Modul OMSI-2
im SoSe 2010

Objektorientierte Simulation mit ODEMx

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

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

Ziele von OMSI-1 und OMSI-2

C++ -Konzepte

ODEMx-Konzepte (Version 2.x)

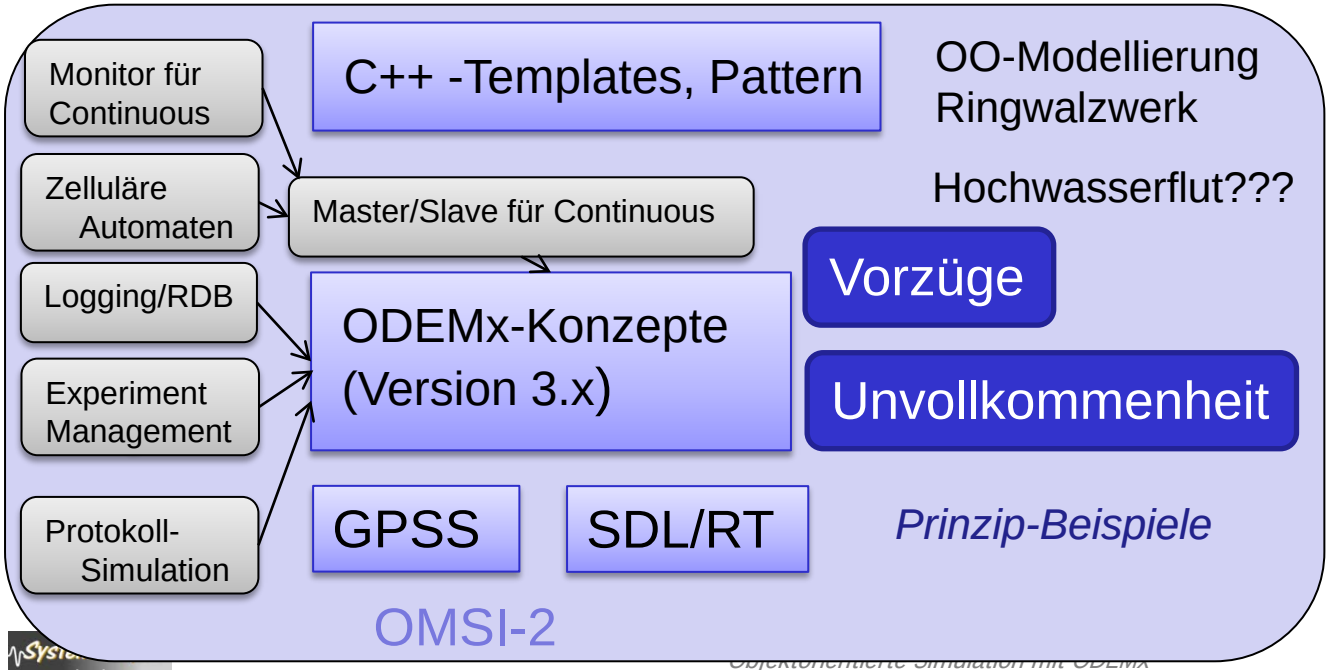
Vorzüge

Unvollkommenheit

OMSI-1

OO-Modellierung von Systemen
- zeitkontinuierlich (GDGL)
- zeitdiskret (Prozesse, Ereignisse)

zum Auftakt nochmal zwei Beispiele



OO-Modellierung Ringwalzwerk

Hochwasserflut???

Vorzüge

Unvollkommenheit

Prinzip-Beispiele

Entwicklung einer an UML/SysML angelehnten ausführbaren DSL mit Laufzeitsystem in ODEMx

aktuelle Forschung

Netzwerksimulatoren

1. WaitQ-, CondQ- Anwendung

Beispiel: Logistikproblem

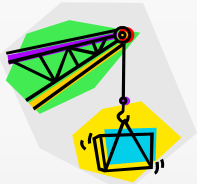
Straßenbelag-Recycling-Anlage

Informales Wortmodell des Systems

7 Bauschutt-Container-Fahrzeuge:
Fassungsvermögen:
je drei Trichterladungen

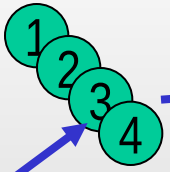
Wiedereintritt:
98 ZE + (0.1)-negexp

Kran



Füllen: normalverteilt
Mahlen: 10 ZE
Entnahme: 15 ZE
Ruhezeit: 10 ZE

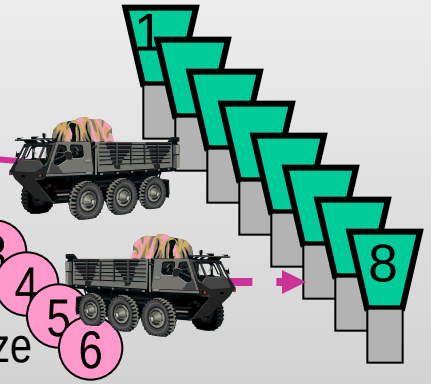
Stellplätze



Kran-Entladeplatz

Wiegen 2 ZE

Waage



Stellplätze

Mahlwerke

Straßenbelag-Fahrzeuge:
Fassungsvermögen
je zwei Trichter

Zwischenankunft:
(0.1)-negexp

- *Ablauf im Mahlwerk:=*
  Zusatzstoffzuführung (bei Erwärmung)+ Mahlen + Entnahme
  als ein geschlossener technolog. Vorgang nach Füllung
- *Nebenbedingung: Energieproblem:=*
  max. 5 solcher Arbeitsgänge können parallel laufen

beladen
leer

Spezifische Probleme

- Zuweisung mehrerer Ressourcen aus einem Spektrum unterschiedlicher **Ressourcenklassen**
  - Stellplätze,
  - Waage,
  - Kran,
  - Trichter (leer, gefüllt, bereit zur Leerung)
  - Energie

- Durchführung unterschiedlicher, aber feststehender **Folgen von Arbeitsgängen** mit Ressourcenforderungen
  a) Stellplatzreservierung – Waage – freier Trichter - Entladung – Waage – Stellplatzfreigabe
  b) Stellplatzreservierung – Waage – voller Trichter - Beladung – Waage – Stellplatzfreigabe
  c) Bereiter Trichter – leeres Fahrzeug- Energie – Produktion

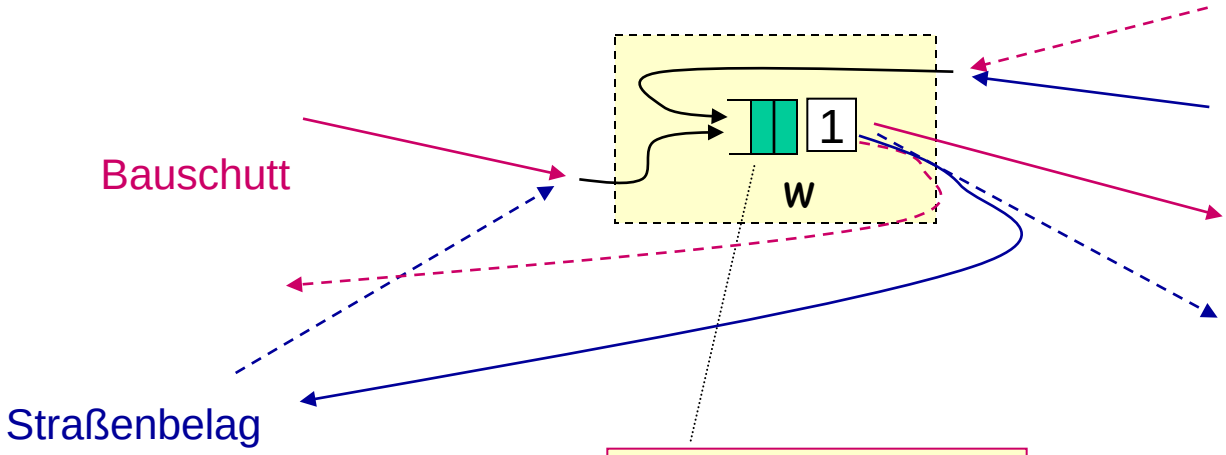
- Unterschiedliche Klassen von **Ankunftsströmen**
  - feste Anzahl von Fahrzeugen mit stochastischer Ankunftszeit
  - unbestimmte Anzahl von Fahrzeugen mit stochastischer Ankunftszeit

Modellierung der Waage (1)

