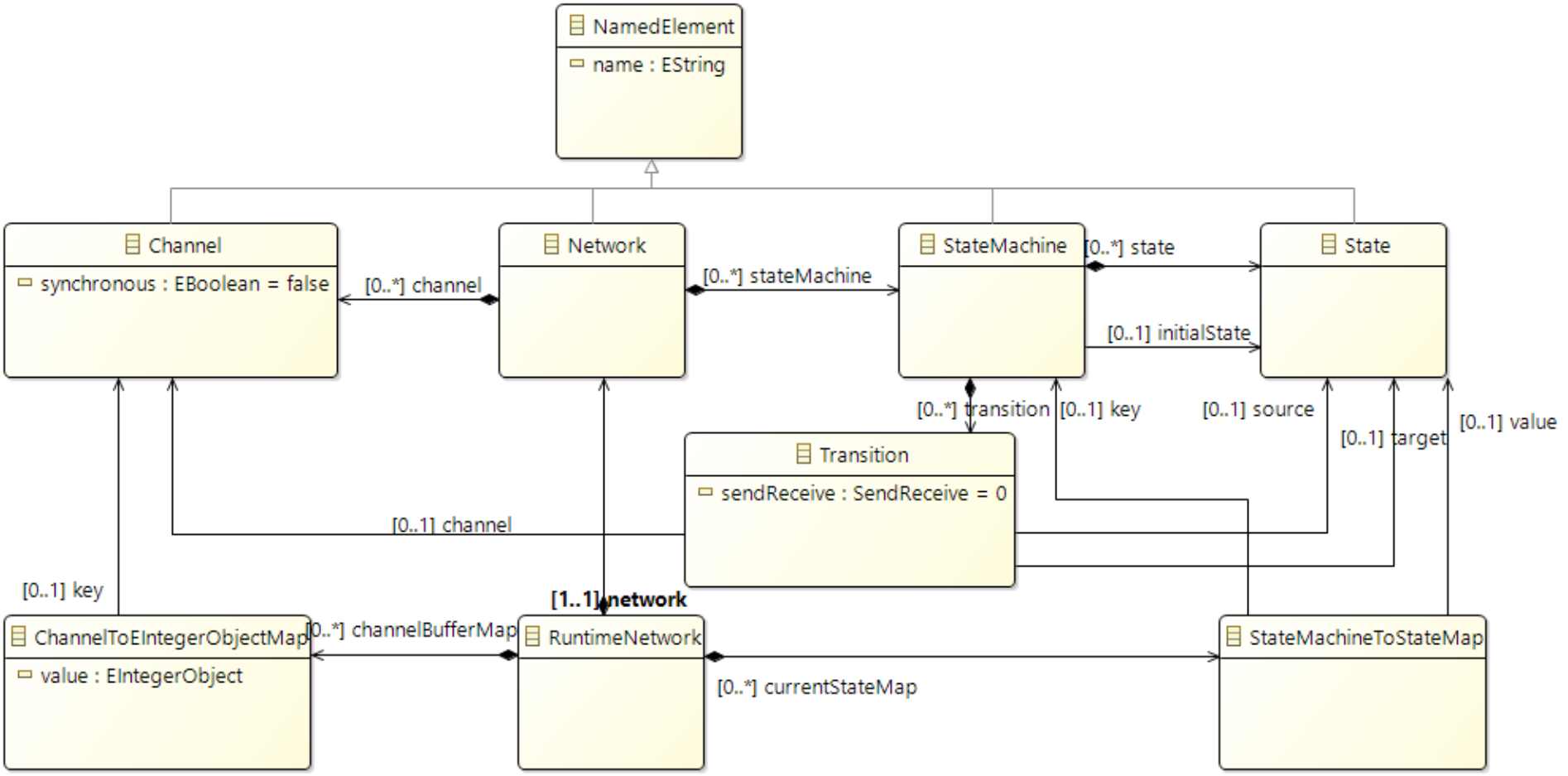
```
private static ArrayList<StateMachine> stateMachines = new ArrayList<StateMachine>();
private static ArrayList<Channel> channels = new ArrayList<Channel>();
private static ArrayList<Pair> transitions = new ArrayList<Pair>();
private static int steps = 20;

public static void main(String[] args) {
    channels.add(new Channel("orderCoffee",true));
    channels.add(new Channel("deliverCoffee",false));
    channels.add(new Channel("payCoffee",false));
    stateMachines.add(new Guest("Guest" ,channels));
    stateMachines.add(new Waiter("Waiter" ,channels));
    for(StateMachine sm : stateMachines) {
        for(Transition t : sm.getTransitions()) {
            transitions.add(new Pair(sm, t));
        }
    }
    new Cafe().run();
    for(int i = 0; i<steps;i++) {
        System.out.println("--------- Step " + (i+1) + " ----------");
        ArrayList<Pair> forFiring = new ArrayList<Pair>();
        for(StateMachine sm : stateMachines) {
            for(Transition t : sm.getEnabledTransitions()) {
                forFiring.add(new Pair(sm,t));
            }
        }
```

1) Output

--------- Step 1 ----------
Firing in Statemachine lock | Current State: locked | Next State: unlocked | with Channel turnLock
Firing in Statemachine door | Current State: locked | Next State: closed | with Channel turnLock
--------- Step 2 ----------
Firing in Statemachine door | Current State: closed | Next State: open | with Channel moveDoor
--------- Step 3 ----------
……..
--------- Step 7 ----------
Firing in Statemachine lock | Current State: locked | Next State: broken | with Channel breakLock
--------- Step 8 ----------
Deadlock in Step 8

2) Extended eCore Model



2) Interpreter Output

Interpreter created for network "Cafe".

