Mini-Project 2

Albert Koch
Hermann Kroll
Leif Wagner

Task 1 – Build a code generator

```
for (nsm : resource.allContents.toIterable.filter(NetworkStatemachine)){
    fsa.generateFile(
        "networkstatemachines/" + nsm.domain + ".java", '''
        «nsm.compile»
        '''
    )
    fsa.generateFile("networkstatemachines/State.java", '''«makeState»''')
    fsa.generateFile("networkstatemachines/Channel.java", '''«makeChannel»''')
    fsa.generateFile("networkstatemachines/Transition.java", '''«makeTransition»''')
    fsa.generateFile("networkstatemachines/Receive.java", '''«makeReceive»''')
    fsa.generateFile("networkstatemachines/Send.java", '''«makeSend»''')
    fsa.generateFile("networkstatemachines/Statemachine.java", '''«makeStatemachine»''')

}

}
```

Task 1 – Build a code generator

```
def Object makeTransition()
'''
package networkstatemachines;


public abstract class Transition {
    public Channel label;
    public State source;
    public State target;
    public boolean synchronous;
}
'''
def Object makeReceive()
'''
package networkstatemachines;


public class Receive extends Transition {
}
'''
def Object makeSend()
'''
package networkstatemachines;


public class Send extends Transition {
}
```

Task 1 – Build a code generator

```
def Object makeStatemachine()
'''
package networkstatemachines;
import java.util.List;
import java.util.ArrayList;

public class Statemachine {
    protected List<State> states = new ArrayList<State>();
    protected List<Transition> transitions  = new ArrayList<Transition>();
    protected List<Channel> channels = null;
    protected State currentState = null;

    public void makeStep(Channel c){
        Transition t = null;
        for(Transition tr: transitions) {
            if(tr.label.equals(c)) {
                t = tr;
                break;
            }
        }
        if(t != null){
            currentState = t.target;
        }
    }


}
'''
```

Task 1 – Build a code generator

```
public class «nsm.domain.toFirstUpper»{

    private List<Channel> channels = new ArrayList<Channel>();
    private List<Channel> asynchronous = new ArrayList<Channel>();
    private List<Channel> synchronous = new ArrayList<Channel>();
    private List<Statemachine> statemachines = new ArrayList<Statemachine>();

    «FOR sm : nsm.statemachines»
        «sm.name.toFirstUpper» «sm.name.toFirstLower»   = new «sm.name.toFirstUpper» ();
    «ENDFOR»

    //main method
        public static void main(String[] args) {
            «nsm.domain.toFirstUpper» «nsm.domain.toFirstLower»
            = new «nsm.domain.toFirstUpper»();

            «nsm.domain.toFirstLower».initialize();

            //make 100 Steps
            for(int i = 0; i< 100; i++)
            {
                «nsm.domain.toFirstLower».makeStep();
                }
        }

    «FOR sm: nsm.statemachines»
    «sm.compile»
    «ENDFOR»
```

Task 1 – Build a code generator

```
asynchronous.clear();
Channel c;

«FOR sm : nsm.statemachines»
    statemachines.add(«sm.name.toFirstLower»);
«ENDFOR»

«FOR ch : nsm.channels»
    «IF (!(ch.synchronous))»
        c = new Channel();
        c.name = "«ch.name»";
        c.synchronous = false;
        c.buffer = 0;
        asynchronous.add(c);
        channels.add(c);
    «ENDIF»
«ENDFOR»

synchronous.clear();
«FOR ch : nsm.channels»
    «IF ((ch.synchronous))»
        c = new Channel();
        c.name = "«ch.name»";
        c.synchronous = true;
        c.buffer = 0;
        synchronous.add(c);
        channels.add(c);
    «ENDIF»
«ENDFOR»
```

