Towards Verified Java Code Generation from Concurrent State Machines

Dan Zhang
Dragan Bosnacki
Mark van den Brand
Luc Engelen
Cornelis Huizing
Ruurd Kuiper
Anton Wijs

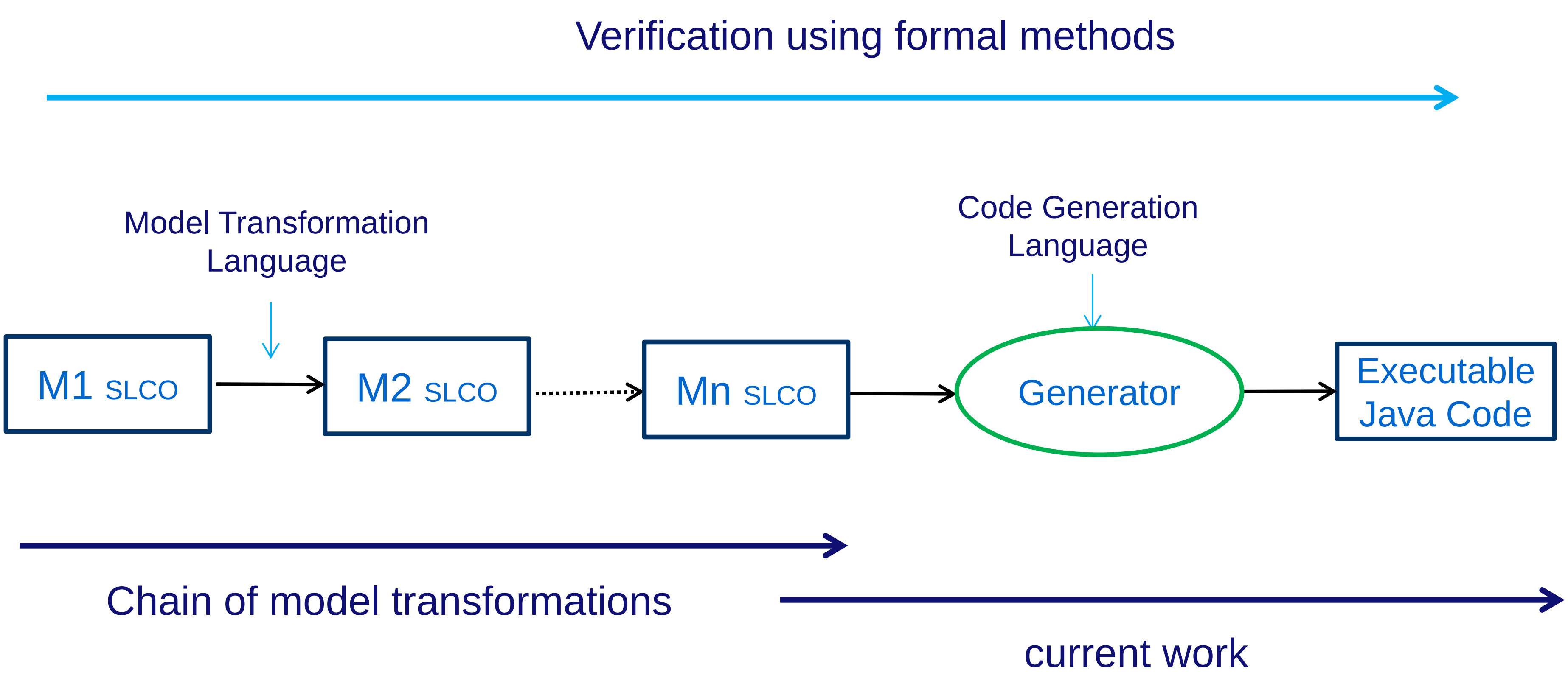
Technische Universiteit Eindhoven
University of Technology

Where innovation starts

Outline

- ✓ Setting the scope
- ✓ A model specification language: Simple Language of Communication Objects (SLCO) based on finite state machines
- ✓ Automated transformation from SLCO model to code (Java).
- ✓ Verification of the transformation (work in progress)
- ✓ Conclusion

Setting the Scope



Simple Language of Communication Objects (SLCO)

