Kurs OMSI
im WiSe 2010/11

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 und CondQ

Tanker – Tank – Raffinerie: Wortmodell

1. Durchführung einer 1000h-Simulation logistischer Abläufe eines **Ölhafen**

2. **Tanker** unterschiedlichen Fassungsvermögens (gleichverteilt 15 tb, 20 tb, 25 tb)
   – treffen zufällig im Öl-Hafen ein und
   – können parallel entladen werden

   1 b ≈ 159 l

3. die **Zwischenankunftszeit** der Tanker ist neg.exponential verteilt
   – im Mittel soll alle 8h ein weiterer Tanker eintreffen

4. zur Entladung stehen maximal **5 Tankbehälter** bereit,
   von denen pro Tanker jeweils immer nur einer zugeordnet wird, und zwar der
   – der die längste Zeit zur Befüllung bereit war und
   – dessen freie Kapazität die Schiffsladung komplett übernehmen kann

   steht kein solcher Tank zur Verfügung, muss der Tanker **warten**

5. die **Befüllung** des Tankbehälters (Entladung des Tankers) erfolgt
   mit einer konstanten Pumprate
   – von 1 tb/h;

   zum Anschluss eines Tankers an einen Tank werden
   – 0,5 h als **Vorbereitungszeit**

   benötigt.

Wortmodell (Forts.)

6. das maximale **Fassungsvermögen** eines jeden Tanks beträgt 70 tb

7. während der **Befüllung des Tankbehälters** erfolgt keine Entnahme durch die angeschlossene Raffinerie

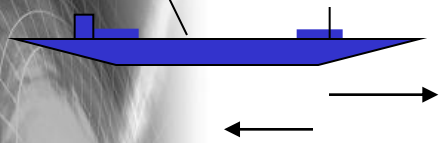
8. die Entnahme von Öl durch die **Raffinerie** erfolgt
   - (nach Abschluss der Befüllung durch einen Tanker), sobald der Tank nur noch 20 tb oder weniger aufnehmen kann
   - mit einer konstanten Pumprate von 4 tb/h;

9. es liegt eine besondere **Ausgangskonfiguration** vor
   - zwei Tanks sind leer
   - einer ist an der Raffinerie angeschlossen und wird in 8h leer
   - zwei werden gefüllt (d.h. zwei Tanker haben angelegt), wobei
     - ein Tank in 3,5h fertig wird mit verbleibender Aufnahmekapazität von 25tb und
     - der andere in 12h mit verbleibender Aufnahmekapazität von 45tb
   - der nächste Tanker wird zur Zeit 0.0 erwartet

