Model-Based Software Engineering

Mini-Project II

Eric Roslin Wete Poaka
Naira Audi

Summary

1. Code generator
2. Interpreter
3. Henshin
4. Final remarks

Code generator

- Transforms model in java code;
- Log information:
  - Current state of each state machine;
  - Channel buffer counter;
  - Last executed transition;
- Step strategy: Channel buffer as close to zero as possible.

Code generator

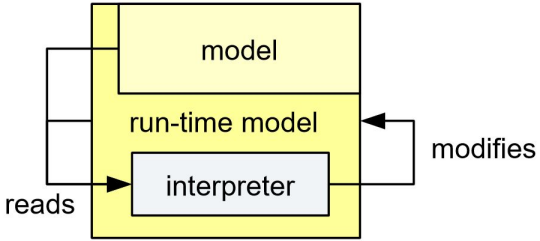
Priority of transitions:

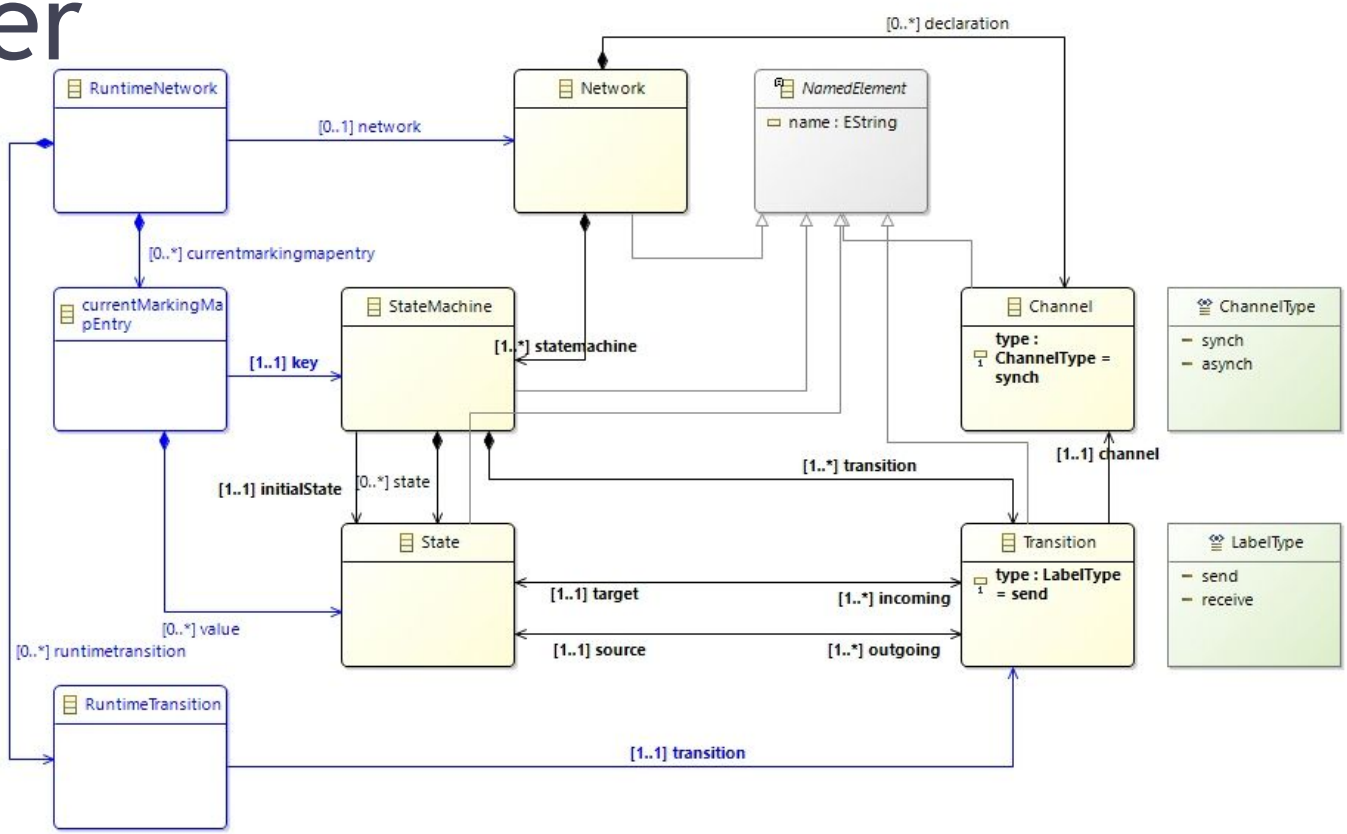
- Asynchronous receive transitions
- Asynchronous send and synchronous transitions

