A Case Study of AOP and OOP Applied to Digital Hardware Design

Tiago R. Mück and Antônio A. Fröhlich

Software/Hardware Integration Lab
Federal University of Santa Catarina (UFSC)

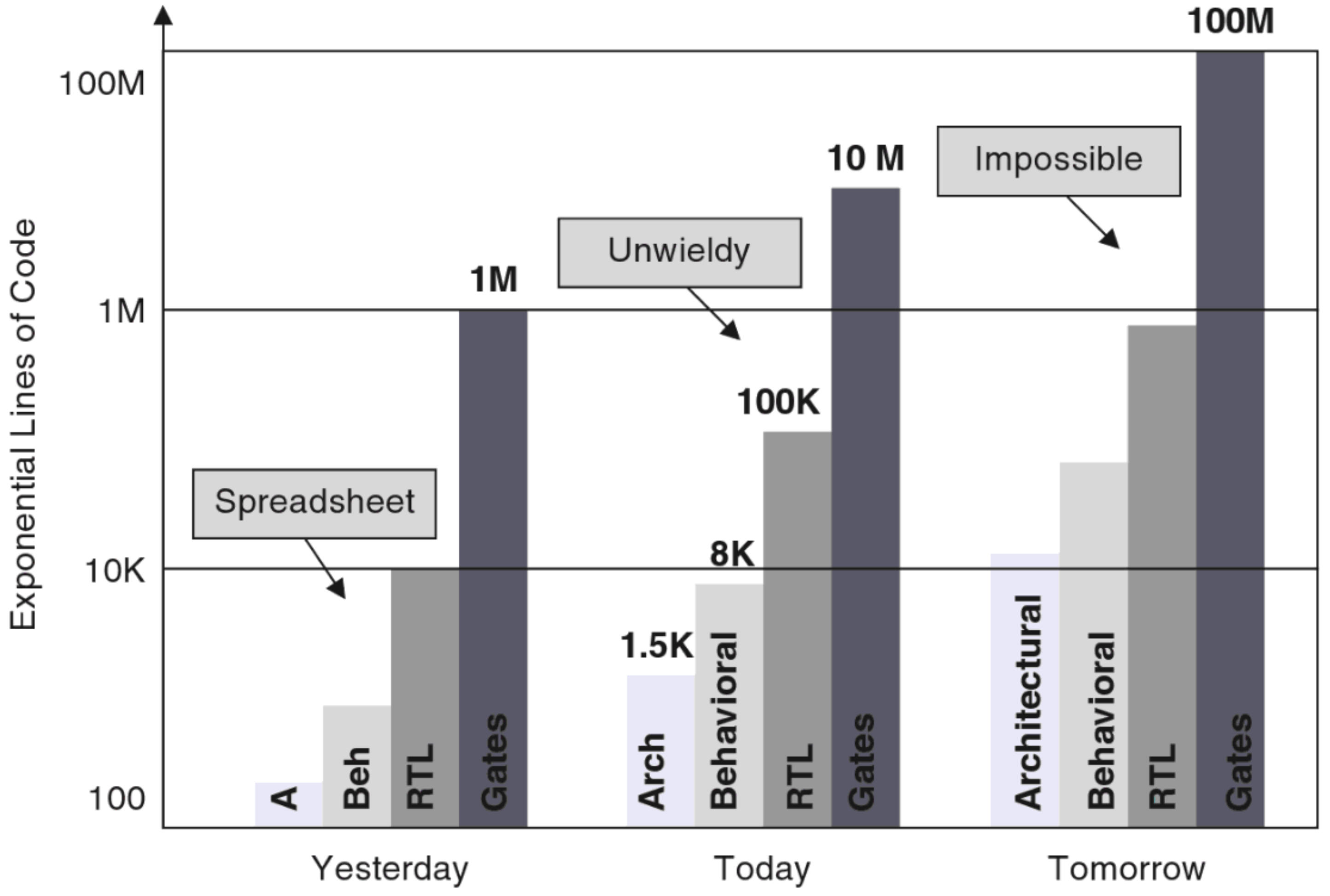
Michael Gernoth and Wolfgang Schröder-Preikschat

Department of Computer Science 4
Friedrich-Alexander University Erlangen-Nuremberg

Introduction

- System complexity is quickly increasing

Code complexity for SoCs designs in various levels of abstraction

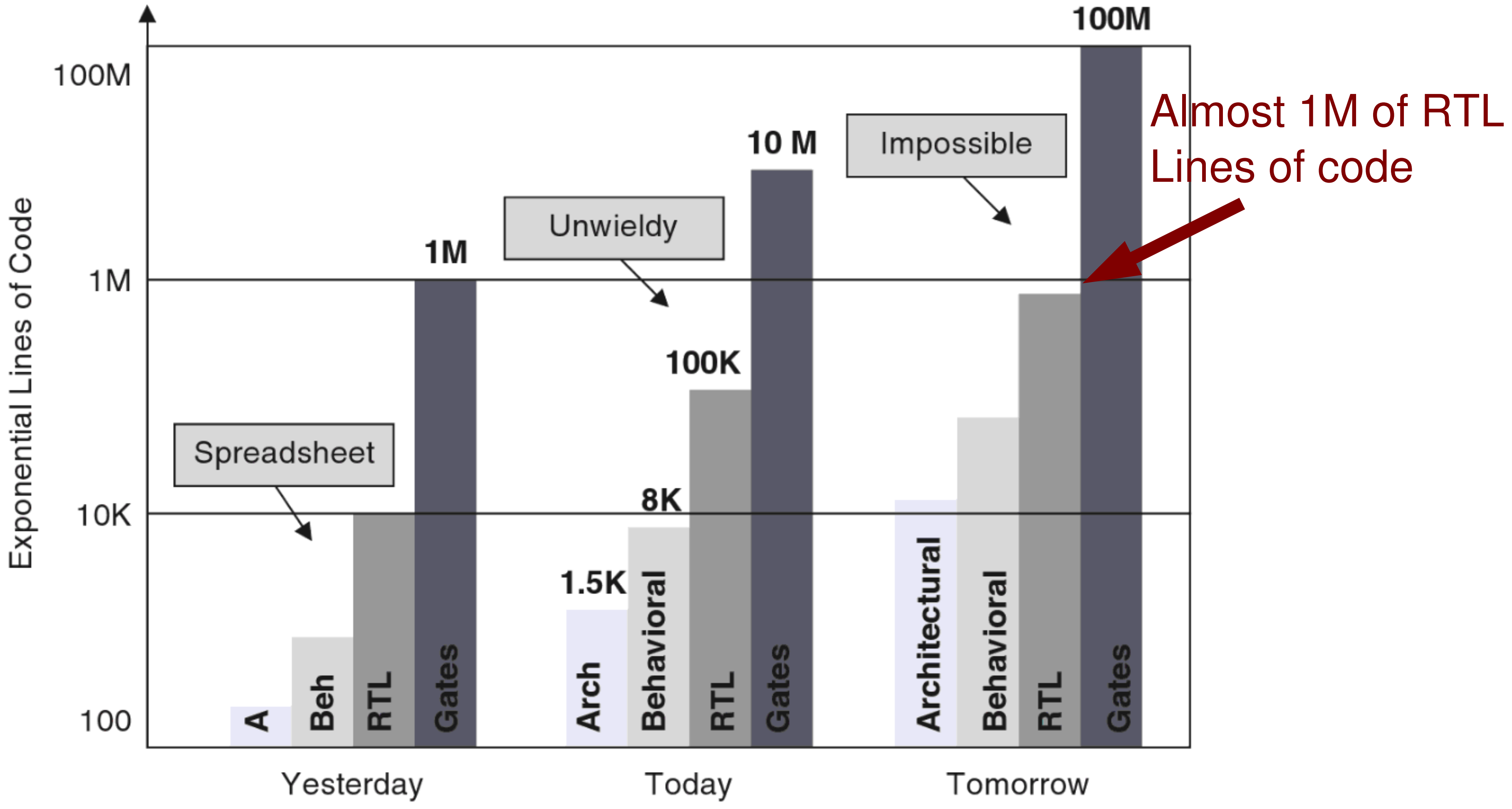


(Black & Donovan, 2009)