Task 1 – Build a code generator

```
List<Statemachine> fireableReceivers = new ArrayList<Statemachine>();
List<Channel> asynchReceiveChannels = new ArrayList<Channel>();
for(Channel ch : asynchronous){
    for(Statemachine s : statemachines){
        for(Transition t : s.transitions){
            if(t.label.equals(ch)){
                if(t.source.equals(s.currentState)&&
                (t instanceof Receive&& ch.buffer>0)){
                    if(!fireableReceivers.contains(s)){
                        if(!fireableReceivers.contains(s)){
                            fireableReceivers.add(s);
                            asynchReceiveChannels.add(ch);
                        } else {
                            int flipcoin = coin.nextInt(1);
                            if(flipcoin == 0){
                                int index = fireableSenders.indexOf(s);
                                fireableSenders.remove(index);
                                asynchSendChannels.remove(index);
                                fireableReceivers.add(s);
                                asynchReceiveChannels.add(ch);
                            }   else {
                                //do nothing
                            }
                        }
                    }
                }
```

Task 1 – Build a code generator

```
List<Statemachine> synchronousSenders = new ArrayList<Statemachine>();
List<Statemachine> synchronousReceivers = new ArrayList<Statemachine>();
List<Channel> syncChannels = new ArrayList<Channel>();

for(Channel ch : synchronous){
    //get possible senders
    List <Statemachine> sendersOfCh = new ArrayList<Statemachine>();
    for(Statemachine s : statemachines){
        for(Transition t : s.transitions){
            if(t instanceof Send){
                if(t.label.equals(ch)&&t.source.equals(s.currentState)){
                    sendersOfCh.add(s);
                }
            }
        }
    }
    //get possible receivers:
    List <Statemachine> receiversOfCh = new ArrayList<Statemachine>();
    for(Statemachine s : statemachines){
        for(Transition t : s.transitions){
            if(t instanceof Send){
                if(t.label.equals(ch)&&t.source.equals(s.currentState)){
                    receiversOfCh.add(s);
                }
            }
        }
    }
```

Task 1 – Build a code generator

```
while(getMinimum(sendersOfCh.size(), receiversOfCh.size())>0){
    int nextOneToAdd = coin.nextInt(getMinimum(sendersOfCh.size(), receiversOfCh.size())-1);

    if(!synchronousSenders.contains(sendersOfCh.get(nextOneToAdd))&&
    !synchronousReceivers.contains(receiversOfCh.get(nextOneToAdd))&&
    !synchronousSenders.contains(receiversOfCh.get(nextOneToAdd))&&
    !synchronousReceivers.contains(sendersOfCh.get(nextOneToAdd)))
    {
        synchronousSenders.add(sendersOfCh.get(nextOneToAdd));
        synchronousReceivers.add(receiversOfCh.get(nextOneToAdd));
        syncChannels.add(ch);
        sendersOfCh.remove(nextOneToAdd);
        receiversOfCh.remove(nextOneToAdd);
    } else {
         int flipcoin = coin.nextInt(1);
         /*
            Okay, either the sender or the receiver is already in the
            sender/receiver list for another channel. We'll flip a coin
            to determine which of the channels we want to stay inside our list
        */
```

Task 1 – Build a code generator

```
if(flipcoin == 0)
{
   int index = -1;
   if(synchronousSenders.contains(sendersOfCh.get(nextOneToAdd))){
       index = synchronousSenders.indexOf(sendersOfCh.get(nextOneToAdd));
   } else if(synchronousSenders.contains(receiversOfCh.get(nextOneToAdd))){
       index = synchronousSenders.indexOf(receiversOfCh.get(nextOneToAdd));
   } else if (synchronousReceivers.contains(receiversOfCh.get(nextOneToAdd))){
       index = synchronousReceivers.indexOf(receiversOfCh.get(nextOneToAdd));
   } else if(synchronousReceivers.contains(sendersOfCh.get(nextOneToAdd))){
       index = synchronousReceivers.indexOf(sendersOfCh.get(nextOneToAdd));
   }
   if(index < 0) {
       System.out.println("Aaach, Mist");
   } else {
       synchronousSenders.remove(index);
       synchronousReceivers.remove(index);
       syncChannels.remove(index);
       synchronousSenders.add(sendersOfCh.get(nextOneToAdd));
       synchronousReceivers.add(receiversOfCh.get(nextOneToAdd));
   }
   sendersOfCh.remove(nextOneToAdd);
   receiversOfCh.remove(nextOneToAdd);

}  else {
   sendersOfCh.remove(nextOneToAdd);
   receiversOfCh.remove(nextOneToAdd);
}
```

