Towards Object-Oriented Modeling in SCCharts

Alexander Schulz-Rosengarten,
Steven Smyth, Kiel University
and Michael Mendler, Bamberg University

Encapsulation

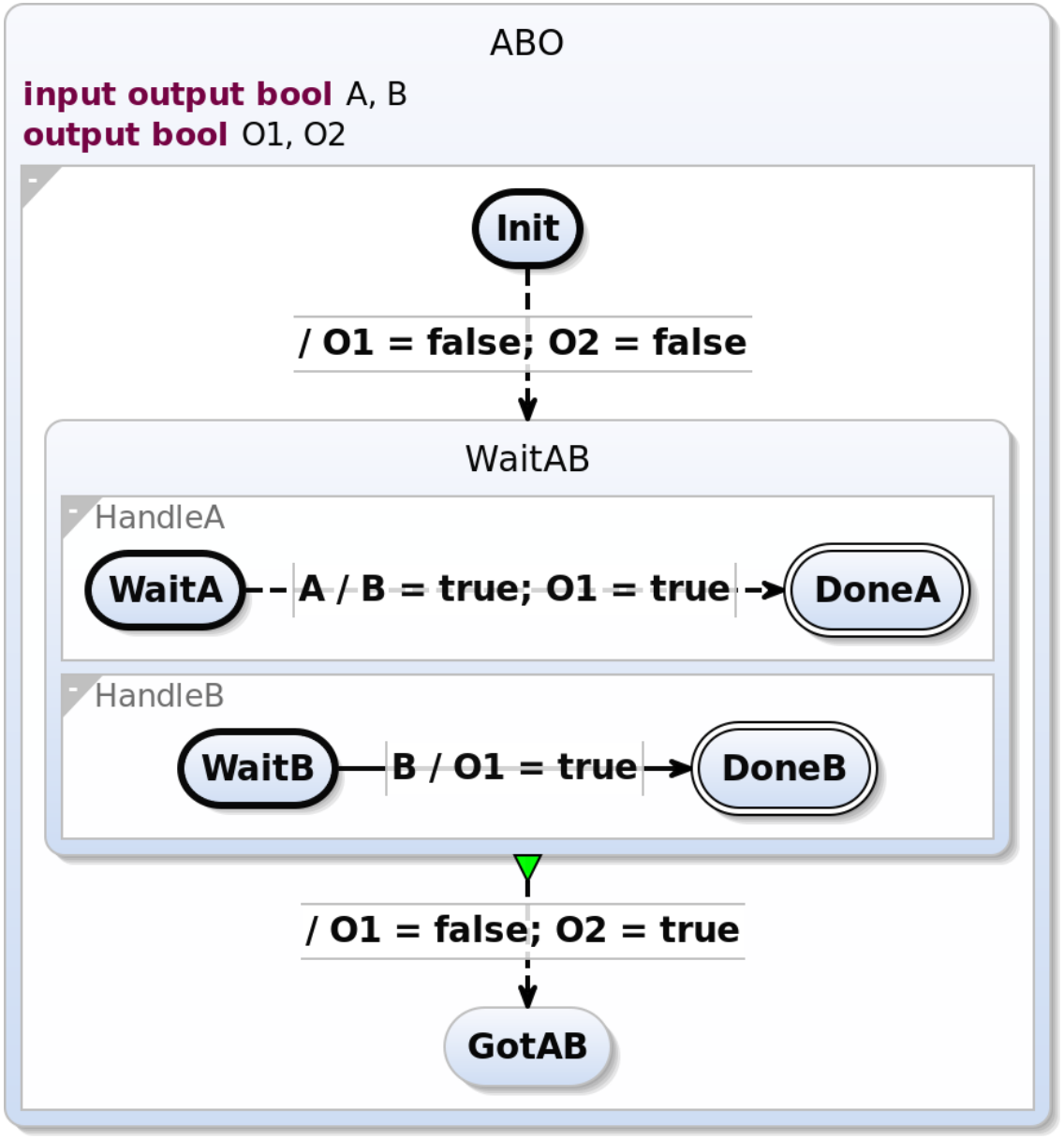
Abstraction Abstract Data Types

OOP Paradigms

Inheritance

Polymorphism

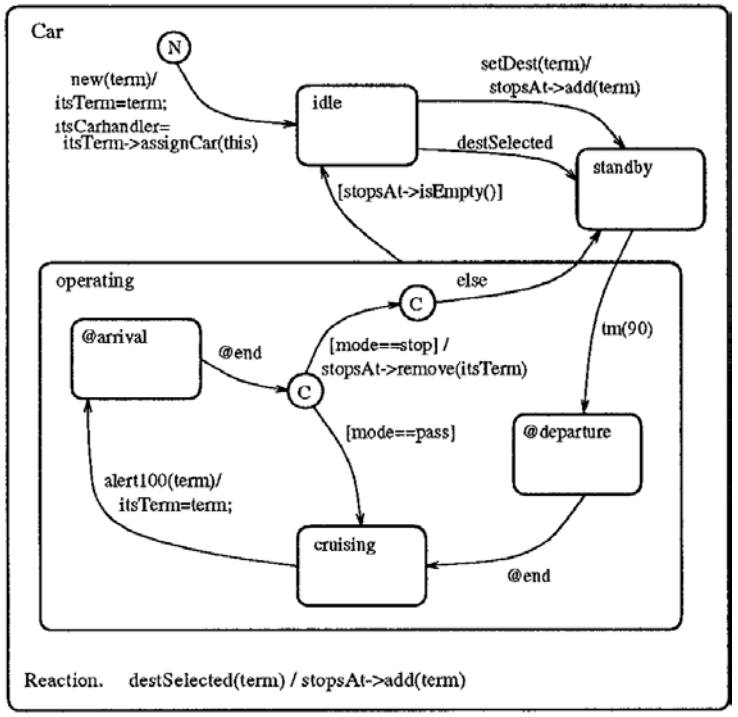
SCCharts

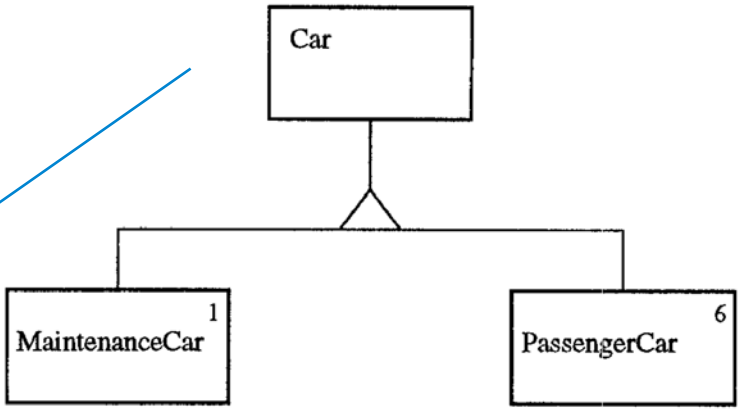

ABO
input output bool A, B
output bool O1, O2
Init
/ O1 = false; O2 = false
WaitAB
HandleA
WaitA
A / B = true; O1 = true
DoneA
HandleB
WaitB
B / O1 = true
DoneB
/ O1 = false; O2 = true
GotAB

