VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- ODEMx- Elemente -

Joachim Fischer

Inhalt letzter Vorlesung

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen

ODEMx- Modellelemente zur Synchronisation (Wdh.)

- **Res** (abstrakte Token) ~ Storage
- **Res-Template**, Token mit Objektidentität ~ keine Entsprechung
- **Bin** (abstrakte Token) ~ keine Entsprechung
  Spezialfall: Bin oder Res mit 1 Token ~ LogicSwitch
- **Bin-Template**, Token mit Objekt-Identität ~ keine Entsprechung
- **WaitQ** (Master-Slave-Synchronisation) ~ keine Entsprechung

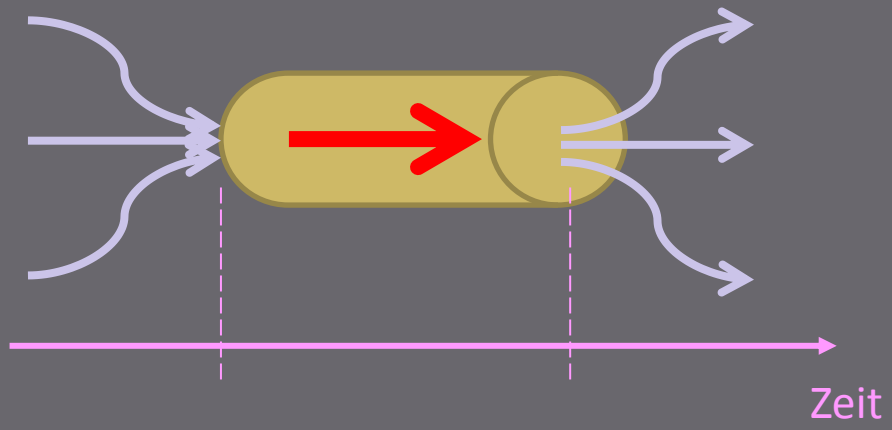
betrachten wir nun näher

- **Asynchrone Kommunikationspuffer** ~ keine Entsprechung
- **CondQ** ~ wait until + Control-Variablen
  (SLX-Lösung aber ausgereifter)

Nützliches Modellierungsmuster (Wdh.)

- n Objekte aktiver Klassen (Prozesse) kooperieren eine bestimmte Zeit

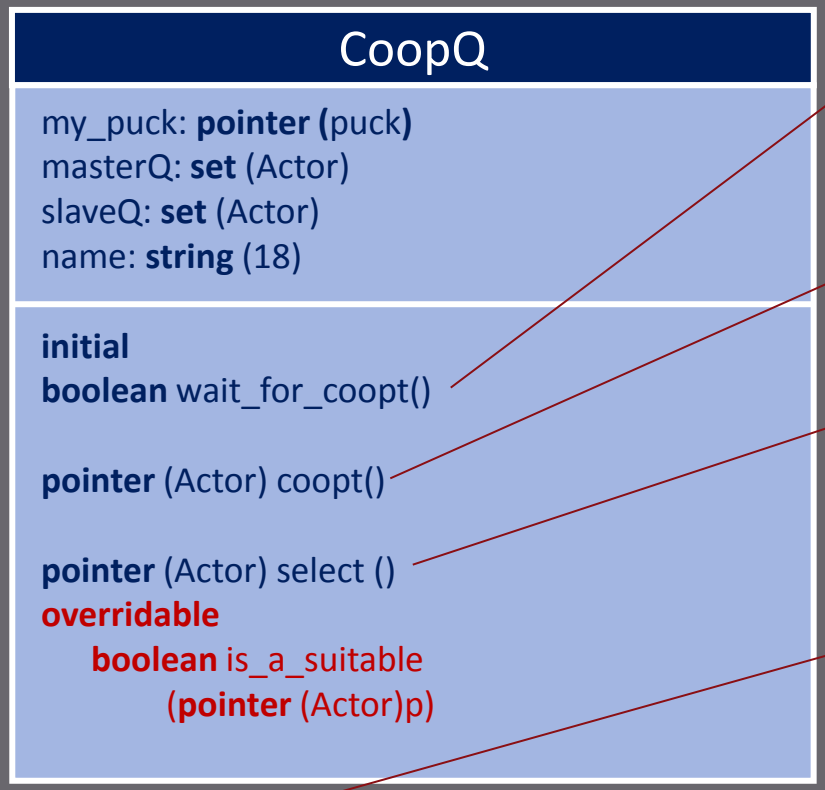
  d.h.: Zustandsänderungen der Objekte sind voneinander abhängig!



- Effiziente simulative Umsetzung auf einer Ein-Prozessor-Maschine
  - Annahme: alle n Objekte sind für die Kooperation bereit
  - genau eines der n Objekte übernimmt als Master die Ausführung sämtlicher Zustandsänderungen (aktiv) in Abhängigkeit der Modellzeit
  - alle anderen Objekte warten (passiv) als Slave auf die Beendigung der Kooperation durch den Master

ACHTUNG: Master und Slave sind Rollen, die Objekte zeitweilig spielen

CoopQ: Grundfunktionalität (Wdh.)



a) Aufrufer wird zum **Slave**
aktiviert den ersten wartenden Master

b) Aufrufer wird zum **Master**
wählt den ersten wartenden Slave

c) Aufrufer wird zum **Master**
wählt den ersten wartenden Slave
für den „is_a_suitable“ TRUE liefert

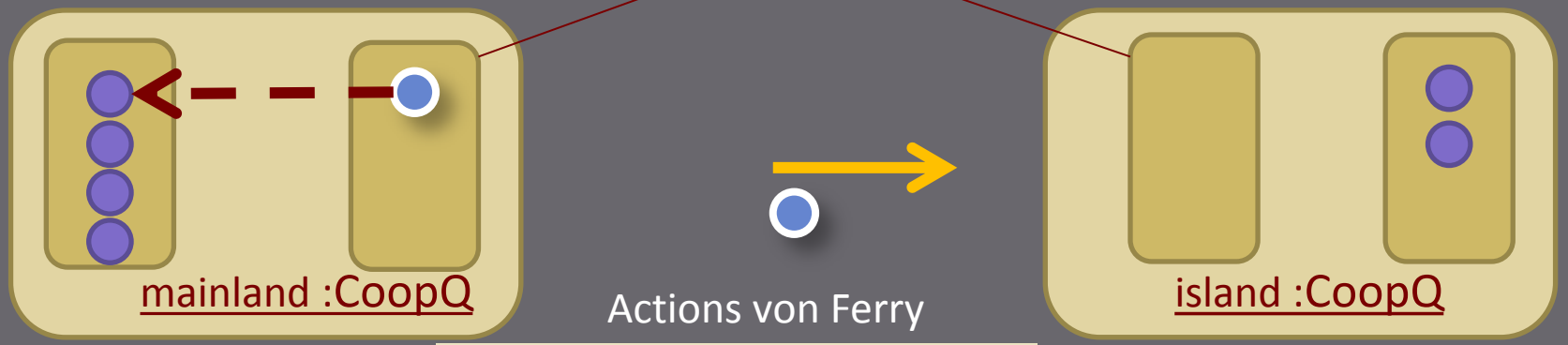
d) Aufrufer wird zum **Master**
wählt den ersten wartenden Slave
für den „Bedingung-an-Slave “ TRUE ist

find *Slave-Pointer* **atleast** Typ **in** CoopQ-Object **with** Bedingung-an-Slave;

*Synchronisation der Kooperationspartner (**ohne** spezifische Bedingungen)*

Beispiel Fähre: Einsatz von zwei CoopQ-Objekten (Wdh.)

Master-Listen für die Fähren-Objekte



Actions von Ferry

```
...
v= mainland->coopt();
... // Fortsetzung der Aktionen
    // bei Ankunft in island
```

Actions von Vehicle

```
...
mainland->waitForMaster();
... // Fortsetzung der Aktionen
    // bei Ankunft in island
```