… als

- Res mit impliziter Warteschlange

```
w->acquire(1);
  holdFor(…);
w->release(…);
```



Gesamterfassung

beladen

leer

```
Res    *w;
w= new Res(…,"Waage", 1)
```

keine separaten Warteschlangen für ankommende und verlassende Fahrzeuge

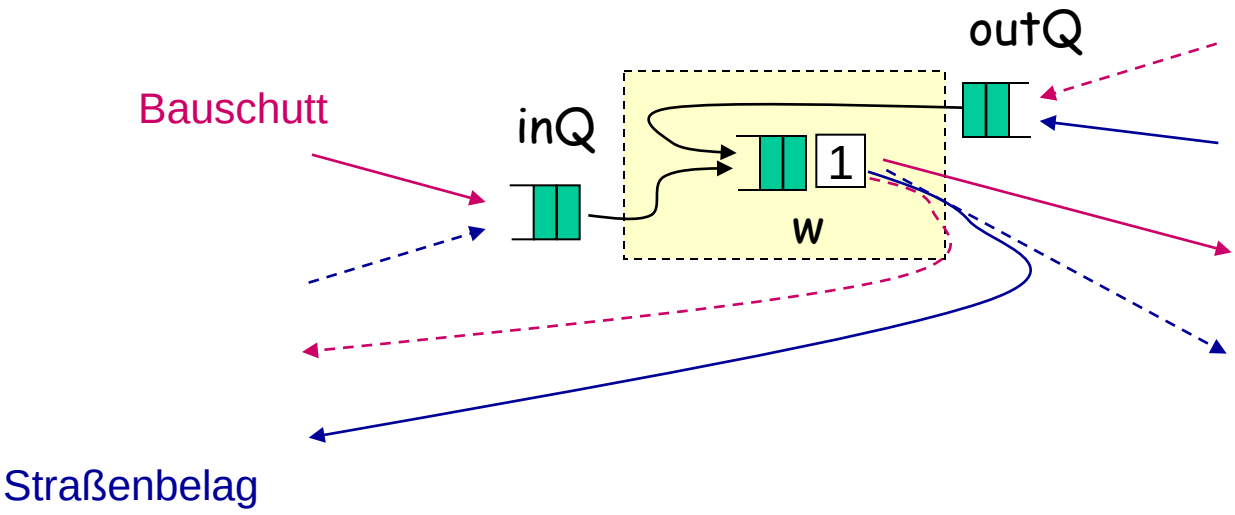
Modellierung der Waage (2)

… als Ensemble aus

- 2 expliziten Warteschlangen
- Res mit impliziter Warteschlange

```
inQ->inSort(getCurrentProcess());
if (! w->getTokenNumber() ) passivate();
                          //---- evtl.Warten ----
w->acquire(1);
inQ->remove(getCurrentProcess());
  holdFor(…);
w->release(1);
```

nicht komplett



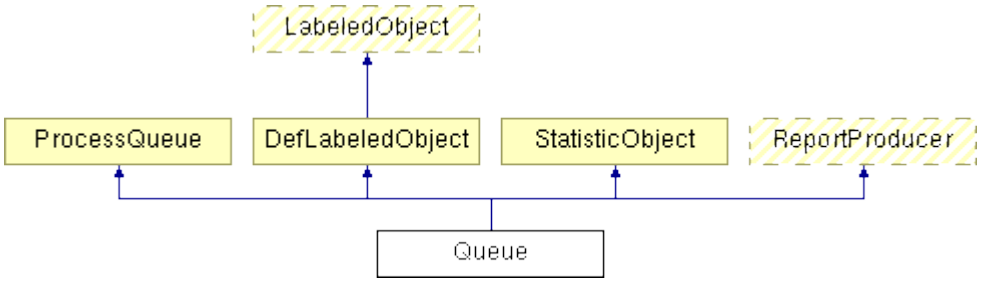
beladen

leer

```
Queue *inQ, *outQ;
Res    *w;
```

Queue als ProcessQueue mit Statistik

Die Klasse Queue



Public Member Functions

Queue (Label l) Contruction for DefaultSimulation.
Queue (Simulation *s, Label l) Contruction for user-defined Simulation.
virtual void popQueue () Remove the first Process from the queue.
virtual void remove (Process *p) Remove a specific Process p from the queue
virtual void inSort (Process *p, **bool** fifo=**true**) add Process p to the queue, considering order and priority
virtual void report (Report *r) generate report

virtual void reset () reset statistics
unsigned int getMin () **const** min queue length
unsigned int getMax () **const** max queue length
unsigned int getNow () **const** current queue length
double getAVLength () **const** average queue length

Modellierung der Waage (3)

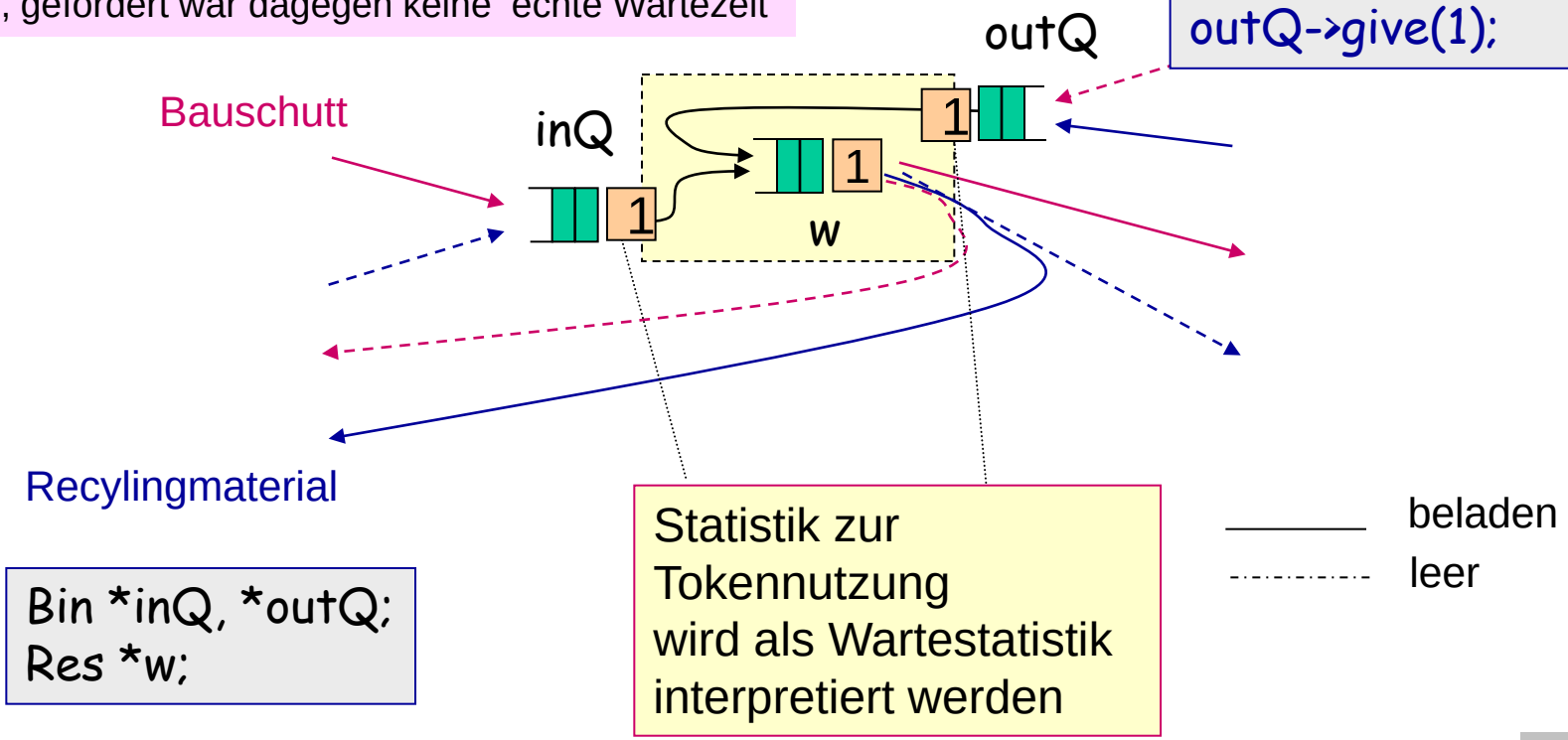
… als Ensemble aus

- 2 Bin-Objekten und 1 Res-Objekt
  (mit jeweils impliziter Warteschlange)

```
inQ->take(1)
  w->acquire(1);
holdFor(…);
  w->release(1);
inQ->give(1);
```

```
outQ->take(1)
  w->acquire(1);
holdFor(…);
  w->release(1);
outQ->give(1);
```

