Kurs OMSI
im WiSe 2011/12

Objektorientierte Simulation mit ODEMx

Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage

fischer|ahrens|eveslage@informatik.hu-berlin.de

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung

WaitQ- Member-Funktionen

```
WaitQ (base::Simulation &sim, const data::Label &label, WaitQObserver *obs=0)
                                    // Construction for user-defined Simulation.

~WaitQ ()
                                    // Destruction.

const base::ProcessList & getWaitingSlaves () const
                                    // List of blocked slaves.

const base::ProcessList & getWaitingMasters () const
                                    // List of blocked masters.
```

Slave-Operationen

```
bool wait ()
                                    // Wait for activation by a 'master' process.

bool wait (base::Weight weightFct)
                                    // Wait for activation by a 'master' process.
```

Master-Operationen

```
base::Process * coopt (base::Selection sel=0)
                                    // Get a 'slave' process by optional evaluating a selection function.

base::Process * coopt (base::Weight weightFct)
                                    // Get a 'slave' process by evaluating a weight function.

base::Process * avail (base::Selection sel=0)
                                    // Get available slaves without blocking (optional: select slave)
```

aktiviert den am längsten wartenden Master (also nicht alle wartenden!)

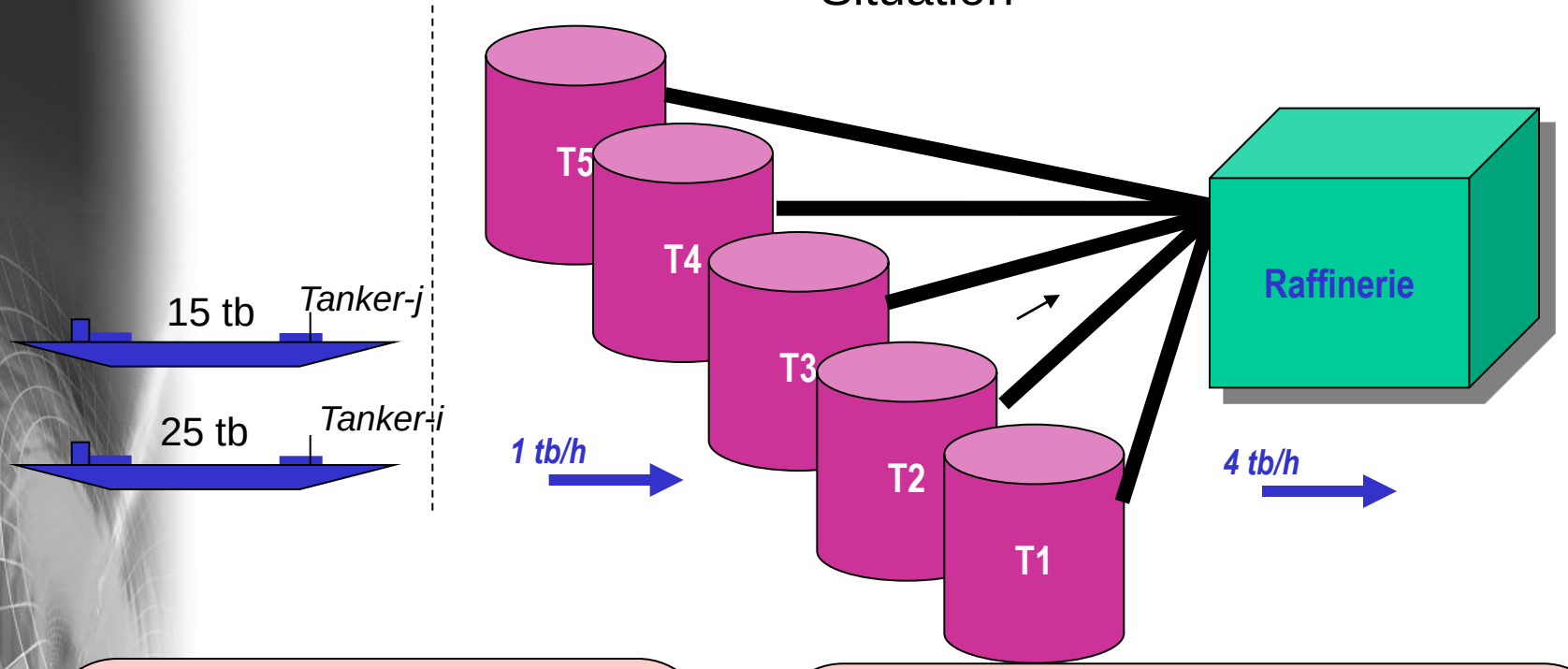
aktiviert den Master, dessen Gewichtsfunktion für den rufenden Slave den Maximalwert liefert