Introduction

- System complexity is quickly increasing

Code complexity for SoCs designs in various levels of abstraction

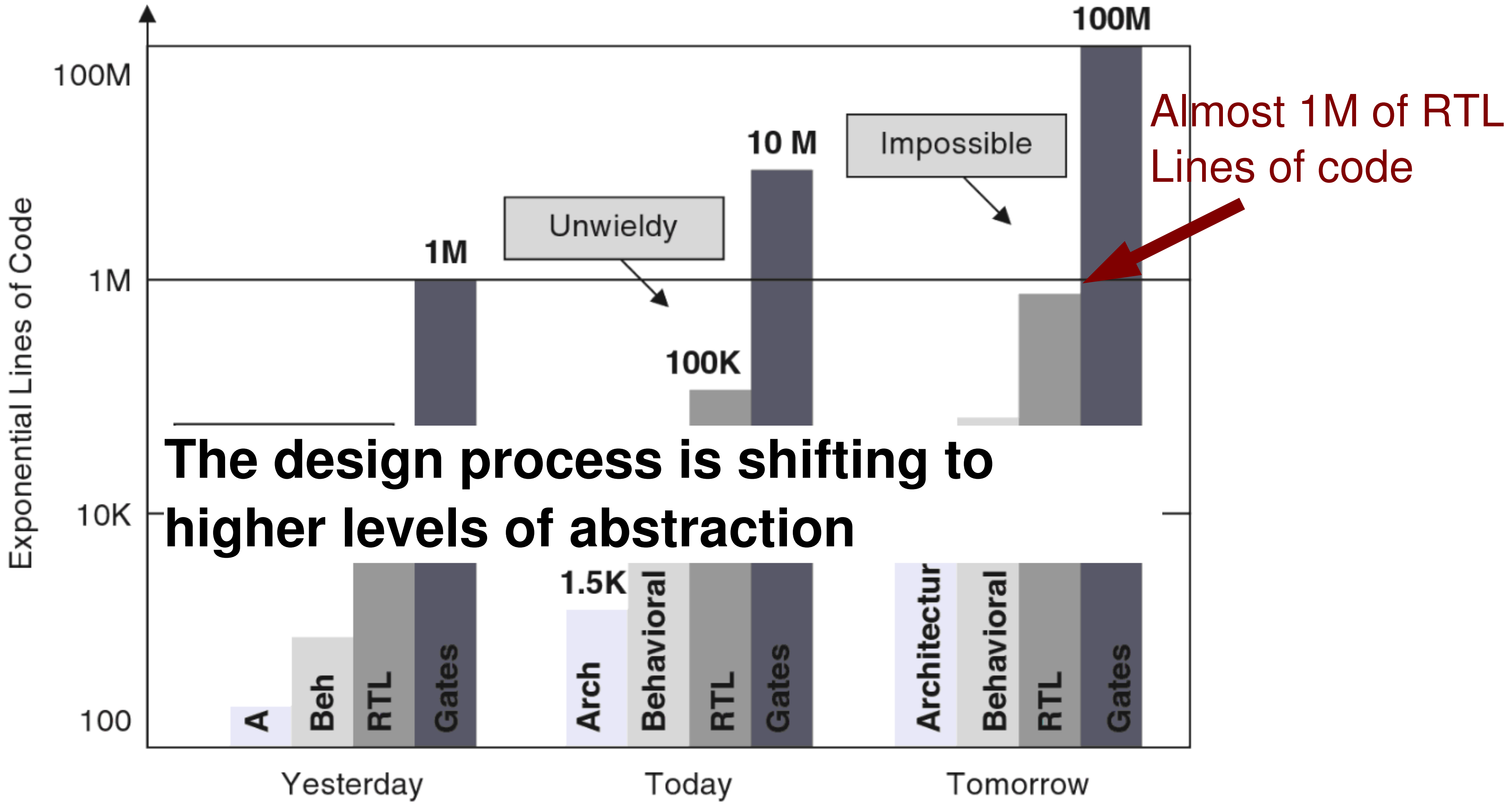


(Black & Donovan, 2009)

Introduction

- System complexity is quickly increasing

Code complexity for SoCs designs in various levels of abstraction



(Black & Donovan, 2009)

Evolution of abstraction levels

Evolution of abstraction levels

- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

Evolution of abstraction levels

- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

$\Rightarrow$

```
For (int i = 0; i < 10; ++i)
     do_something(i);
```

Procedural language

Evolution of abstraction levels

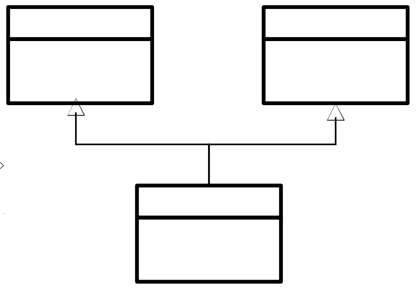
- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

```
For (int i = 0; i < 10; ++i)
     do_something(i);
```

Procedural language



OOP, AOP, UML, etc

Evolution of abstraction levels

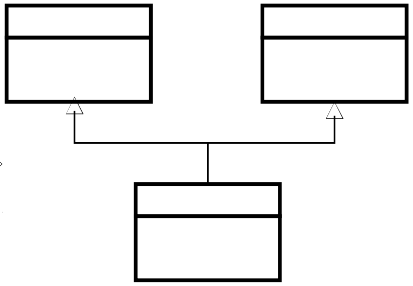
- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

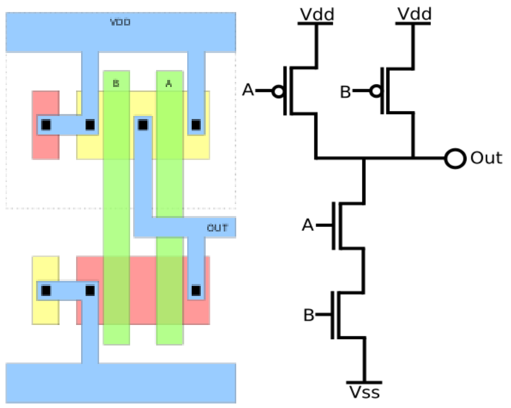
```
For (int i = 0; i < 10; ++i)
    do_something(i);
```

Procedural language



OOP, AOP, UML, etc

- For hardware:



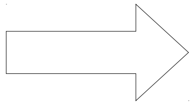
Electric level

Evolution of abstraction levels

- For software:

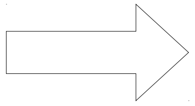
```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

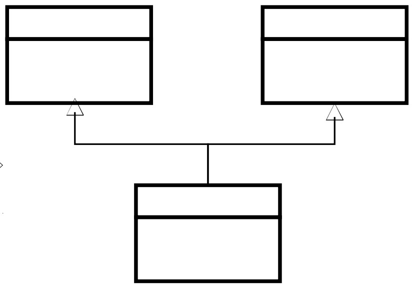
Machine language



```
For (int i = 0; i < 10; ++i)
    do_something(i);
```

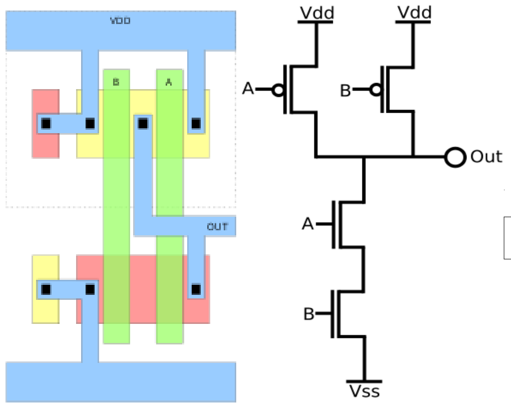
Procedural language



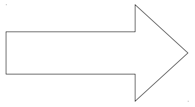


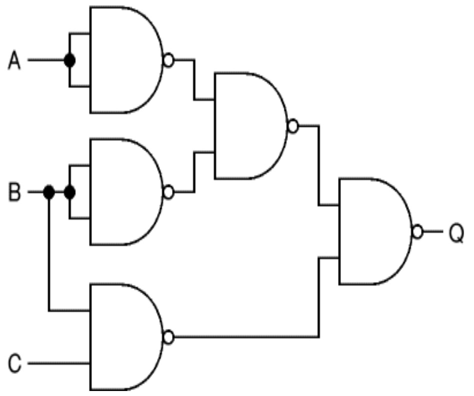
OOP, AOP, UML, etc

- For hardware:



Electric level





Gate level

Evolution of abstraction levels

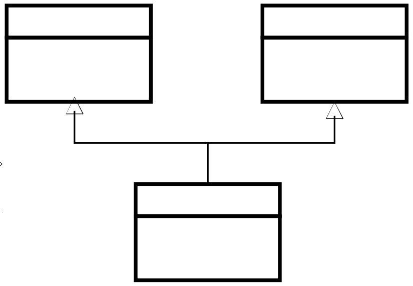
- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

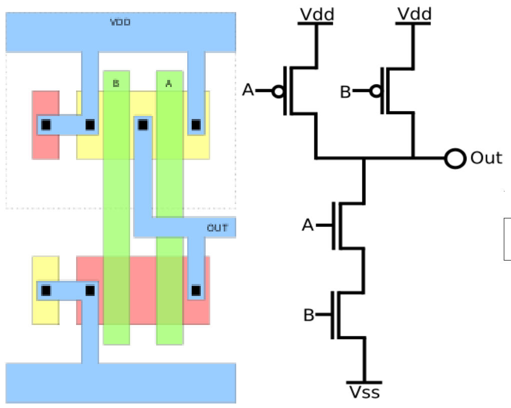
```
For (int i = 0; i < 10; ++i)
    do_something(i);
```

Procedural language

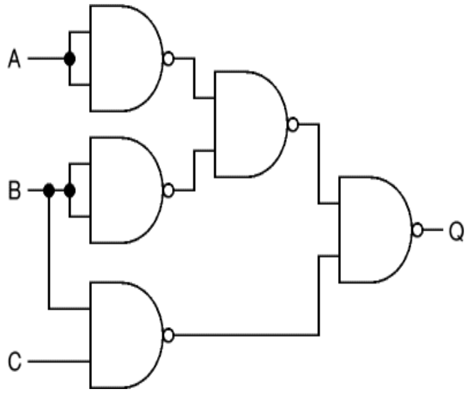


OOP, AOP, UML, etc

- For hardware:



Electric level



Gate level

```
always @(sel or a or b)
begin
  if (sel == 1)
    f = a;
  else
    f = b;
end
```

Register transfer level

Evolution of abstraction levels

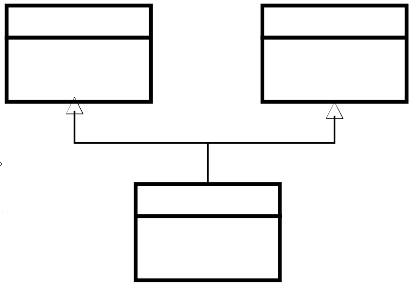
- For software:

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

