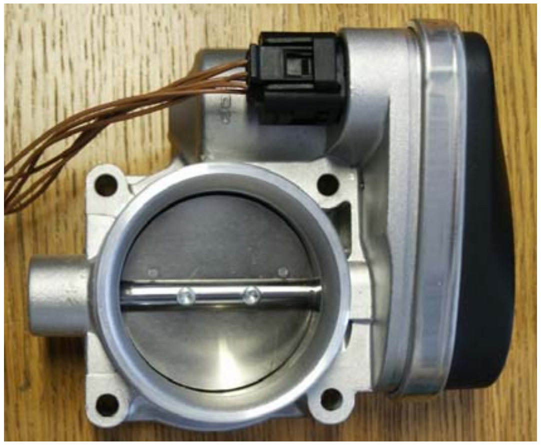
Modularization and Software Architectures

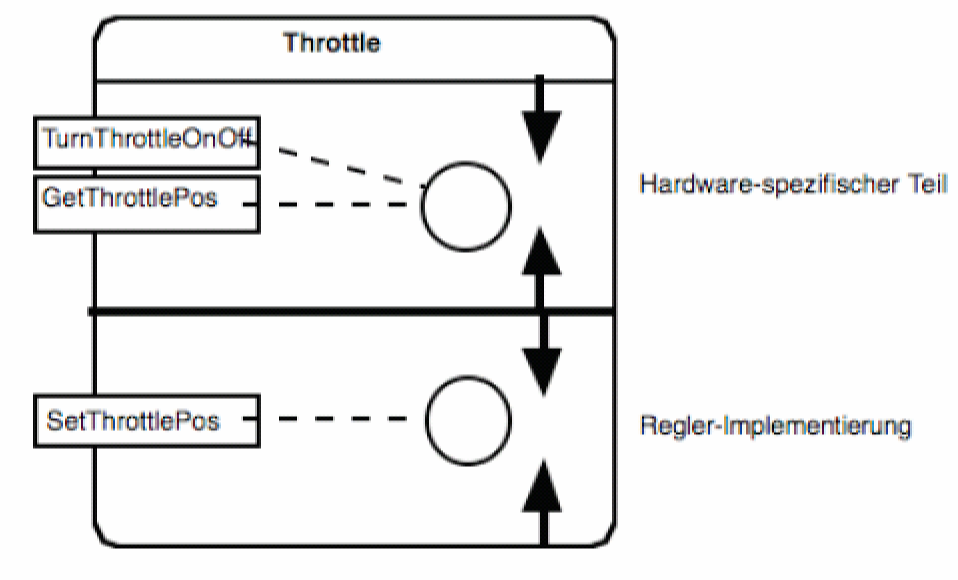
UNIVERSITÄT SALZBURG

Improving Cohesion in the Butterfly Valve Example

```
interface Throttle {
  bool TurnThrottleOnOff(bool onOff);
  bool SetThrottlePosition(float angle);  // 0..90 Grad
  float GetThrottlePosition();
}
```