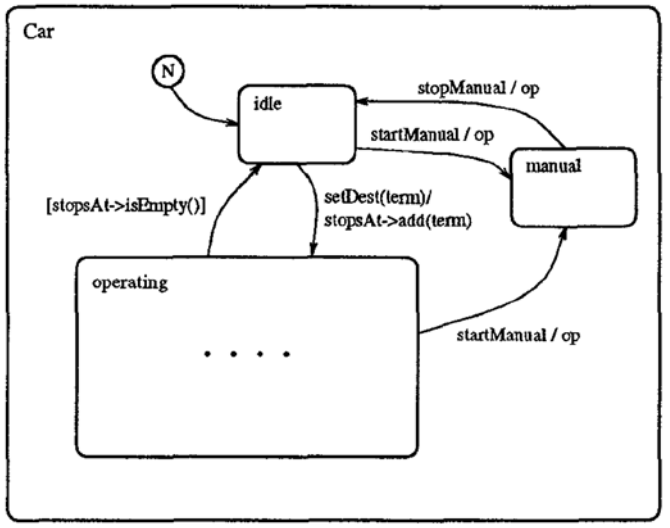
Related Work

O-charts

D. Harel and E. Gery.
Executable object modeling with statecharts.
ICSE '96



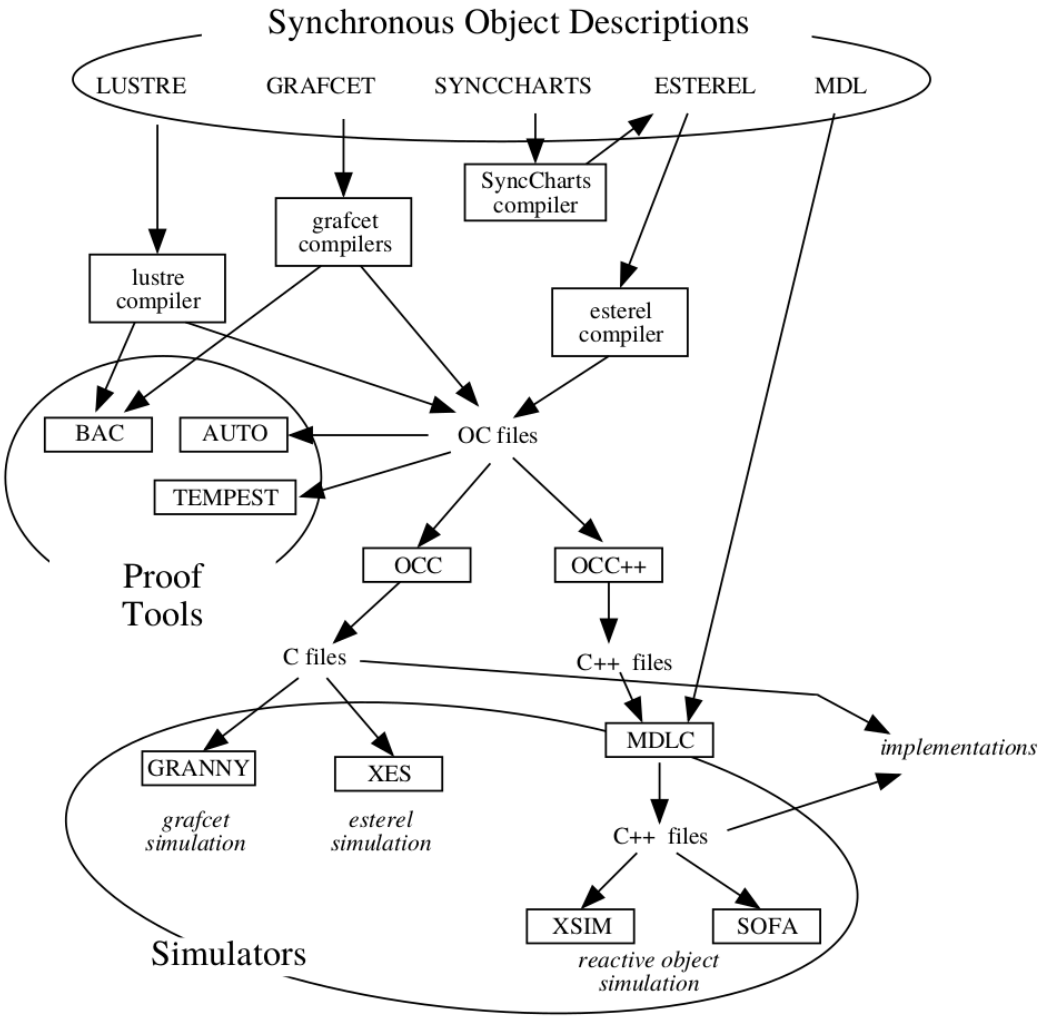




Related Work

OO Code Generation for Synchronous Languages

C. André, F. Boulanger, M.-A. Péraldi, J. P. Rigault, and G. Vidal-Naquet. *Objects and synchronous programming.* RAIRO-APII-JESA-Journal Europeen des Systemes Automatises, 1997



Related Work

O. Hainque, L. Pautet, Y. Le Biannic, É. Nassor:
Cronos: A Separate Compilation Toolset for Modular Esterel Applications.
In: World Congress on Formal Methods. September 1999.

Esterel module = Ada package {signal accessors,
accessors for partial evaluation nodes, Initialise, Reset, Run, Clear}

... See also object modularisation Obc in the Vélus compilation chain

P. Caspi, J-L. Colaco, L. Gérard, M. Pouzet, P. Raymond :
Synchronous Objects with Scheduling Policies (Introducing safe shared memory in Lustre).
In: LCTES 2009, Dublin.

Related Work

Abstract Data Types in Synchrouns Languages

F. Gretz and F. Grosch.
Blech, imperative synchronous programming!
FDL'18

Also:

Sant'Anna, R. Ierusalimschy, N. Rodriguez.
Structured synchronous reactive programming with Céu.
Proc. of the 14th International Conference on Modularity.

```
struct PID
  var KI: float32 = 0.0
  var KD: float32 = 0.0
  var KP: float32 = 1.0
  var dt: float32 = 1.0
with
  function this:calc
           ( pv: float32, sp: float32 )
           ( priErr: float32,
             intl: float32, cmd: int32 )
    let error = sp - pv
    let der = (error - priErr) / this.dt
    intl = intl + (error * this.dt)
    cmd = this.KP * error + this.KI * intl
          + this.KD * der
    priErr = error
  end

  activity this:control(pv:float32,sp:float32)
                        (cmd:int32)
    var priErr: float32 = 0
    var intl: float32 = 0
    repeat
      this:calc(pc, sp)(priErr, intl, cmd)
      await true
    end
  end
end
```