```
For (int i = 0; i < 10; ++i)
    do_something(i);
```

Procedural language

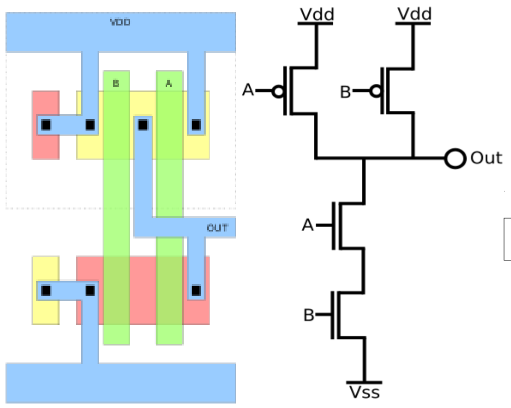


OOP, AOP, UML, etc

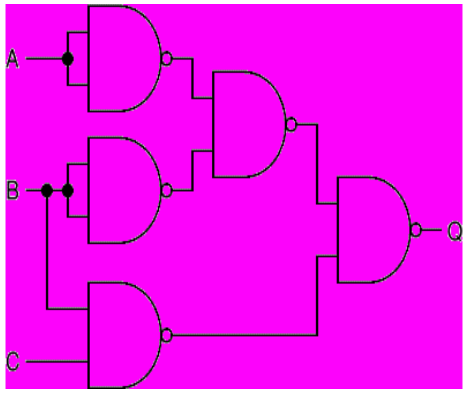
Behavioral synthesis

High-level modeling

- For hardware:



Electric level



Gate level

```
always @(sel or a or b)
begin
  if (sel == 1)
    f = a;
  else
    f = b;
end
```

Register transfer level

Evolution of abstraction levels

- For software:

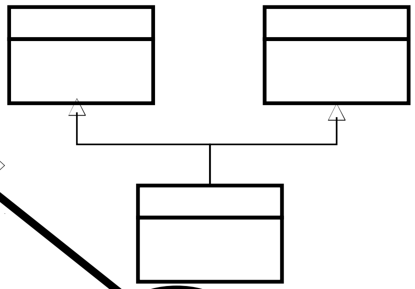
Aspect-oriented programming

```
ADDI $SP, $SP, -8
SW   $A0, 4($SP)
SW   $RA, 0($SP)
```

Machine language

```
For (int i = 0; i < 10; ++i)
    do_something(i);
```

Procedural language

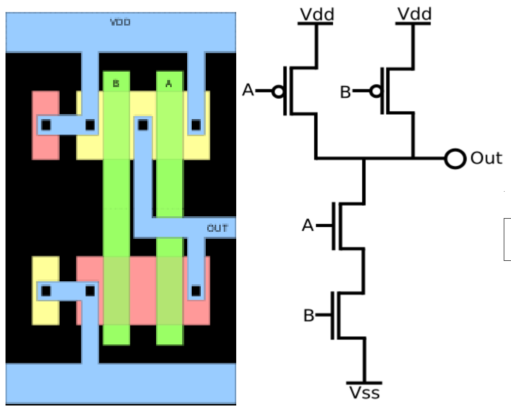


OOP, AOP, UML, etc

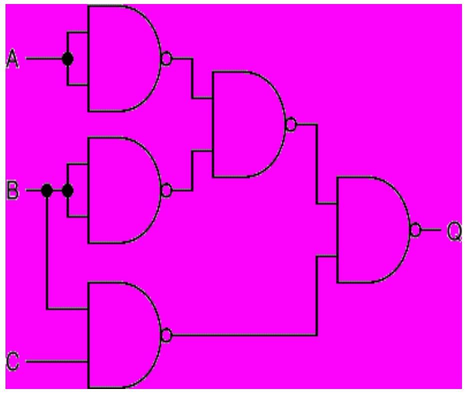
- For hardware:

Behavioral synthesis

High-level modeling



Electric level



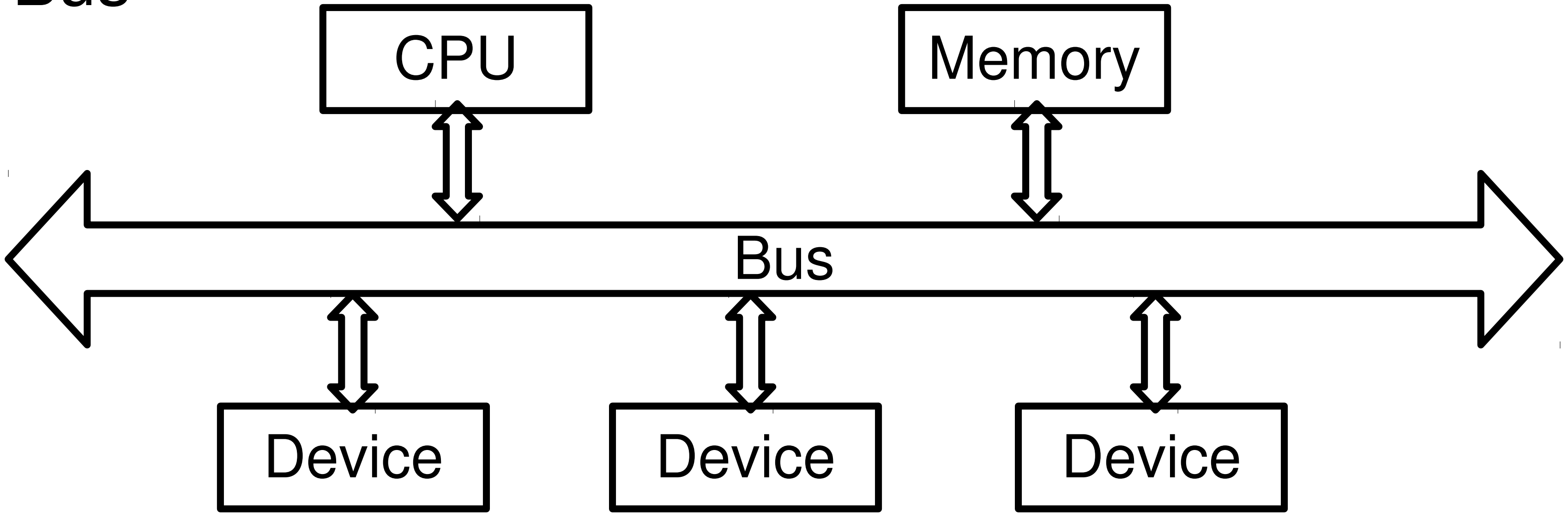
Gate level