Actions von Vehicle

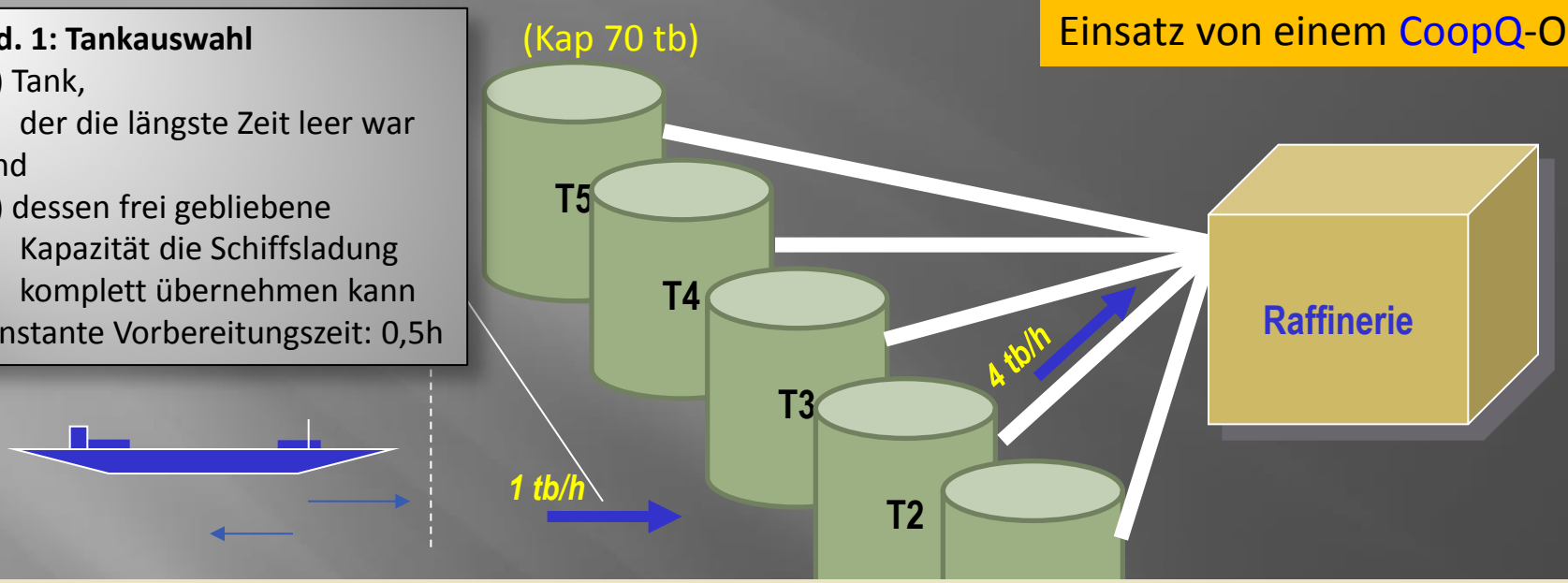
```
...
island->waitForMaster();
... // Fortsetzung der Aktionen
    // bei Ankunft in mainland
```

CoopQ funktioniert insbesondere für mehrere Master (hier: Fähren) !

*Synchronisation der Kooperationspartner (**mit** spezifischer Slave-Bedingungen)*

Beispiel: Tanker – Tank – Raffinerie

Bed. 1: Tankauswahl
(1) Tank,
der die längste Zeit leer war
und
(2) dessen frei gebliebene
Kapazität die Schiffsladung
komplett übernehmen kann
konstante Vorbereitungszeit: 0,5h



Einsatz von einem CoopQ-Objekt

```
class Tank_Management subclass (CoopQ) {
        override method is_a_suitable (pointer (Actor) p) returning boolean {
                pointer (Tanker) tanker;
                pointer (Tank) tank;
                tanker= ACTIVE->puck_object;  //caller is a Tanker object
                tank= p;                      // all slaves have to be of type tank
                return tanker->load   <   tank->freeCap;
        };
}
```

Tanker wird Aufrufer von select(), select wendet is_a_suitable auf Tank-Objekte an

```
method select() returning pointer(Actor) {
        pointer (Actor) slave;
        pointer(Actor) master;
        pointer (puck) master_puck;

        master_puck= ACTIVE;  // caller becomes a master
        master= master_puck->puck_object;


        place master into  masterQ;

        forever {
                for (slave= each Actor in slaveQ with is_a_suitable (slave)) {
                        remove slave from slaveQ;
                        remove master from masterQ;
                        return slave; //by the first slave which was found
                } // for
                slave=NULL; //no slave was found

                wait;
                // continue when activation by a new arrived SLAVE or TIMEOUT

                if (master->waitInterrupted) {
                        master->waitInterrupted= FALSE;
                        return NULL;
                }
        }
}; //cond find
```

CoopQ

my_puck: **pointer (**puck**)**
masterQ: **set** (Actor)
slaveQ: **set** (Actor)
name: **string** (18)

initial
pointer (Actor) coopt()
boolean wait_for_coopt()
pointer (Actor) select ()
overridable
boolean is_a_suitable
(**pointer** (actor)p)

SLX-Modellentwicklung:

Tank→Slave, Tanker→Master

```
class Tank (double l) subclass (Actor) {
        static double maxCap= 70;
        double freeCap;

        initial {
            freeCap= l;
        }

        actions {
            my_puck= ACTIVE;

            forever {
              while (freeCap > 20.0) {
                    tankControl.wait_for_coopt();
                }
                advance (1/4*(maxCap-freeCap));
                freeCap= maxCap;
            }
        }
}
```

Bed.2 ist Tank nahezu voll (≈70%), wird Öl zur Raffinerie abgepumpt

Reaktivierung durch Tanker (Master) nach beendeter Ölübernahme

```
class Tanker  subclass  (Actor) {
        static rn_stream size;
        double load;
        pointer (Tank) my_tank;

        initial {
            load= 10+
             rv_discrete_uniform  (size, 1, 3) * 5;
        }

        actions {
            my_puck= ACTIVE;
            my_tank= tankControl.select();
        // begin of unloading
            tank_report(my_tank, ME);
            advance (load);
            my_tank->freeCap -= load;
            reactivate my_tank->my_puck;
        // end of unloading
        }
}
```

Bestimmung verfügbarer Tanks (Slaves) per virtueller Methode

SLX-Modellentwicklung: Tanker-Ankunft

Zwischenankunftszeit:
n.-exponential verteilt
Mittelwert 8h

```
// 1 ZE= 1 h
rn_stream arrivals;

random_input arrive_time=  rv_expo( arrivals , 8.0 )
                                    histogram start=0 width=1 count=24;
```

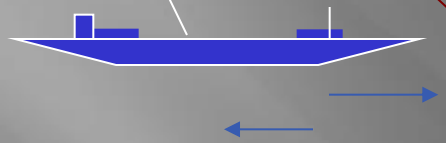
```
class Arrival ()  {
        int tankers;

        actions {
                forever {
                        tankers++;
                        activate new Tanker; // Create new Tanker at 0.0
                        advance sample_arrive_time() ; // ca. 8h
                };
        }
}
```

SLX-Modellentwicklung: Tanker-Initialisierung

zusätzlich:
Typ-bezogene Erfassung ankommender Tanker

Ladung: Gleichverteilung
15tb, 20tb, 25tb



```
passive class Harbor {
        //counter
        int load_15,  load_20,  load_25;
        …
}
Harbor harbor;
```

```
class Tanker  subclass  (Actor) {
        static rn_stream size;

        double load;
        static double preparation= 0.5;
        pointer (Tank) my_tank;
        initial {
                load= 10 + 5 * rv_discrete_uniform (size, 1, 3);
                switch (load) {
                        case 15.0: { harbor.load_15++; break; }
                        case 20.0: {harbor.load_20++; break;}
                        default: {harbor.load_25++;}
                }
        }
```

Systemanalyse

SLX-Modellentwicklung: Tanker-Erfassung

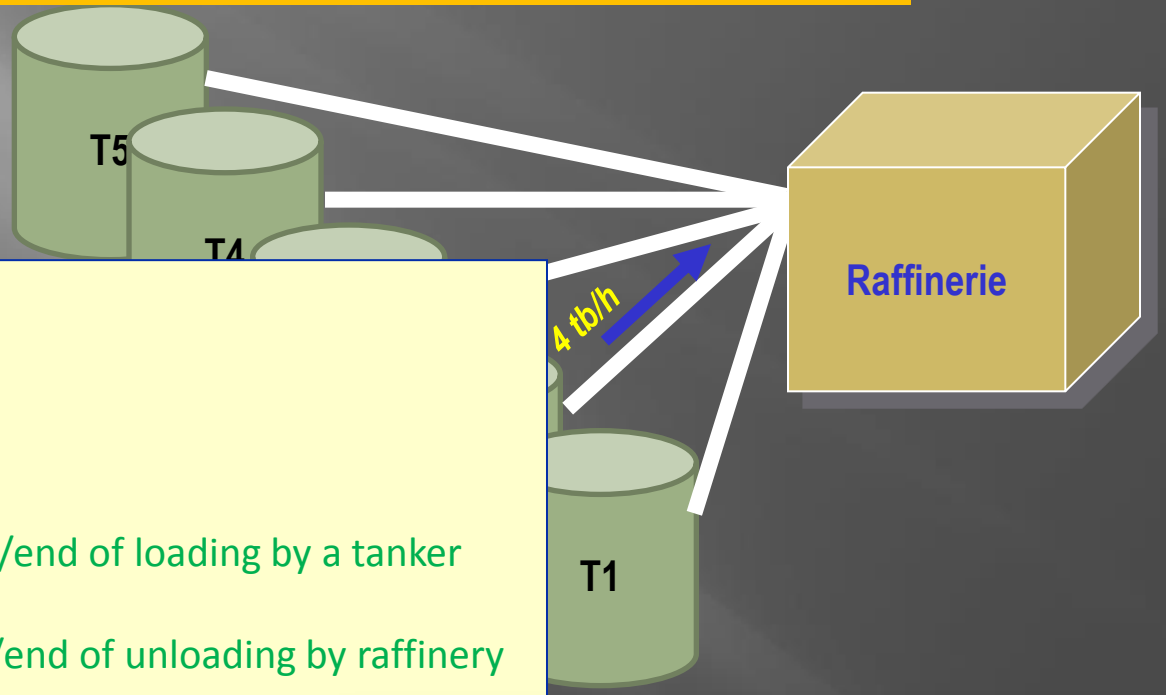
zusätzlich:
Klasse Harbor

```
passive class Harbor {
        int load_15, load_20, load_25;
        procedure tanker_report (){
                print "\n";
                print options=bold (time, load_15, load_20, load_25)
                        "\B___._  ... \BHarbor:\B   Tankers(15tb)= _    Tankers(20tb)= _   Tankers(25tb)=_  \n";
        }
}
Harbor harbor;
```