Towards Object-Oriented Modeling in SCCharts

OO Statecharts Design

Inheritance

Goals

Abstraction / Structure

Robust Semantics

Objects and Classes

Class Types
Hostcode Integration

Goal

Determinism

Object-Oriented Statecharts Design in SCCharts

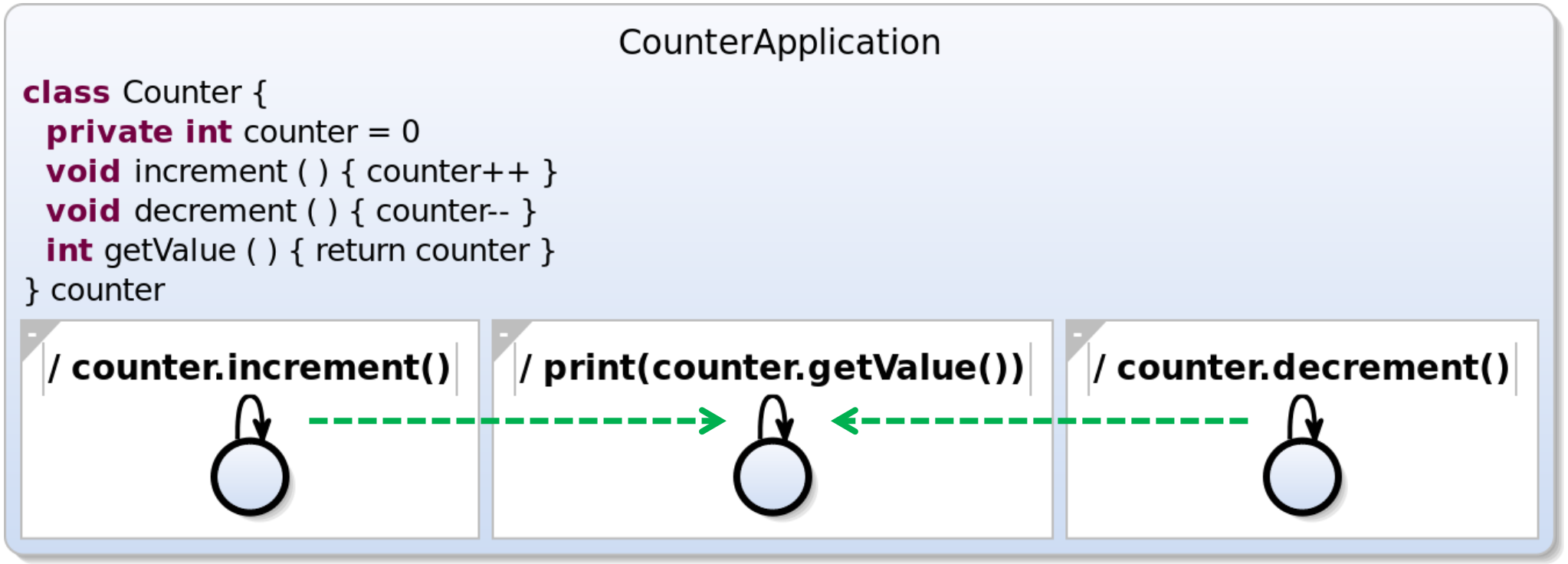
Class Types

Class Types

CounterApplication

```
class Counter {
  private int counter = 0
  void increment ( ) { counter++ }
  void decrement ( ) { counter-- }
  int getValue ( ) { return counter }
} counter
```

Using Objects



IUR scheduling dependencies obtained from method code

Hostcode Classes

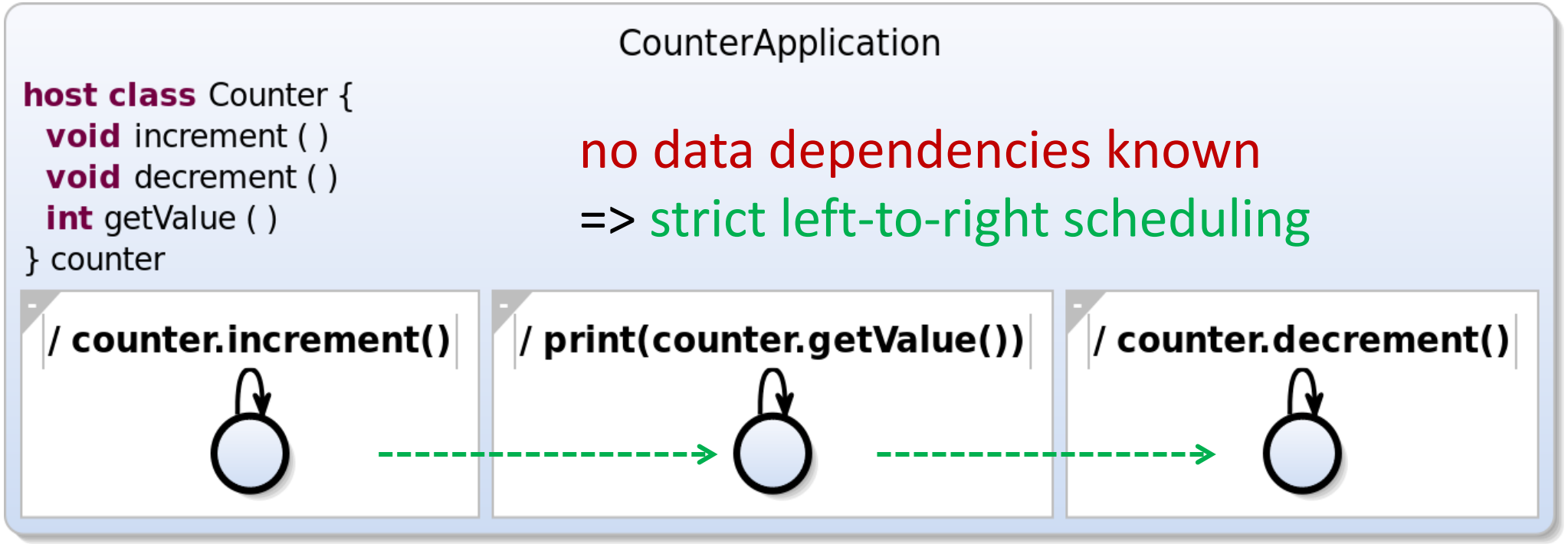
```
class Counter {
    private int value = 0;

    public void increment() {
        value++;
    }
    public void decrement() {
        value--;
    }
    public void getValue() {
        return value;
    }
}
```

CounterApplication

```
host class Counter {
  void increment ( )
  void decrement ( )
  int getValue ( )
} counter
```

Black Box vs. White Box Scheduling



S. Andalam, P. S. Roop, A. Girault, Alain:
Deterministic, predictable and light-weight multithreading using PRET-C.
[DATE'10]

M. Lohstroh, M. Schoeberl, A. Goens, A. Wasicek, C. Gill, M. Sirjani, and E. A. Lee.
Invited: Actors revisited for time-critical systems. [DAC'19]