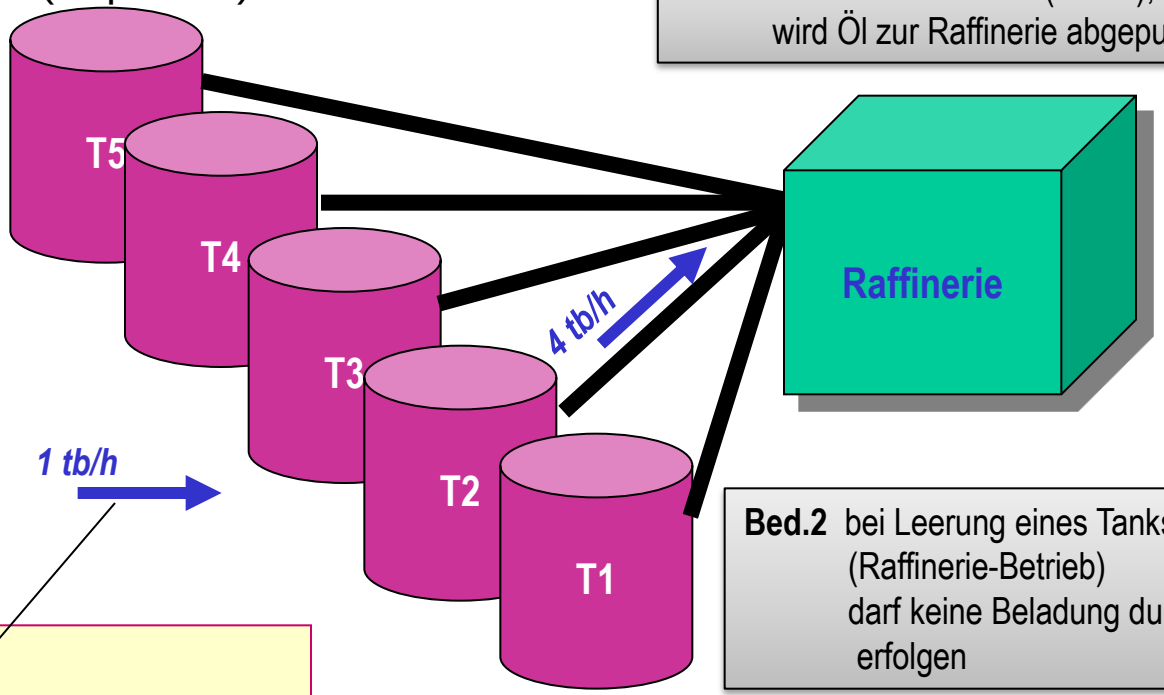
Beispiel: Tanker – Tank – Raffinerie

Zwischenankunftszeit:
n.-exponential verteilt

Ladung: Gleichverteilung
15tb, 20tb, 25tb



(Kap 70 tb)



Bed.1 ist Tank nahezu voll (≈70%),
wird Öl zur Raffinerie abgepumpt

Bed.2 bei Leerung eines Tanks
(Raffinerie-Betrieb)
darf keine Beladung durch Tanker
erfolgen

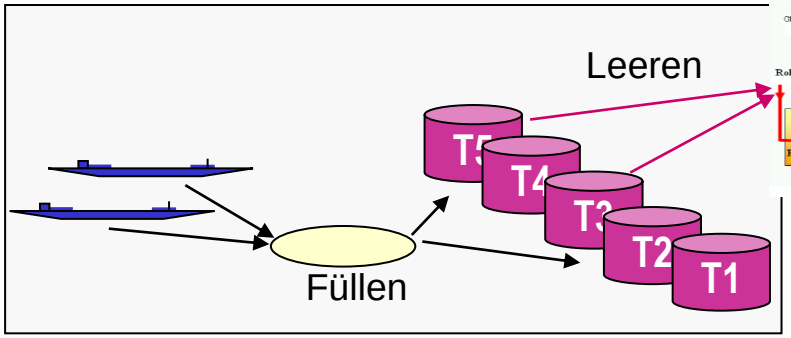
Tankauswahl:
- (1) Tank,
  der die längste Zeit leer war
  und
- (2) dessen frei gebliebene
  Kapazität die Schiffsladung
  komplett übernehmen kann

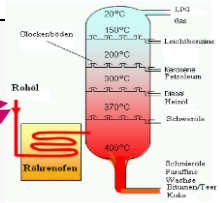
konstante Vorbereitungszeit: 0,5h

Ziel:
1000 h Simulation, bei besonderer Ausgangssituation:
(1) Füllstand der Tankbehälter
- zwei sind leer (70tb frei)
- einer wird in 8h leer (70tb frei)
- einer wird in 12 h mit Befüllung fertig (45tb frei)
- einer wird in 3.5h mit Befüllung fertig (25tb)

(2) erste Tankerankunft: 0.0

Informales Modell → Simulationsmodell



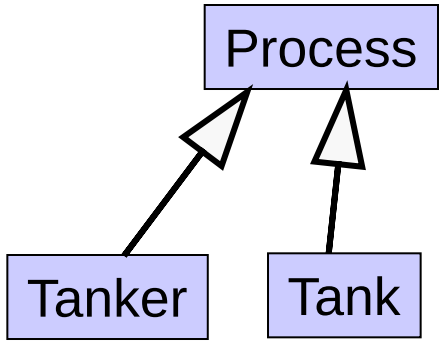


- Kooperationsaktivität: Füllen
  - Tanker und leerer Tank
- Kooperationsaktivität: Leeren
  - „voller“ Tank (und Raffinerie, die aber außerhalb des Systems liegt)
- Master/Slave-Prinzip beim Füllen