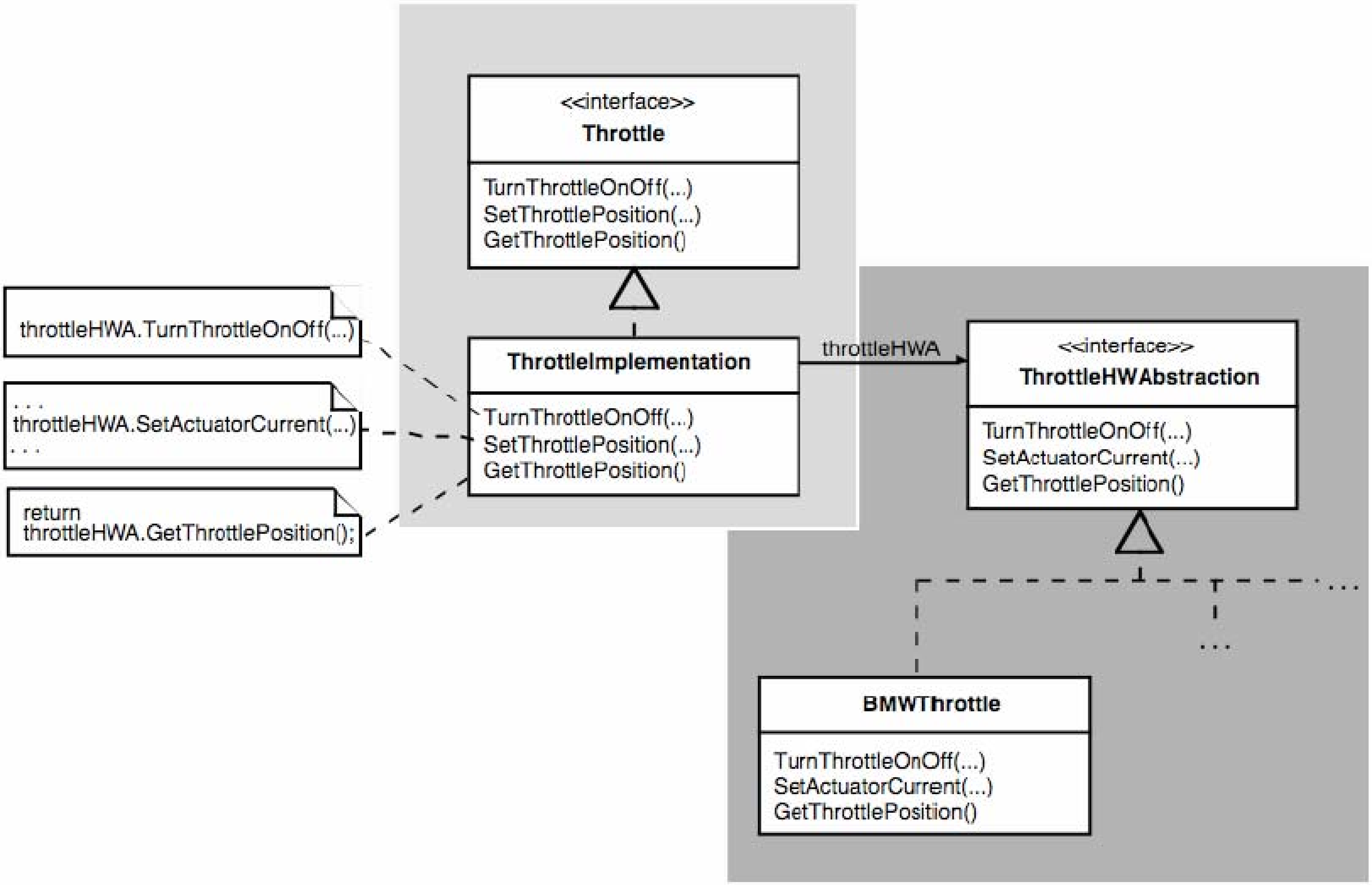


Low cohesion:



UNIVERSITÄT SALZBURG

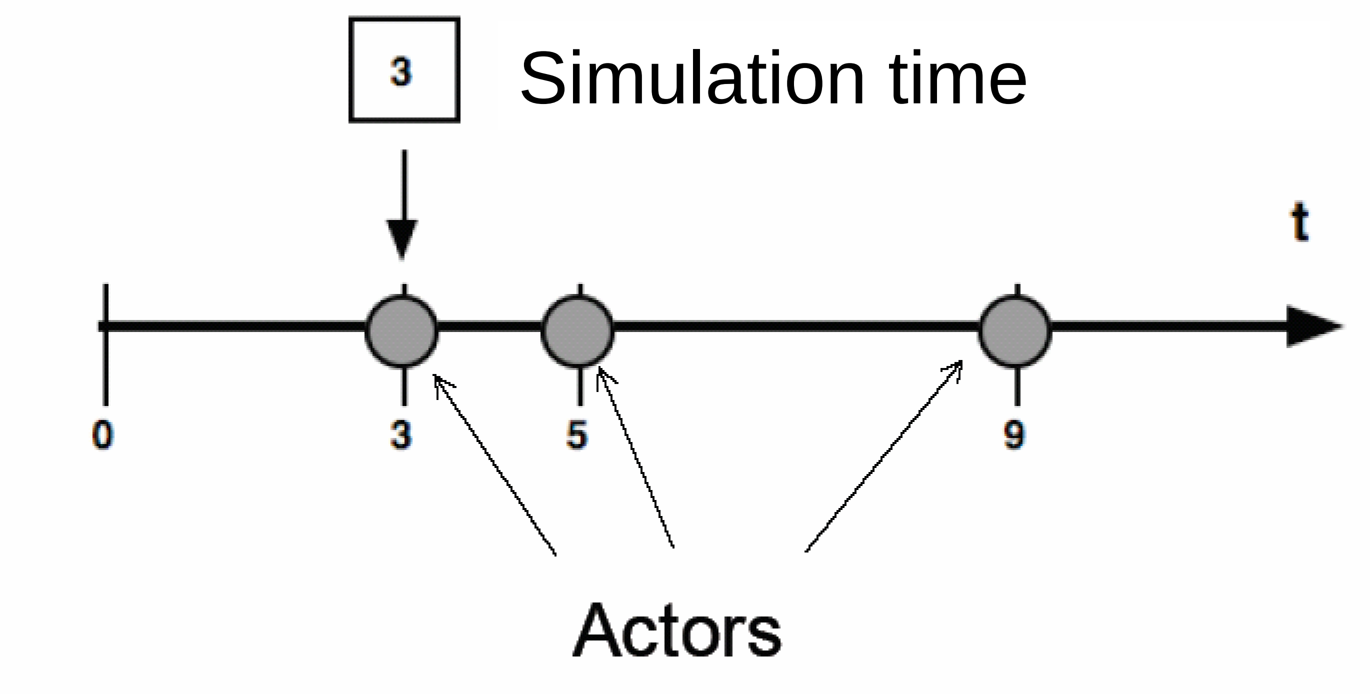
Split up the module Throttle while maintaining the interface