10003.8 ... **Harbor:** Tankers(15tb)= 375 Tankers(20tb)= 442 Tankers(25tb)= 424

SLX-Modellentwicklung: Startkonfiguration

ÜA: Erweiterung des h7-Standard-Reports um den spezifischen Report




```
procedure main() {
    activate new Arrival();
    activate new Tank(70.0);
    activate new Tank(70.0);
    advance 3.5;
    activate new Tank(25.0);  //end of loading by a tanker
    advance 4.5;
    activate new Tank(70.0); //end of unloading by raffinery
    advance 4.0;
    activate new Tank(45.0);
                          //end of loading by
                           //another tank
        wait until (time > 10000);
        report (system);
        tankControl. report_();
        harbor. tanker_report();
}
```

Ziel:

10.000 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

Simulationsexperimente

Execution begins
*** Slave-Bestimmungs-Variante: select, mitl. Tankerankunft= 8 h

System Status at Time 1000.5693

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	129	400000	400129	OFF	0.66
size	129	200000	200129	OFF	0.24

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	129	7.848	8.276		0.07	47.95

Lower	Upper	Frequency	Percent	
0.0	2.0	27	20.930	**********
2.0	4.0	23	17.829	*********
4.0	6.0	21	16.279	********
6.0	8.0	15	11.628	******
8.0	10.0	10	7.752	****
10.0	12.0	8	6.202	***
12.0	14.0	6	4.651	**
14.0	16.0	6	4.651	**
16.0	18.0	1	0.775	*
18.0	20.0	4	3.101	**
24.0	26.0	1	0.775	*
26.0	28.0	1	0.775	*
28.0	30.0	1	0.775	*
30.0	32.0	2	1.550	*
34.0	36.0	1	0.775	*
38.0	40.0	1	0.775	*
46.0	48.0	1	0.775	*

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	5	5	4.40	126	0	0.00	34.925	34.925
q_tanker	2	3	0.09	128	102	79.69	0.724	3.565

1000.6 ... **Harbor:** Tankers(15tb)= 48 Tankers(20tb)= 40 Tankers(25tb)= 41
Execution complete
Objects created: 186 passive and 2 active Pucks created: 137 Memory: 5 MB Time: 0.19 seconds

Execution begins
*** Slave-Bestimmungs-Variante: select, mitl. Tankerankunft= 6 h

System Status at Time 1001.0175

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	164	400000	400164	OFF	0.80
size	164	200000	200164	OFF	0.40

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	164	6.120	6.238		0.05	35.96

Lower	Upper	Frequency	Percent	
0.0	2.0	45	27.439	**********
2.0	4.0	34	20.732	********
4.0	6.0	23	14.024	******
6.0	8.0	17	10.366	****
8.0	10.0	17	10.366	****
10.0	12.0	10	6.098	***
12.0	14.0	4	2.439	**
14.0	16.0	3	1.829	**
16.0	18.0	1	0.610	*
18.0	20.0	1	0.610	*
20.0	22.0	2	1.220	*
22.0	24.0	3	1.829	**
24.0	26.0	1	0.610	*
28.0	30.0	2	1.220	*
34.0	36.0	1	0.610	*

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	4	5	4.18	168	0	0.00	24.928	24.928
q_tanker	0	7	0.72	164	81	49.39	4.424	8.742

1001.0 ... **Harbor:** Tankers(15tb)= 62 Tankers(20tb)= 52 Tankers(25tb)= 50
Execution complete
Objects created: 225 passive and 2 active Pucks created: 172 Memory: 5 MB Time: 0.09 seconds

Inhalt C.4

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen

Nachteil von *select()* in Kombination mit *is_a-suitable()*

1. Die Bedingung je Ableitungsklasse von CoopQ gilt

   für alle Typen von

   - Tankern(in Rolle Master) und

   für alle Typen von

   - Tanks (in Rolle Slave).

2. Beim Aufruf von select
   ist Bedingung weder sichtbar
   noch lässt sie sich ändern

   (Alle Fälle müssen vorher in einer
   Funktion
   is_a_suitable erfasst sein)

Problem bei weiterer Modellverfeinerung:

Tankerflotte (15, 20, 25 tb),
bezieht von unterschiedlichen Erdölquellen Mengen Rohöl
verschiedener Güteklassen,
die in einem Tank **nicht gemischt** werden dürfen

Lösung

suitable-Methode von TankManagement erweitert Fallunterscheidung
um Vergleich des Ölttyps des Aufrufers und des Slaves

Definition von find als Anweisung

Tank→Slave, Tanker→Master

```
class Tank (double l) subclass (Actor) {
        static double maxCap= 70;
        double freeCap;


        initial {
            freeCap= l;
        }

        actions {
            my_puck= ACTIVE;

            forever {
               while (freeCap > 20.0) {
                        tankControl.wait_for_coopt();
                }
                advance (1/4*(maxCap-freeCap));
                freeCap= maxCap;
            }
        }
}
```

```
class Tanker  subclass  (Actor) {
        static rn_stream size;
        double load;
        pointer (Tank) my_tank;

        initial {
            load= 10+
             rv_discrete_uniform  (size, 1, 3) * 5;
        }
        actions {
            my_puck= ACTIVE;
        find slave  atleast Tank in tankControl
            with  load  <  slave->freeCap; // begin of unloading
            tank_report(my_tank, ME);
            advance (load);
            my_tank->freeCap -= load;
            reactivate my_tank->my_puck;
        // end of unloading
        }
}
```

Bestimmung verfügbarer Tanks (Slaves) per Anweisung (Spracherweiterung)

```
find tank  atleast Tank in tankQ with  load   <   tank->freeCap;
```

soll automatisch ersetzt **w**erden durch

Nutzung der Möglichkeit einer SLX-Spracherweitertung

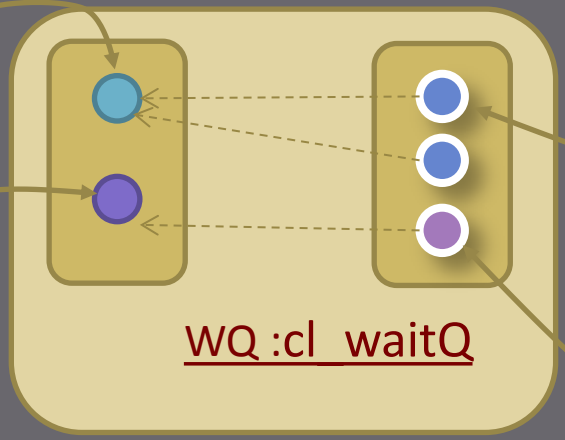
```
pointer (Tank)  tank;
pointer (Actor) master;
pointer (puck) master_puck;
boolean found;
my_puck= ACTIVE;
master= my_puck->puck_object;
place master into  tankControl.masterQ;
enqueue tankControl.mQ;
trace(master->name, "waits for a suitable coopt in masterQ of ", name);
forever {
        for (tank = each Tank in tankControl.slaveQ with load < tank->freeCap) {
                remove tank from tankControl.slaveQ;
                remove master from tankControl.masterQ;
                trace (master->name, "founds in slaveQ", tankControl.name );
                found= TRUE;
                goto end; //by the first slave which was found
        } // for
        if (not found) {
                trace_red (master->name, "founds no suitable coopt ", "" );
                wait;
                trace_red (master->name, "continues by  an arrived coopt peer or cooptInterupt", "");
                if (master->coopInterrupted) {
                        master->waitInterrupted= FALSE;
                        tank = NULL;
                        goto end;
                }
        }
}
end:;
```

SLX-Spracherweiterung wird A. Blunk präsentieren

SLX-Lösungen (Fazit)

sollte gleichzeitig für mehrere Master-Slave-Ensemble nutzbar sein:

1. mehr als ein Master pro CoopQ-Objekt



WQ->wait_for_coopt()

WQ->wait_for_coopt()

S1= WQ->coopt()

S2= WQ->select()

find S3 **atleast** …**in** WQ **with** …;