Object-Specific Scheduling Regimes

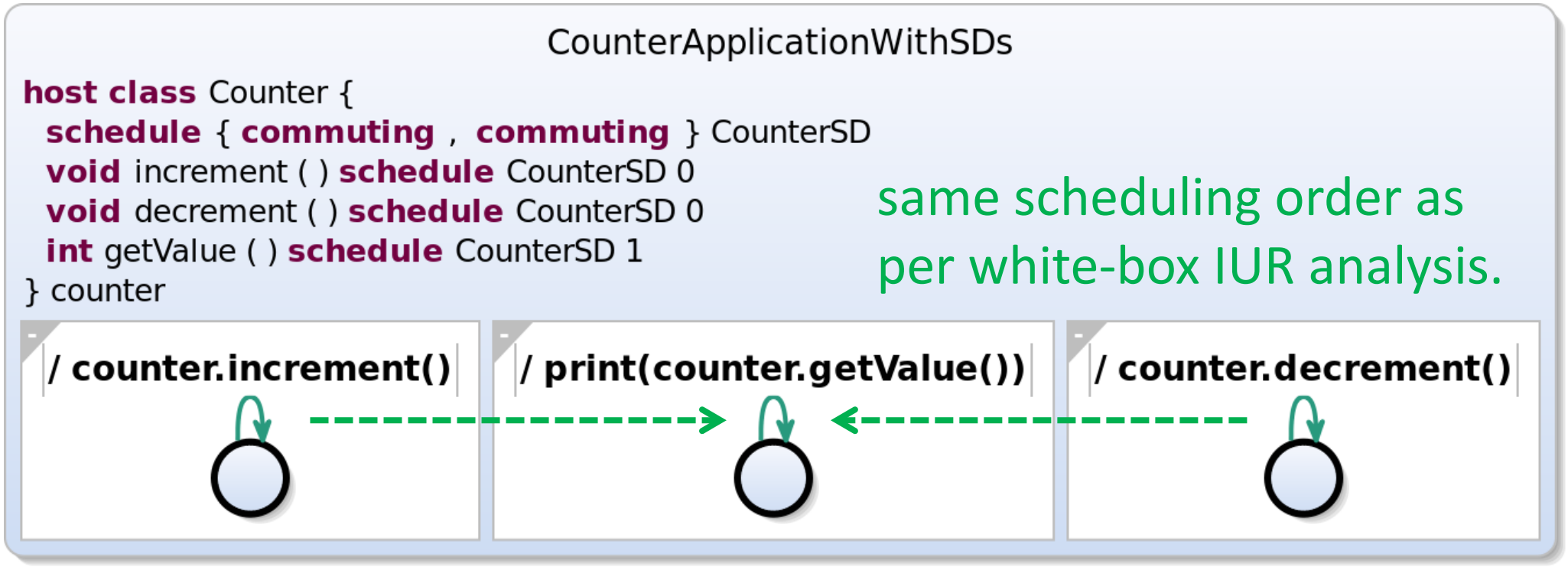
Scheduling Directives

S. Smyth, A. Schulz-Rosengarten, and R. von Hanxleden.
Practical causality handling for synchronous languages.
In Proc. Design, Automation and Test in Europe Conference (DATE '19).