```
always @(sel or a or b)
begin
  if (sel == 1)
    f = a;
  else
    f = b;
end
```

Register transfer level

Aspect-oriented programing

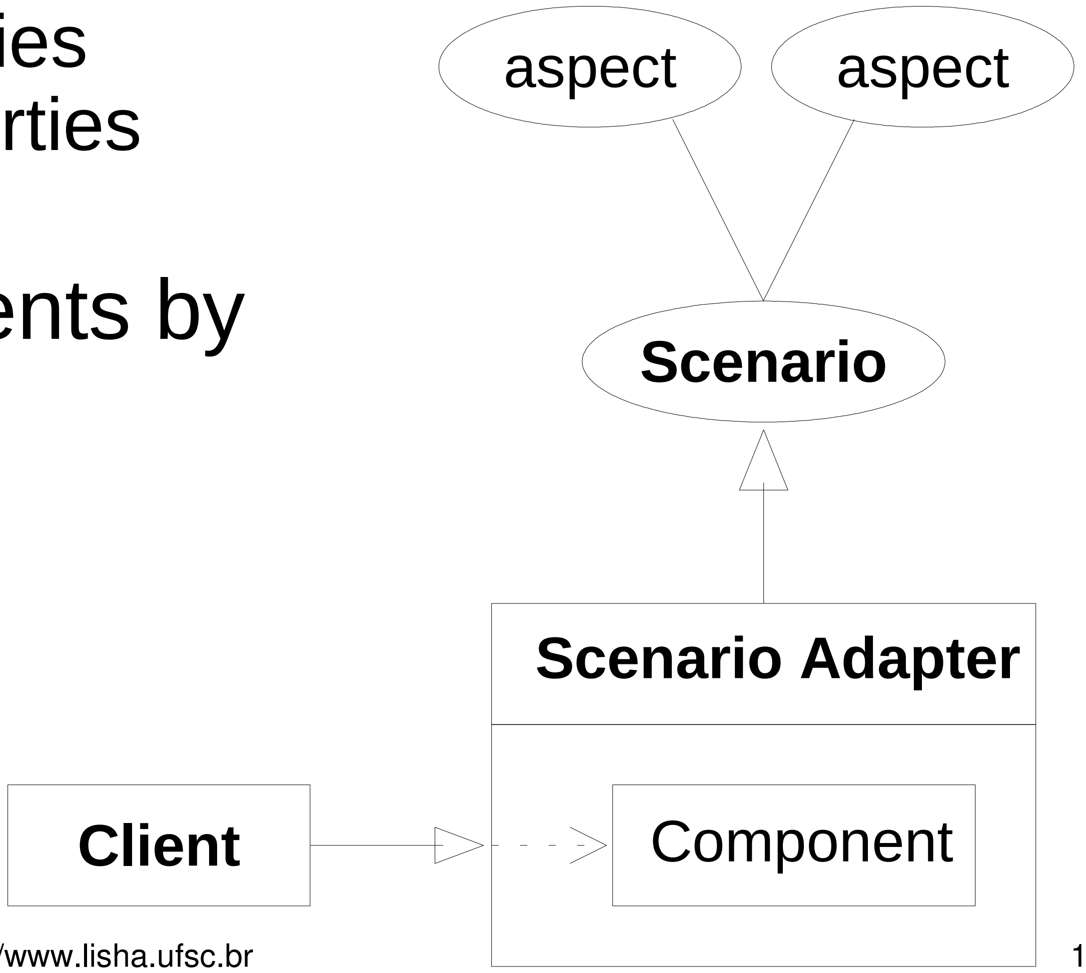
- Aspect-oriented programming (AOP) extends OOP to deal with **crosscutting concerns**
- Crosscutting in HW:
  - Bus



  - Other examples
    - Power management, HW debugging, etc ...

Dealing with crosscutting concerns

- ADESD's way:
  - Model as aspects the properties that transcend the scope of single abstractions
    - Scenario dependencies
    - Non-functional properties
  - Applied to components by
    - Weavers
    - Scenario adapters



Proposal

- Is it possible to apply scenario adapters to hardware design ?

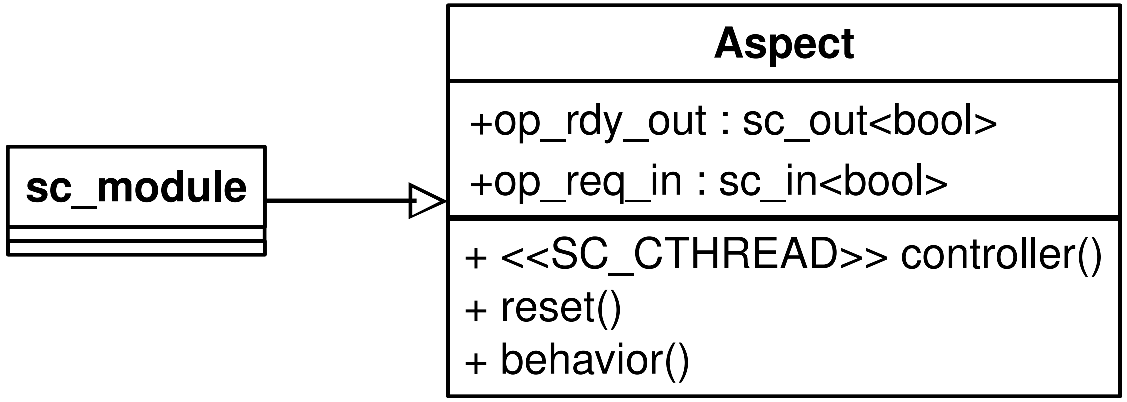
- 1º step: define the HDL
  - SystemC
    - A library and simulation environment that turns C++ into a HDL
    - Supports RTL and higher levels of abstractions

- 2º step: how a scenario adapter could work in hardware ?

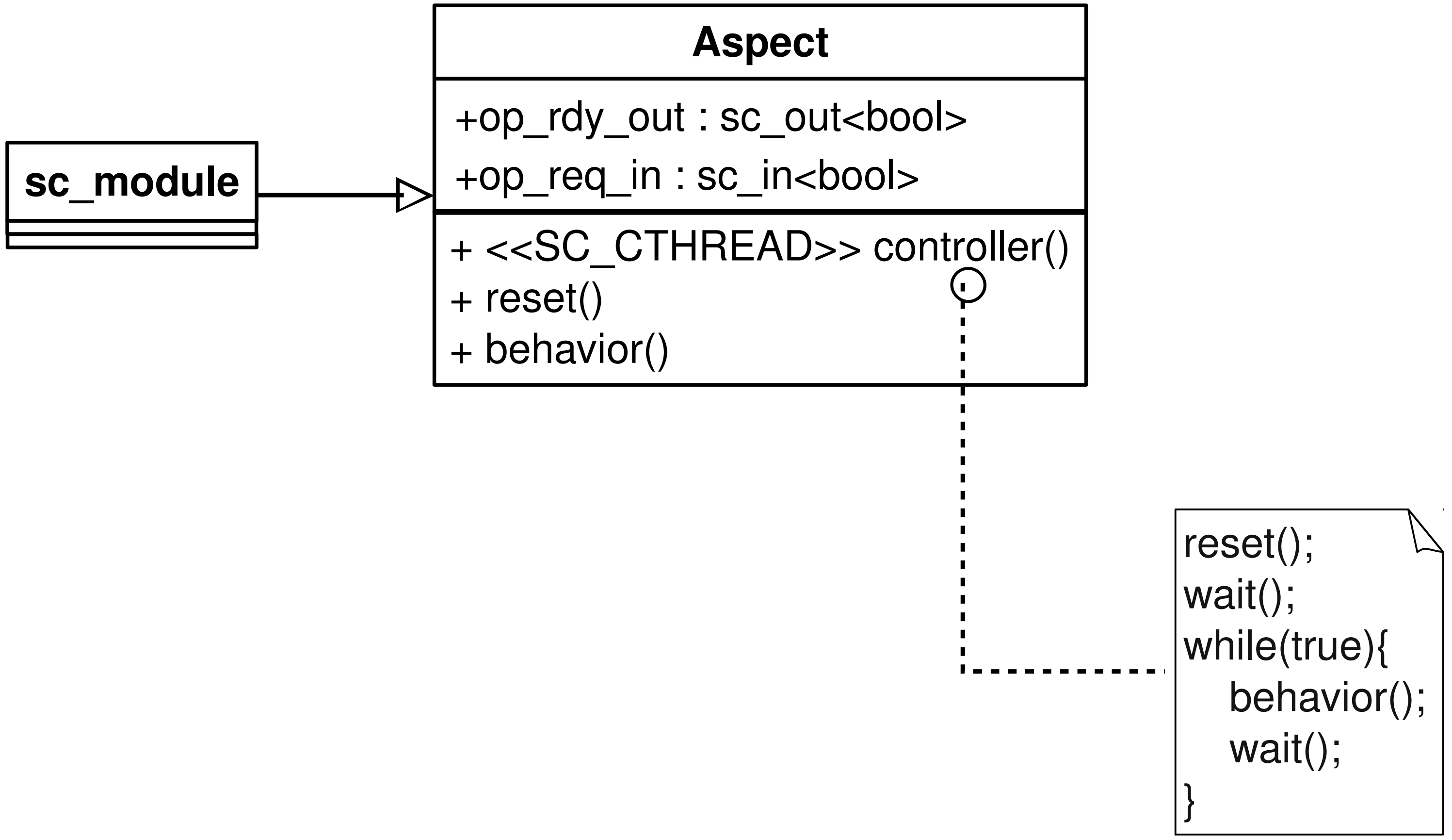
Differences between HW and SW

- Software
  - Intrinsically sequential
  - Timeless operations
  - Interfaces based on invocation of methods and functions
- Hardware (RTL)
  - Intrinsically parallel
  - Timed operations
  - Interfaces based on input/output signals
- Scenario adapters redesigned to match these differences

Aspect definition

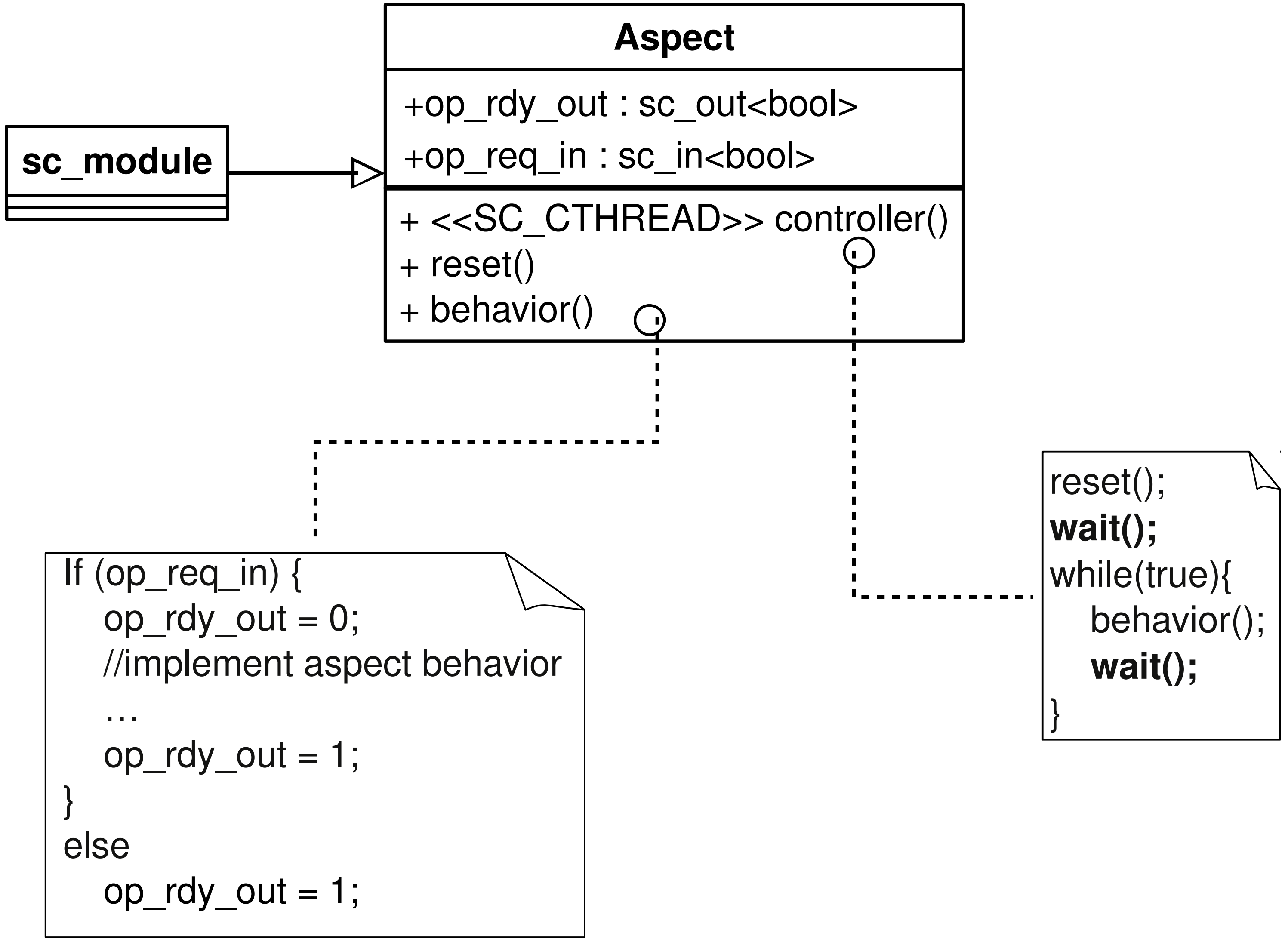


Aspect definition