Task 1 – Build a code generator

```
/*
Sodele...now we have a set of synchronous senders and receivers and a set of asynchronous senders and
receivers. We still have to check if any statemachine is in both sets, sync and async:
*/
for(Statemachine s : fireableSenders){
    if(synchronousSenders.contains(s)) {
        int flipcoin = coin.nextInt(1);
        if(flipcoin == 0) {
            int index = synchronousSenders.indexOf(s);
            synchronousSenders.remove(index);
            synchronousReceivers.remove(index);
            syncChannels.remove(index);
        } else {
            fireableSenders.remove(s);
        }
    }
    if(synchronousReceivers.contains(s)) {
        int flipcoin = coin.nextInt(1);
        if(flipcoin == 0) {
            int index = synchronousReceivers.indexOf(s);
            synchronousSenders.remove(index);
            synchronousReceivers.remove(index);
            syncChannels.remove(index);
        } else {
            fireableSenders.remove(s);
        }
    }
}
```

Task 1 – Build a code generator

```
for(Statemachine s : fireableReceivers){
    if(synchronousSenders.contains(s)) {
        int flipcoin = coin.nextInt(1);
        if(flipcoin == 0) {
            int index = synchronousSenders.indexOf(s);
            synchronousSenders.remove(index);
            synchronousReceivers.remove(index);
            syncChannels.remove(index);
        } else {
            fireableReceivers.remove(s);
        }
    }
    if(synchronousReceivers.contains(s)) {
        int flipcoin = coin.nextInt(1);
        if(flipcoin == 0) {
            int index = synchronousReceivers.indexOf(s);
            synchronousSenders.remove(index);
            synchronousReceivers.remove(index);
            syncChannels.remove(index);
        } else {
            fireableReceivers.remove(s);
        }
    }
}
```

Task 1 – Build a code generator

```
for(int i = 0; i< syncChannels.size(); i++){
    synchronousSenders.get(i).makeStep(syncChannels.get(i));
    synchronousReceivers.get(i).makeStep(syncChannels.get(i));
}
for(int i = 0; i< asynchSendChannels.size(); i++) {
    fireableSenders.get(i).makeStep(asynchSendChannels.get(i));
    asynchSendChannels.get(i).buffer++;
}

for(int i = 0; i< asynchReceiveChannels.size(); i++) {
    fireableReceivers.get(i).makeStep(asynchReceiveChannels.get(i));
    asynchReceiveChannels.get(i).buffer--;
}
```