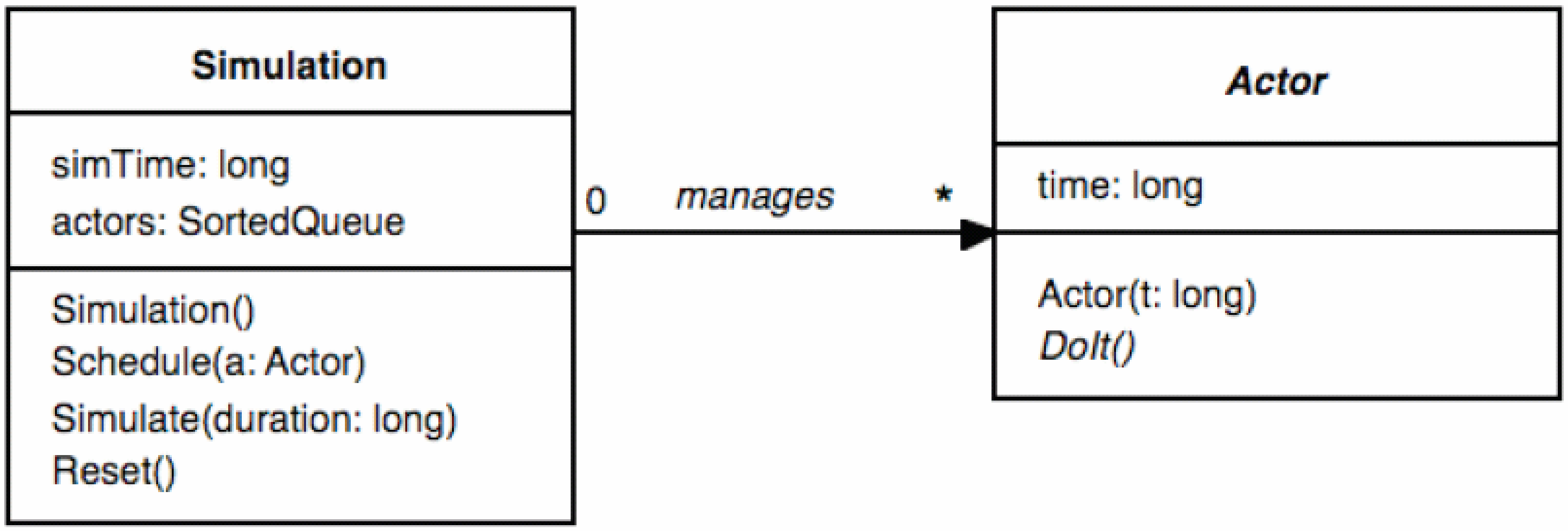
UNIVERSITAT SALZBURG

Example: Simulation of Discrete Events

Discrete Events on a Time Axis



Framework for discrete event simulation



Simulation

simTime: long
actors: SortedQueue

Simulation()
Schedule(a: Actor)
Simulate(duration: long)
Reset()