Scheduling Policies

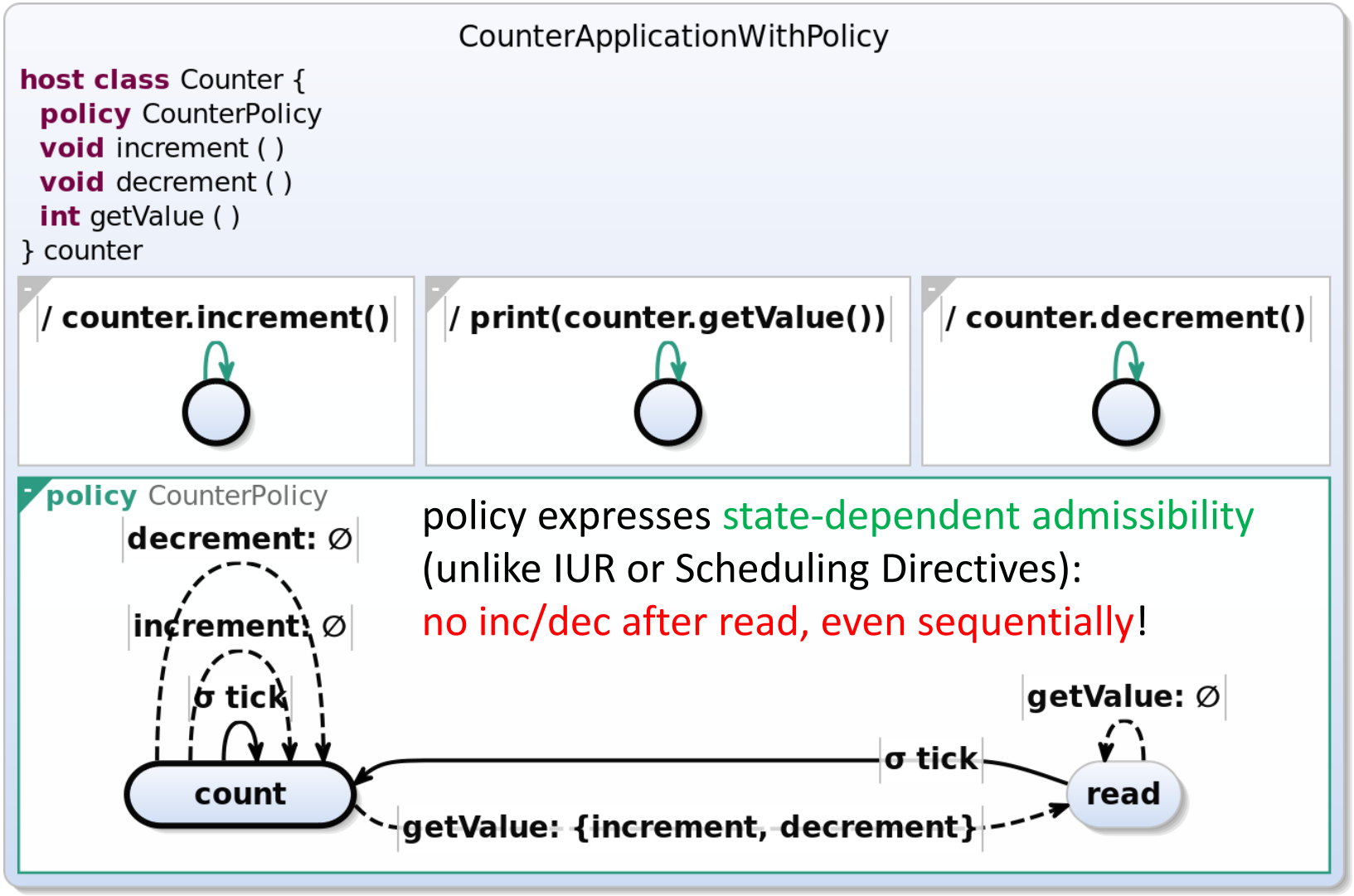
J. Aguado, M. Mendler, M. Pouzet, P. S. Roop, and R. von Hanxleden.
Deterministic concurrency: A clock-synchronised shared memory approach.
In 27th European Symposium on Programming (ESOP'18).