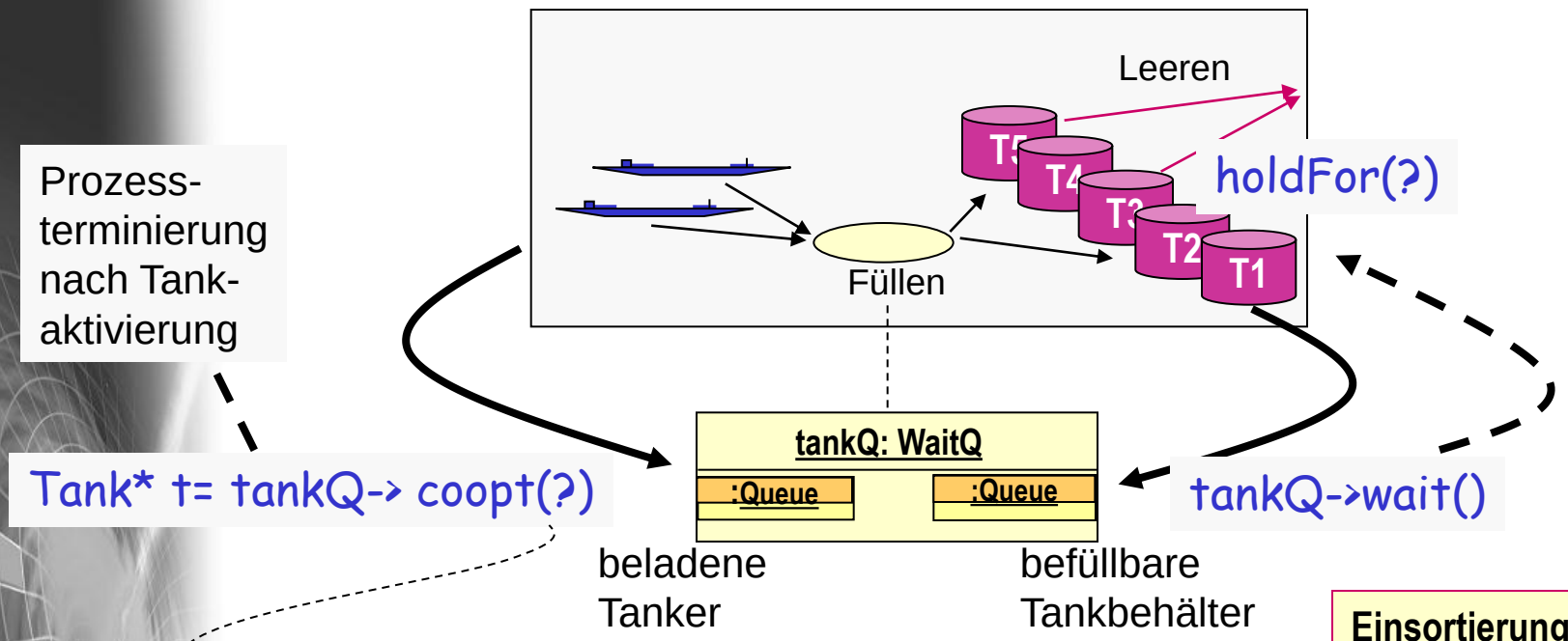
  Frage: warum sollten Tanker die Master-Rolle übernehmen ?

  haben Tank-Objekte nach Kriterien auszuwählen!

  ➔ Anwendung von coopt mit selection-Funtion



Informales Modell → Simulationsmodell



Tankauswahl:
- (1) Tank,
  der die längste Zeit auf Befüllung wartete
- (2) dessen frei gebliebene Kapazität die Schiffsladung komplett übernehmen kann