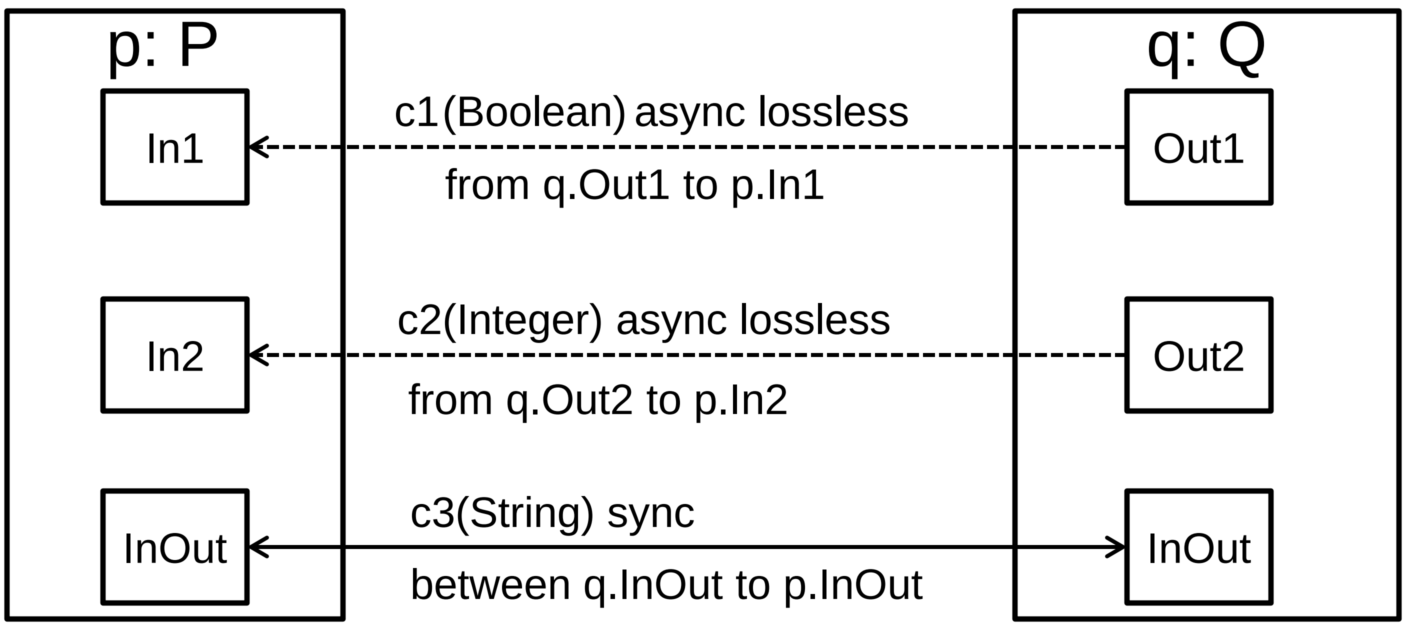
- ✓ SLCO is a small domain-specific modeling language
- ✓ SLCO models are collections of concurrent objects
- ✓ The dynamics of objects is given by state machines
- ✓ The state machines can communicate via
  - Shared memory(class variables)
  - Message passing(channels)

An SLCO Model Using the Textual Syntax

```
model PaperExample1 {
  classes
    P {
        variables
           Integer m = 0
        ports
            In1 In2 InOut
        state machines
          Rec1 {
                variables Boolean v = true
                initial Rec1
                transitions
                   Rec102Rec1 from Rec1 to Rec1 {
                      receive P(v| v == false) from In1
                   }
                }
          }
          Rec2 {
                initial Rec2a
                state Rec2b
                transitions
                  Rec2a2Rec2b from Rec2a to Rec2b {
                    receive P(m| m >= 0) from In2
                  }
                  Rec2b2Rec2a from Rec2b to Rec2a {
                        m := m+1
                  }
          }
         SendRec {…}
    }
Q {
    ports
        Out1 Out2 InOut
    state machines
       Com {
            variables String s = ""
            initial Com0
            state Com1 Com3 Com4
            final
              Com2
            transitions
              Com02Com1 from Com0 to Com1 {
                send P(true) to Out1
              }…
              Com02Com2 from Com0 to Com2 {
                after 5 ms
              }
       }
}
objects
  p: P
  q: Q
channels
c1(Boolean) async lossless from q.Out1 to p.In1
c2(Integer) async lossless from q.Out2 to p.In2
c3(String) sync between q.InOut and p.InOut
}
```