2. Slave muss einer Bedingung genügen

3. Warte-Aktivität des Master auf Slaves bzw. der Slaves auf Master und die eigentliche Master-Slave-Kooperation ist unterbrechbar

Varianten zur Slave-Bestimmung

- coopt() Memberfunktion von CoopQ
- select() + Bedingungsfunktion als virtuelle Funktion von CoopQ oder Actor
- **find**-Statement (SLX-Spracherweiterung erlaubt Angabe einer Bedingung beim Aufruf)

Simulationsexperimente

oil-harbor--with-new-master-slave.slx: SLX-64 UI 211 Lines: 8,087 Errors: 0 Warnings: 0 Lines/Second: 464,202 Memory: 5 MB
Execution begins
*** Slave-Bestimmungs-Variante : find SLX-Erweiterung, mitl. Tankerankunft= 8 h

System Status at Time 1000.5693

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	129	400000	400129	OFF	0.66
size	129	200000	200129	OFF	0.24

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	129	7.848	8.276		0.07	47.95

Lower	Upper	Frequency	Percent	
0.0	2.0	27	20.930	**********
2.0	4.0	23	17.829	*********
4.0	6.0	21	16.279	********
6.0	8.0	15	11.628	******
8.0	10.0	10	7.752	****
10.0	12.0	8	6.202	***
12.0	14.0	6	4.651	**
14.0	16.0	6	4.651	**
16.0	18.0	1	0.775	*
18.0	20.0	4	3.101	**
24.0	26.0	1	0.775	*
26.0	28.0	1	0.775	*
28.0	30.0	1	0.775	*
30.0	32.0	2	1.550	*
34.0	36.0	1	0.775	*
38.0	40.0	1	0.775	*
46.0	48.0	1	0.775	*

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	5	5	4.39	126	0	0.00	34.896	34.896
q_tanker	2	2	0.10	128	101	78.91	0.817	3.872

1000.6 ... Harbor: **total-in-Oil=** 2515 tb **total-stored-Oil=** 2525 tb **total-out-Oil=** 2685 tb
1000.6 ... Harbor: **Tankers(15tb)=** 48 **Tankers(20tb)=** 40 **Tankers(25tb)=** 41
Execution complete
Objects created: 187 passive and 2 active Pucks created: 137 Memory: 5 MB Time: 0.14 seconds

oil-harbor--with-new-master-slave.slx: SLX-64 UI 211 Lines: 8,060 Errors: 0 Warnings: 0 Lines/Second: 301,184 Memory: 5 MB
Execution begins
*** Slave-Bestimmungs-Variante : select, mitl. Tankerankunft= 8 h

System Status at Time 1000.5693

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	129	400000	400129	OFF	0.66
size	129	200000	200129	OFF	0.24

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	129	7.848	8.276		0.07	47.95

Lower	Upper	Frequency	Percent	
0.0	2.0	27	20.930	**********
2.0	4.0	23	17.829	*********
4.0	6.0	21	16.279	********
6.0	8.0	15	11.628	******
8.0	10.0	10	7.752	****
10.0	12.0	8	6.202	***
12.0	14.0	6	4.651	**
14.0	16.0	6	4.651	**
16.0	18.0	1	0.775	*
18.0	20.0	4	3.101	**
24.0	26.0	1	0.775	*
26.0	28.0	1	0.775	*
28.0	30.0	1	0.775	*
30.0	32.0	2	1.550	*
34.0	36.0	1	0.775	*
38.0	40.0	1	0.775	*
46.0	48.0	1	0.775	*

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	5	5	4.39	126	0	0.00	34.896	34.896
q_tanker	2	2	0.10	128	101	78.91	0.817	3.872

1000.6 ... Harbor: **total-in-Oil=** 2515 tb **total-stored-Oil=** 2525 tb **total-out-Oil=** 2685 tb
1000.6 ... Harbor: **Tankers(15tb)=** 48 **Tankers(20tb)=** 40 **Tankers(25tb)=** 41
Execution complete
Objects created: 187 passive and 2 active Pucks created: 137 Memory: 5 MB Time: 0.12 seconds

Simulationsexperimente

oil-harbor--with-new-master-slave.slx: SLX-64 UL211 Lines: 8,060 Errors: 0 Warnings: 0 Lines/Second: 357,260 Memory: 5 MB
Execution begins
*** Slave-Bestimmungs-Variante : select, mitl. Tankerankunft= 4 h

System Status at Time 1000.0248

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	252	400000	400252	OFF	0.60
size	252	200000	200252	OFF	0.32

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	252	3.985	4.088		0.03	23.98

Lower	Upper	Frequency	Percent	
0.0	2.0	102	40.476	**
2.0	4.0	54	21.429	**************************
4.0	6.0	41	16.270	********************
6.0	8.0	26	10.317	************
8.0	10.0	11	4.365	*****
10.0	12.0	4	1.587	*
12.0	14.0	4	1.587	*
14.0	16.0	4	1.587	*
16.0	18.0	2	0.794	
18.0	20.0	2	0.794	
20.0	22.0	1	0.397	
22.0	24.0	1	0.397	

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	3	5	4.01	202	0	0.00	19.832	19.832
q_tanker	50	50	20.34	252	5	1.98	80.710	82.343

1000.0 ... Harbor: **total-in-Oil=** 4030 tb **total-stored-Oil=** 4095 tb **total-out-Oil=** 5100 tb
1000.0 ... Harbor: **Tankers(15tb)=** 97 **Tankers(20tb)=** 74 **Tankers(25tb)=** 81
Execution complete
Objects created: 357 passive and 2 active Pucks created: 260 Memory: 5 MB Time: 0.11 seconds

oil-harbor--with-new-master-slave.slx: SLX-64 UL211 Lines: 8,060 Errors: 0 Warnings: 0 Lines/Second: 301,184 Memory: 5 MB
Execution begins
*** Slave-Bestimmungs-Variante : select, mitl. Tankerankunft= 8 h

System Status at Time 1000.5693

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrival	129	400000	400129	OFF	0.66
size	129	200000	200129	OFF	0.24

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	129	7.848	8.276		0.07	47.95

Lower	Upper	Frequency	Percent	
0.0	2.0	27	20.930	**
2.0	4.0	23	17.829	***
4.0	6.0	21	16.279	***************************************
6.0	8.0	15	11.628	***************************
8.0	10.0	10	7.752	******************
10.0	12.0	8	6.202	**************
12.0	14.0	6	4.651	**********
14.0	16.0	6	4.651	**********
16.0	18.0	1	0.775	*
18.0	20.0	4	3.101	*******
24.0	26.0	1	0.775	*
26.0	28.0	1	0.775	*
28.0	30.0	1	0.775	*
30.0	32.0	2	1.550	***
34.0	36.0	1	0.775	*
38.0	40.0	1	0.775	*
46.0	48.0	1	0.775	*

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
q_tank	5	5	4.39	126	0	0.00	34.896	34.896
q_tanker	2	2	0.10	128	101	78.91	0.817	3.872

1000.6 ... Harbor: **total-in-Oil=** 2515 tb **total-stored-Oil=** 2525 tb **total-out-Oil=** 2685 tb
1000.6 ... Harbor: **Tankers(15tb)=** 48 **Tankers(20tb)=** 40 **Tankers(25tb)=** 41
Execution complete
Objects created: 187 passive and 2 active Pucks created: 137 Memory: 5 MB Time: 0.12 seconds

Queue-Anwendung aus Modul h7

```
//*****************************************************************************
// Module Master Slave Module, Version 2
//*****************************************************************************
#define SLX2 ON
import <stats>
import <h7>
...
passive class CoopQ {
        set (Actor) masterQ;
        set (Actor) ranked (descending priority) slaveQ ;
        string (18) name;
        queue sQ title= "SlaveQ";
        queue mQ title= "MasterQ";
...
method wait_for_coopt() returning boolean {
                pointer (Actor) slave;
                pointer(Actor) master;
                pointer ( puck ) slave_puck;
                ...

                slave_puck= ACTIVE; // caller-Puck
                slave= slave_puck->puck_object;

                place slave into  slaveQ;
                enqueue sQ;
                if (masterQ.size >0 ) {
                        master= first object in masterQ;
                        reactivate master->my_puck;                    }
                wait;                // continue by reactivation of a master or a wait_interrupt
                depart sQ;
                if (slave->waitInterrupted) {
                        return FALSE;
                }
                else {
                        wait;
                        return TRUE;
                }
        }
...
```

Report-Ausgabe

Frage: Was ist zu tun, um separate Erfassung der Queue-Objekte von CoopQ-Objekten zu erreichen?