Restproblem:
Interne w-Warteschlange kann max. 1 Process-Objekt enthalten, gefordert war dagegen keine echte Wartezeit



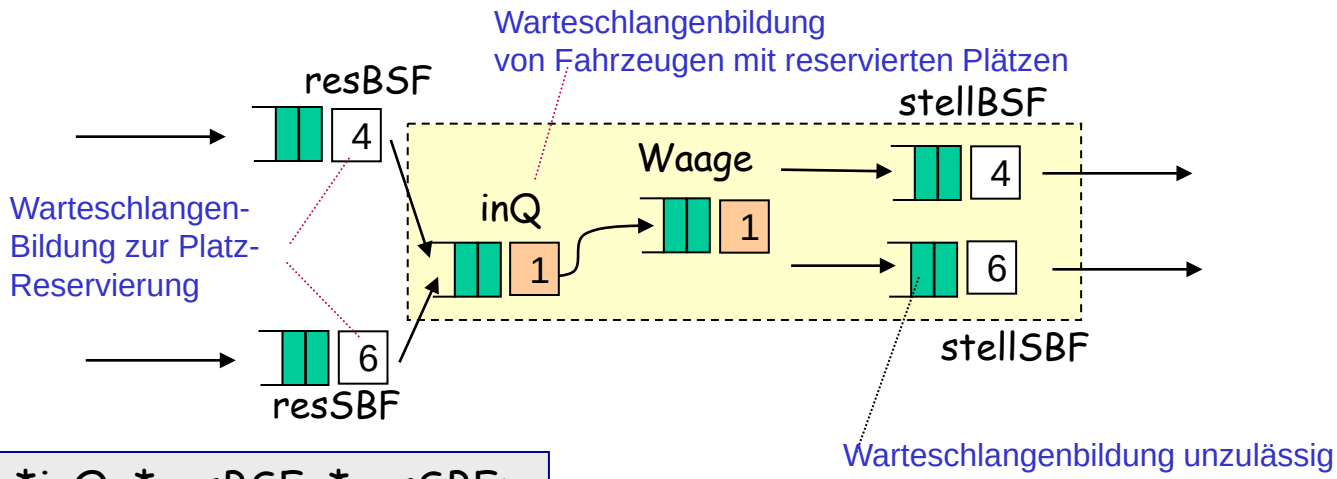

```
Bin *inQ, *outQ;
Res *w;
```

Statistik zur Tokennutzung wird als Wartestatistik interpretiert werden

Modellierung der Stellplätze

BSF=Bauschuttfahrzeuge
SBF=Straßenbelagfahrzeuge

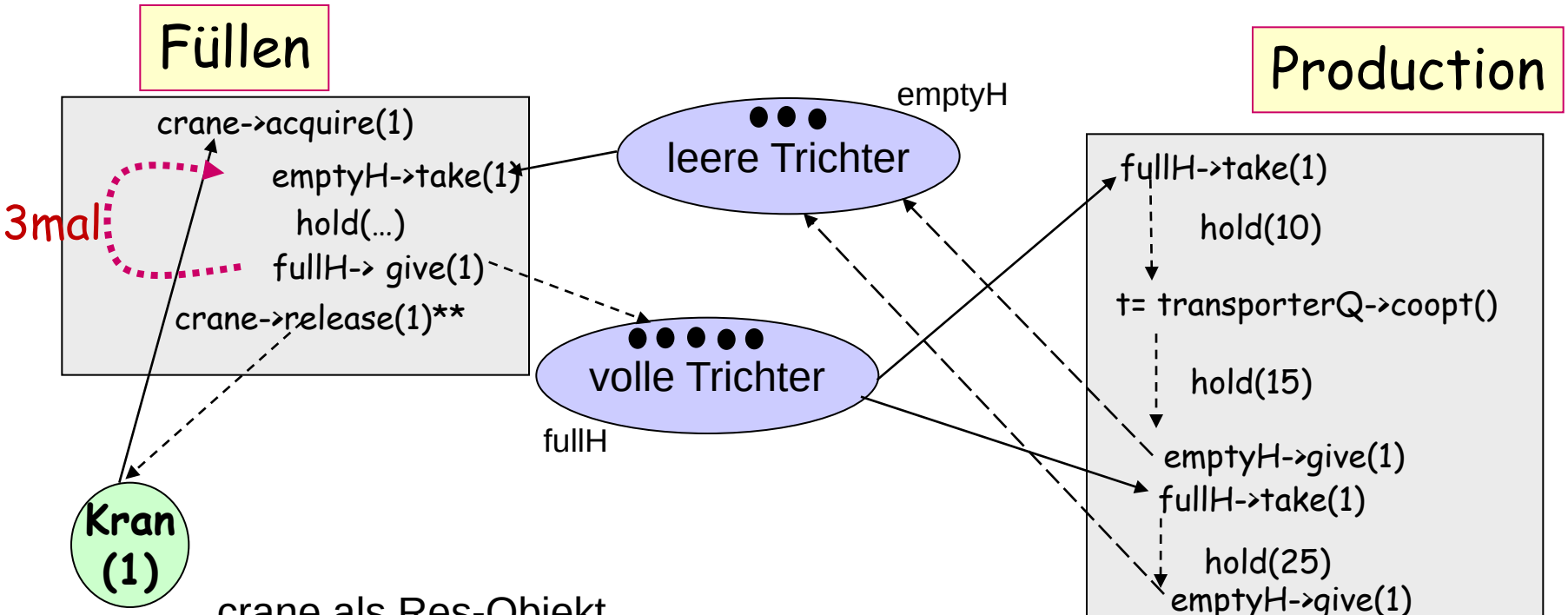
```
resBSF->take(1);        //ziehe Parkplatz-Ticket
inQ->take(1);           //ziehe Waage-Ticket
w->acquire(1);          //Waage-benutzung
holdFor(...);           //Waage-belegung
w->release(1);          //Waage-freigabe
inQ->give(1);           //gebe Waage-Ticket zurück
stellBSF->acquire(1);   //Parklplatzwahl
holdFor(...);           //Parkplatzbelegung
stellBSF->release(1);   //Parkplatzfreigabe
resBSF->give(1);        //gebe Parkplatz-Ticket zurück
```




```
Bin *inQ, *resBSF, *resSBF;
Res *w, *stellBSF, *stellSBF;
```

Modellierung der Trichter

…als feste Anzahl von 8 Token (je Trichter ein Token),
die in Abhängigkeit ihres Zustandes
in unterschiedlichen Bin-Objekten verwaltet werden.



crane als Res-Objekt

Begrenzung der Produktionslinien
→ Anzahl der Objekte= 5

2. ODEMx-Modul Statistik

- Auswertungsmöglichkeiten in ODEMx (Rückblick)
- Berichtserweiterung (Report, Tab) um statistische Profile
- Tab-Spezialisierungen

Experimentauswertungsmöglichkeiten

Allgemeine Vorgehensweise

- Ausführung des Simulationsmodells in Modellzeit,

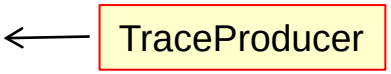
  dabei
  - Beobachtung von Modellgrößen zu bestimmten oder allen möglichen Modellzeitpunkten
  - Analyse der Beobachtungen/ Komprimierung/ Filterung
  - evtl. Einfluss auf weiteren Simulationslauf
  - flexible Zusammenstellung der Daten in Dateien

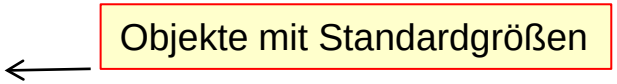
- Ausgabe/Visualisierung nach evtl. weiterer Komprimierung zu bestimmten Modellierungszeitpunkten (z.B. Ende der Simulation)

Auswahl: fest verdrahtet, tatsächliche Aufnahme in ein Trace-File dynamisch

Formen

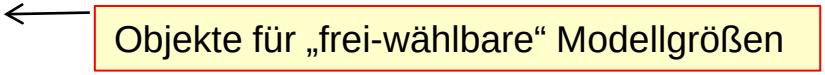
- Trace





- Report

Auswahl: dynamisch



2. ODEMx-Modul Statistik

- Auswertungsmöglichkeiten in ODEMx (Rückblick)
- Berichtserweiterung (Report, Tab) um statistische Profile
- Tab-Spezialisierungen