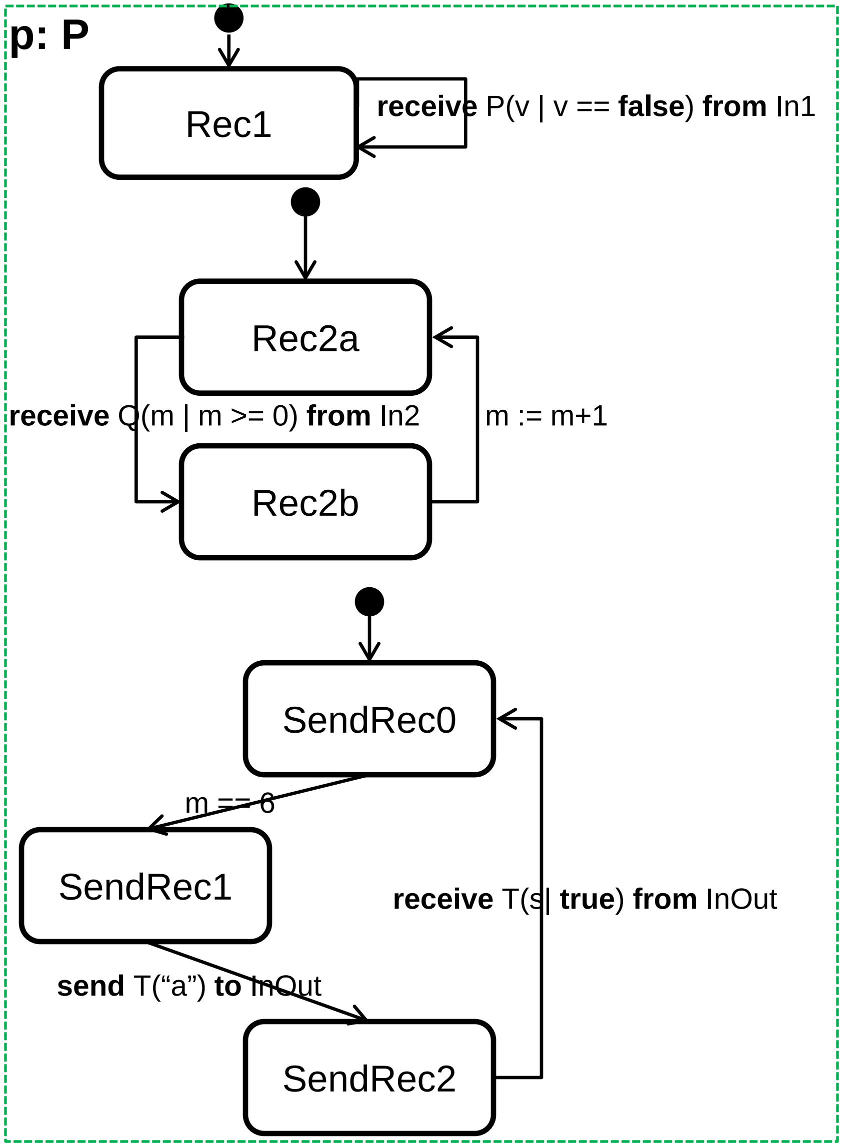
Channels

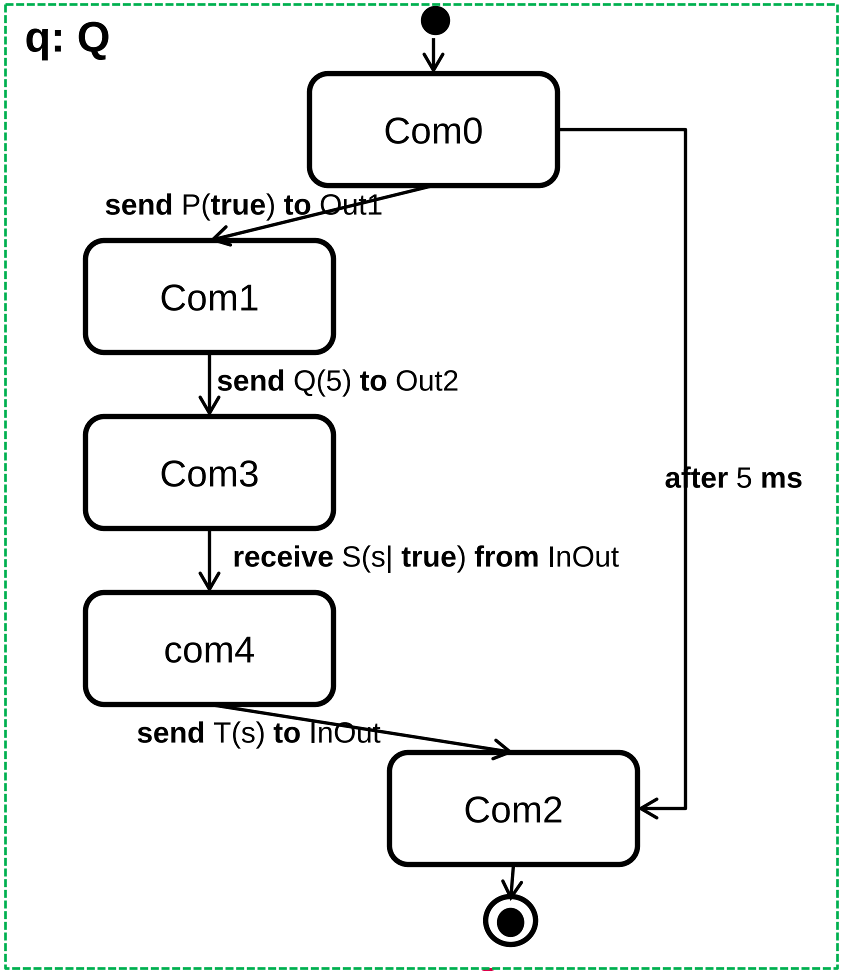
- Objects, instances of classes, communicate with each other via channels.
- SLCO offers three types of channels:
  - Synchronous channel
  - Asychronous lossless channel
  - Asychronous lossy channel