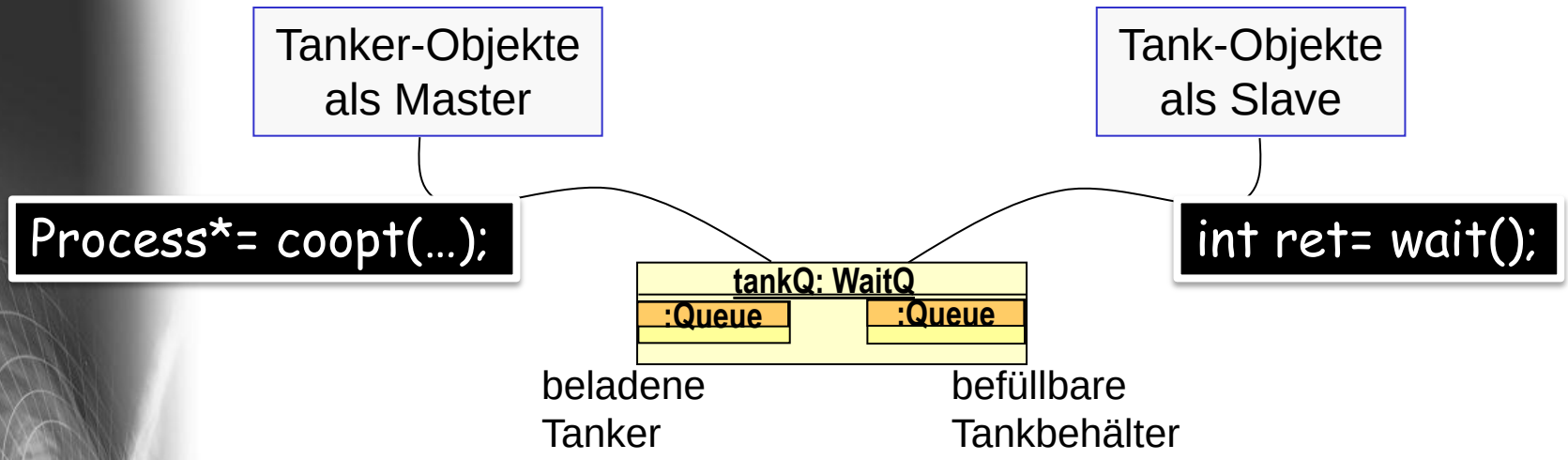
Tanks, die entleert werden, stehen nicht zur Auswahl

Einsortierung
FIFO: Reihenfolge
nach der Wartedauer

```
//Selektionsfunktion als Member-Fkt von Tanker
bool suitableTank (Process* partner) {
      Tank*  selected= dynamic_cast<Tank*> (partner);
      return selected->freeCap >= load;
};
```

freie Tankkapazität — Tankerladung

Umsetzung: Master-Slave-Synchronisation (1)



Frage: Wann wird 0-Zeiger bzw. int 0 zurückgegeben ?

```
...
WaitQ* tankQ= new  WaitQ (defaultSimulation(), "shoreTanks");
...
```

Bem.: Warteschlangen für Master- und Slave-Prozesse
werden durch den WaitQ-Konstruktur angelegt

"shoreTanks_master_queue" "shoreTanks_slave_queue"

Umsetzung: Master-Slave-Synchronisation (2)

Tanker-Objekte als Master

```
DiscreteDist *size=
                    new Randint(defalutSimulation(), "size", 3, 5)
```

Tank-Objekte als Slave

```
class Tanker : public Process {
public:
      Tank *myTank; //Tank zur Entladung
      double load; //Fassungsvermoegen[tb]

      Tanker() : Process(defaultSimulation(),
                         "Tanker"),
                 load (5.0*size->sample()){}
protected:
      int main();

//Selektionsfunktion
      bool suitableTank (Process* partner) {
            ...
      };
};
```

```
class Tank : public Process {
      double maxCap;    //max. Fassungsverm.
public:
      double freeCap; //akt. Freiraum[tb]

      Tank (double f) :
      Process(defaultSimulation(), "Tank"),
            maxCap(70), freeCap(f) {}
protected:
      int main();
};
```

- Adresse dieser Funktion ist coopt beim Aufruf als Parameter zu übergeben
- coopt- Implementierung iteriert über Menge verfügbarer Slave-Prozesse und wendet auf jedes Element suitableTank() an
- coopt liefert Prozess-Zeiger, für den Slave-Prozess, für den als erster die Bedingung gilt

Bedingungen zur Prozessauswahl (Wdh.)

```
class Process : …. {
    public:
        // Funktionstypen zur Codierung von Bedingungen für Prozessauswahl
        typedef bool (Process::*Selection)(Process* partner);
        typedef bool (Process::*Condition)();

        // Prozessgrundzustand
        enum ProcessState {CREATED, CURRENT, RUNNABLE,
                           IDLE, TERMINATED                    };

        Process (Simulation* s, Label l, ProcessObserver* o = 0);
        ~Process();

        ProcessState getProcessState() const;
```

- Funktionstyp heißt Selection,
- Wert einer Variable oder eines Parameters *s* von diesem Typ muss eine Adresse einer Memberfunktion einer Prozess-Ableitung sein mit der Signatur (Process*):bool
- **Beispiel:** Selection s = & processSpecial::mF
  ein Aufruf erfolgt mittels (p->*s) (aktuellerPartner)

Umsetzung: Master-Slave-Synchronisation (3)