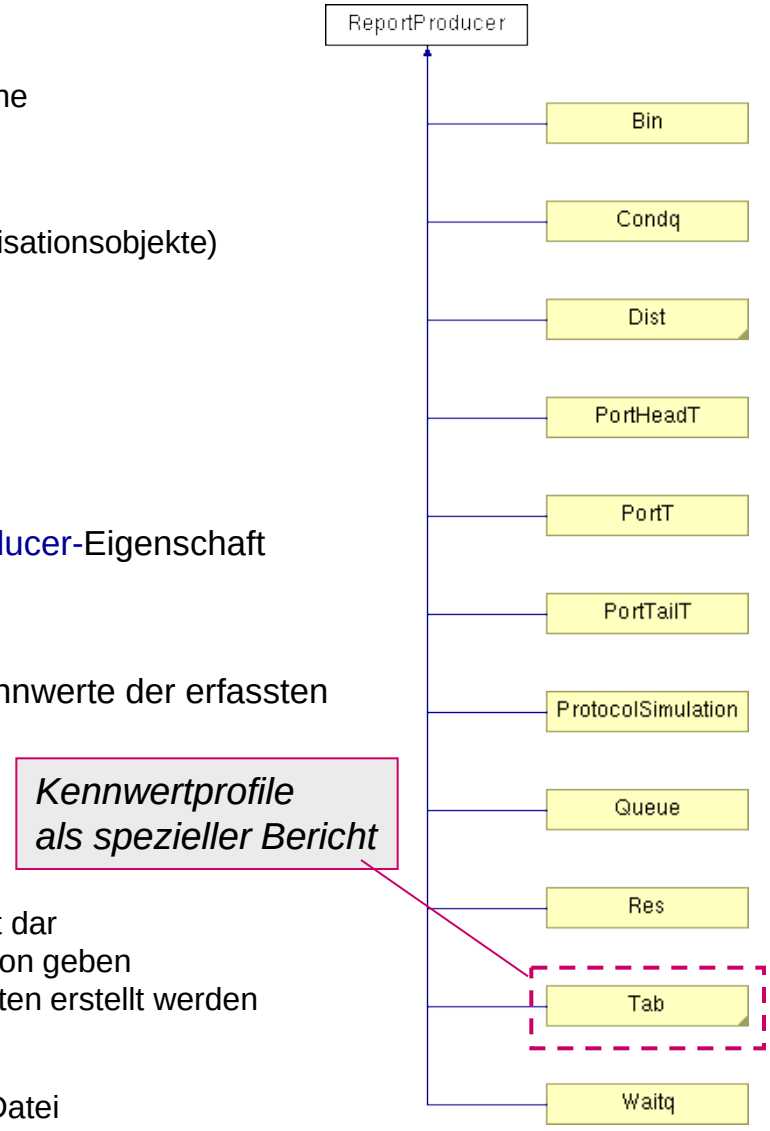
Report (Simulationsbericht)

- **Zweck**
  Klasse für zusammenfassende Berichte über einzelne Modellierungselemente Objekte von
  - Zufallszahlengeneratoren,
  - Prozesswarteschlangen,
  - Ressourcenverwaltungsobjekte, … (Synchronisationsobjekte)
  - Modellvariablen-Beobachter (Tab)

- **Organisation**
  Report verwaltet je Modellelementetyp
  - Tabellen
    zur dynamischen Erfassung der Elemente
  - Tabelleneinträge (Tab) haben ReportProducer-Eigenschaft
    addProducer(), removeProducer()
  - Berichtserzeugung
    strukturierte Ausgabe der spezifischen Kennwerte der erfassten Elemente
  - Rücksetzen von Kennwerten (Reset)

- **Ausgabedatei**
  jedes Report-Objekt stellt einen individuellen Bericht dar
  - es kann mehrere **Report**-Objekte pro Simulation geben
  - Berichte können zu beliebigen Modellzeitpunkten erstellt werden

- **HtmlReport**
  Spezialisierung von Report: Ausgabedatei ist Html-Datei



Berichtsauszug (Beispiel Autofähre)

Random Number Generators

Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3
mainland	0	Negexp	95	33427485	0.1	0	0

Bin Statistics

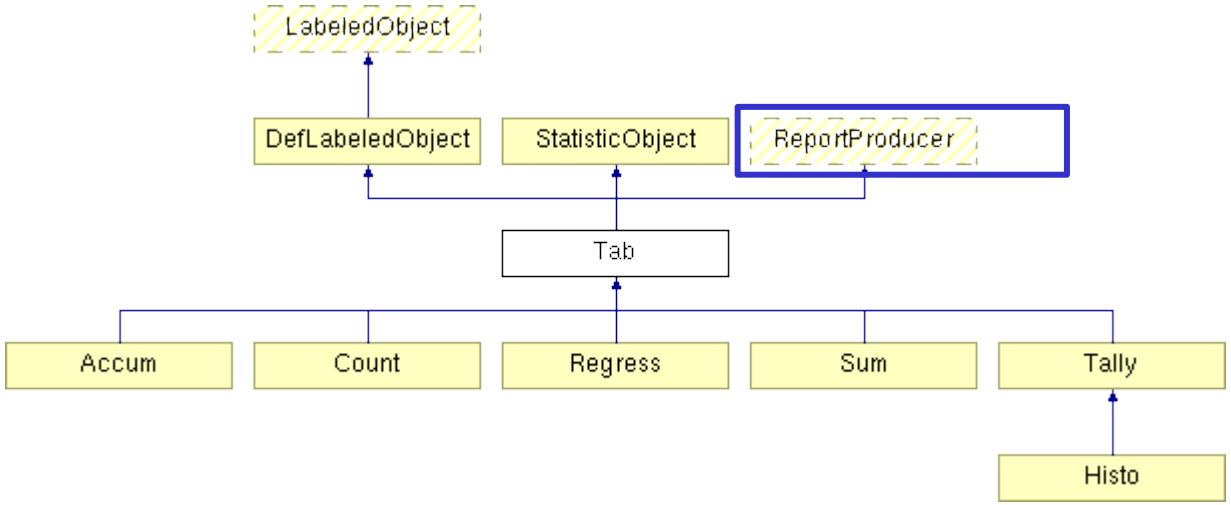
Name	Reset at	Queue	Users	Provider	Init number of token	Min number of token	Max number of token	Now number of token	Avg number of token	Avg waiting time
mainland_1	0	mainland_queue	92	94	3	0	7	2	1.99533	0

Queue Statistics

Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length
mainland_queue	0	0	1	0	0

Report-Objekt hat zum Zeitpunkt der Dateierzeugung drei Objekte von drei verschiedenen Typen: Dist, Bin, Queue
(wobei das Queue-Objekt die Warteschlange vom Bin-Objekt darstellt

Erstellung von Kennwertprofilen



- Einrichtung eines Objektes
  bei Zuordnung einer oder mehrerer Modellvariablen (int/double)
- Beobachtung der Modellvariablen mit Erfassung der Werteänderung
- Bestimmung des spezifischen Kennwertprofils

Stichprobenerstellung/
Beobachtung

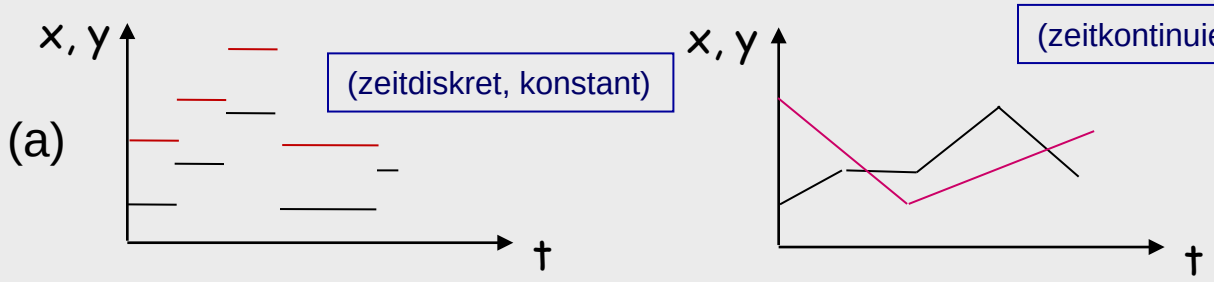
Statistische
Auswertung

Profile zu beobachtender Modellgrößen

int- **oder** **double-** Modellgrößen x, y mögen sich im Laufe der Simulation ändern