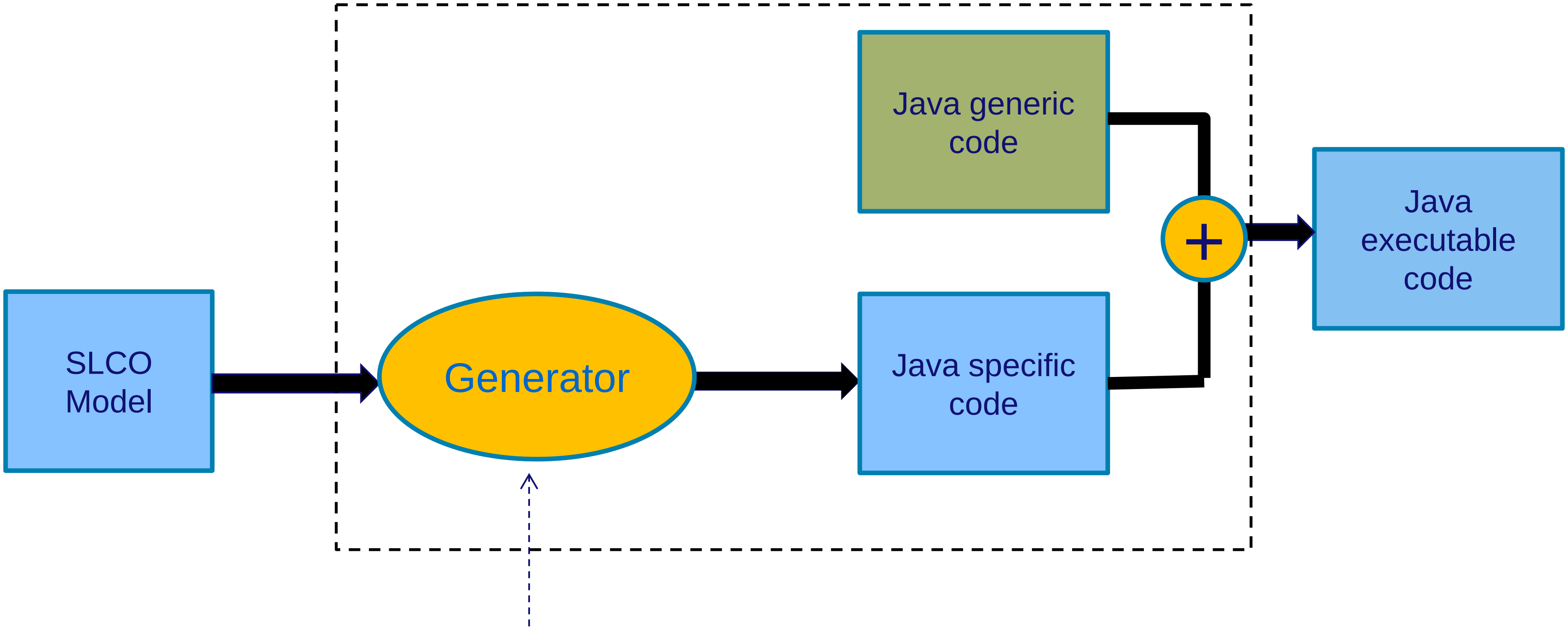
Technische Universiteit Eindhoven University of Technology

Graphical Representation





From SLCO Model to Java Code



This part is created in the Epsilon Generation Language (EGL) tailored for model-to-text transformation.

From SLCO Model to Generated Java Code

State Machine Com in Class Q

```
Com {
   initial
      Com0
   state
      Com1 Com2 Com3
   final
      Com3
   transitions
      Com02Com1 from Com0 to Com1 {
         send P(true) to Out1
       }
       Com02Com2 from Com0 to Com2 {
         after 5 ms
       }
       Com12Com3 from Com1 to Com3 {
         …
       }
       Com22Com3 from Com2 to Com3 {
         …
       }
}
```

Java code

```
String currentState = "Com0";
  switch(currentState){
    case "Com0":
      String nextTransition;
      String transitions[] =
            {"Com02Com1", "Com02Com2"};
      …
      int idx =
         new Random().nextInt(transitions.length);
      nextTransition = transitions[idx];

      switch(nextTransition){
        case "Com02Com1":
          …
        case "Com02Com2":
          …
      }
    }

    case "Com1":
      …
    case "Com2":
      …
}
```

Non-deterministic transitons

How should the states look like in Java?