System Status at Time 157.6569

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Uniformity
arrival	19	400000	400019	OFF	N/A
size	19	200000	200019	OFF	N/A

Random Variable	Mean #Observed	Std Dev or ~Value	or ~Error	Sig. Digits	Minimum	Maximum
arrive_time	19	8.777	9.398		0.28	31.98

Lower	Upper	Frequency	Percent	
0.0	1.0	2	10.526	| **********
1.0	2.0	2	10.526	| **********
2.0	3.0	2	10.526	| **********
3.0	4.0	2	10.526	| **********
4.0	5.0	2	10.526	| **********
5.0	6.0	1	5.263	| **********
7.0	8.0	1	5.263	| **********
8.0	9.0	1	5.263	| **********
11.0	12.0	1	5.263	| **********
13.0	14.0	1	5.263	| **********
15.0	16.0	1	5.263	| **********
18.0	19.0	1	5.263	| **********

Overflow: 2 Average Overflow: 30.95

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
MasterQ	0	1	0.05	19	18	94.74	0.434	8.253
SlaveQ	1	5	1.98	20	1	5.00	15.580	16.400

Statistik-Funktionalität für Warteschlangenvorgänge aus Modul <h7>

Inhalt C.4

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen

CoopQ-Unterbrechungsoperationen

Actor ~ ODEMx-Klasse *Process*

CoopQ ~ ODEMx-Klasse *WaitQ*

Abstrakte Klasse

Konkrete Klasse

Actor
my_puck: **pointer** (puck) name: **string**(18) coopInterrupted: **Boolean** waitInterrupted: **Boolean** advanceInterrupted: **Boolean**
initial coop_interrupt() wait_interrupt() advance_interrupt()

CoopQ
masterQ: **set** (Actor) slaveQ: **set** (Actor) name: **string** (18)
initial **boolean** wait_for_coopt() **pointer** (Actor) coopt() **pointer** (Actor) select() **overridable** **boolean** is_a_suitable (**pointer**(Actor) p)

Anwendung bei folgenden Puck-Zustände möglich

Wartender Master	waiting	Wartender Slave	waiting
Aktiver Master	moving	Inaktiver, zugeordneter Slave	waiting

Implementierung

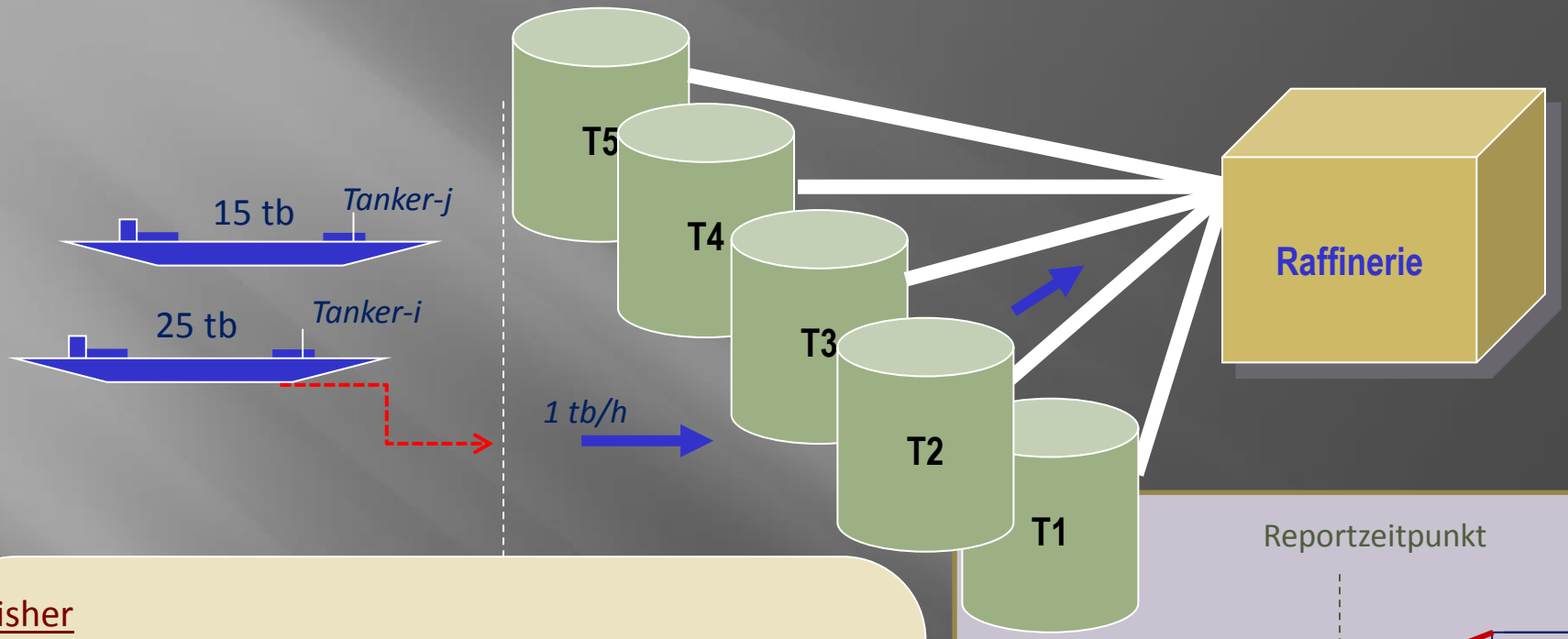
```
method wait_interrupt() {
        pointer (Actor) caller= ACTIVE->puck_object;
        #ifdef ODEM_DEBUG trace_bold (caller->name,
                "interrupts the <wait_for_coopt> activity of", name );
        #endif
        waitInterrupted= TRUE;
        reactivate my_puck;
}

method coop_interrupt() {
        pointer (Actor) caller= ACTIVE->puck_object;
        #ifdef ODEM_DEBUG trace_bold (caller->name, "interrupts the <coopt> activity of", name );#endif
        coopInterrupted= TRUE;
        reactivate my_puck;
}

method advance_interrupt() {
        pointer (Actor) caller= ACTIVE->puck_object;
        #ifdef ODEM_DEBUG trace_bold (caller->name, "interrupts the <advance> activity of", name );#endif
        advanceInterrupted= TRUE;
        reschedule my_puck at time;
}
```

Zustandsaktualisierung der Anwendung ist vom Master zu organisieren (für sich selbst und die beteiligten Slaves)

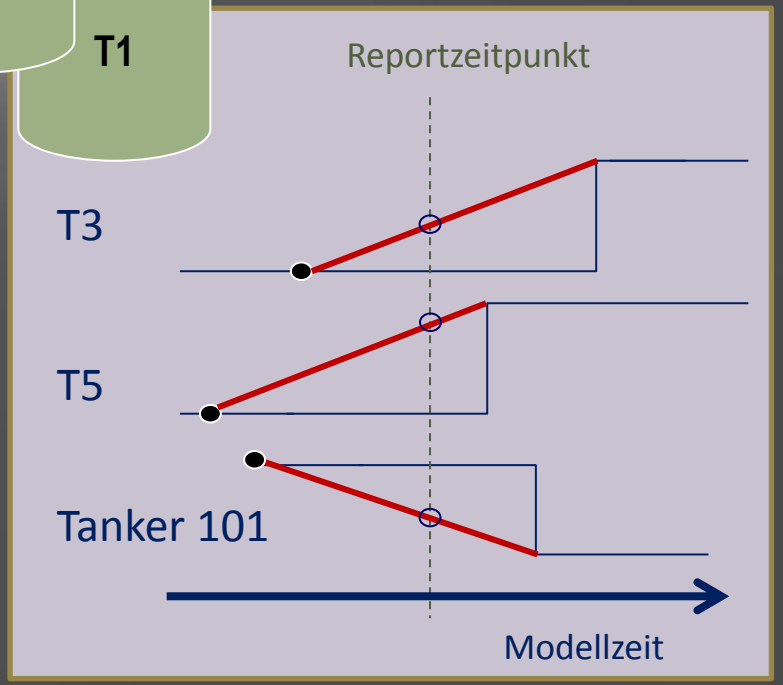
Problem: korrekte Zustandserfassung zu einem beliebigen Zeitpunkt



bisher

- Belegungszustände und deren Änderungen werden exakt erfasst
- Zustände der jeweils aktuellen
  - Tank-Inhalte,
  - Tanker-Inhalte und
  - die Menge geflossenen Öls an die Raffinerie

  werden **nicht synchron** dargestellt,
- stetige Zustandsänderungen in der Realität sind **Sprungfunktionen** im Modell