Code generator

```
'''
package «packagename»;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.ArrayList;
import java.util.List;

public class «networkClassName» {

    // states machines current states
    private Map<String, String> «currentStatesGlobal»;
    // channels buffer (declaration)
    private Map<String, Integer> «channelBuffersGlobal»;
    // state machines
    private List<String> «stateMachinesGlobal»;
    // transitions
    private Map<String, List<String>> «transitionsGlobal»;
    // transitions sources
    private Map<String, String> «sourcesGlobal»;
    // transitions sources
    private Map<String, String> «targetsGlobal»;
    // states
    private Map<String, List<String>>  «statesGlobal» = new HashMap<String, List<String>>();
    // for ramdomly chooose
    private Random «randomPairsSendVarGlobal»;
    private Random «randomReceiveVarGlobal»;
    private Random «randomPairsVarGlobal»;
    private Random «randomSendVarGlobal»;
```

Interpreter





Interpreter

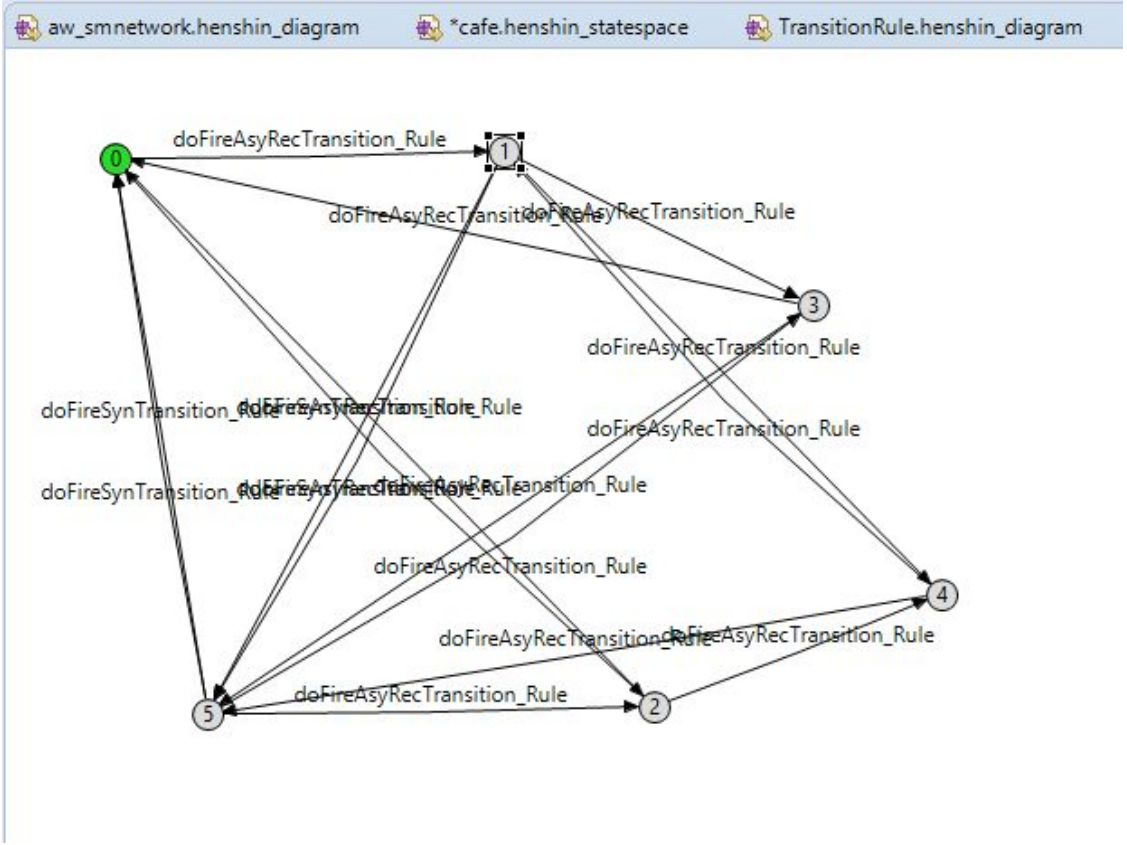
```
private def Network getNetwork(String fileName) {

    SmnetworkPackage.eINSTANCE.eClass

    // registering
    SMNLStandaloneSetup.doSetup();

    // load file
    //val URI uri = URI.createPlatformResourceURI(fileName, true);
    //val URI uri = URI::createURI(fileName);
    val URI uri = URI::createFileURI(fileName);
    System.out.println(uri);
    var resource = new ResourceSetImpl().getResource(uri, true);

    // get network package
    val networkPackage = resource.contents.get(0) as Network

    return networkPackage
}

def EList<RuntimeTransition> makeStep() {
    var EList<RuntimeTransition> enabledTransitions = getEnabledTransitions()
    if (enabledTransitions.size < 0) {
        throw new Exception("Deadlock")
    }

    // Our strategy to minimize the number of unreceived messages
    // Asynch Receive events have the priority
    var BasicEList<RuntimeTransition> receiveTransitions = new BasicEList<RuntimeTransition>();
    for (transition : enabledTransitions.filter[t|t.type == LabelType.RECEIVE && t.channel.type == ChannelType.ASYNCH]) {
        receiveTransitions.add(transition);
    }
```