- State machine structure preserving
- Understanding easy
- Verification feasible

Generated Code from SLCO Model

Sending Statement in SLCO

```
Com42Com2 from Com4 to Com2 {
  send T(s) to InOut
}
```

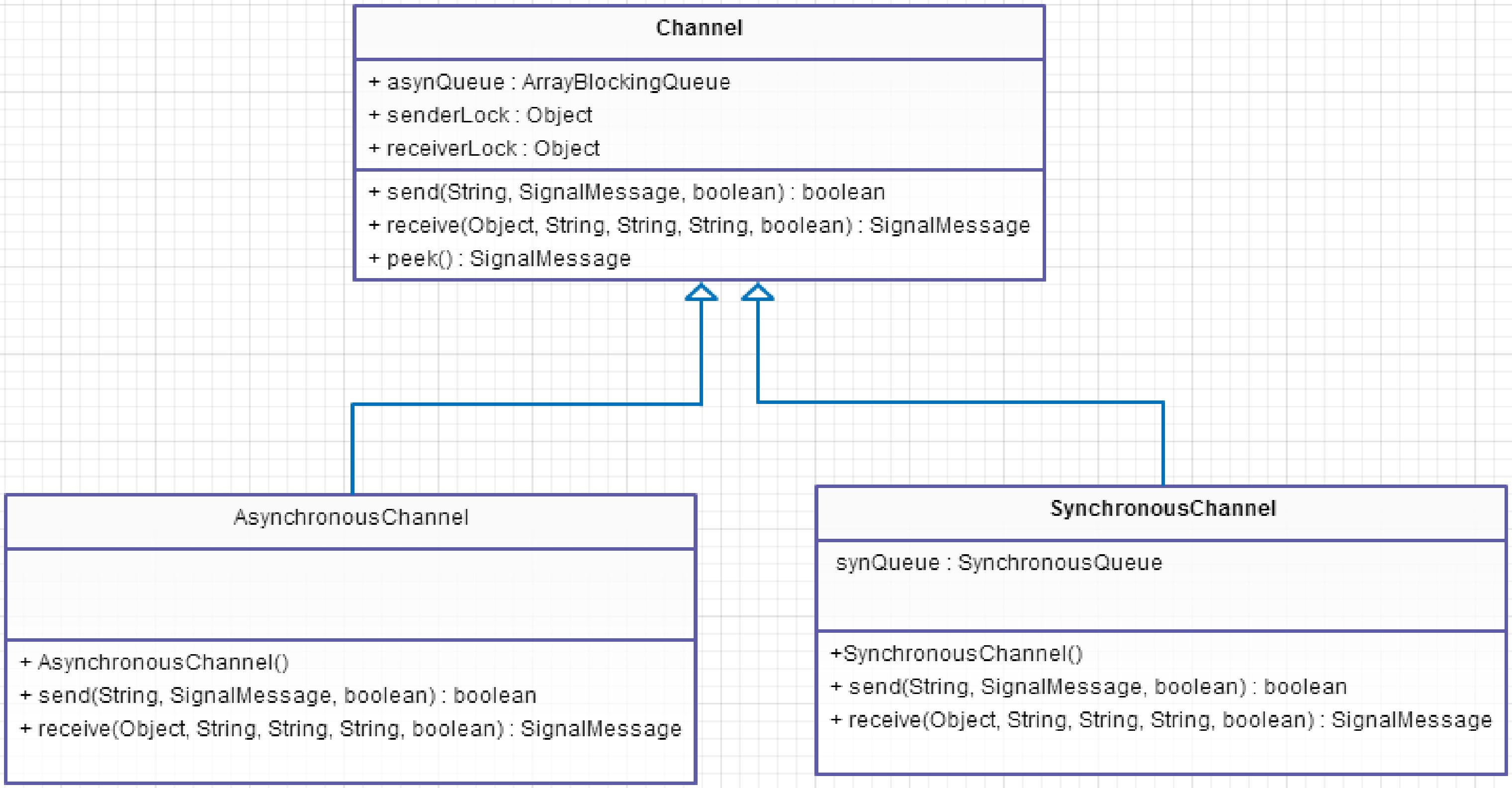
```
case "Com4":
    try {
        port_InOut.channel.send("Com42Com2", new
        SignalMessage("T",new Object[]{s}), false);
        currentState = "Com2";
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    break;
```

The implementation of channel should be hidden in the generic code.