```
reset();
wait();
while(true){
   behavior();
   wait();
}
```

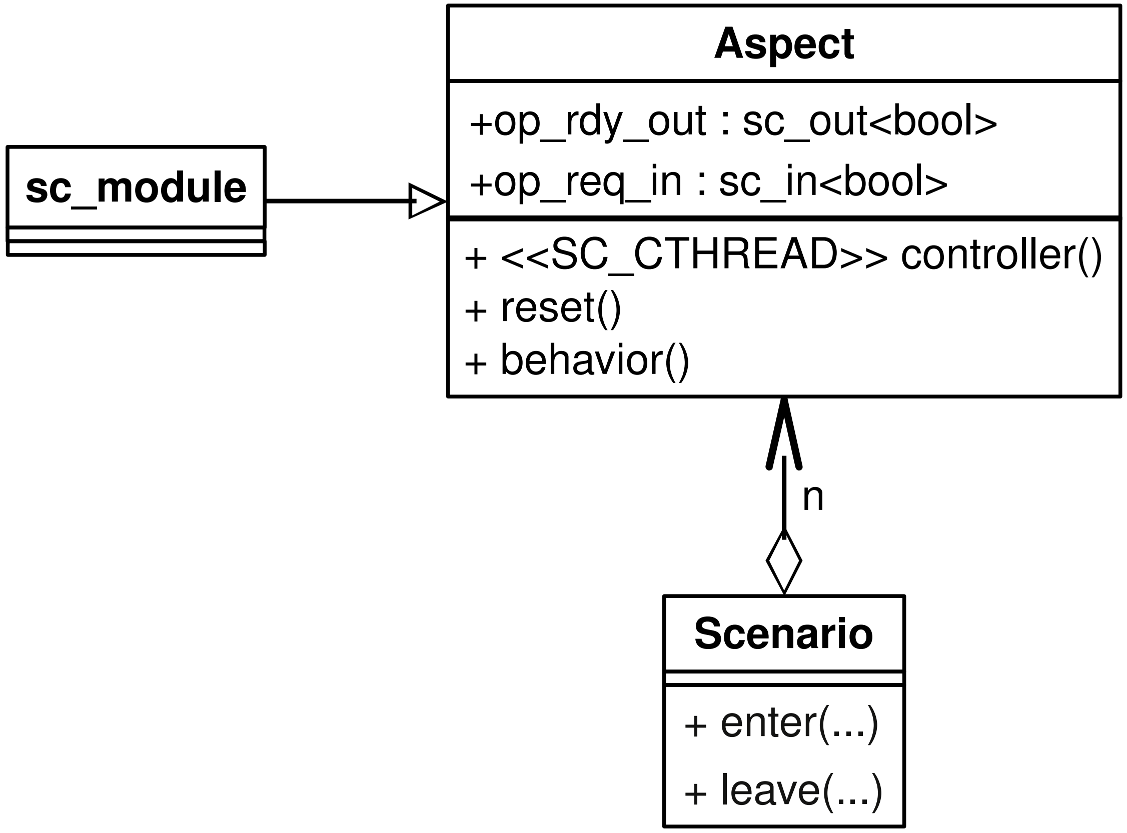
Aspect definition



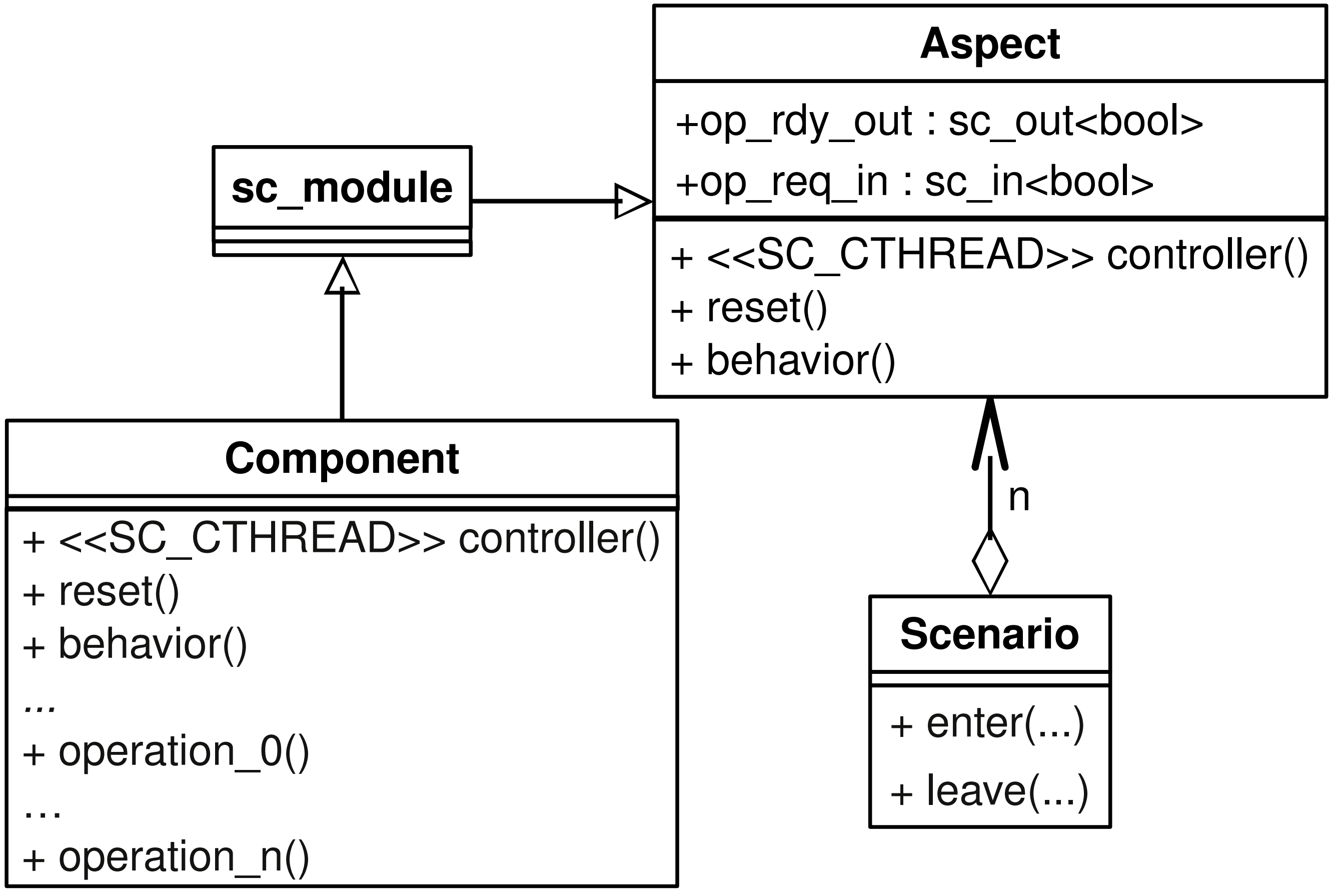
```
If (op_req_in) {
    op_rdy_out = 0;
    //implement aspect behavior
    …
    op_rdy_out = 1;
}
else
    op_rdy_out = 1;
```