Object-Specific Scheduling Regimes



S. Smyth, A. Schulz-Rosengarten, and R. von Hanxleden.
Practical causality handling for synchronous languages.
In Proc. Design, Automation and Test in Europe Conference (DATE '19).

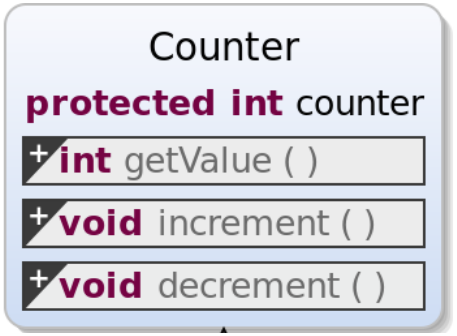
Object-Specific Scheduling Regimes

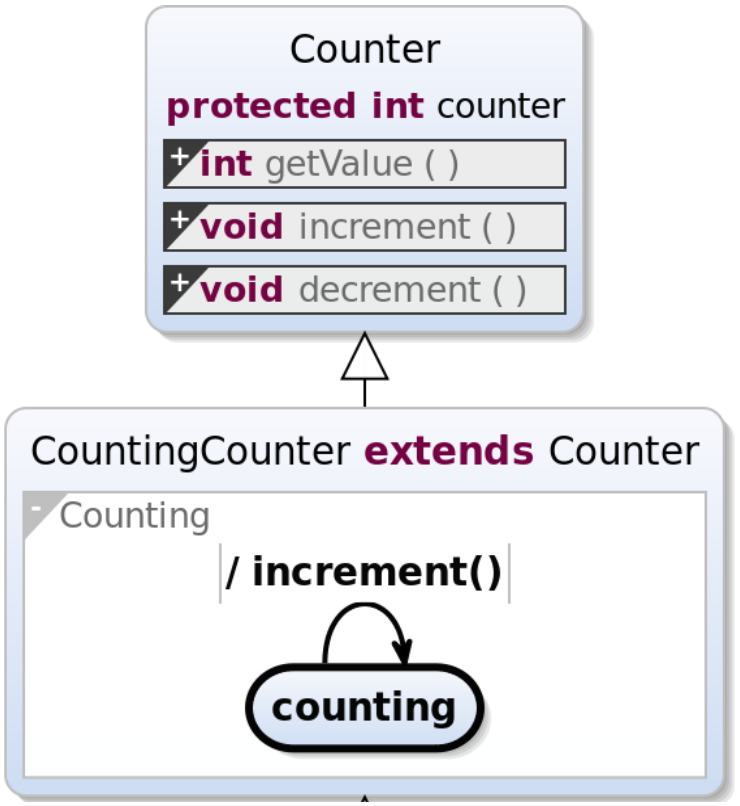