- user
- verification

Technische Universiteit Eindhoven University of Technology

Generic Code Structure of Channels



Technische Universiteit
Eindhoven
University of Technology

Generic Code of Asynchronous Channel

```java
import java.lang.reflect.Method;
import java.util.concurrent.ArrayBlockingQueue;

class AsynchronousChannel extends Channel {

    public AsynchronousChannel() {
        asynQueue = new ArrayBlockingQueue<SignalMessage>(1);
    }

    @SuppressWarnings("unchecked")
    @Override
    public boolean send(String transitionName, SignalMessage s,
            boolean isNonDeterministicTransition) throws InterruptedException {
        // TODO Auto-generated method stub
        synchronized (senderLock) {
            SignalMessage signal = peek();
            if (isNonDeterministicTransition) {
                if (signal == null) {
                    asynQueue.put(s);
                    System.out.println("Transition: " + transitionName);
                    return true;
                } else {
                    return false;
                }
            } else {
                asynQueue.put(s);
                System.out.println("Transition: " + transitionName);
                return true;
            }
        }
    }

    public SignalMessage receive(Object object, String conditionName,
}
```

Results

Previous results

- ✓ Java channel implementation
- ✓ Java channel specification with Separation Logic
- ✓ Verified the channel using VeriFast tool

Current results

new generic code

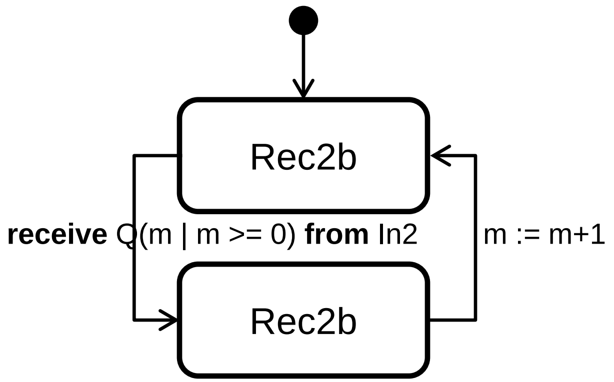
- ✓ Verification oriented OO design
- ✓ Considering fairness
- ✓ More efficiency
    - Java synchronization construct

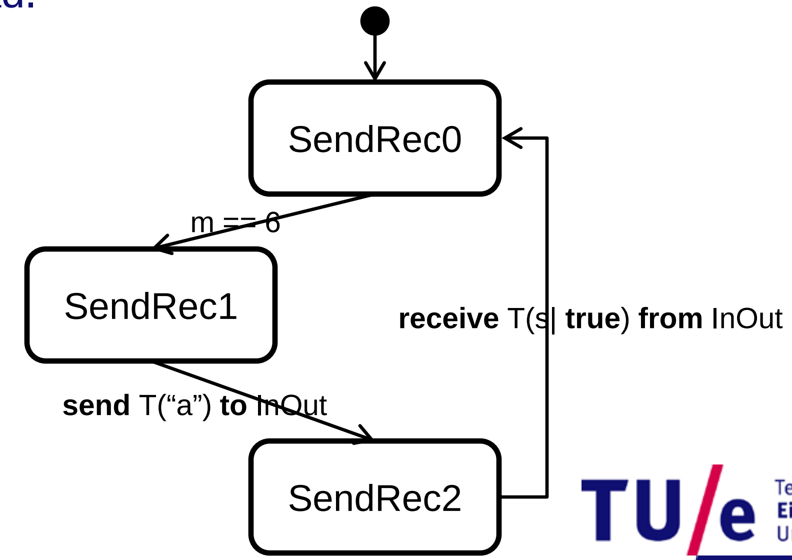
Challenges

Shared variables - atomicity

In SLCO, the class variables can be accessed and /or modified by multiple state machines.

- ✓ **Locking constructs** limit the number of threads that can perform some activity.
- ✓ **Signaling constructs** used to let a thread pause until receiving a notification from another thread.





Challenges

Channels - synchronization

In SLCO, signals can be sent over synchronous channels and asynchronous channels. Determining when both sender and receiver are available for sending and receiving is difficult.

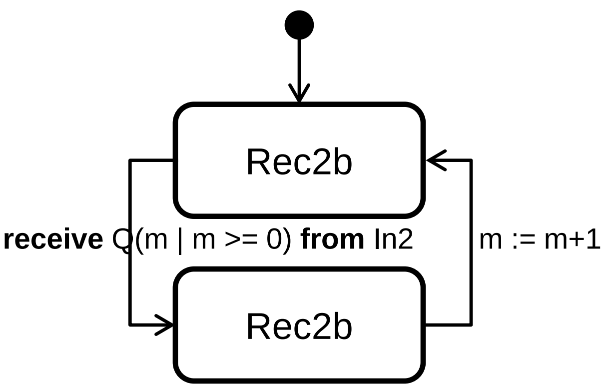
- ✓ Synchronous communication
  - Both receiving and sending party need to be available before a signal can be sent
  - The condition of the signal should be satisfied
- ✓ Asynchronous communication
  - The condition of the signal needs to be checked before exchanging the message
- ✓ Aiming at a generic solution for conditional synchronous and asynchronous communication