Actor

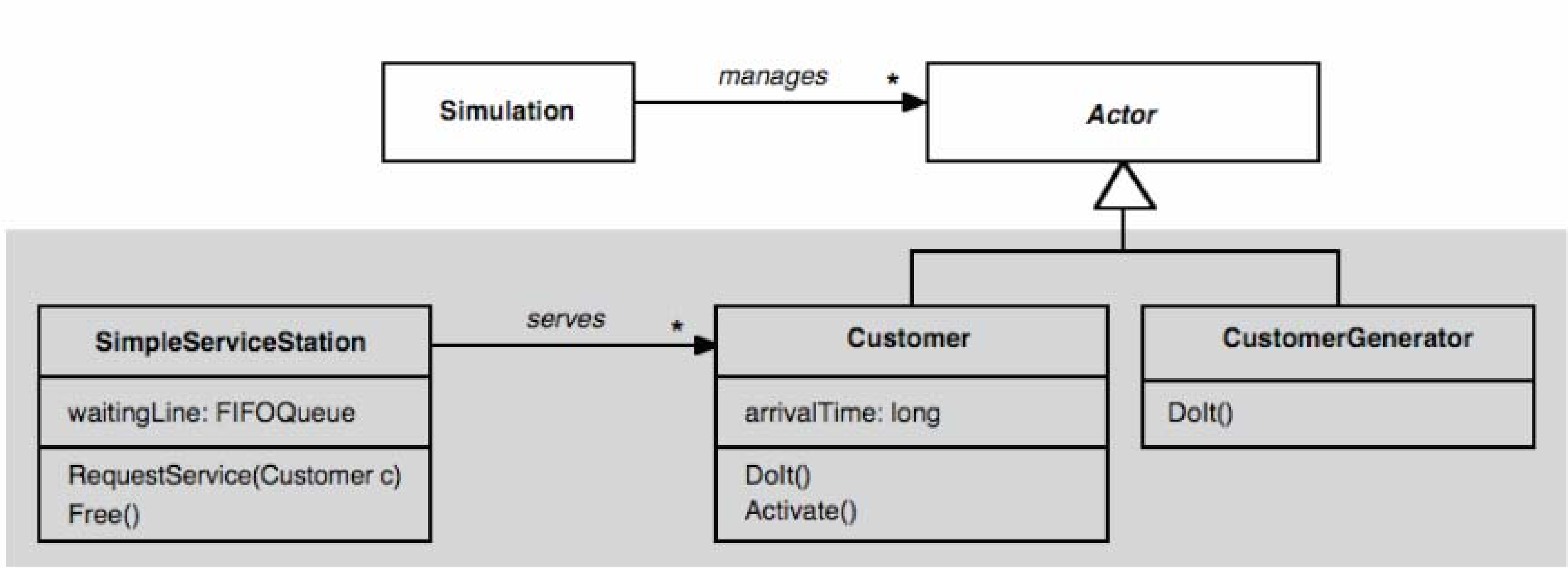
time: long

Actor(t: long)
DoIt()