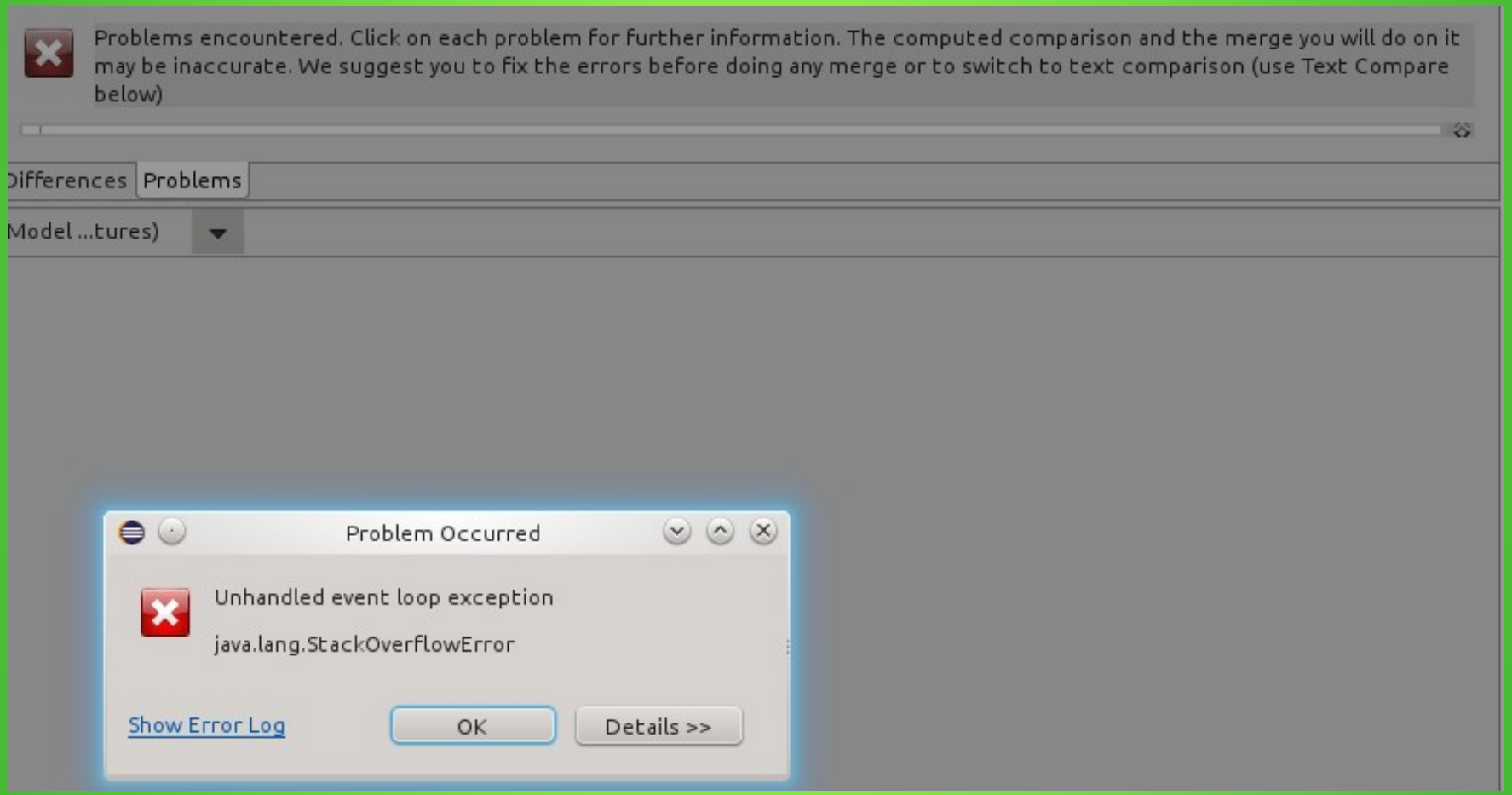
Task 1 – Build a code generator

```
def Object makeStatemachine()
'''
package networkstatemachines;
import java.util.List;
import java.util.ArrayList;

public class Statemachine {
    protected List<State> states = new ArrayList<State>();
    protected List<Transition> transitions  = new ArrayList<Transition>();
    protected List<Channel> channels = null;
    protected State currentState = null;

    public void makeStep(Channel c){
        Transition t = null;
        for(Transition tr: transitions) {
            if(tr.label.equals(c)) {
                t = tr;
                break;
            }
        }
        if(t != null){
            currentState = t.target;
        }
    }


}
'''
```