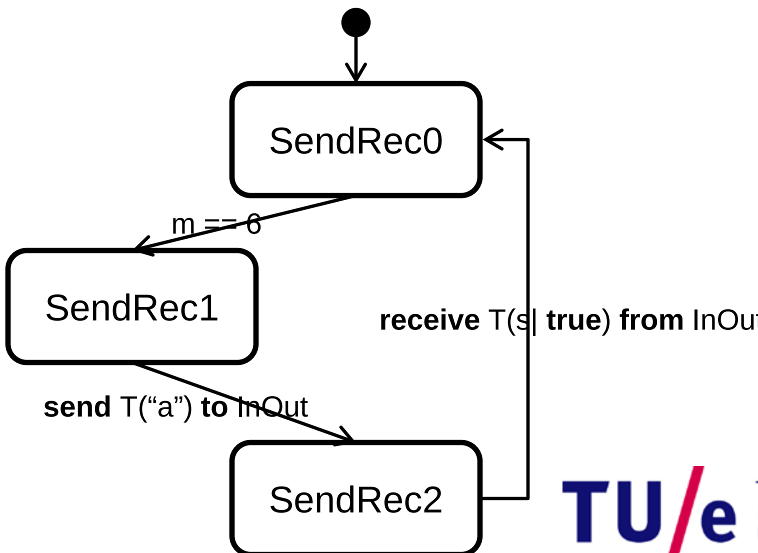
Challenges

Conditional transition

Each statement in SLCO is either blocked or enabled. we need to find a construct to simulate the blocking in Java

- ✓ busy-waiting
- ✓ Wait-notify
- ✓ ?





Challenges

Fairness

- ✓ We use an interleaving semantics for SLCO with weak fairness.
  - ❑ if at some time point a transition becomes continuously enabled, this transition will at some later time point be taken.

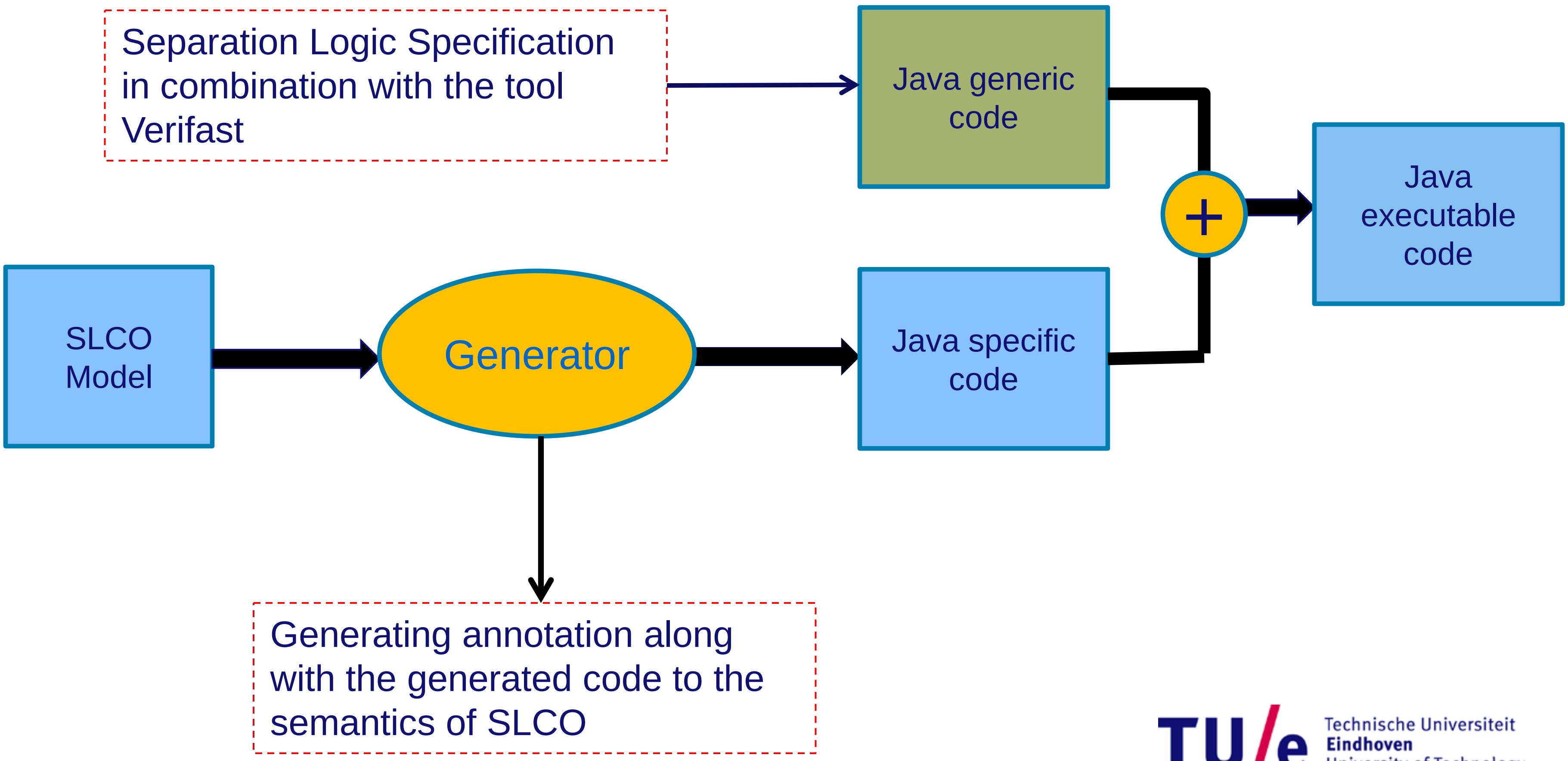
- ✓ We need stronger fairness in Java.
  - ❑ The granularity in Java is much finer than in SLCO, more progress is enforced by weak fairness in SLCO than in Java.