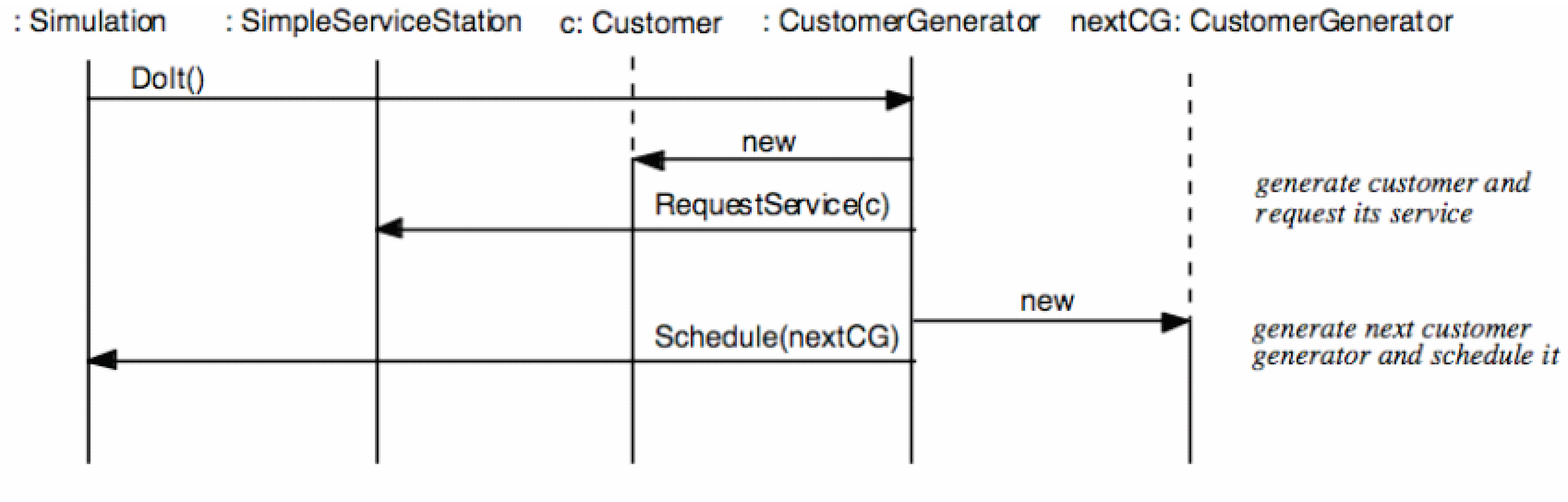
C# Implementation of Simulate()

```
public void Simulate(long duration) {
        long endOfSimulation= simTime + duration;
        do {
          if (actors.Count() != 0) {
                Actor actor= (Actor) actors.Dequeue();
                simTime = actor.time;
                actor.DoIt();
          } else      // no more actors enqueued
                break;      // exit loop
        } while (simTime <= endOfSimulation);
   }
```

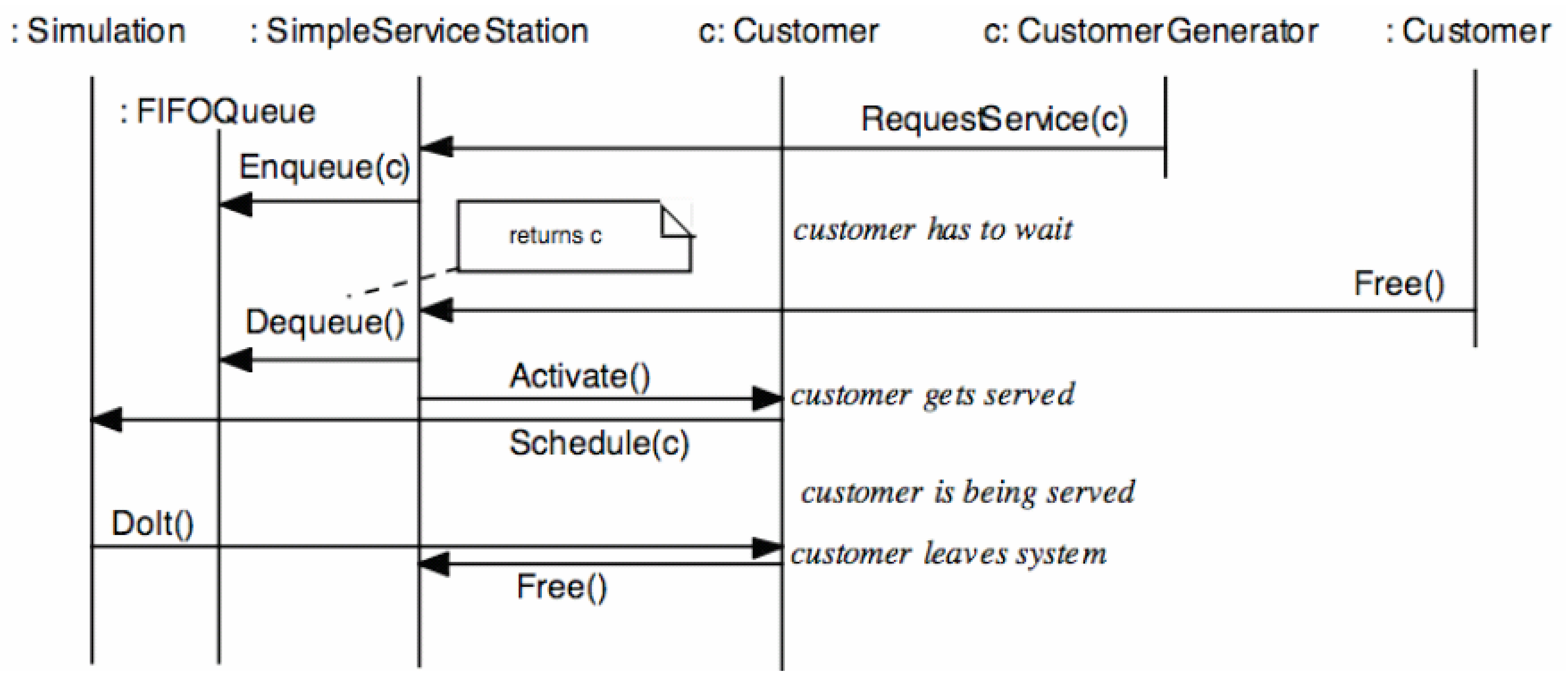
Classes for the simulation of a simple bank service counter