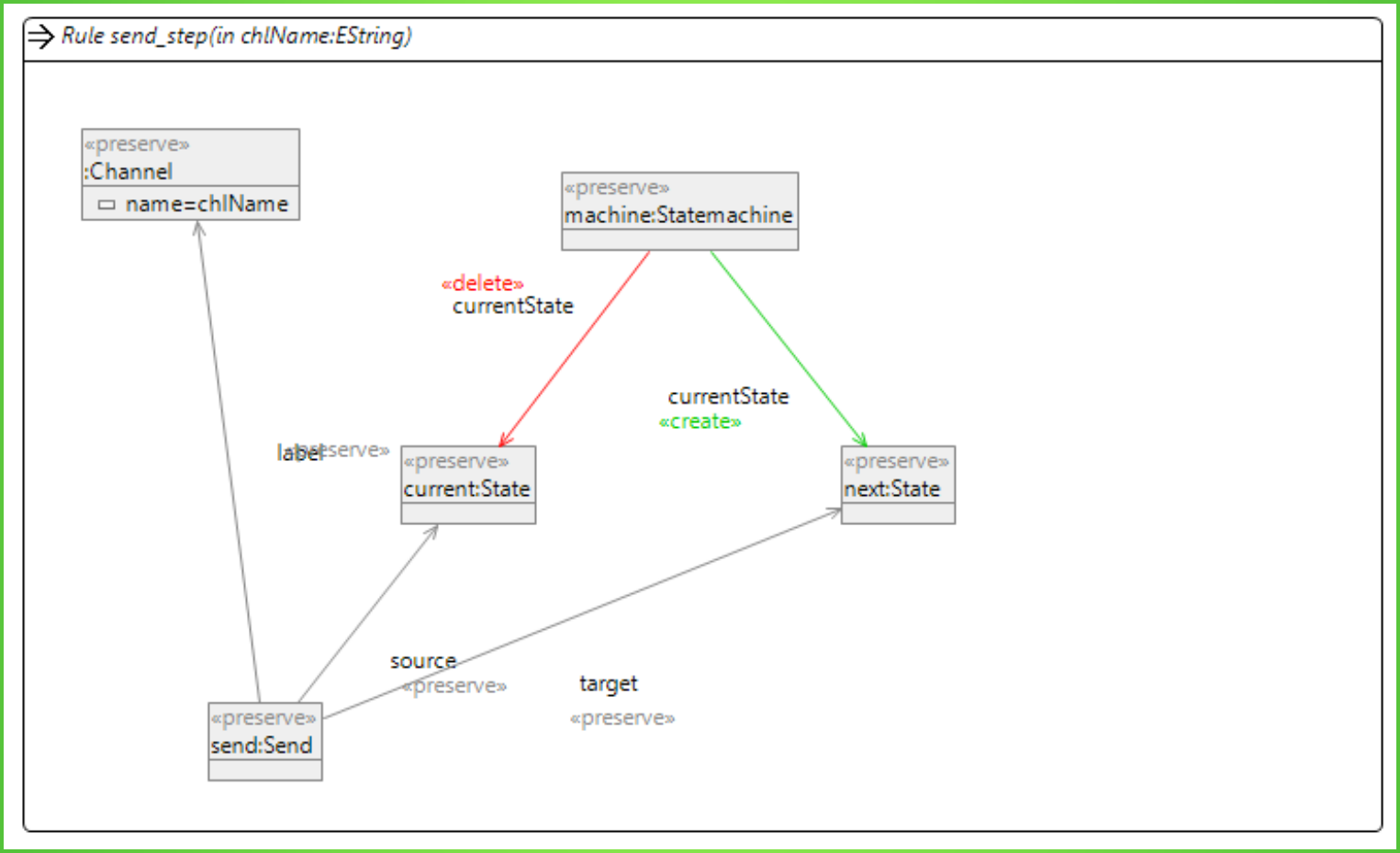
Task 2 – Build an interpreter

```
//Well...that doesn't work at all...
public static void main(String[] args) {
    MiniProject1Package.eINSTANCE.eClass();

    Resource.Factory.Registry registry = Resource.Factory.Registry.INSTANCE;
    Map<String, Object> map = registry.getExtensionToFactoryMap();
    map.put("nsm", new XMIResourceFactoryImpl());

    ResourceSet resourceSet = new ResourceSetImpl();
    Resource resource = resourceSet.getResource(URI.createURI("Semester.xmi"), true);

    EObject res = resource.getContents().get(0);
    NetworkStatemachine rtn = (NetworkStatemachine) res;
    System.out.println(rtn.getDomain());
}
```