wählt den ersten Slave, für den die Auswahlfunktion des rufenden Masters wahr ist

wählt unter allen Slaves denjenigen, für den die Auswahlfunktion des rufenden Masters den Maximalwert liefert

Beispiel: Tanker – Tank – Raffinerie

Situation



Zustand zum Zeitpunkt t

- alle Tanks sind belegt oder mit Raffinerie verbunden
- zwei wartende Tanker i und j, wobei Tanker i am längsten gewartet hat

Zustandsänderung zum Zeitpunkt t + dt

- einer der Tank-Behälter wird durch Tanker freigegeben (gebliebene Aufnahmekapazität = 20tb)

Wie geht es weiter?

Kritik an der aktuellen ODEMx-Lösung

- WaitQ sollte im Bedienungsmodus veränderbar sein
  - setStatus_OneMaster()

    so wie aktuelle Semantik:
    Der am längsten wartende Master erhält die Steuerung durch den nächsten eintreffenden Slave.
    Sollte die Selection- Funktion für alle erfassten Slaves False ergeben, blockiert dieser Master wieder ohne seine Nachfolger in der masterQ zu aktivieren

  - setStatus_AllMaster()

    Der am längsten wartende Master erhält die Steuerung durch den nächsten eintreffenden Slave (wie oben).
    Sollte die Selection- Funktion für alle erfassten Slaves False ergeben, blockiert dieser Master, aktiviert aber zuvor seinen masterQ-Nachfolger.

6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung

Begriffe

- **Ereignis**

  allgemein:

  … ist ein Geschehen, das in einem gegebenen Kontext eine Bedeutung hat und das sich räumlich und zeitlich lokalisieren lässt.

  OO-Verhaltensmodellierung:

  … ist die zu einem Zeitpunkt ohne Modellzeitverbrauch stattfindende Realisierung von Aktionen zur Attributänderung eines oder mehrerer Objekte

  - Auslösen neuer Ereignisse (Instanziierung von Ereignissen)
  - Start oder Abbruch von (zeitverbrauchender, zustandsverändernden) **Aktivitäten**, i.allg. beschrieben durch ein Start- und Ende-Ereignis

- **Ereignisklassen**

  … sind das Ergebnis einer Klassifikationsabstraktion von Ereignissen bei Parametrisierung der Ereigniszeit und der beteiligten Objekte (Objektreferenzen). Ereignisse sind Instanzen ihrer jeweils beschreibenden Ereignisklasse.