```
reset();
wait();
while(true){
    behavior();
    wait();
}
```

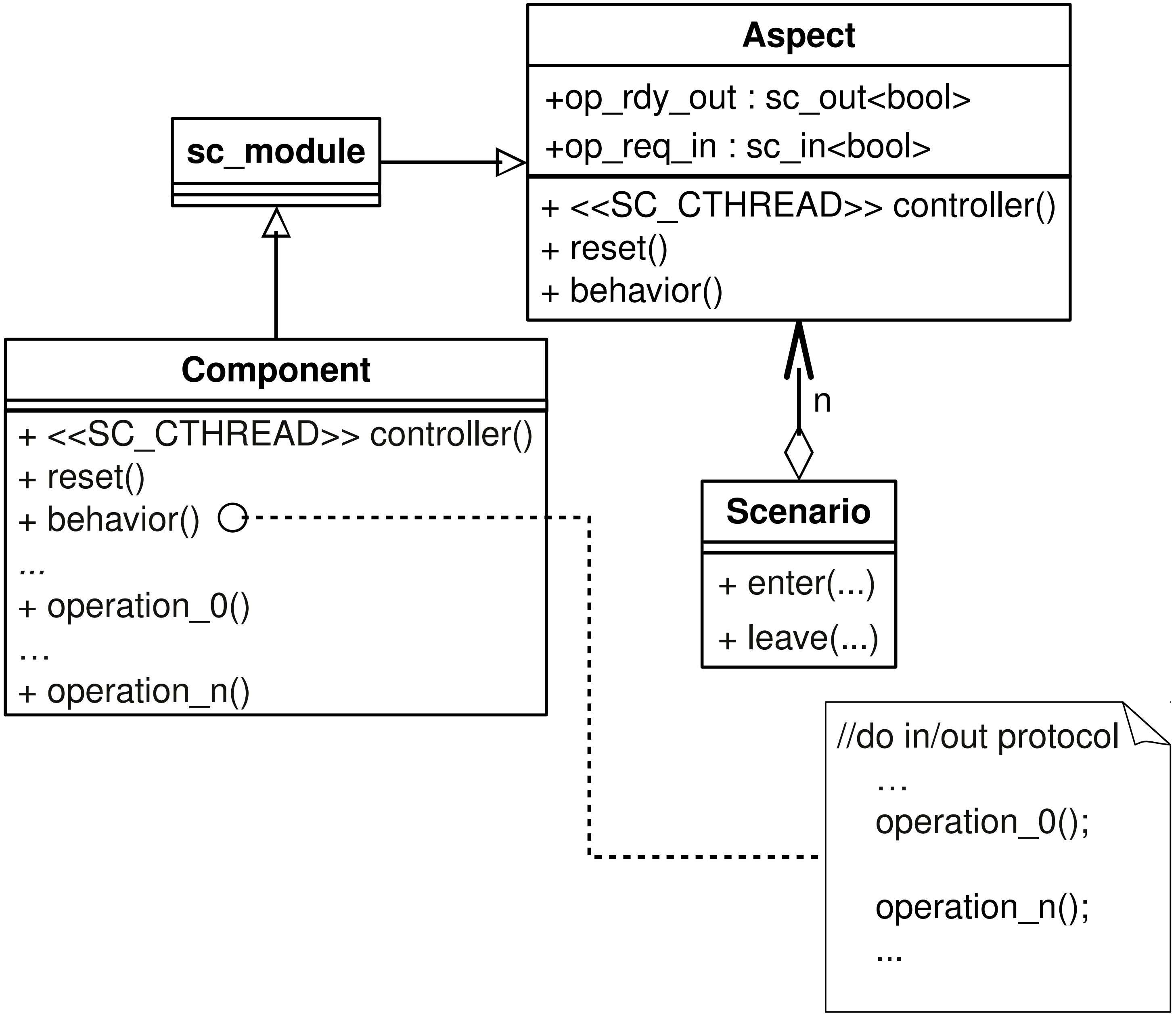
Scenario definition



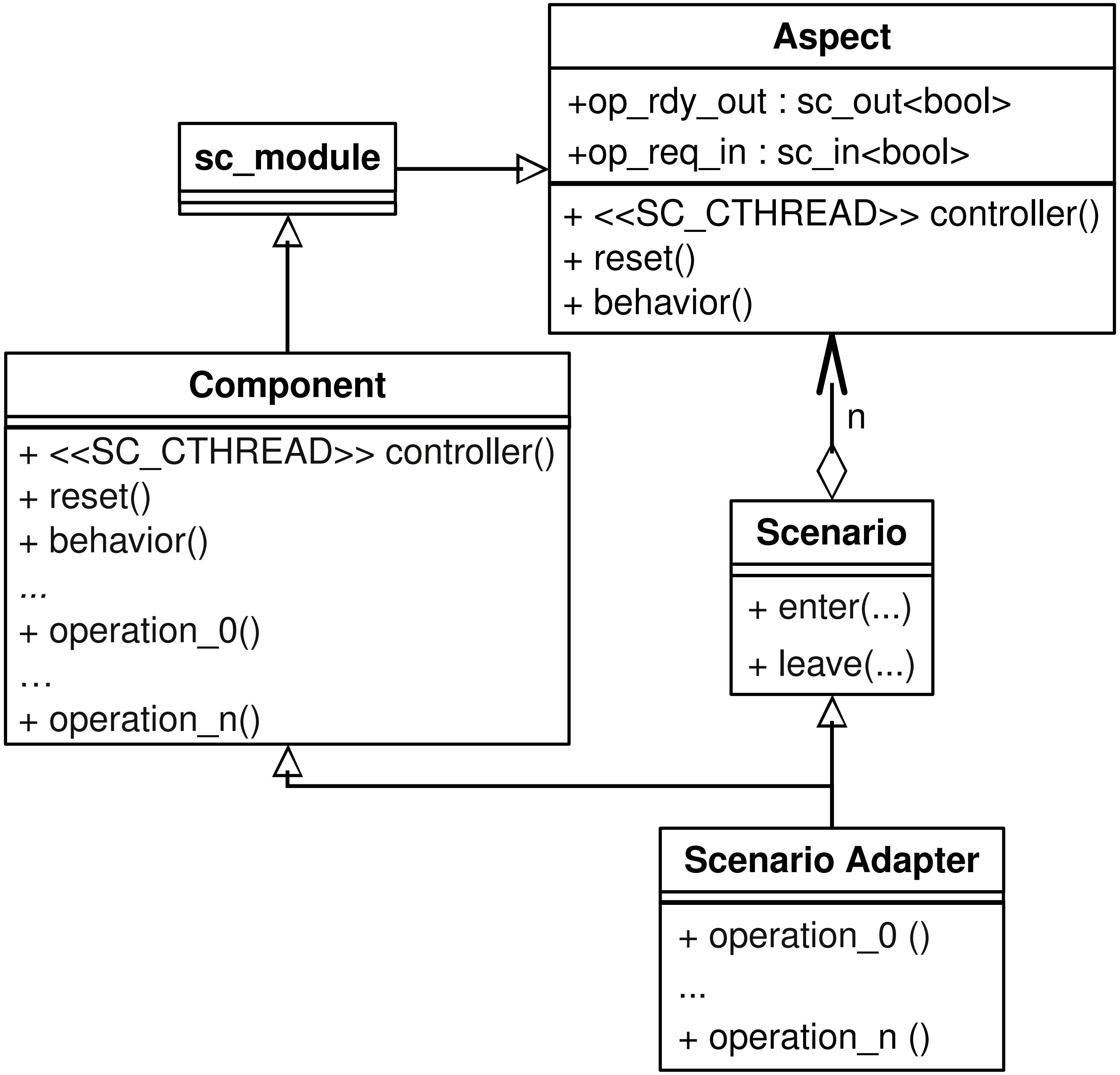
Component definition



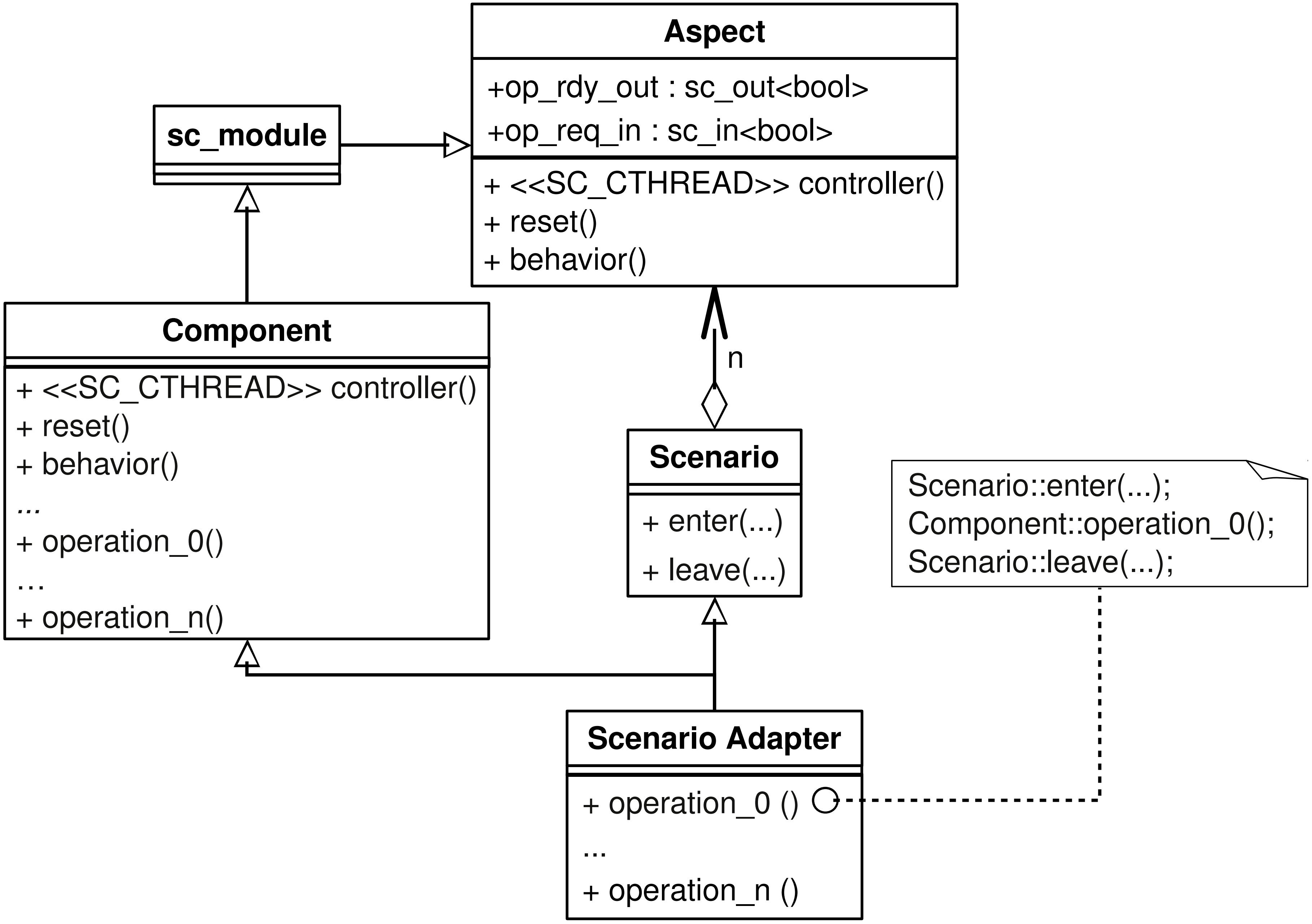
Component definition



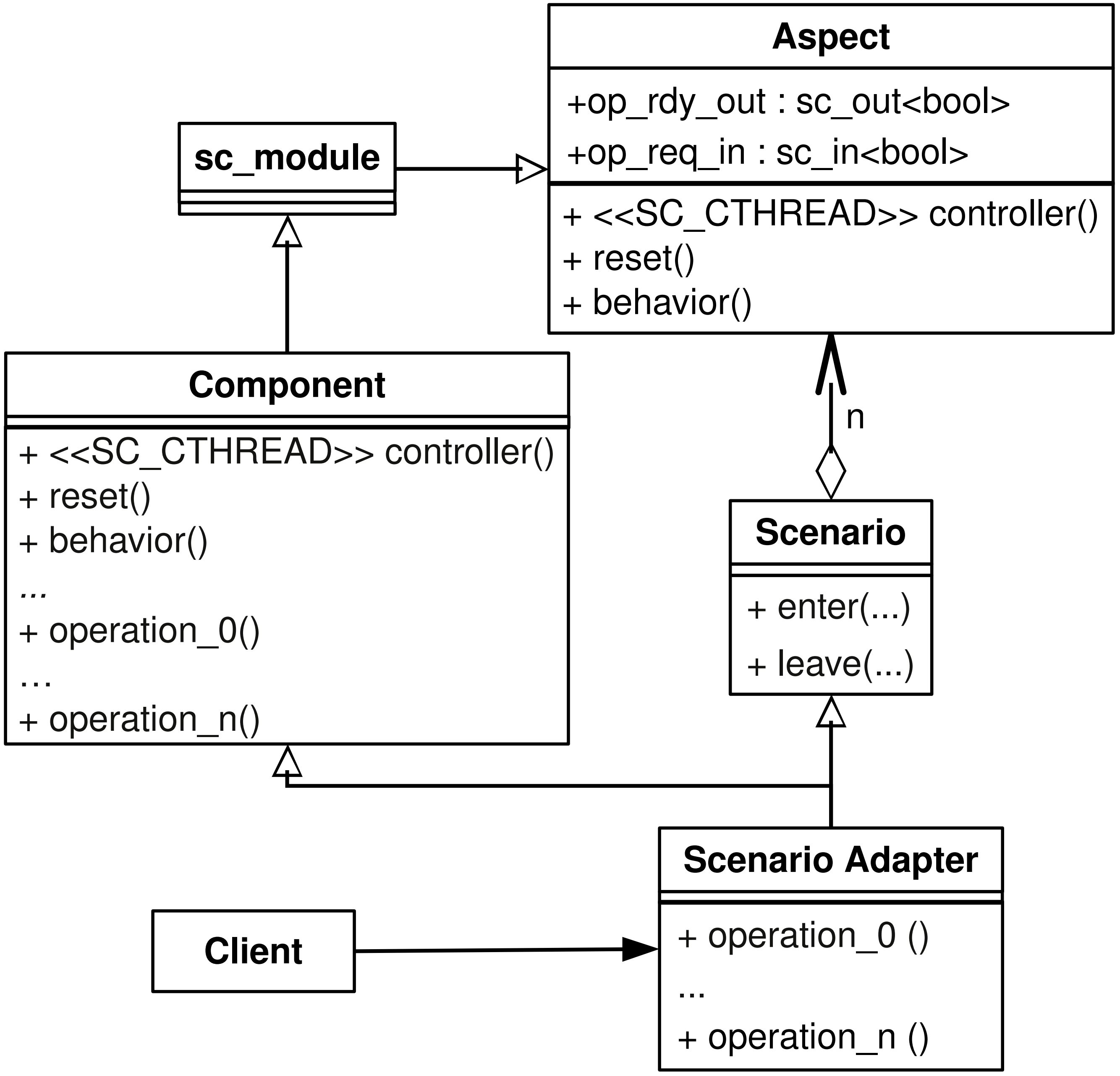
Scenario adapter definition



Scenario adapter definition

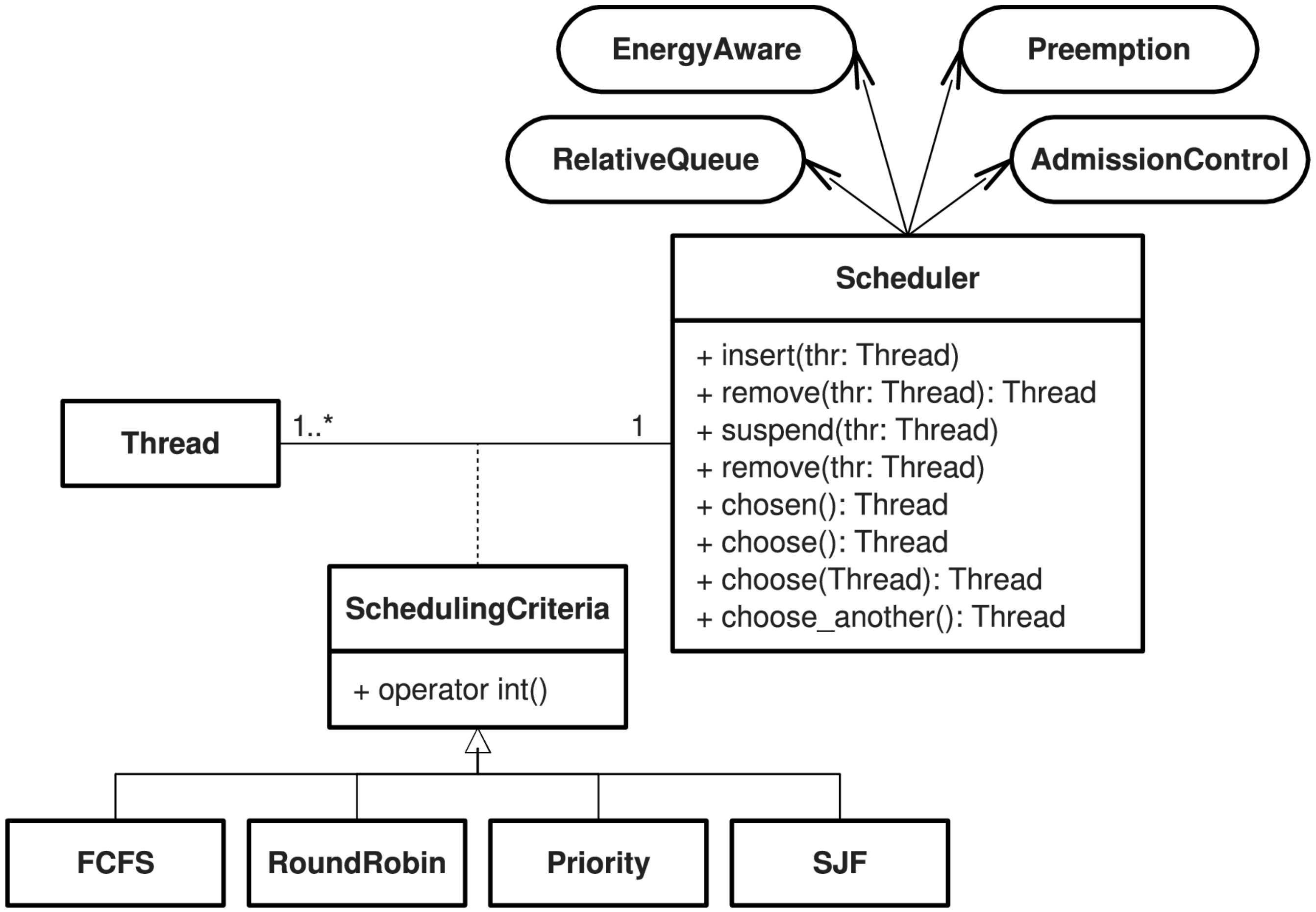


Scenario adapter definition



Case study

- Scenario adapters were applied to a HW implementation of an operating system scheduler



Why a HW scheduler ?

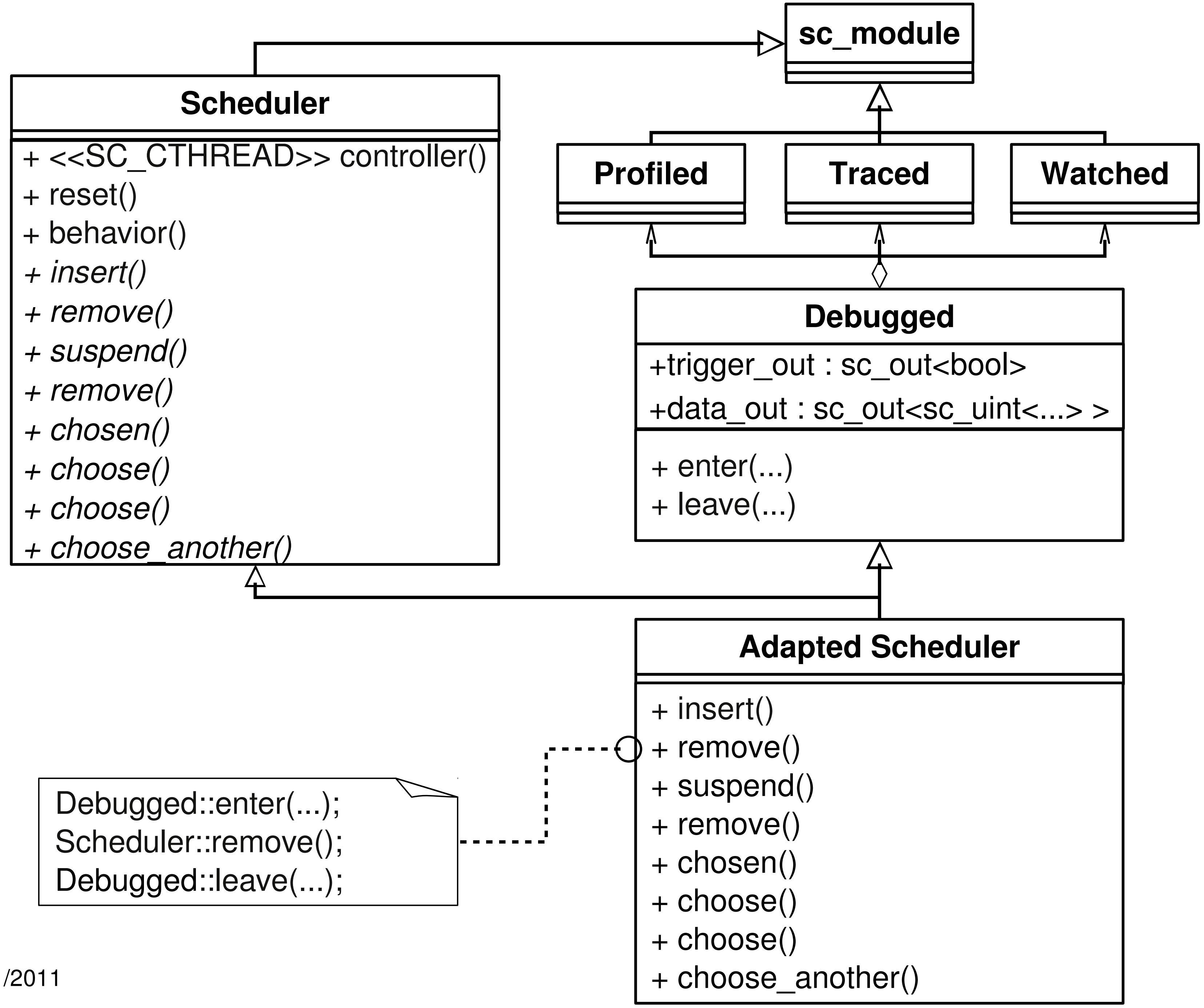
- Its complex behavior
  - Synchronous operations
    - Upon request by another component
  - Asynchronous operations
    - By preempting another component
- Also an interesting approach for real-time systems

Scenarios and aspects identified

- Design for testability
  - A common practice in digital hardware design
  - Real-time debugging

- Case study:
  - A **debugged scenario** targeting the integration with JTAG scan chains
  - Aspects:
    - **Watched:** monitor state changes in the component
    - **Traced:** signalize each execution of an operation
    - **Profiled:** counts the number of clock cycles used for each operation

Scenario-adapted scheduler