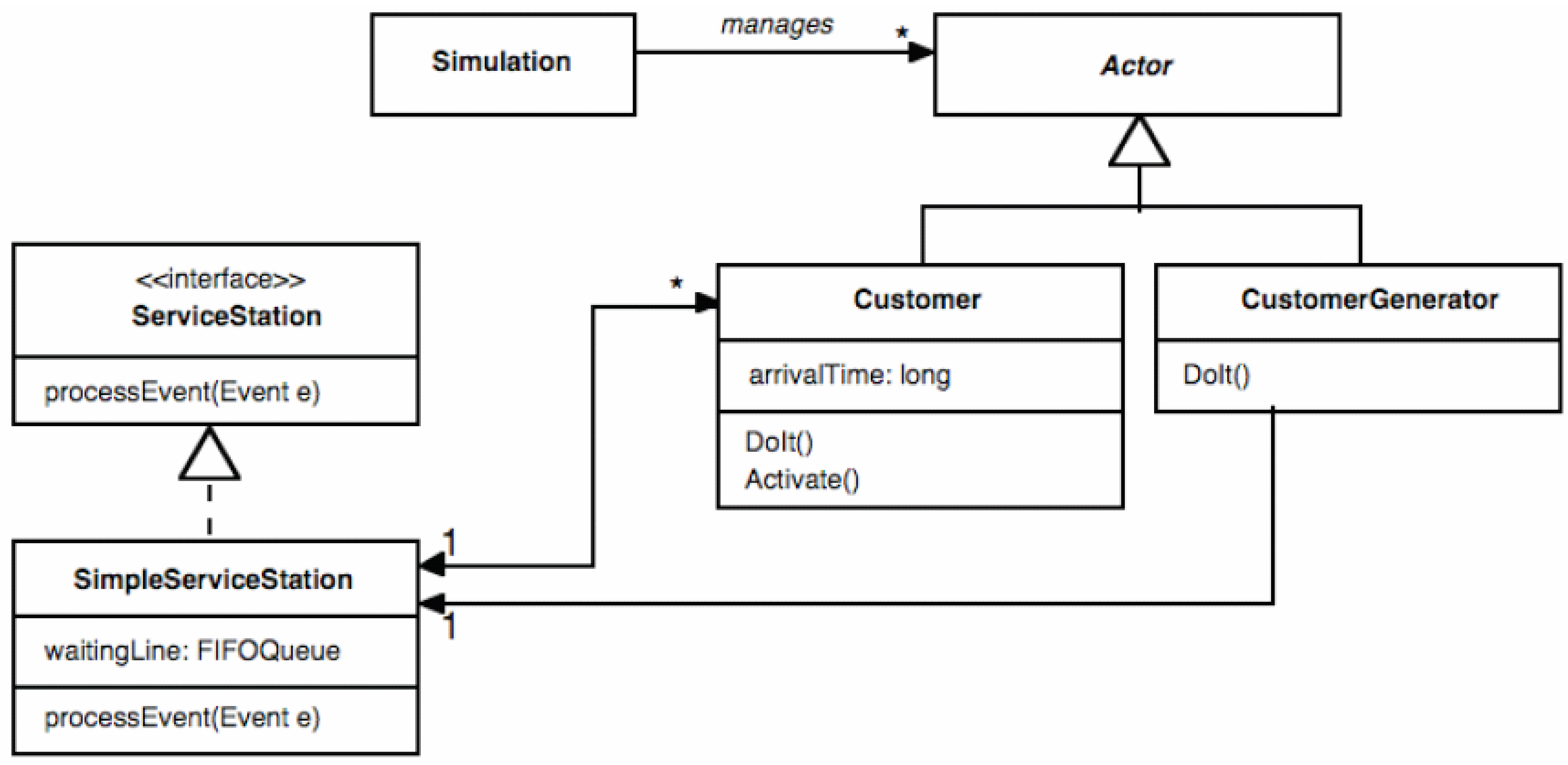
The DoIt() Method of Class CustomerGenerator