(z.B. x Member der Klasse X,
y Member der Klasse Y

unterstützte Arten von Werterfassungen (Beobachtungen)



man erfasst nur Stützwerte und nimmt lineare Übergänge an

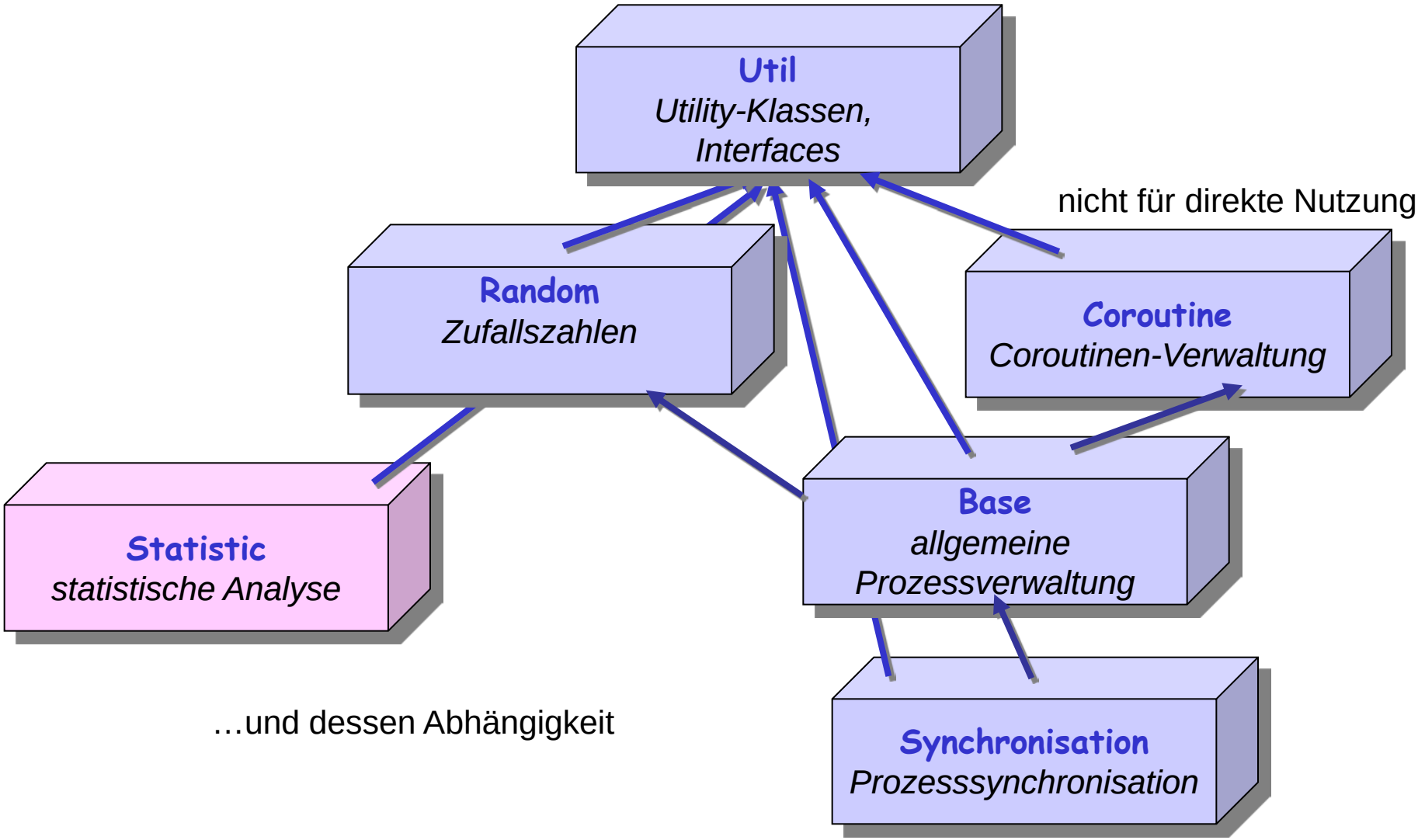
(1) Modellbeobachtungen: $x_1, x_2, x_3, \ldots \quad y_1, \ldots$

(2) Modellbeobachtungen: $(x_1, t_1), (x_2, t_2), (x_3, t_3)$

(3) Modellbeobachtungen: $(x_1, y_1)^{t1}, (x_2, y_2)^{t2}, (x_3, y_3)^{t3}, \ldots$

typische Profile enthalten Mittelwert, Standardabweichung, Minimum, Maximum

Der ODEMx- Modul Statistic

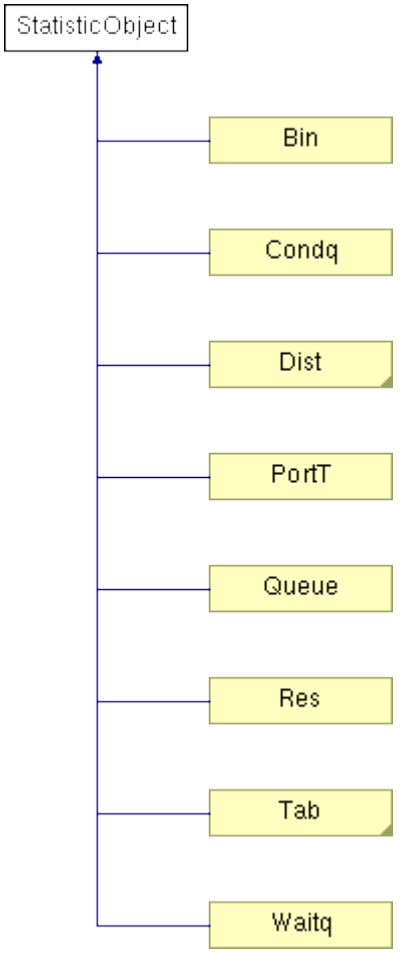


…und dessen Abhängigkeit

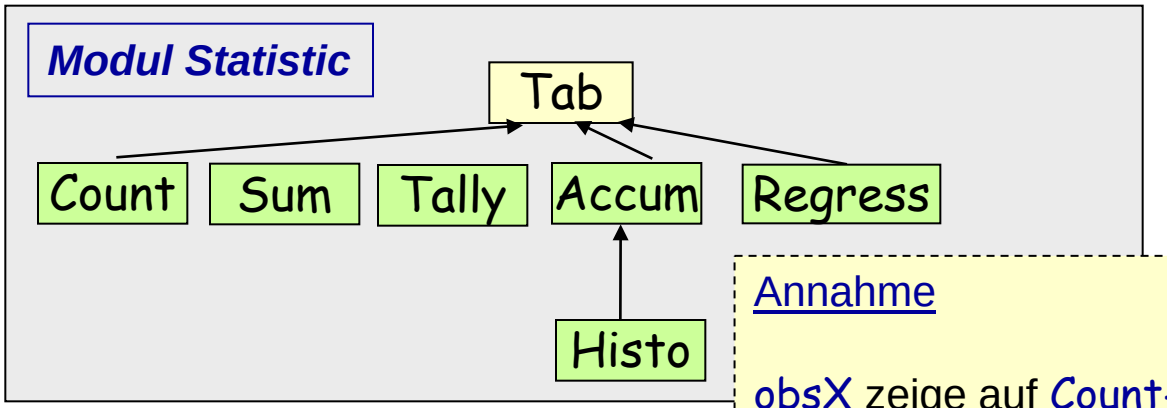
StatisticObject

– ermöglicht
  Rücksetzen von Kennwerten (Reset)
  der in lokalen Tabellen von Report erfassten
  Elemente

(bei Dist-Objekten: wird nur die Anzahl der bisherigen Aufrufe auf Null zurückgesetzt, die zuletzt generierte (0,1)-Basiszufallszahl wird nicht auf den Startwert zurückgesetzt



Anwendung der Statistic-Klassen



Annahme

obsX zeige auf Count-,..,Histo-Objekt

obsXY zeige auf Regress-Objekt

einheitliches Nutzungsschema

- pro Variable x bzw. Variablenpaar (x,y) ist
  - ein Beobachter-Objekt obsX bzw. obsXY zu konstruieren,
  - explizite Erfassung der x-Werte in obsX bzw. der (x,y)-Werte in obsXY
- zu diskreten Zeitpunkten wird x beobachtet: obsX->update(x)

  oder x und y: obsXY->update(x,y)
- zu gewissen Zeitpunkten können die Profile der Größen als Tabellen in Reports generiert werden: obsX->report(), ...
- zu gewünschten Zeitpunkten können Einschwingphasen ausgeblendet werden: obsX->reset(), ...