```
Debugged::enter(...);
Scheduler::remove();
Debugged::leave(...);
```

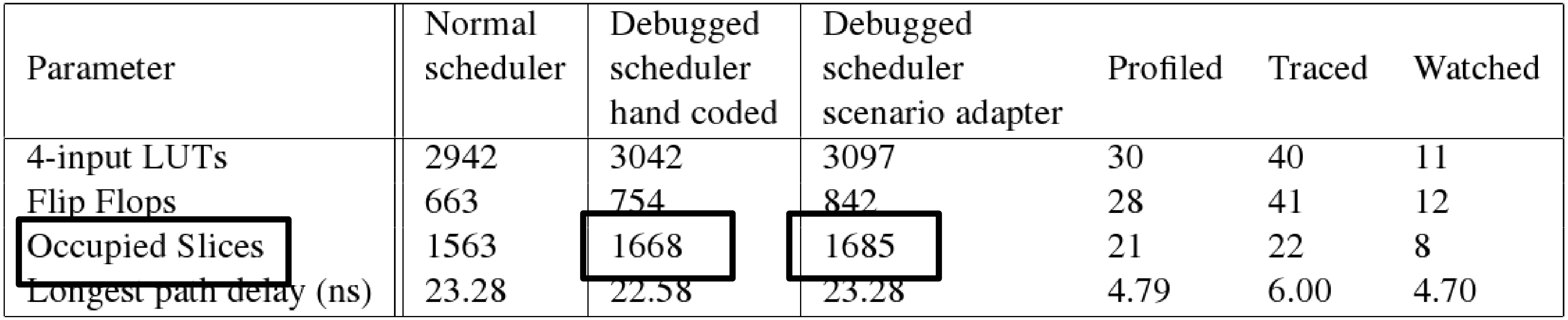
Results

- Circuits synthesized targeting a Xilinx FPGA

Parameter	Normal scheduler	Debugged scheduler hand coded	Debugged scheduler scenario adapter	Profiled	Traced	Watched
4-input LUTs	2942	3042	3097	30	40	11
Flip Flops	663	754	842	28	41	12
Occupied Slices	1563	1668	1685	21	22	8
Longest path delay (ns)	23.28	22.58	23.28	4.79	6.00	4.70

Results

- Circuits synthesized targeting a Xilinx FPGA

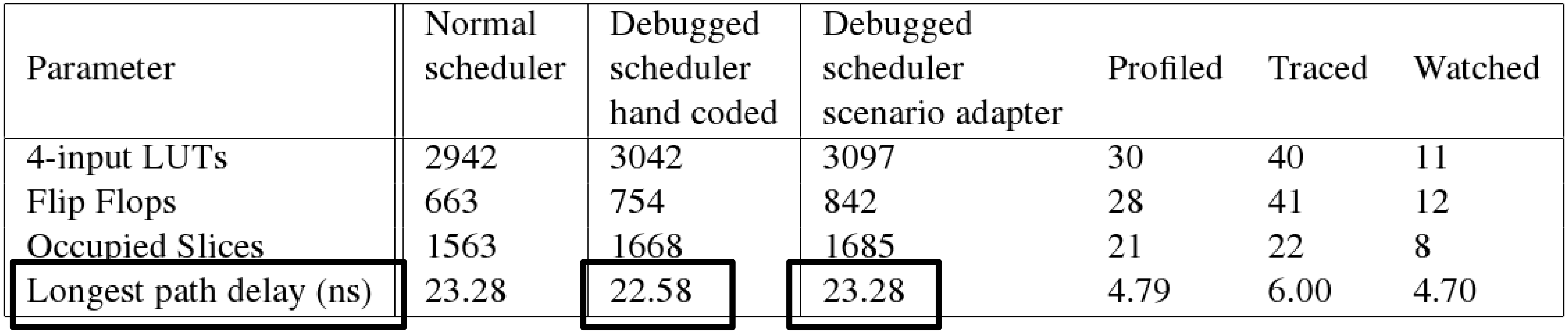


Parameter	Normal scheduler	Debugged scheduler hand coded	Debugged scheduler scenario adapter	Profiled	Traced	Watched
4-input LUTs	2942	3042	3097	30	40	11
Flip Flops	663	754	842	28	41	12
Occupied Slices	1563	1668	1685	21	22	8
Longest path delay (ns)	23.28	22.58	23.28	4.79	6.00	4.70

- Scenario-adapted scheduler uses about 1% more area then the hand-coded scheduler

Results

- Circuits synthesized targeting a Xilinx FPGA



Parameter	Normal scheduler	Debugged scheduler hand coded	Debugged scheduler scenario adapter	Profiled	Traced	Watched
4-input LUTs	2942	3042	3097	30	40	11
Flip Flops	663	754	842	28	41	12
Occupied Slices	1563	1668	1685	21	22	8
Longest path delay (ns)	23.28	22.58	23.28	4.79	6.00	4.70

- Scenario-adapted scheduler uses about 1% more area then the hand-coded scheduler
- The difference in performance is negligible

Conclusion

- Successfully separated the Scheduler dependency from a debugged scenario using a scenario adapter
- Very small overhead for both silicon area and performance
- Scenario adapters seems like an efficient way to increase HW designs quality