: Simulation
: SimpleServiceStation
c: Customer
: CustomerGenerator
nextCG: CustomerGenerator
DoIt()
new
RequestService(c)
new
Schedule(nextCG)
generate customer and request its service
generate next customer generator and schedule it

Lodging a customer in the queue



Decrease of the coupling between service station and the actors



UNIVERSITÄT SALZBURG

Description of Software Architectures

UNIVERSITÄT SALZBURG

Definition of Software Architecture

The assembly of all the components (modules) of a software system together with their interactions.

Architectural styles

- In the 90s, The Software Engineering Institute (SEI) of the Carnegie Mellon University in Pittsburgh, Pennsylvania, considerably contributed to the establishment of **architectural styles** for the description of software architectures.
- SEI originally suggested a dedicated notation for architecture description; since 2003, SEI has used also the UML for that.

Examples of well described software architectures

- *Project Oberon—The Design of an Operating System and Compiler* by Wirth and Reiser (Addison-Wesley 1992).

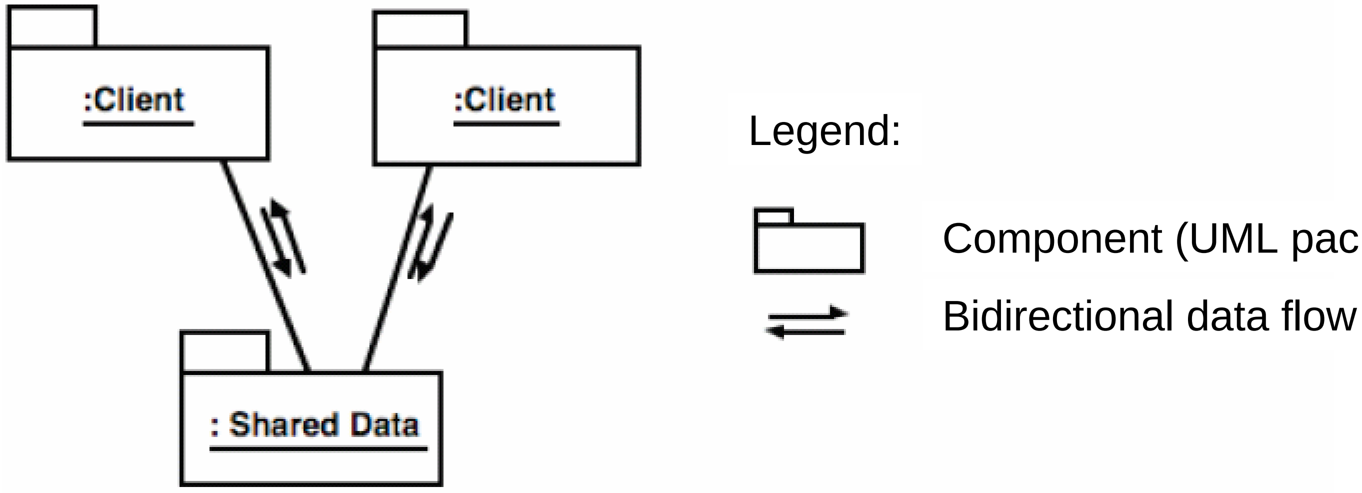
  The informal description is supplemented by schematic representations, screenshots, and source text.

- *Design Patterns* of Gamma et al. (Addison-Wesley 1995)

Overview of SEI architectural styles

Architectural style	**Characteristics**
Data-centered	Repository Architecture Blackboard Architecture
Data-flow	Batch/Sequential Architecture Pipes&Filters Architecture
Call & Return	Top-Down Architecture Network Architecture (Object oriented) Layered Architecture
Virtual Machine	Interpreter Architecture Rule-based Architecture
Independent Components	Event-driven Architecture

Data-centered (I)



Legend:

Component (UML package)

Bidirectional data flow

- In a Repository architecture the data is passive.
- A Blackboard architecture has quasi-active data, which informs the clients interested in changes. The Blackboard architecture style is similar to the Observer design pattern (Gamma et al., 1995).