Die abstrakte Klasse Tab

```
class Tab : public DefLabeledObject,
                                public StatisticObject,
                                public virtual ReportProducer

    {
    public:
            Tab (Simulation* s, Label label="");
            virtual ~Tab();

    protected:
            void update() {obs++;}
            virtual void reset() {obs=0; StatisticObject::reset();}

            unsigned int getObs() const {return obs;}

    protected:
            Simulation* env;
            unsigned int obs;
            SimTime resetAt;
    };
```

erst Ableitungen stellen eigentliche update()- Funktionalität mit spezifischer Signatur bereit

per Überladung, nicht per Redefinition

8. ODEMx-Modul Statistik

- Auswertungsmöglichkeiten in ODEMx (Überblick)
- Berichte (Report, Tab)
- Tab-Spezialisierungen

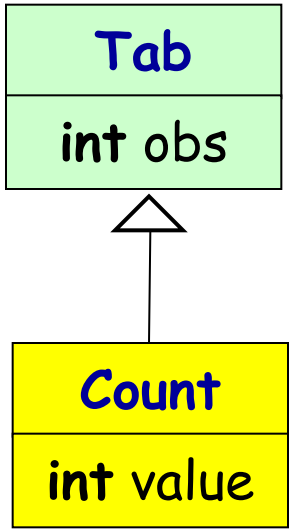
1. Profilbestimmung von Modellgrößen mit Count

- **Vor.:** zeitdiskrete Variable vom Typ **int**
- **Funktion**: Zähler
  Bestimmung des Profils zu einem beliebigen Zeitpunkt
- **Kennwertprofil**
  Stichprobenumfang (Anzahl von Änderungen), aktueller Wert
- **Beobachtung**

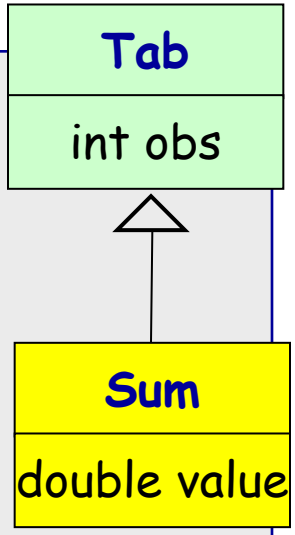
```
Simulation *sim;

Count *c= new Count(sim, "Zähler");

c->update (-3); // bei einer Reduktion von c um 3
```

Die Klasse Count

```
class Count : public Tab {
    public:
        Count (Simulation* s, Label title="");
        virtual ~Count();

        void update (int v=1) {
                Tab::update();
                value += v;
        };
        virtual void reset() {
                Tab::reset();
                value = 0;
        };
        int getValue() {return value;};
        virtual void report(Report* r);
    private:
        int value;
    };
```



2. Profilbestimmung von Modellgrößen mit Sum



- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Summenbildung
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

- **Kennwertprofil**
  Stichprobenumfang (Anzahl von Änderungen),
  aktueller Wert

- **Beobachtung**

```
Simulation *sim;

Sum *s= new Sum(sim, "Menge");

s->update (15.3); // bei einer Erhöhung von s um 15.3
```

3. Profilbestimmung von Modellgrößen mit Tally

- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

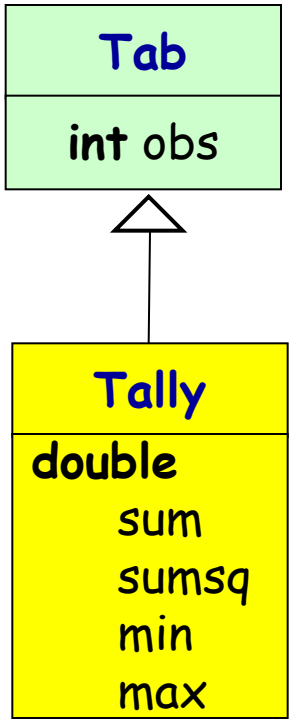
- **Tally-Kennwertprofil**
  unabhängig von der jeweiligen Dauer der Wertebelegungen
  Stichprobenumfang, Min, Max, Mittelwert, Standardabweichung

- **Beobachtung**

```
double x;
Simulation *sim;

Tally *t= new Tally(sim, "Wartezeiten");

t->update (x); // x gemessene Wartezeit eines Autos
               // Aufruf für jedes Auto nach Beendigung des Wartens
```

Die Klasse Tally

```
class Tally : public Tab {
    public:
        Tally (Simulation* s, Label title="");
        virtual ~Tally();

        virtual void update (double v);
        virtual void reset();

        unsigned int getSize()
            {return getObs();}
        double getMin() {return min;}
        double getMax() {return max;}
        double getMean()
            {return getObs() ?
                sum/getObs() : 0.0;}
        double getDivergence();
        virtual void report(Report* r);

    protected:
        double sum, sumsq, min, max;
};
```



kommen nur mit 4 Speicherplätzen aus

Die Klasse Tally

```
class Tally : public Tab {
    public:
        Tally (Simulation* s
        virtual ~Tally();

        virtual void update
        virtual void reset()

        unsigned int getSi
            {return getOb
        double getMin() {re
        double getMax() {r
        double getMean()
            {return getOb
                sum/getC
        double getDiverge
        virtual void report

    protected:
        double sum, sumsq;
};
```

5 Speicherplätze zur Profilbestimmung

- Anzahl der Beobachtungen
- Summe der bisher beobachteten Werte
- Summe der Quadrate der bisher beobachteten Werte
- Minimum
- Maximum

Berechnung erfolgt bei Aufruf von getMean, getDivergence

Mittelwert m

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Streuung/Varianz s^2

Standardabweichung s

$$s^2 = \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n} \sum_{i=1}^{n} x_i \right)^2 \right)$$

Vorteilhafte Berechnung der Standardabweichung

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Definition der Standardabweichung s,
ungünstig zu beliebigen Zeitpunkten zu berechnen
(s2 ist die Streuung)

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2) = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - 2\sum_{i=1}^{n}x_i\,\bar{x} + \sum_{i=1}^{n}\bar{x}^2\right)$$

$$s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - 2n\bar{x}^2 + n\bar{x}^2\right) = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - n\bar{x}^2\right)$$

Summe der Quadrate
der beobachteten Werte

Summe der
beobachteten Werte

$$s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - n\left[\frac{1}{n}\sum_{i=1}^{n}x_i\right]^2\right) = \frac{1}{n-1}\left(\sum_{i=1}^{n}x_i^2 - \left[\frac{1}{n}\sum_{i=1}^{n}x_i\right]^2\right)$$

Die Klasse Tally

```
class Tally : public Tab {
    public:
        Tally (Simulation* s, Label title="");
        virtual ~Tally();
        virtual void update (double v);
        virtual void reset();
        unsigned int getSize()
              {return getObs();}
        double getMin() {return min;}
        double getMax() {return max;}
        double getMean()
              {return getObs() ?
                  sum/getObs() : 0.0;}
        double getDivergence();
        virtual void report(Report* r);

    protected:
        double sum, sumsq, min, max;
};
```

```
void Tally::update (double v) {
        Tab::update();

        sum += v;
        sumsq += v*v;

        if (getObs() == 1) min = max = v;
        else if (v < min) min = v;
        else if (v > max) max = v;
}
```

```
void Tally::reset() {
        Tab::reset();
        sum = sumsq = min = max = 0.0;
}
```

```
double Tally::getDivergence() {
        if (getObs()<=1)
                return 0.0;

        double v=fabs(sumsq/getObs() -
                getMean()*getMean());
        return sqrt(v);
}
```

$$s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \left[\frac{1}{n}\sum_{i=1}^{n} x_i\right]^2\right)$$

Profilbestimmung von Modellgrößen mit Accum

- **Vor.:** a) zeitdiskrete Variable vom Typ double
  oder
  b) zeitkontinuierliche Variable vom Typ double
- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