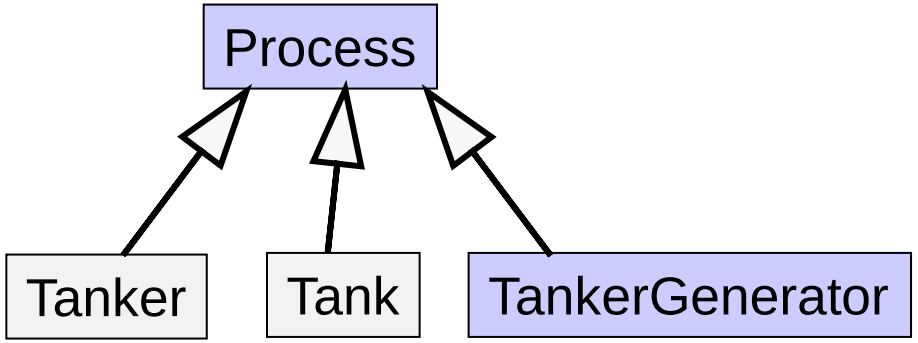
Tanker-Objekte als Master

```
int Tanker::main() {
    //Ankunft im Hafen

    //Auswahl des Tanks und Synchronisation
    myTank = dynamic_cast <Tank*>
             (tankq->coopt(
              (Selection) &Tanker::suitableTank));
    //Entladung
    holdFor(setuptime +
                    load*tankerPumpRate);
    //Zeitverbrauch zur Entladung
    myTank->freeCap =
                    myTank->freeCap - load;
    //Beendigung der Kopplung zum Tank
    myTank->holdFor();

    return 0;
}
```

Tank-Objekte als Slave

```
int Tank::main() {
  for (;;) {
    // Warten auf Synchronisation mit Tanker
    // bei anschl. Befuellung des Tanks
    // durch Tanker
    // bis weniger als 20 tb frei sind
    while (freeCap > 20.0) tankq->wait();

    // Entleerung des Tanks
    holdFor( (maxCap - freeCap) *
                              raffPumpRate);
    freeCap = maxCap;
  }
  return 0;
}
```

Umsetzung: Tankerankunft




```
int TankerGenerator::main() {
  Tanker *t;
   ContinuousDist *arr= new NegExp(defaultSimulation(), "nextTanker", 1.0/8.0);

   for (;;) {
      t= new Tanker;
      t->activateAt(now);
      holdFor(arr->sample());
    }

    return 0;
}
```

Umsetzung: Startsituation

Ziel:
1000 h Simulation, bei **besonderer Startsituation**:

(1) Füllstand der Tankbehälter
- zwei sind leer (70tb frei)
- einer wird in 8h leer (70tb frei)
- einer wird in 12 h mit Befüllung fertig (45tb frei)
- einer wird in 3.5h mit Befüllung fertig (25tb)

(2) erste Tankerankunft: 0.0

```
Tank *t;
    t = new Tank(70.0); t->hold();
    t = new Tank(70.0); t->hold();
    t = new Tank(45.0); t->holdUntil(12.0);
    t = new Tank(25.0); t->holdUntil(3.5);
    t = new Tank(70.0); t->holdUntil(8.0);
```

Aktionen können entweder
- im Hauptprogramm vor Start des Simulationskontextes
  oder
- von einem speziellen Konfigurationsprozess übernommen werden,
  der wiederum vom Hauptprogramm zur Zeit 0 in den Kalender aufzunehmen ist

Report-File

Random Number Generators

Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3
nextTanker	0	Negexp	128	33427485	0.125	0	0
size	0	Randint	128	22276755	3	5	0

Queue Statistics

Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length
shoreTanks_master_queue	0	0	5	0	0.184572
shoreTanks_slave_queue	0	0	5	2	1.82777

Waitq Statistics

Name	Reset at	Master Queue	Slave Queue	Number of Synch.	Zero wait masters	Avg masters wait	Zero wait slaves	Avg slaves wait
shoreTanks	0	shoreTanks_master_queue	shoreTanks_slave_queue	128	102	1.46019	26	14.2281

6. ODEMx-Modul Synchronisation:

WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele

für WaitQ und CondQ

Zustandsereignisse

beliebige Komplexität

Aktion:
- Start/Stop oder
- Aktivierung/Unterbrechung eines oder mehrerer Prozesse