Begriffe

zwei Spielarten von Ereignissen

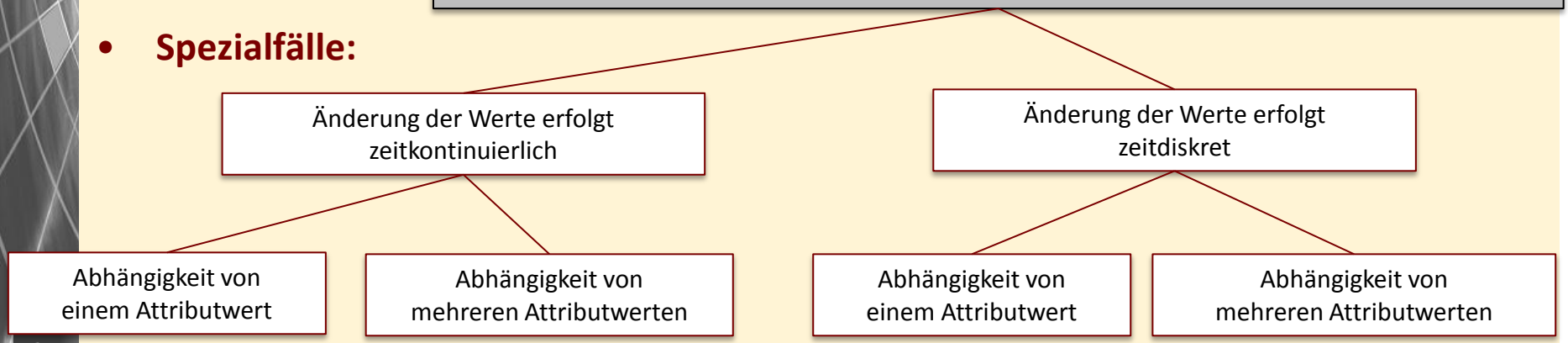
- **Zeitereignis:**

  bei der Instanziierung ist der Ereigniszeitpunkt bekannt

- **Zustandsereignis:**

  bei der Instanziierung ist der Ereigniszeitpunkt nicht bekannt, dieser ergibt sich vielmehr durch die Erfüllung einer Bedingung, die an Attributwerte der beteiligten Objekte geknüpft ist, die sich in Abhängigkeit der Modellzeit ändern.
  Der erste Zeitpunkt, zu dem diese Bedingung erfüllt ist oder der erste Zeitpunkt der Feststellung der Bedingungserfüllung ist dann der Ereigniszeitpunkt

- **Spezialfälle:**



Zustandsereignisse

Aktion: *beliebige Komplexität*
(1) Start/Stop oder Aktivierung/Unterbrechung eines oder mehrerer Prozesse
(2) elementare Aktion

- Ereignis mit Zustandsbedingung
  (erst mit Erfüllung der Bedingung soll Ereignis eintreten)

Annahme
- System besteht aus Prozessen
- Zustand ändert sich mit Zeitfortschritt durch Realisierung der Prozesse

Fragen
- wo werden Zustandsbedingungen vermerkt?
- wer überprüft wann die Zustandsbedingungen ?
- wer löst die Ereignisse aus?

ODEMx unterscheidet zwischen der Behandlung
- zeitkontinuierlicher und
- zeitdiskreter

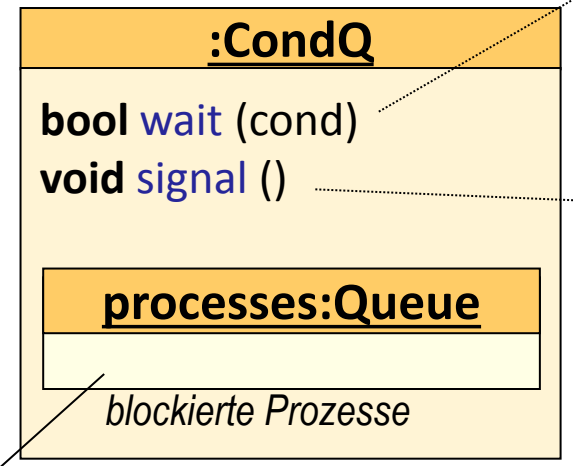
Zustandsgrößen

```
int Contiuous:: integrate
                (SimTime timeEvent, Condition stateEvent=0);
```

```
bool CondQ:: wait (Condition cond);
```

```
typedef bool (Process::*Condition)();
//definiert in Process
```

CondQ-Konzept



Aufrufer-Prozess wird blockiert,
- wartet auf Erfüllung der Bedingung cond

Aufrufer-Prozess
- reaktiviert **alle** blockierten Prozesse (bei erneuter Auswertung der Nutzerbedingung)

noch **nicht implementiert**
- setAll() : alle blockierten Prozesse
- setOne(): der am „längsten“ wartet