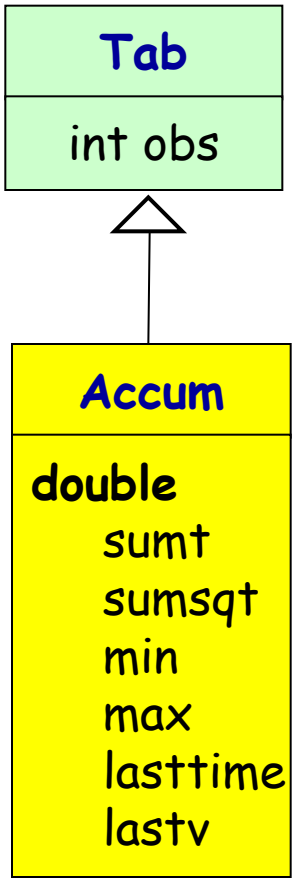
- **Accum-Kennwertprofil**
  abhängig von der jeweiligen Dauer der Wertebelegungen
  Stichprobenumfang, Min, Max, Mittelwert, Standardabweichung

- **Beobachtung**

```
double x /*zeitdiskret*/ , y /*zeitkontinuierlich*/;
Simulation *sim;

Accum *a1= new ACCUM(sim, "Warteschlangenlänge");
a1->update (x); // bei jeder Änderung von x

Accum *a2= new ACCUM(sim, "Tankinhalt");
a2->integrate (y); // Annahme eines linearen Übergangs von y zwischen zwei
                   // Beobachtungen
```

Die Klasse Accum

```
class Accum : public Tab {
    public:
        Accum (Simulation* s, Label title="");
        virtual ~Accum();

        virtual void update (double v);
        virtual void integrate (double v);

        virtual void reset();
        unsigned int getSize() const
                                {return getObs();}
        double getMin() const {return min;}
        double getMax() const {return max;}
        double getMean() const;
        double getDivergence() const;

        virtual void report(Report* r);

    protected:
        double sumt, sumsqt, min, max,
               lasttime, lastv;
    };
```



Die Klasse Accum

```
class Accum : public Tab {
    public:
        Accum (Simulation* s, Label title="");
        virtual ~Accum();

        virtual void update (double v);
        virtual void reset();
        unsigned int getSize() const
                              {return getObs();}
        double getMin() const {return min;}
        double getMax() const {return max;}
        double getMean() const;
        double getDivergence() const;
        virtual void report(Report* r);

    protected:
        double sumt, sumsqt, min, max,
               lasttime, lastv;
    };
```

```
void Accum::update (double v) {
        double now, span;
    //Zeitintegral einer Treppenfunktion
        Tab::update();
        now     = env->getTime();
        span    = now - lasttime;
        lasttime= now;

        if (getObs()>1) {
            sumt   += lastv * span;
            sumsqt += lastv * lastv * span;
        }
        lastv   = v;

        if (getObs() == 1) min = max = v;
        else if (v < min) min = v;
        else if (v > max) max = v;
}
```

```
void Accum::reset() {
        Tab::reset();
        sumt = sumsqt =
                min = max = lastv = 0.0;
        lasttime = env->getTime();
}
```

Profil von Modellgrößen mit Histo

- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

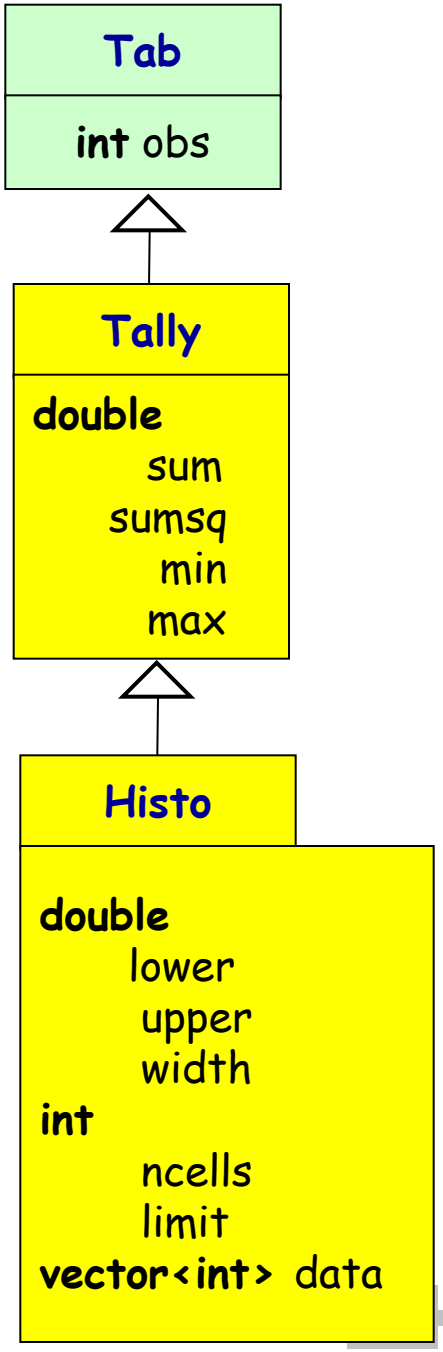
- **Kennwertprofil** (wie bei Tally)
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  mit zusätzlicher Erfassung in vorgegebene Werteklassen

- **Beobachtung**

```
double x;
Simulation *sim;

Histo *h= new Histo(sim, "Wartezeiten", 1.0, 300.0, 25);
h->update (x); // bei jeder Änderung von x
```

Die Klasse Histo

```
class Histo : public Tally {
public:
        Histo(Simulation* s, Label title,
                        double low, double up, int n);
        virtual ~Histo();
        virtual void update(double v);
        virtual void reset();

        const Tally* getTally() {//cast
          return reinterpret_cast<const Tally*> (this);}

        const std::vector<int>& getData() {return data;}
        int maximumelem();
        virtual void report(Report* r);

protected:
        double lower, upper;
        int ncells;
        std::vector<int> data;
        int limit;
        double width;
};
```



Die Klasse Histo

```
class Histo : public Tally {
public:
        Histo(Simulation* s, Label title,
                        double low, double
        virtual ~Histo();
        virtual void update(double v);
        virtual void reset();
        const Tally* getTally() {
           return reinterpret_cast<const T
        const std::vector<int>& getData() {r
        int maximumelem();
        virtual void report(Report* r);

protected:
        double lower, upper;
        int ncells;
        std::vector<int> data;
        int limit;
        double width;
};
```

Basistyp

```
Histo::Histo(Simulation* s, Label title,
   double low, double upp, int n):
   Tally(s, title), data(n+2)
{
      lower=low;
      upper=upp;
      ncells=n;

      if (upper==lower) {
         upper=lower+100.0;
         warning("Histo: borders are equal; ...");

      }

      if (upper < lower) {
         std::swap(lower, upper);
         warning("Histo: upper borders < ...");
      }

      if ((ncells < 1)) {
         warning("Histo: number of cells < 1 ...");
         ncells=10;

         data.resize(ncells+2);
      }

      width = (upper-lower)/ncells;
      limit = ncells+1;
};
```

Kontruktor mit Dimensionsangabe

Erweiterung (evtl. Umspeicherung)

Profil von Modellgrößen mit Regress

- **Vor.:** Paar zeitdiskreter Variablen vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen des Variablenpaares (x, y)
  Bestimmung einer angenommenen linearen Abhängigkeit für ein Beobachtungsintervall

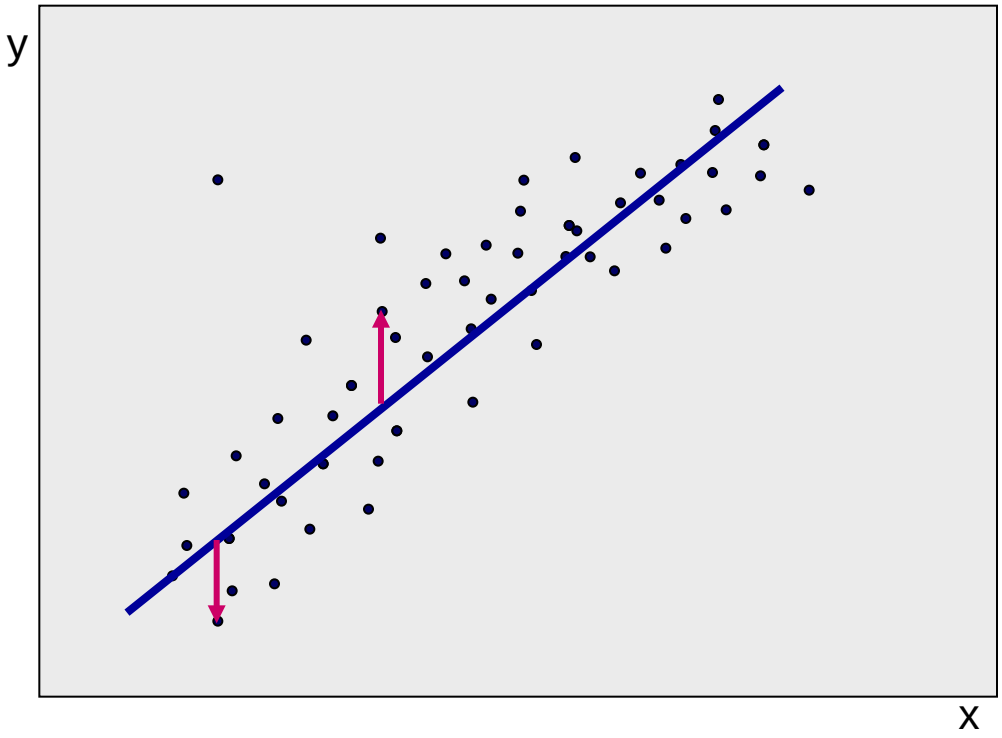
- **Kennwertprofil**
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  Parameterschätzung des linearen Zusammenhangs:
  - Erwartungswert, Standardabweichung für
    Schätzungen von m und b (bei Annahme von y= mx + b),
  - Regressionskoeffizient