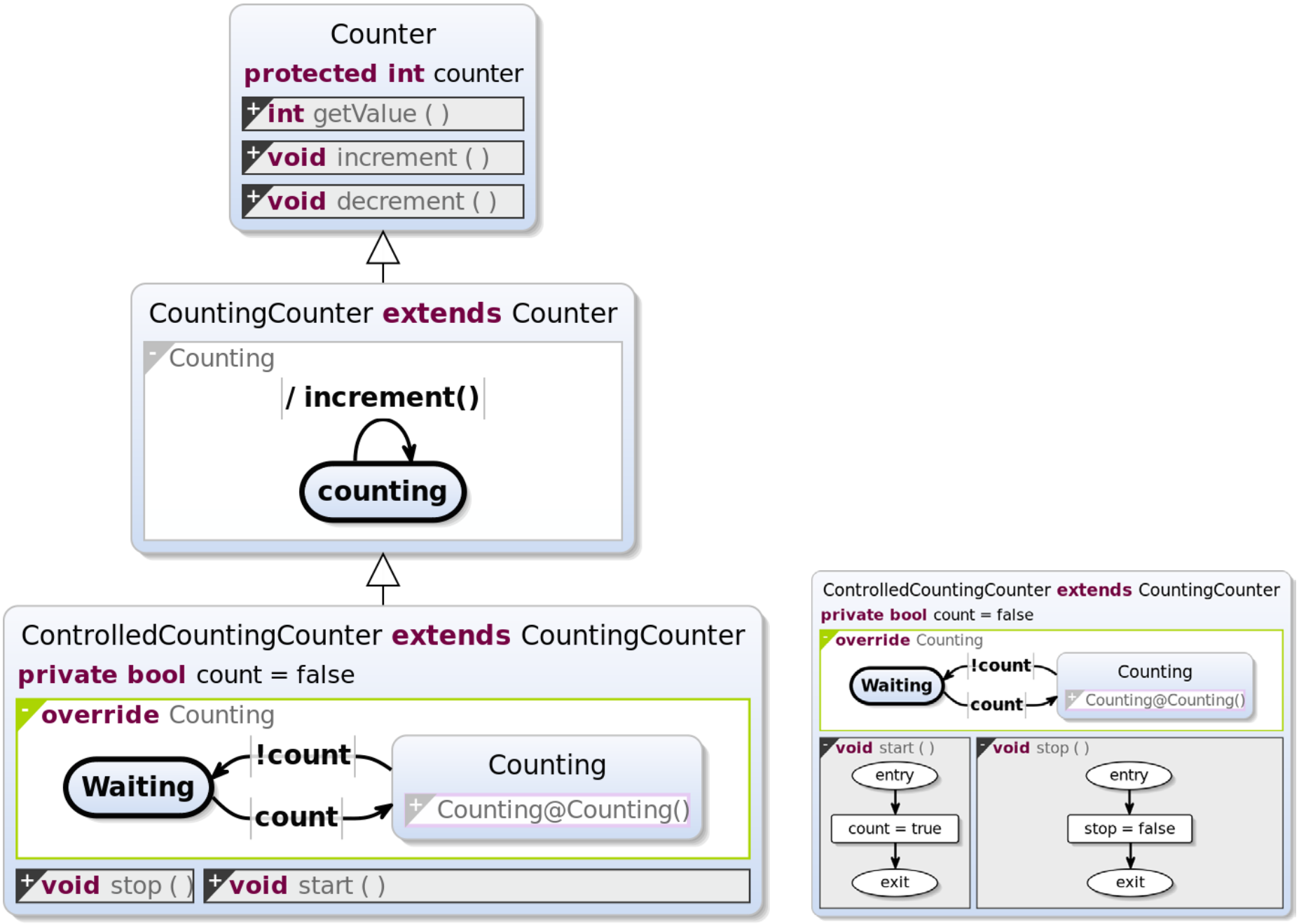
J. Aguado, M. Mendler, M. Pouzet, P. S. Roop, and R. von Hanxleden.
Deterministic concurrency: A clock-synchronised shared memory approach.
In 27th European Symposium on Programming (ESOP'18).

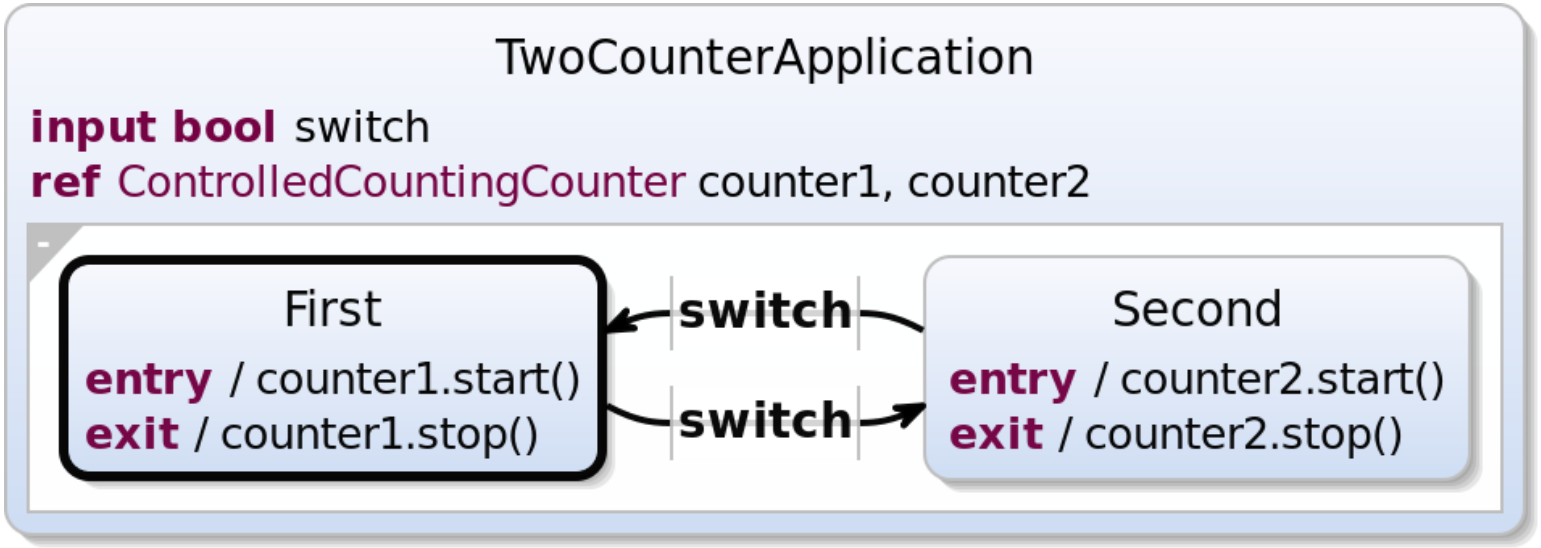
Object-Oriented Statecharts Design in SCCharts