ACHTUNG
sinnvollerweise sollten signal()-Rufer zumindest
partiellen Einfluss auf die Zustands-bedingung der blockierten Prozesse haben

jeder hier vermerkte, blockierte Prozess kennt seine individuelle Fortsetzungsbedingung

aktuelle ODEMx-Semantik von signal

Weitere Anforderungen an CondQ

1. über ein CondQ-Objekt sollen sich

   gleichzeitig / nacheinander

   beliebig viele Prozesse (einer oder unterschiedlicher Klassen) registrieren können, die auf individuelle Zustandsbedingungen zu warten haben.

   über die dynamische Vereinigung der Prozesse in einem CondQ-Objekt wird ein einheitlicher Zugang für ihre Reaktivierung (bei erneuter Berechnung der Fortsetzungsbedingung) geschaffen

2. die Warte-Aktivität soll für jeden blockierten Prozess, der in einem CondQ-Objektes erfasst ist, vorzeitig **unterbrechbar** sein,
   der Rückkehrcode informiert den Rufer von wait über die jeweilige Warte-Beendigung

Member-Funktionen von CondQ

```
CondQ (base::Simulation &sim, const data::Label &label, CondQObserver *obs=0)
                    // Construction for user-defined Simulation.
~CondQ ()
                    // Destruction.

getWaitingProcesses () const
                    // Get a list of blocked process objects.


wait (base::Condition cond)
                    // Wait for cond.
signal ()
                    // Trigger condition check.
```

```
bool CondQ::wait (Condition cond) {

      processes. inSort (getCurrentProcess());

      // statistics
      SimTime t=env->getTime();

      while (!(getCurrentProcess()->*cond)() ) {
                getCurrentProcess()->sleep();

                if (getCurrentProcess()->isInterrupted()) {
                          processes. remove (getCurrentProcess());

                          return false;
                }
      }

      processes. remove(getCurrentProcess());

      // statistics
      t = env->getTime() - t;
      sumWaitTime += t;
      if (t==0) zeros++;

      users++;
                …
      return true;
}
```

```
void CondQ::signal() {
    // trace
    getTrace()->mark( this, markSignal, getCurrentProcess());

    // observer
    …

    // statistics
    signals++;

    if (processes.isEmpty())
        return;

    // test conditions
    awakeAll(&processes);
}
```