Allg. Vorgehensweise

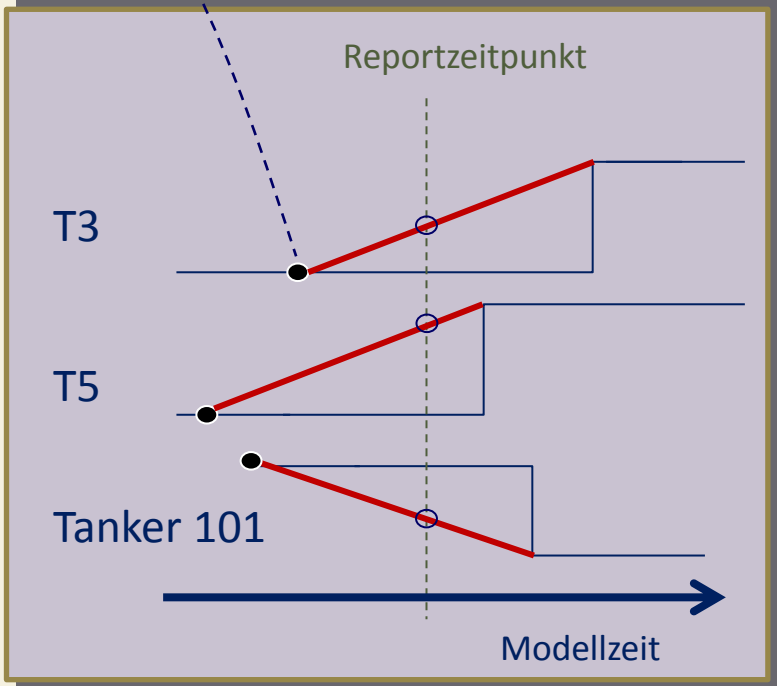
- **Ankommene Tanker**
  registrieren sich in Harbor,
  Dabei vermerken sie die jeweiligen **Startzeit**
  der geplanten advance-Aktivität
  sie verlassen Harbor, sobald sie leer geworden sind

- Erhält der **Report-Prozess** zum geplanten Report-Zeitpunkt die Steuerung,
  - **a)** **unterbricht** er per advance_interrupt **alle momentan** Tanker von Harbor im Moving-Zustand
  - b) diese werden **re-scheduled** (erkennen bei Fortsetzung nach advance) am Flag, dass eine Unterbrechung vorliegt und
  - **c)** **Berechnen** Ihre Zustände neu aus
    - altem Zustand,
    - Änderungsrate und
    - Zeitintervall (= Reportzeitpunkt – Startzeitpunkt)

    und aktualisieren auch den Tankbehälter-Zustand
  - **d)** **Berechnen** die **Restzeit**

  ➔ Damit liegen die aktuellen Zustandswerte für den Report-Prozess abholbereit vor.

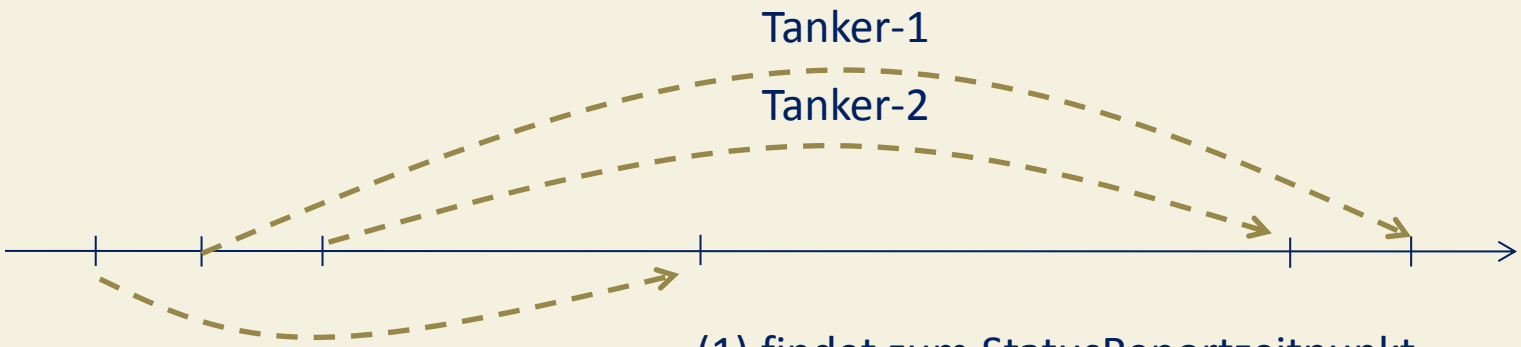
- Die **unterbrochenen** (aber aktiven Prozesse) setzen danach die Koop.Phase mit der **Restzeit** fort



Änderung von Tanker-actions

```
my_puck= ACTIVE;
habor->register(ME);
find slave  atleast Tank in tankControl
                        with  load   <   slave->freeCap;
while (load >0.0) {
        start_time= time;
        advance (load);
        if (advanceInterrupted) {
                moved_load= time – start_time;  // load_rate =1.0
                my_tank->freeCap -= moved_load;
                load-= moved_load;
                advanceInterrupted= FALSE;
      }
      my_tank->freeCap-= load;
      load= 0.0;
}
reactivate my_tank->my_puck;  // Freigabe des zugeordneten Tankbehälter
habor->de_register(ME);
```

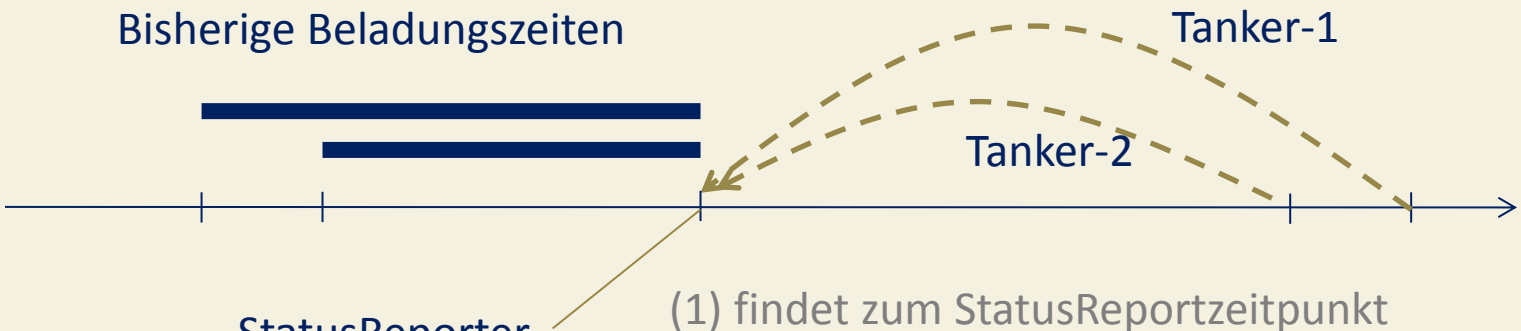
Momentaufnahme [1]



(1) findet zum StatusReportzeitpunkt
- Tanker-1 mit zugeordnetem Tank
- Tanker-2 und zugeordnetem Tank
- 5 Tankbehälter

(2) advance_interrupt-Anwendung auf
- Tanker-1 und
- Tanker-2

Momentaufnahme [2]



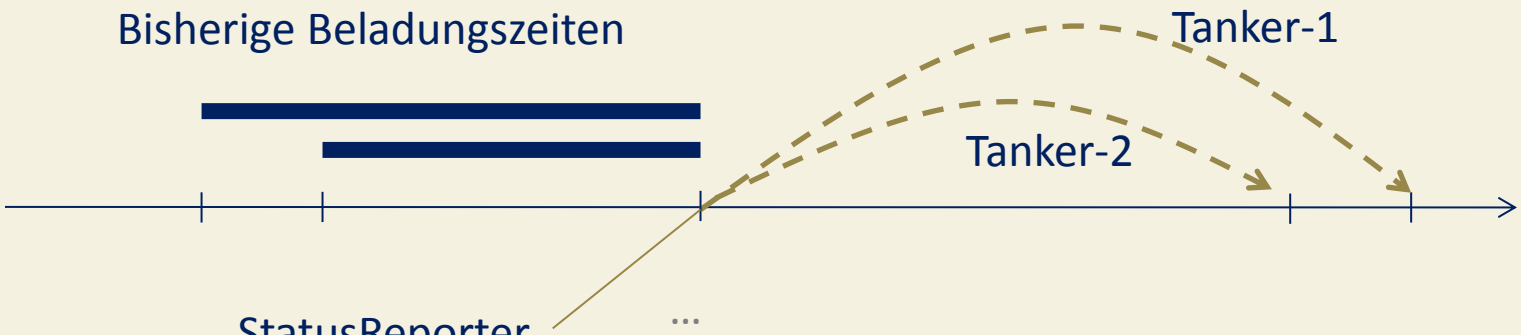
(1) findet zum StatusReportzeitpunkt
- Tanker-1 mit zugeordnetem Tank
- Tanker-2 und zugeordnetem Tank
- 5 Tankbehälter

(2) advance_interrupt-Anwendung auf
- Tanker-1 und
- Tanker-2

(3) Tanker-1 und Tanker-2
- erkennen Unterbrechung
- aktualisieren die Zustände
- Setzen mit reduziertem Advance fort

bevor StatusReporter (als Unterbrecher) fortsetzen kann

Momentaufnahme [3]



...