Benchmark Analysis

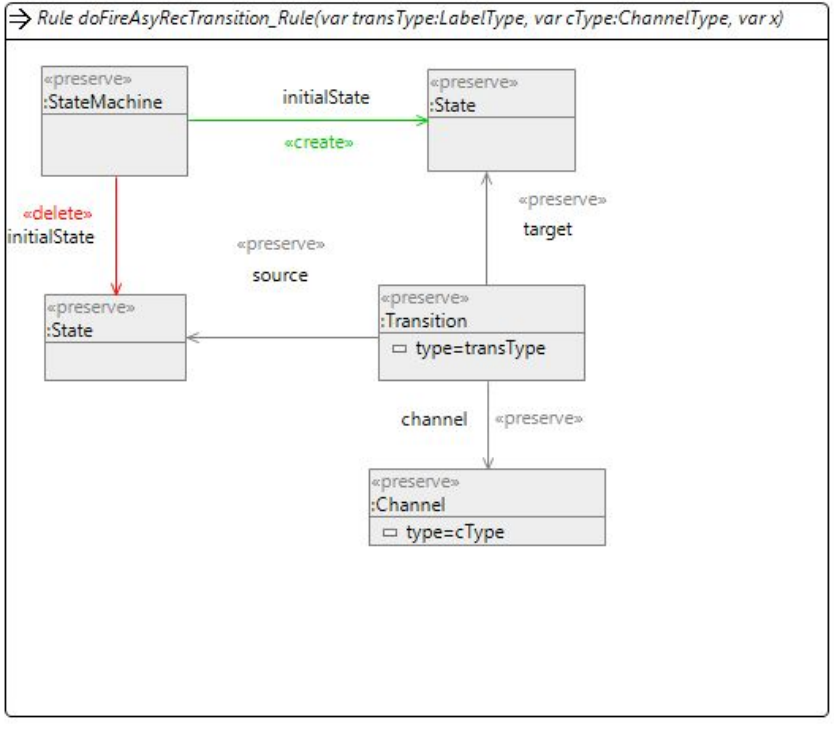
- Code Generator: Cafe
  - Total Time: 171 ms
- Interpreter: Cafe
  - Total Time: 1567 ms

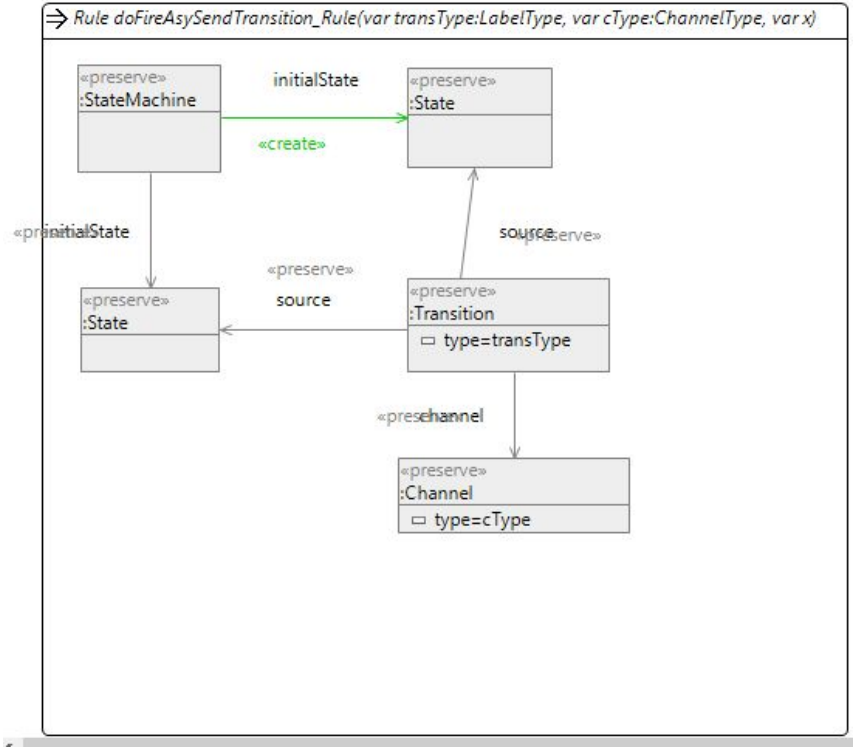
Henshin

- Visual editor



Henshin


Rule doFireAsyRecTransition_Rule(var transType:LabelType, var cType:ChannelType, var x)
«preserve»
:StateMachine
initialState
«create»
«preserve»
:State
«delete»
initialState
«preserve»
target
«preserve»
source
«preserve»
:State
«preserve»
:Transition
type=transType
channel
«preserve»
«preserve»
:Channel
type=cType


Rule doFireAsySendTransition_Rule(var transType:LabelType, var cType:ChannelType, var x)
«preserve»
:StateMachine
initialState
«create»
«preserve»
:State
initialState
source
«preserve»
source
«preserve»
:State
«preserve»
:Transition
type=transType
channel
«preserve»
:Channel
type=cType

Final remarks

- Interpreter take more time than the code generator to run
- Difficult to find helpful information for Henshin.
- Difficult to make Rule conditions with Enum types

Thank you for your attention!

Do you have any questions?