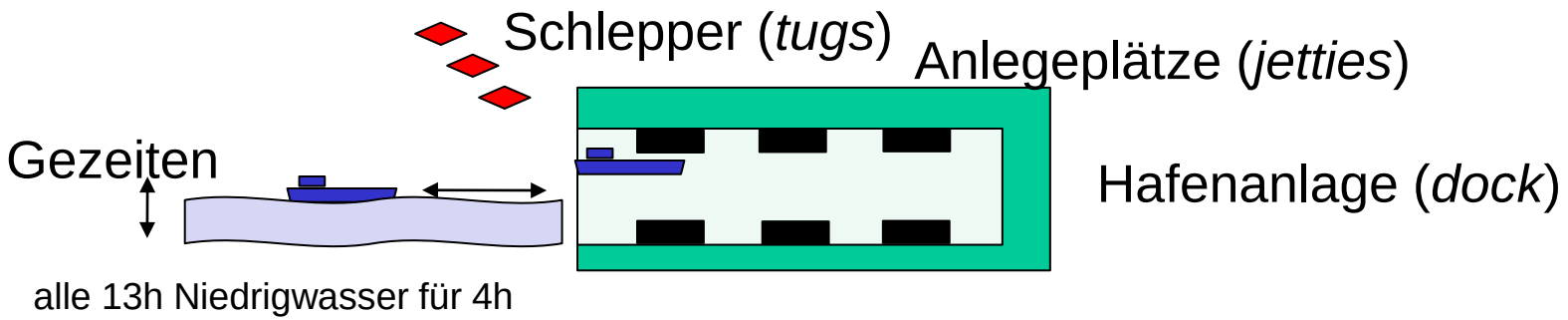
Kritik an der aktuellen ODEMx-Lösung

- Warte-Operation sollte waitUntil heißen (frühere ODEMx-Variante)
  - drückt Semantik prägnanter aus
  - so hat ODEMx nun 3 Kategorien von wait()-Funktionen
    - Globale Funktion: Parameter sind Memory-Objekte
    - Member-Funktion von WaitQ
    - Member-Funktion von CondQ
- setOne(), setAll() wäre nützlich

6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel (Res, CondQ): Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung

Wortmodell: Hafenabfertigung mit Gezeiten



Einlauf von Schiffen
Bedingungen/Aktionen:
- freier Anlegeplatz
- zwei freie Schlepper
- Wasserstand: hoch
- Anlegemanöver= 2h
- Schlepperfreigabe

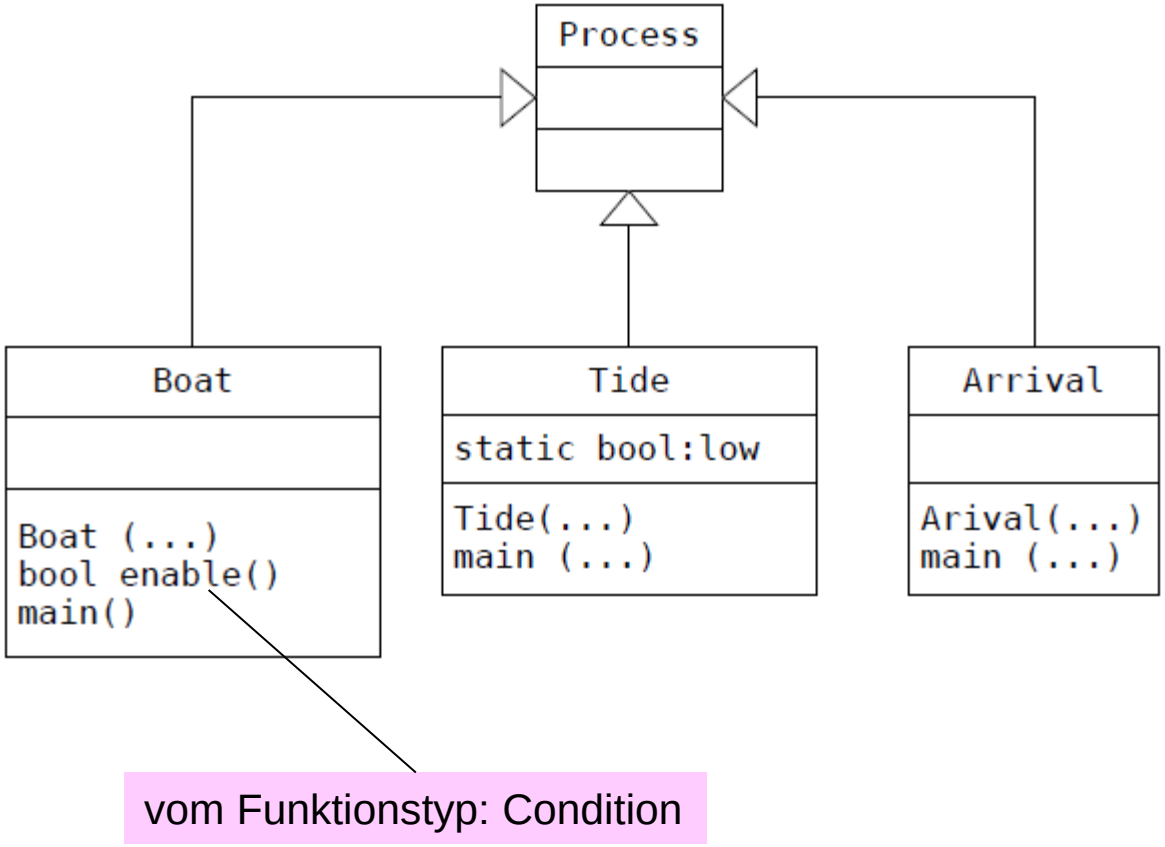
Entladung von Schiffen
Bedingungen/Aktionen:
- stochastische Entladungszeit