(3) Tanker-1 und Tanker-2
- erkennen Unterbrechung
- aktualisieren die Zustände
- Setzen mit reduziertem Advance fort

bevor StatusReporter (als Unterbrecher) fortsetzen kann

(4) StatusReport findet nun zum geplanten StatusReport-Zeitpunkt aktuelle Tanker- und aktuelle Tankbehälter-Zustände vor

Frage:
Was ist mit den Tankbehältern, die von der Raffinerie entleert werden?

Vervollständigen Sie den Lösungsansatz !

Unterbrechungen der Master-Slave-Kooperation

… kann in 2 Formen auftreten

1. Unterbrechung der gestarteten Kooperation eines Masters mit n (n>=1) Slaves
   per advance_interrupt()

   advanceInterrupted → **True**

   Unterbrochen wird der aktive Master (seine Slaves sind passiv) in einer Advance-Aktion.

   Er sollte in seinem Actions-Teil diese Situation durch Auswertung des Flag-Wertes berücksichtigen

2. Unterbrechung der Warte-Aktionen von Slave bzw. Master
   per wait_interrupt() bzw. coopt_interrupt()

   Die unterbrochene Prozesse (Pucks im waiting-Zustand) werden reaktiviert.
   Diese sollten in ihren jeweiligen Actions-Teil endiese Situation durch Auswertung des Rückgabe-Wertes erfassen

   coopt() → NULL-Zeiger
   wait_for_coopt() → **False**

Unterbrechung der Master-Slave-Kooperation und Fortsetzung bei Bestimmung neuer Slaves

nach erkanntem Interrupt für eine Kooperationsphase mit advance-Interrupt

- aktualisiert der Master die Zustände von Master und der zugeordneten Slaves
- gibt die Slaves frei
- Bestimmt neue Slaves und setzt die Kooperation mit den neuen Partnern fort

Inhalt C.4

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen

Beispiel: Transportfahrzeuge – Bagger – Beladung

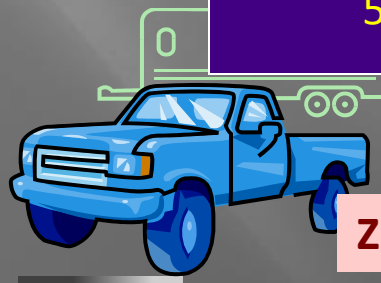
große Lastwagen
20 t

alle 3 min

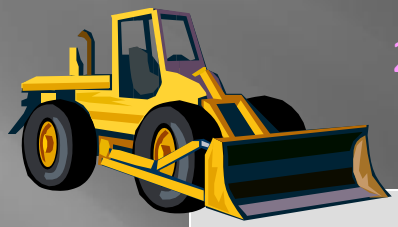
Ankunftsraten

alle 22 min

kleine Lastwagen
5 t



Ziel: 8h -Simulation



240 t/h (4t/min)

60 t/h (1t/min)

Beladungsraten

kleinerBagger

großerBagger-1

großerBagger-2

Beladung

- große Lastwagen bevorzugt von großen Baggern
- kleine Lastwagen **nur** vom kleinen Bagger
- sofortige Unterbrechung der Beladung eines großen Lastwagens durch kleinen Bagger, sobald kleiner Lastwagen eintrifft oder großer Bagger frei wird
- priorisierte Beladung unterbrochener Fahrzeuge

Systemanalyse

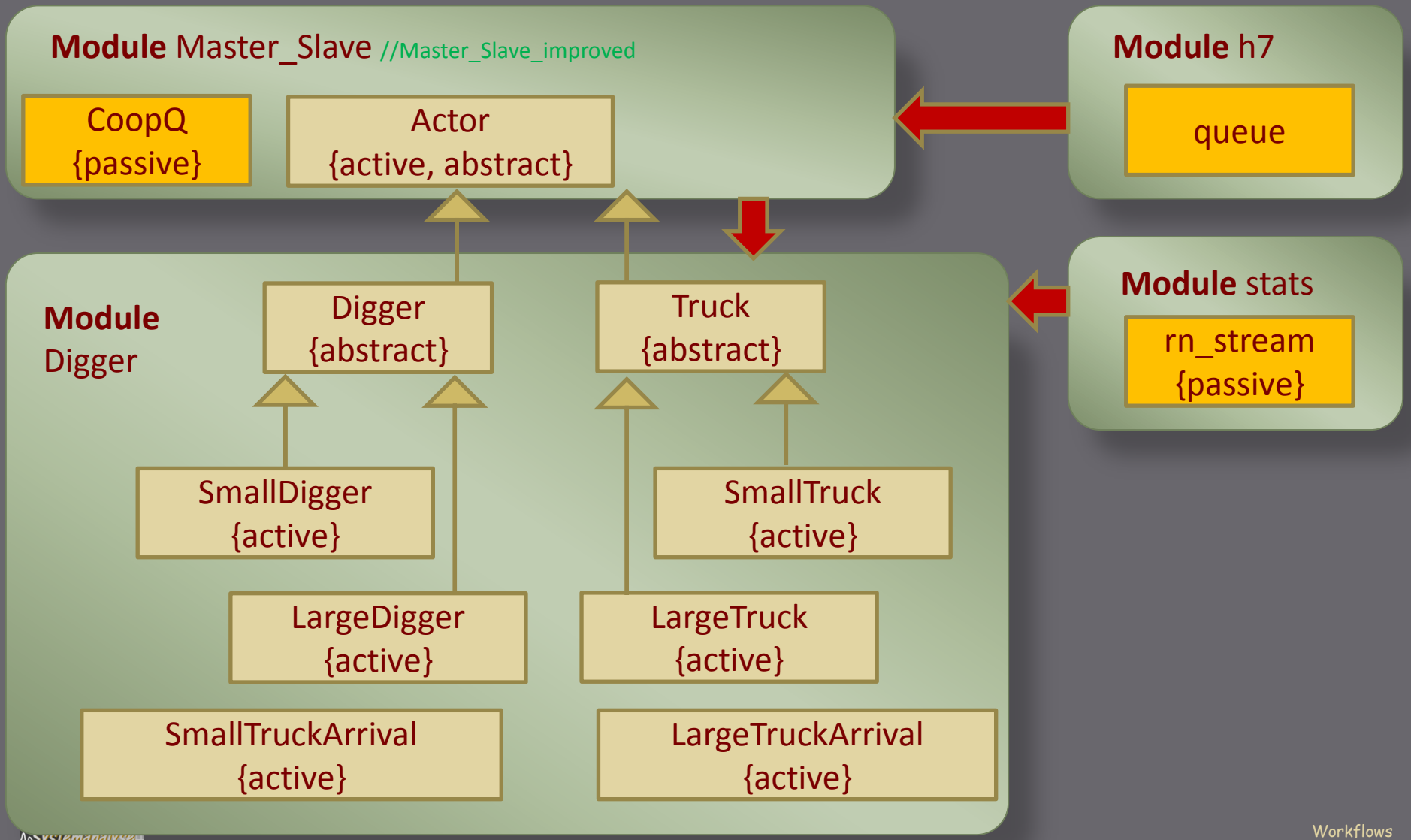
Typische Probleme

- Zuweisung einer Ressource aus einem Spektrum unterschiedlicher **Ressourcenklassen**
  für die Durchführung spezifischer Arbeitsgänge