- ✓ We aim to achieve this through a combination of fairness in
  - ❑ scheduling threads, obtained by choosing the right JVM
  - ❑ fair locks, obtained from the package java.util.concurrent.locks.

TU/e Technische Universiteit Eindhoven University of Technology

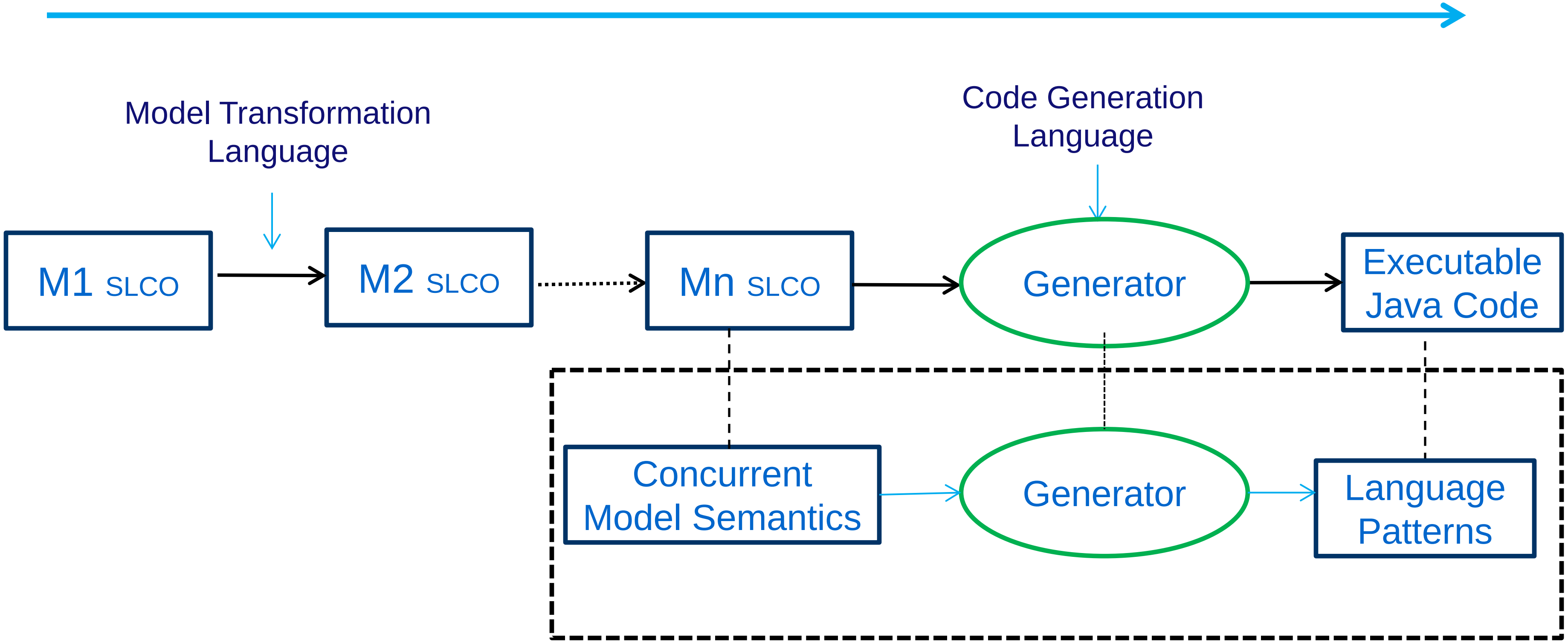
Challenges

- Verification



Generalization

Verification using formal methods



A basis for developing efficient simulation, formal verification and other analysis tools

Conclusion

- ✓ Investigated fairness aspects of a model specification
- ✓ Changed automated transformation to more verification oriented OO code
- ✓ Identified and presented tentative solutions to challenges

Technische Universiteit
Eindhoven
University of Technology

Questions

Thank you very much!

TU/e Technische Universiteit Eindhoven University of Technology