Auslauf von Schiffen
Bedingungen/Aktionen:
- ein freier Schlepper
- Auslaufmanöver: 2h
- Schlepperfreigabe
- Platzfreigabe

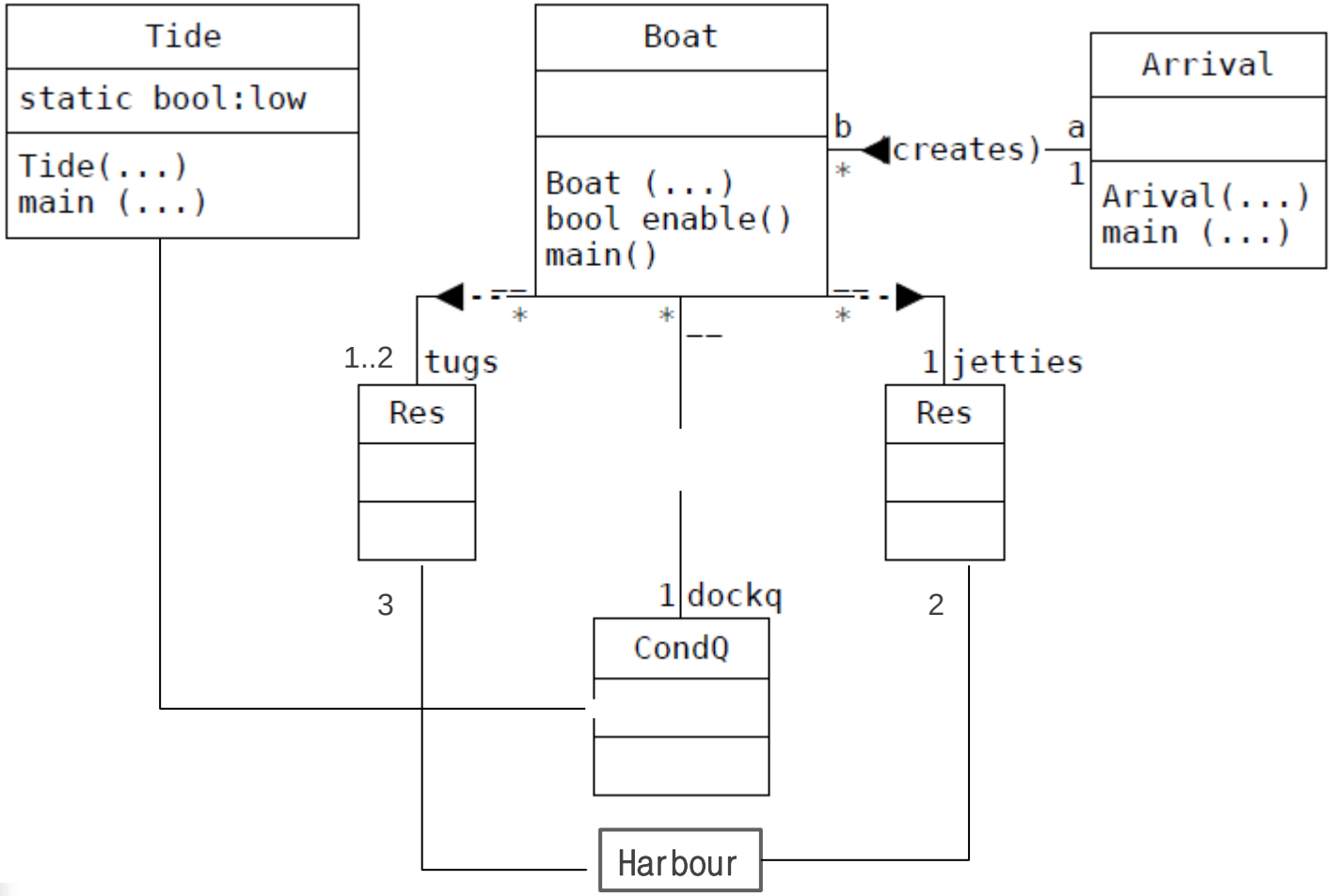
Ziel: simulative Workflow-Nachbildung bei Bestimmung der Auslastung eingesetzter Ressourcen