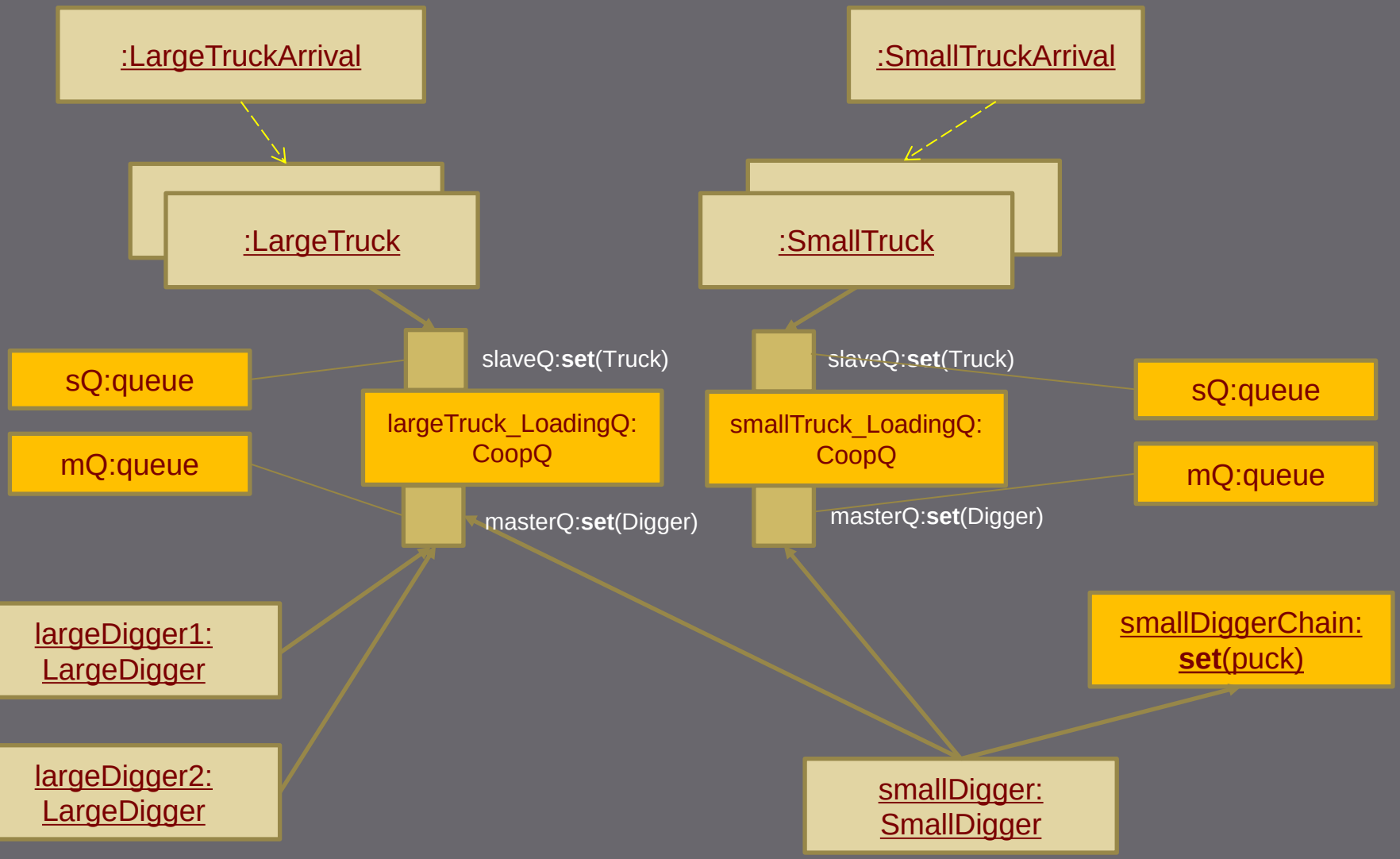
- **dynamischer Austausch** der eingesetzten Ressourcen (bei gegebener Verfügbarkeit) um Effizienz des Arbeitsganges zu erhöhen

- **Unterbrechung** von Ressourcennutzungen bei Neuzuteilung von Ressourcen

Modellelemente-Typen (Klassen)



Systemkonfiguration



Master-Slave-Bindung

- NORMALFALL

  LargeDigger – LargeTruck: - Bindung per coopt
  - advance (ohne Unterbrechung)
  - Zustandsänderung
  - Truck-Freigabe

  SmallDigger – SmallTruck: - Bindung per coopt
  - advance (ohne Unterbrechung)
  - Zustandsänderung
  - Truck-Freigabe

- SONDERFALL

  SmallDigger – LargeTruck: **a) ohne Unterbrechung**
  - Bindung per coopt
  - advance
  - Zustandsänderung
  - Truck-Freigabe

  b) mit Unterbrechung durch largeDigger oder smallTruck
  - Bindung per coopt
  - advance mit Unterbrechung
  - korrigierte Zustandsänderung beim Truck
  - Prioritätserhöhung beim Truck
  - Fortsetzung mit Warten in smallDiggerChain
  - Truck-Freigabe für Fortsetzung mit largeDigger1 oder largeDigger2

SLX-Umsetzung

- folgt demnächst …

Programm-Erweiterungen

- **Queue-Statistik** aus Modul „h7“ bleibt
- Einzelne **Puck-Kette** zur Erfassung wartender SmallDigger **entfällt,**
  dafür gibt es ein **Konfigurationsobjekt**
  (Übernahme der beiden Listen **SD_list, LD_list**)
- Das **Konfigurations-Objekt** (besser dessen Klasse) besitzt zunächst zwei Member-Funktionen:
  - eine liefert bei Bedarf ein freies SmallDigger-Objekt (falls vorhanden)
  - die andere ein SmallDigger-Objekt, das ein großes Fahrzeug belegt (falls vorhanden)

```
passive class System_Configuration {
   set (Large_Digger) LD_list;
   set (Small_Digger) SD_list;

   procedure free_digger() returning pointer (Small_Digger) {}
   procedure LT_loading_digger() returning pointer (Small_Digger) {}

   procedure digger_report () {}
   procedure truck_report () {}

   procedure state_report () {
                        digger_report();
                        truck_report();
   }
} // class
```

- Darüber hinaus, sind die Report-Funktionen der Reporterklassen zu Memberfunktion der Konfigurationsklasse geworden,
  die **Report-Klasse** entfällt damit.
- Erweitert wurde
  a) **Digger-Summary** ~ Ausgabe von Zähl-Attributen der installierten Digger
  b) **Truck-Summary** ~ Ausgabe von globalen Zählgrößen

Inhalt C.4

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen
   - Abschließende Betrachtung

Ausstehende Re-Implementation von CoopQ (Ideensammlung)

- **h7-konforme-Report-Funktionalität** für CoopQ-Objekte
  bei Sicherung, dass lokale Queue-Objekte nur 1-mal im Report erscheinen
  (nicht nochmal bei Darstellung sämtlicher Queue-Objekte)

  remove &masterQ **from** queue_set;

- **Verzicht** auf **Super-Klasse Actor**, Operation über rufenden Puck,
  Dazu müsste die Klasse Puck um Unterbrechungsflags von Actor erweitert werden

  augment class puck { … } ;

- **coopt**, und **find** könnten zu einer Operation zusammengelegt werden:

  SLX-Erweiterung erlaubt Anweisung mit optionalen Parametern

- ODEMx bietet weitere Operationen für Master-Slave
  - Master kann **Verfügbarkeit** von Slves bzw. Passenden Slaves prüfen, ohne zu blockieren
  - Slaves kann bei der **Reaktivierung schlafender Master**, eine Reihenfolge der Master beachten, die sich durch dynamische veränderbare Gewichte (Prioritätsänderungen) ergeben.

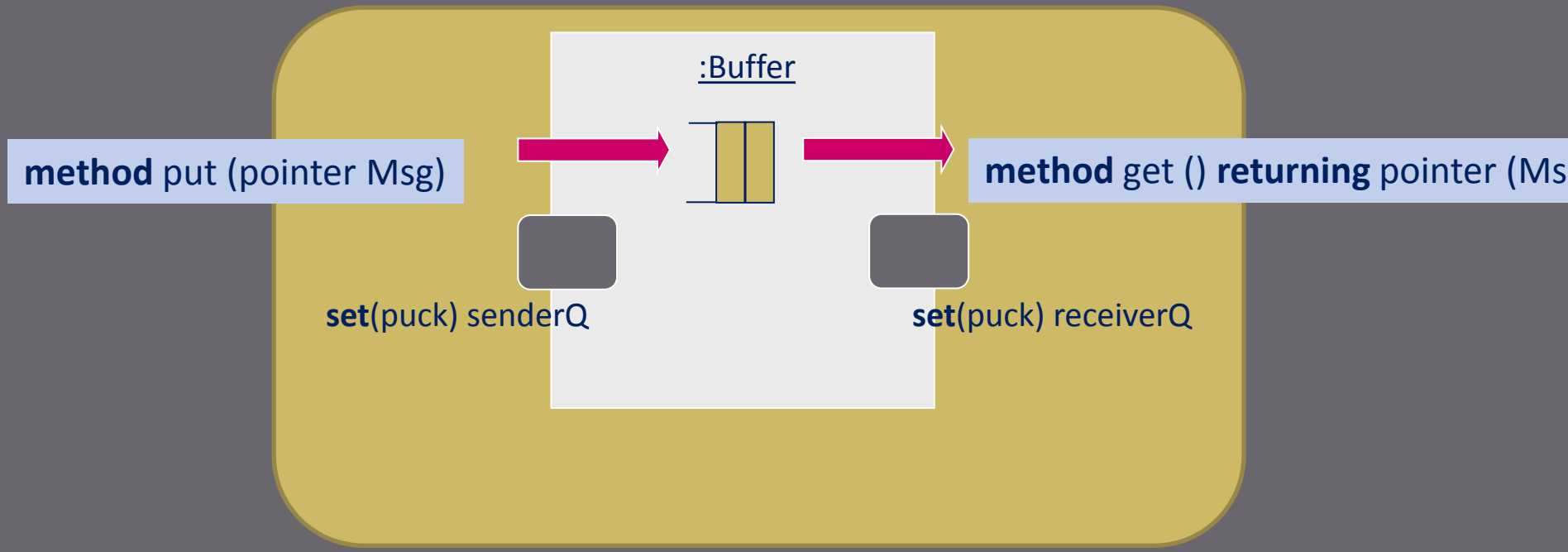
Inhalt C.4

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  DISCO-Elemente

1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
3. Asynchrone Kommunikation

Buffer ~ vereinfachtes ODEMx-Portkonzept



Blockierung wenn voll

Blockierung wenn leer

Msg ist Basisklasse für Buffer-Einträge