UNIVERSITÄT SALZBURG

Data-centered (II)

Advantage:

- Clients are independent from each other. Thus, a client can be changed, without affecting the others. Also further clients can be added.

- This advantage pales if the architecture is changed in such a way that clients are coupled closely (thus deviating from the recommended architecture style), for example in order to improve the performance of the system.

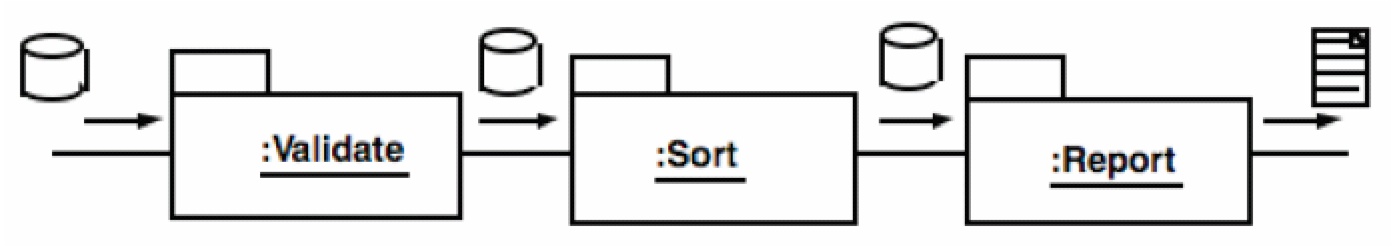
UNIVERSITÄT SALZBURG

Data-centered (III)

Issues that must be addressed:

- Data consistency - synchronization of read/write operations
- Data security, access control
- Single point of failure

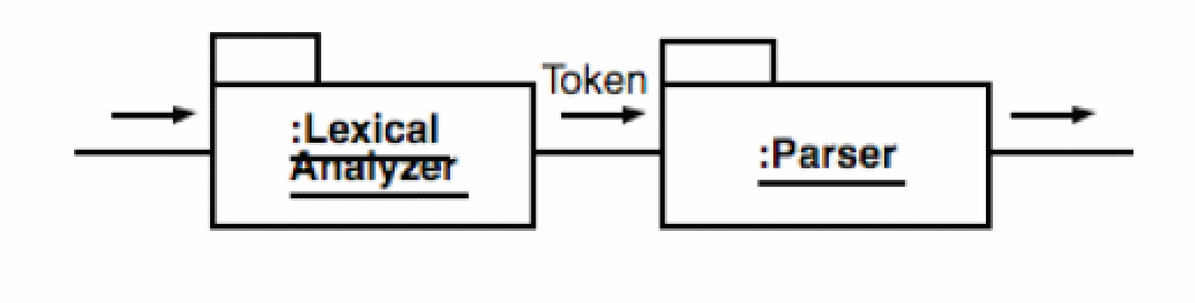
Data-flow style: Batch/Sequential



- The style describes a succession of transformations of input data.
- Data flow-oriented architecture parts are particularly characterized by reusability and modifiability.
- In the Batch/Sequential form, each transformation procedure must be terminated before the next one begins.

UNIVERSITÄT SALZBURG

Data-flow style: Pipes&Filters



- In the Pipes&Filters form, data is incrementally (not sequentially in blocks) transformed. That is, the data is divided into smaller units and these units are processed by the processes.
- Pipes are stateless and transport the data from filter to filter in such a way that each filter autonomously determines when it needs the next element (input) of the data stream from the preceding filter.
- The difference between Pipes&Filters and Batch/Sequential is not evident in a UML representation.

UNIVERSITÄT SALZBURG

Data-flow style: Advantages and Disadvantages

- The main advantage of data-flow is the low complexity of interactions between components. The processing modules are black boxes.
- The data-flow-oriented architecture style is unsuitable for modeling interactive applications.
- A further disadvantage is the frequently insufficient performance and efficiency. If filters need the entire input stream as context, appropriate buffers must be used. That affects the memory efficiency negatively.
- The data-flow style is well suited as basis for visual-interactive composition. It is used for example in the tool Simulink (from MathWorks).