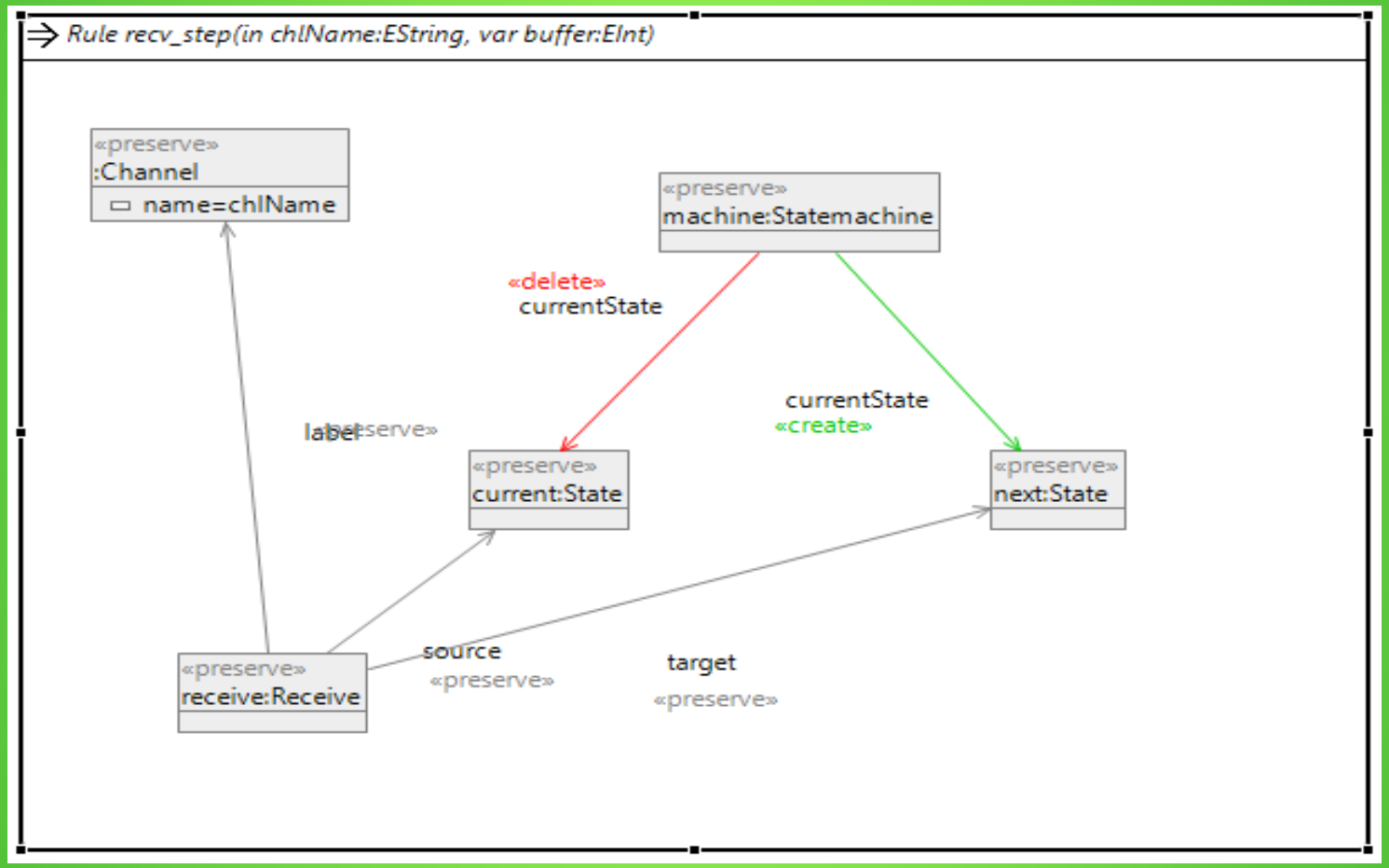
Task 3 – Henshin



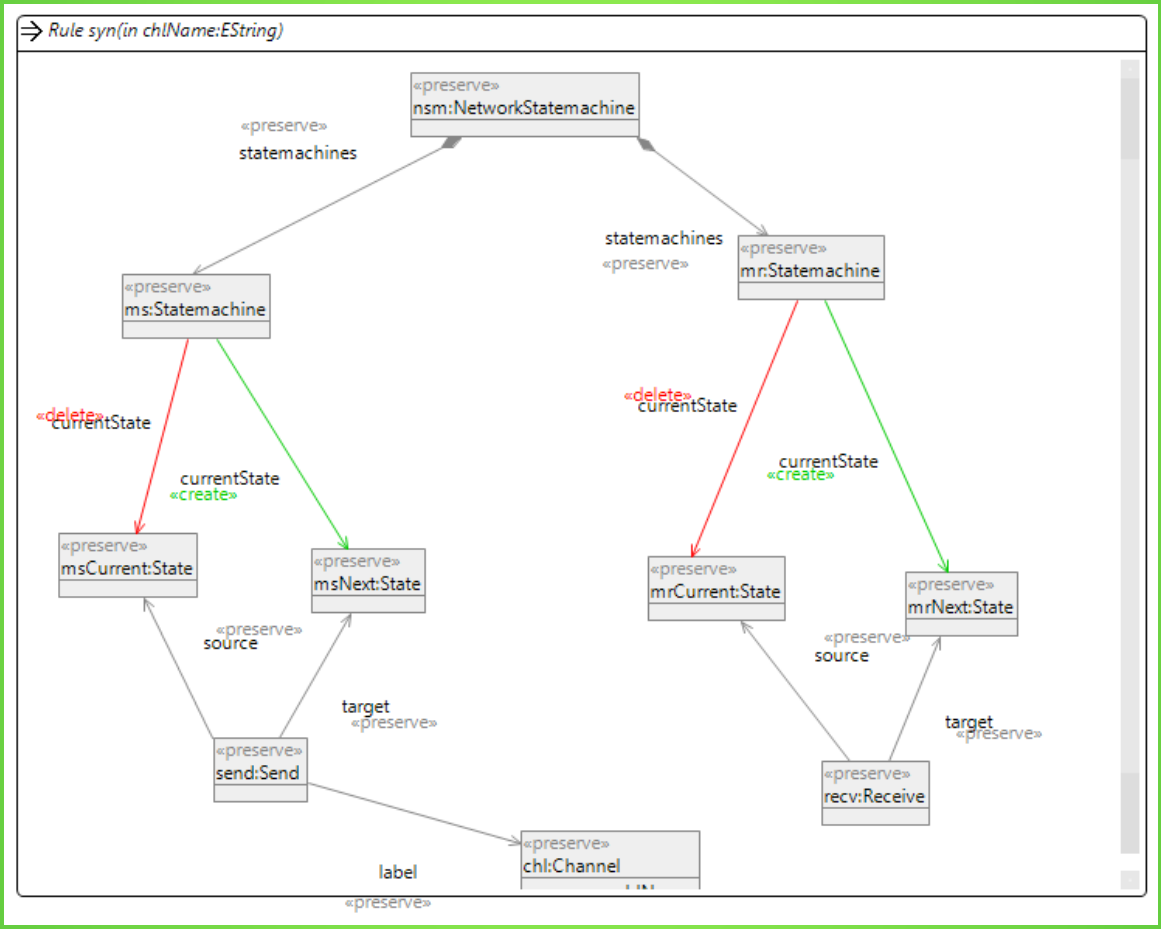
Task 3 – Henshin



Task 3 – Henshin



Task 3 – Henshin



That's it!

Thank you for

Jura-Tännschen!