Systemstruktur (semiformal)

Strukturkomponenten (Typbeschreibung, Objektkonfiguration)



Objektkonfiguration



Verhalten von Boat

```
Simulation* sim = getDefaultSimulation();
Res   *tugs;                                    // Schlepper
Res   *jetties;                                 // Anlegeplaetze
CondQ *dockq;
ContinuousDist *next, *discharge;

class Boat : public Process {
public:
        int main ();
        Boat() : Process(sim, "boat"){}
        bool enable();
};
```

```
bool Boat::enable() {
   return (tugs->getTokenNumber() >=2) && !Tide::low;
}
```

```
int Boat::main() {
// im Dock anlegen
        jetties->acquire(1);
        dockq->wait((Condition)&Boat::enable);
        tugs->acquire(2);
        holdFor(2.0);
        tugs->release(2);
        dockq->signal();
// loeschen der Ladung
        holdFor(discharge->sample());
// ablegen
        tugs->acquire(1);
        holdFor(2.0);
        tugs->release(1);
        jetties->release(1);
        dockq->signal();

        return 0;
}
```

Verhalten von Tide

```
class Tide : public Process {
public:
        static bool low;
        Tide(): Process(sim, "tide") {};
        int main ();
};

bool Tide::low = false;
```

```
int Tide::main() {
        for (;;) {
                // low
                low = true;
                holdFor(4.0);
                // high
                low = false;
                dockq->signal();
                holdFor(9.0);
        }
        return 0;
}
```

```
int main( int argc, const char* argv[]) {
        next = new Negexp(sim, "next boat", 0.1);
        discharge = new Normal(sim, "discharge", 14.0, 3.0);
        tugs = new Res(sim, "tugs", 3, 3);
        jetties = new Res(sim, "jetties", 2, 2);
        dockq = new CondQ(sim, "dockq");

        report.addProducer(next);
        report.addProducer(discharge);
        report.addProducer(tugs);
        report.addProducer(jetties);
        report.addProducer(dockq);
        sim->startTrace();
        a->activate();
        t->activateIn(1.0);

        sim->runUntil(50.0);
        sim->stopTrace();
        sim->runUntil (28.0*24.0);
        report.generateReport();

        delete a;
        delete t;
        delete next;
        delete discharge;
        delete tugs;
        delete jetties;
        delete dockq;

        return 0;
}
```

a= **new** Arrival();
t= **new** Tide();

Simulation: DefaultSimulation SimTime: 670

HtmlReport ODEMx Version: 2.2

Random Number Generators							
Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3
next boat		Negexp	64	33427485	0.1	0	0
discharge		Normal	58	22276755	14	3	0

Queue Statistics					
Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length
tugs queue		0	1	0	0
jetties queue		0	7	6	0.634897
dockq queue		0	2	0	0.0770625

Res Statistics											
Name	Reset at	Queue	Acquires	Releases	Init token number	Limit token number	Min token number	Max token number	Now token number	Avg token number	Avg waiting time
tugs		tugs queue	114	114	3	3	0	3	3	2.48657	0
jetties		jetties queue	58	56	2	2	0	2	0	0.373191	6.34256

CondQ Statistics						
Name	Reset at	Queue	Users	Signals	Zero wait users	Avg waiting time
dockq		dockq queue	58	166	36	0.890205