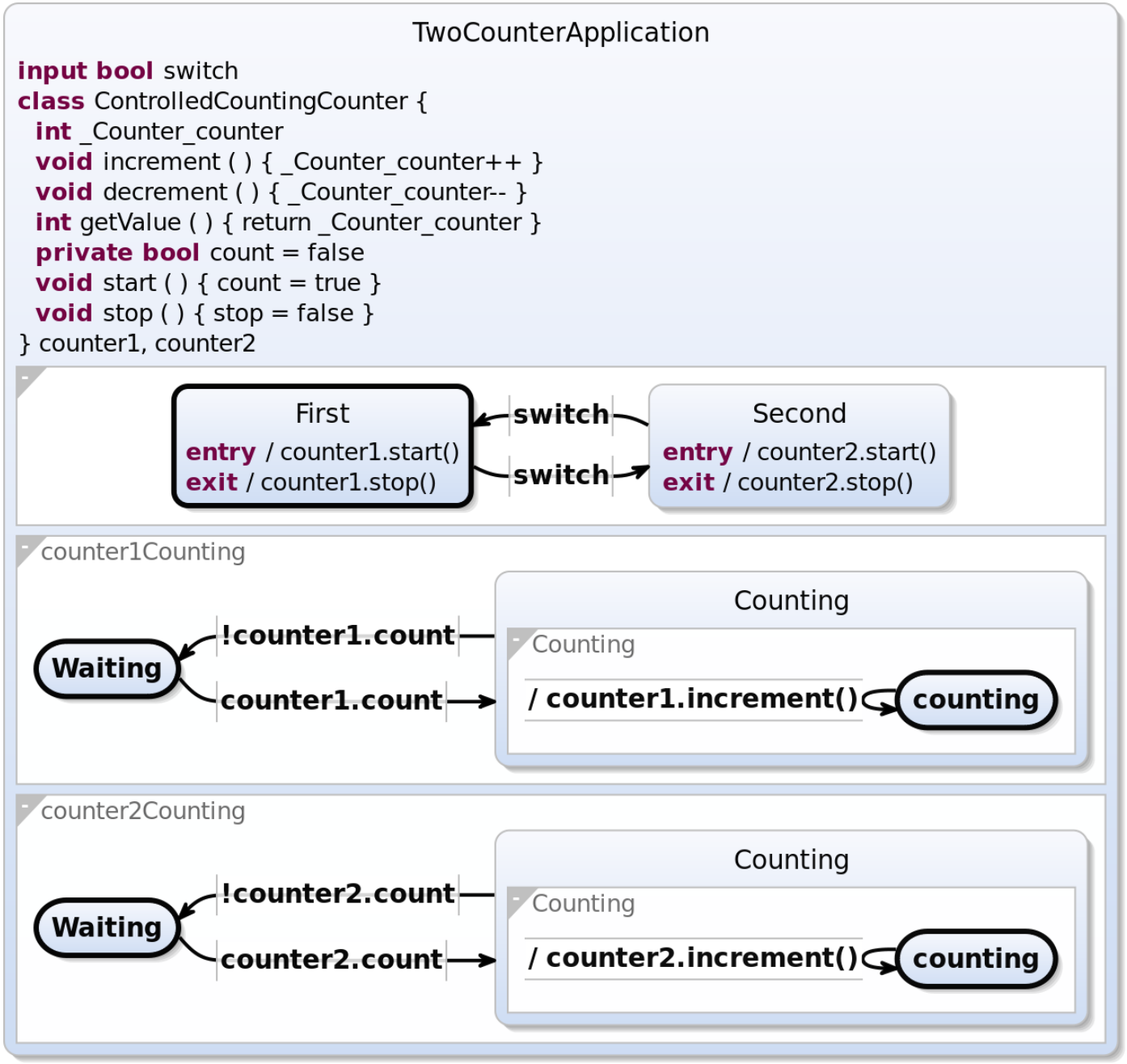
Inheritance


Counter
protected int counter
+int getValue ( )
+void increment ( )
+void decrement ( )


Counter
protected int counter
+int getValue ( )
+void increment ( )
+void decrement ( )
CountingCounter extends Counter
-Counting
/ increment()
counting


Counter
protected int counter
+int getValue ( )
+void increment ( )
+void decrement ( )
CountingCounter extends Counter
-Counting
/ increment()
counting
ControlledCountingCounter extends CountingCounter
private bool count = false
-override Counting
Waiting
!count
count
Counting
+Counting@Counting()
+void stop ( )
+void start ( )
ControlledCountingCounter extends CountingCounter
private bool count = false
-override Counting
Waiting
!count
count
Counting
+Counting@Counting()
-void start ( )
entry
count = true
exit
-void stop ( )
entry
stop = false
exit


TwoCounterApplication
input bool switch
ref ControlledCountingCounter counter1, counter2
First
entry / counter1.start()
exit / counter1.stop()
switch
switch
Second
entry / counter2.start()
exit / counter2.stop()


TwoCounterApplication
input bool switch
class ControlledCountingCounter {
int _Counter_counter
void increment ( ) { _Counter_counter++ }
void decrement ( ) { _Counter_counter-- }
int getValue ( ) { return _Counter_counter }
private bool count = false
void start ( ) { count = true }
void stop ( ) { stop = false }
} counter1, counter2
First
entry / counter1.start()
exit / counter1.stop()
switch
switch
Second
entry / counter2.start()
exit / counter2.stop()
counter1Counting
Waiting
!counter1.count
counter1.count
Counting
Counting
/ counter1.increment()
counting
counter2Counting
Waiting
!counter2.count
counter2.count
Counting
Counting
/ counter2.increment()
counting

Summary

Conservative Extension

Static Handling of OO Features

Abstraction and Structure

Hostcode Integration

Determinism

The End