- 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

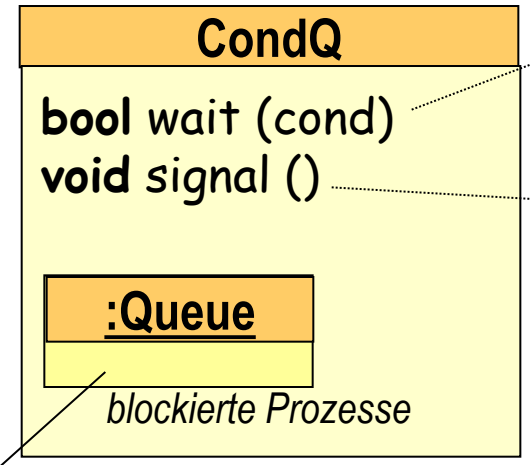
Zustandsgrößen

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

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

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

Zustandsabhängige Synchronisation über zeitdiskrete Größen



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

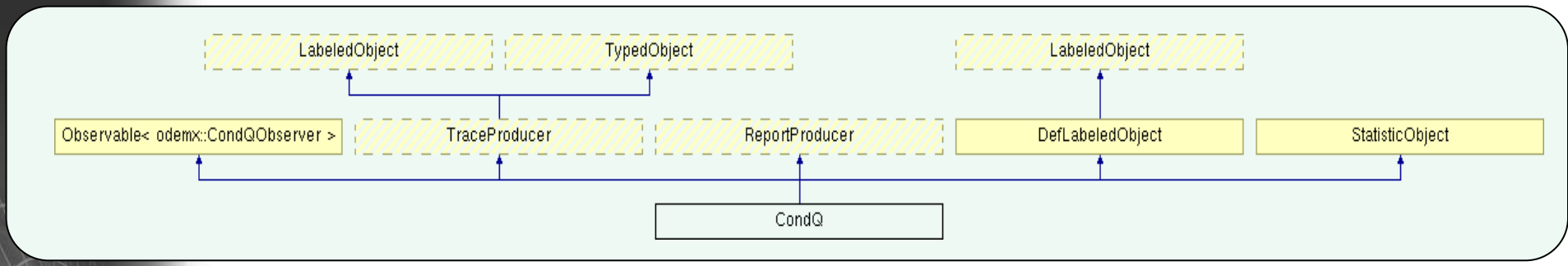
Aufrufer-Prozess
- reaktiviert **alle** blockierten Prozesse (bei erneuter Auswertung der Nutzerbedingung)

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

ACHTUNG

sinnvoller Weise sollte signal()-Rufer zumindest
partiellen Einfluss auf die Zustands-bedingung der blockierten Prozesse haben

jeder vermerkte, blockierte Prozess kennt seine Fortsetzungs-Bedingung



CondQ (Label l, CondQObserver *o=0) *Construction for DefaultSimulation.*
CondQ (Simulation *s, Label l, CondQObserver *o=0) *Construction for user-defined Simulation.*
~CondQ () *Destruction.*
const std::list< Process * > & getProcessList () const *Get list of blocked process objects.*
virtual Trace * getTrace () const *Get pointer to Trace*.
virtual void report (Report *r) *Generate report*.

bool wait (Condition cond) *Wait for cond.*
void signal () *Trigger condition check.*

```
class Condq : public ...
    {
    public:
        bool wait(Condition cond);
        void signal();

        const std::list<Process*>& getProcessList() const {return processes.getList();}
        virtual void reset();
        virtual void report(Report* r);

        unsigned int getZeroWait() const {return zeros;}
        double getAVWaitTime() const {return sumWaitTime / users;}
        unsigned int getUsers() const {return users;}
        unsigned int getSignals() const {return signals;}
        virtual Trace* getTrace() const {return env;}

        static const MarkTypeId baseMarkId;
        static const MarkType markCreate;
        static const MarkType markDestroy;
        static const MarkType markWait;
        static const MarkType markContinue;
        static const MarkType markSignal;

        // Implementation
    private:
        Simulation* env;
        Queue processes;

        // statistics
        double sumWaitTime;
        unsigned int users;
        unsigned int zeros;
        unsigned int signals;

        // help methods
        Process* getCurrentProcess() {return env->getCurrentProcess();}
    };
```

Liste blockierter Prozesse

jeder Modellelementetyp liefert eigene Kennungen für Trace-Aufbau

```
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);
}
```