Guest:waiting->waiting(send) on orderCoffee (new buffer: 1).

Waiter:waiting->preparing_coffee(receive) on orderCoffee (new buffer: 0).

Guest:waiting->waiting(send) on orderCoffee (new buffer: 1).

Waiter:preparing_coffee->waiting_for_payment(send) on deliverCoffee (new buffer: 1).

Guest:waiting->drinking_coffee(receive) on deliverCoffee (new buffer: 0).

Guest:drinking_coffee->waiting(send) on payCoffee (new buffer: 1).

Waiter:waiting_for_payment->waiting(receive) on payCoffee (new buffer: 0).

Runtime: about 550ms (for 30 steps)

Comparison Interpreter VS Generated Code

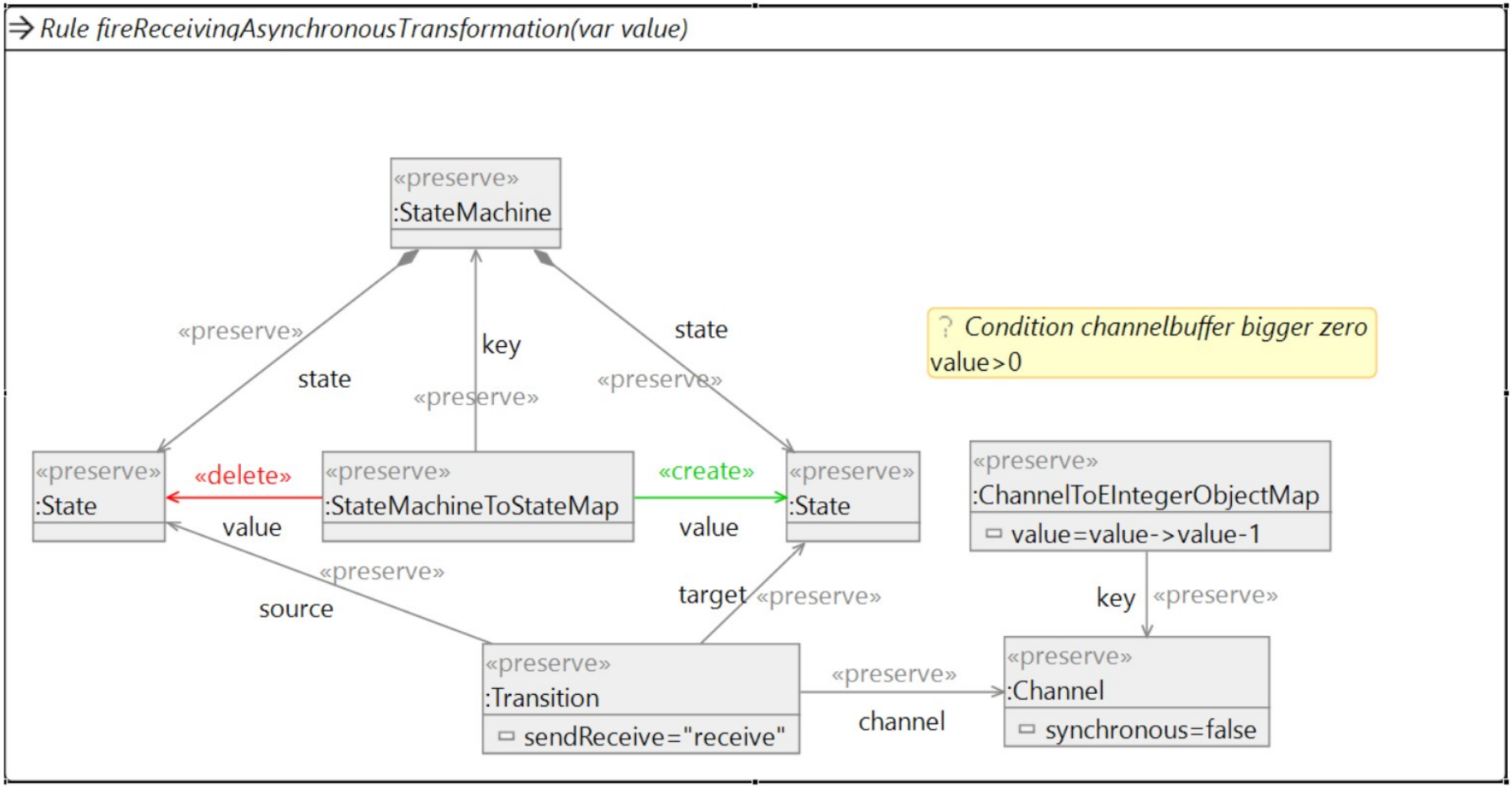
- Run of the Cafe model
- 30 steps were taken in each model

- Runtime in interpreter: about 550 ms

- Runtime generated code: about 10 ms

- Time was measured with Java system time.

3) Henshin


Rule fireReceivingAsynchronousTransformation(var value)
«preserve»
:StateMachine
«preserve»
state
key
state
«preserve»
«preserve»
«preserve»
«preserve»
:State
«delete»
«preserve»
:StateMachineToStateMap
«create»
«preserve»
:State
value
value
«preserve»
source
target
«preserve»
«preserve»
:Transition
sendReceive="receive"
«preserve»
channel
«preserve»
:Channel
synchronous=false
Condition channelbuffer bigger zero
value>0
«preserve»
:ChannelToEIntegerObjectMap
value=value->value-1
key
«preserve»

3) Statespace