- **Beobachtung**

```
double x, y;
Simulation *sim;
Regress *r= new regress ("Abh", sim);
r->update (x,y);  //bei jeder Änderung von x oder y
```

Lineare Regressionsanalyse

Punkteschwarm (als Ergebnis einer mit Messfehlern behafteten Beobachtung)

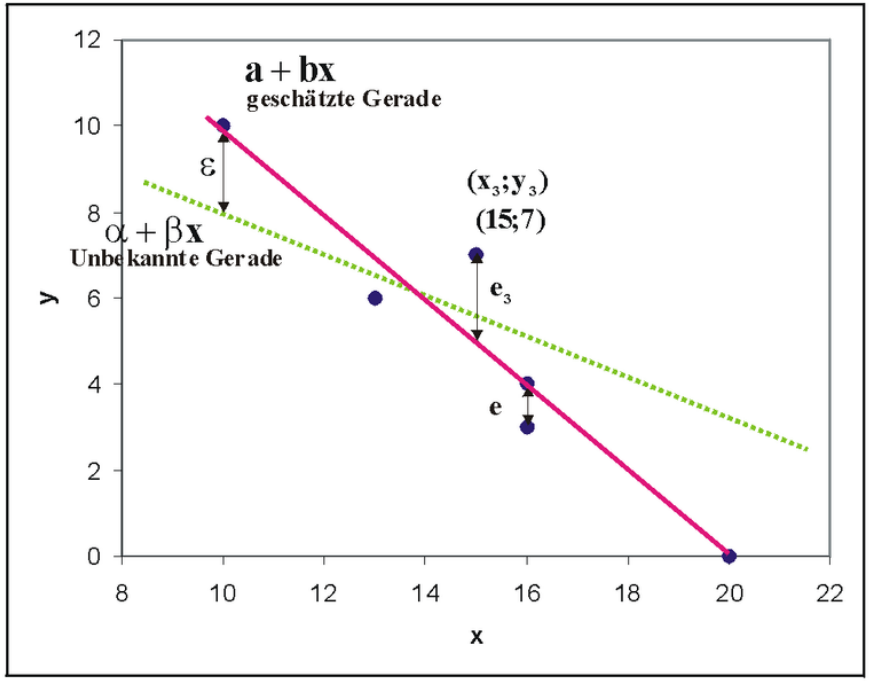


Annahme:
bei festem (aber beliebigen) x
ist y normalverteilt !!!

Koeffizienten
eines angenommenen linearen Zusammenhangs müssen geschätzt werden

Koeffizienten für $y = mx + b$

so bestimmen, dass die Summe der einzelnen quadrierten Abstände der Punkte(Beobachtungen) zur angenommenen Geraden minimal wird

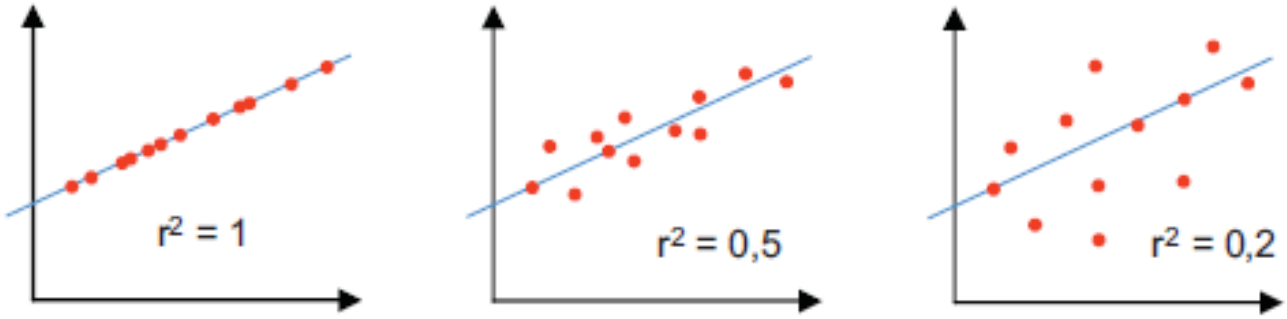


$$a = \bar{y} - b\bar{x}$$

$$b = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Korrelation

Punktwolken haben die gleiche Regressionsgerade,
aber Güte der Korrelation ist bei der ersten am höchsten.



$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

$$r_{xy} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sqrt{\left(\sum_{i=1}^{n} x_i^{\,2} - n\bar{x}\right)\left(\sum_{i=1}^{n} y_i^{\,2} - n\bar{y}\right)}}$$

Regressionskoeffizient
$-1 \leq r \leq 1$

Maß für Zusammenhang von x und y

Regressions- oder Korrellationskoeffizient

- **Wert: +1 (bzw. −1)**
  → vollständig positiver (bzw. negativer) linearer Zusammenhang zwischen den betrachteten Merkmalen

- **Wert: 0**
  → Merkmale hängen überhaupt nicht linear voneinander ab.

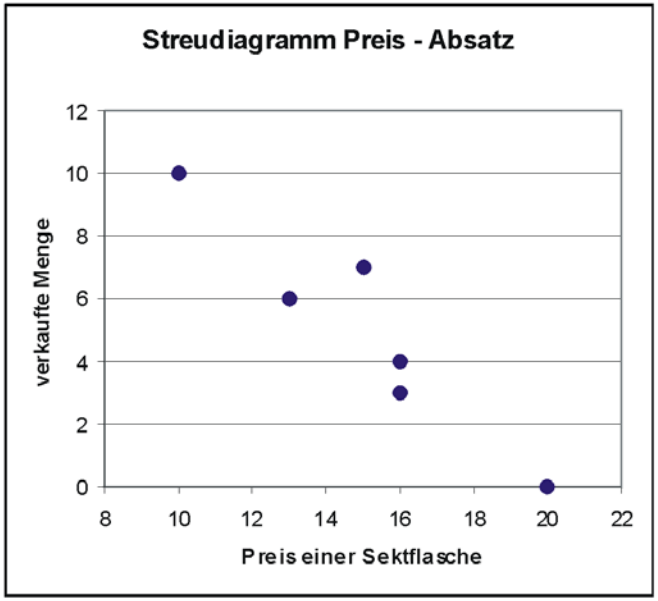
  Allerdings können x und y in *nicht-linearer* Weise voneinander abhängen.

  Achtung: der Korrelationskoeffizient ist kein geeignetes Maß für die (reine) stochastische Abhängigkeit von Merkmalen

Laden	i	1	2	3	4	5	6
Preis einer Flasche	x_i	20	16	15	16	13	10
verkaufte Menge	y_i	0	3	7	4	6	10

$\bar{x} = 15$

$\bar{y} = 5$



i	Flaschenpreis x_i	verkaufte Menge y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})(x_i - \bar{x})$	$(y_i - \bar{y})(y_i - \bar{y})$	$\hat{y}_i$
1	20	0	5	-5	-25	25	25	0,09
2	16	3	1	-2	-2	1	4	4,02
3	15	7	0	2	0	0	4	5,00
4	16	4	1	-1	-1	1	1	4,02
5	13	6	-2	1	-2	4	1	6,96
6	10	10	-5	5	-25	25	25	9,91
Total	90	30	0	0	-55	56	60	30,0

Die geschätzte Regressionsgerade lautet somit $\hat{y}_i = 19{,}73 + (-0{,}98) \cdot x_i$