

7. Szenario-Sprachen

Fortsetzung Abschn. 2

geklärt waren:

- Konzept ereignisorientierte Simulation
- wünschenswerte Fähigkeiten/Hilfsmittel (von PS) zur Spezifikation ("Programmierung") Simulatoren, insbesondere Manipulation der Zeit in "Denkwelten"
 - event-scheduling
 - activity-scanning
 - process-interaction
- programmiertechnische Aspekte
 - Einbettung event-scheduling in allgemeine PS
 - process-interaction am Beispiel SIMULA (SIMULA: allgemeine HLL, Konzepte über normale HLL hinaus, insbes. auch Simulations-Unterstützung)

Wie SIMULA aus ALGOL60 "hervorgegangen"
(und weit mächtiger als Basissprache),

so andere Simulationssprachen aus
FORTRAN, PL/I , PASCAL, ... hervorgegangen
(immer mächtiger in Syntax und Semantik als Basis);

daher auch immer:

"komplizierter" zu verwenden
(programmieren → spezifizieren)

alternativ zu
Erweiterung allgemeiner HLL
in allgemein Simulations-unterstützendem Sinn

ist
Erweiterung allgemeiner HLL
(oder Definition von Spezialsprachen)
zur speziellen Simulations-Unterstützung
für ganz bestimmte Problemklassen,
bestimmte **Szenarios**

solche Sprachen: **Szenariosprachen**

Konkretes, bei Anwendung ereignisorientierter Simulation

- häufig auftretendes
- praktisch breit eingesetztes

Szenario ist das der

Verkehrsnetze / Warteschlangennetze /
flow-shops / job-shops / ...

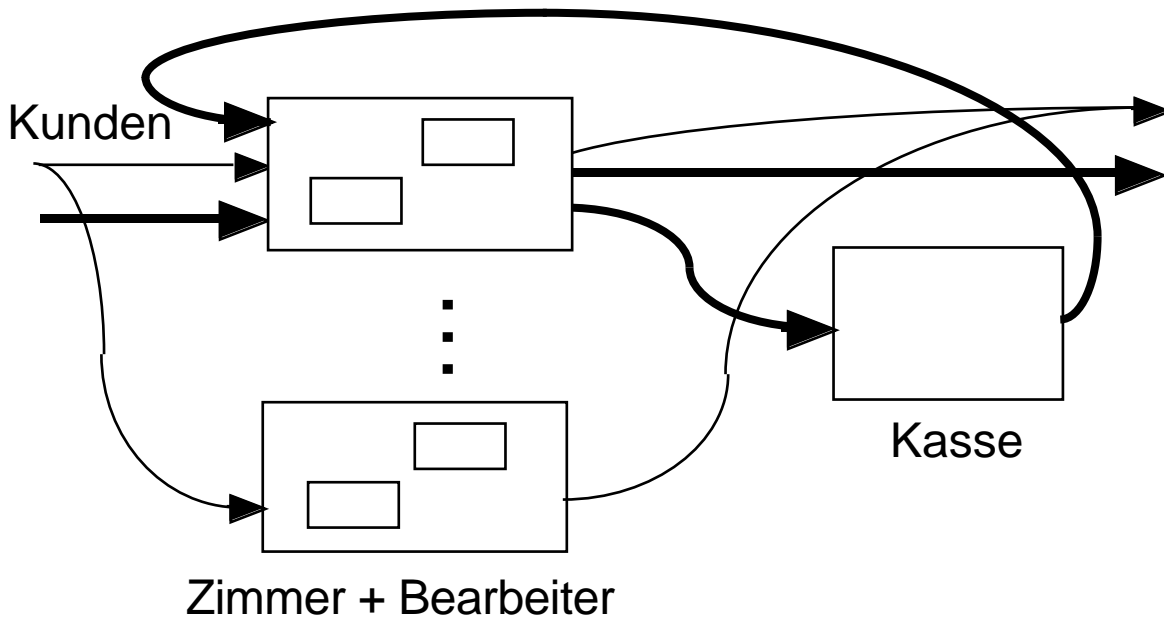
Charakteristika dieses Szenarios:

- **Verkehrseinheiten** (Kunden, Material,...)
bewegen sich / werden bewegt
- in Menge / **Netz von Betriebsmitteln**
(Bearbeitungseinheiten, Bedieneinheiten,
Ressourcen, Stationen, Maschinen,...)
- und verlangen je "Besuch" eines Betriebsmittels
gewisse **Bearbeitung** / Bedienung

viele praktische Beispiele:

- Bankschalter (wie gehabt)
- Fabrikhalle
- Supermarkt
- ...

z.B. Straßenverkehrsamt



verschiedene Bearbeiter-Funktionen,
verschiedene Kunden-Typen,
unterschieden nach Wünschen (Wunsch-Mustern)

es existiert eine lange Historie
der Modellierung
und Simulation (neben anderen Analyse-Techniken)
solcher Systeme

gerne zwei (Modellierungs-) "Sichten" unterschieden

machine-oriented:

Bearbeitungseinheiten
sind Akteure:
Prozesse

Verkehrseinheiten
sind passiv

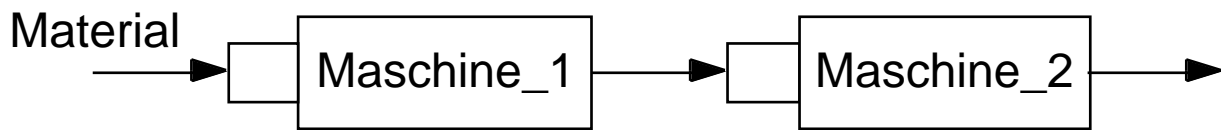
material-oriented:

Verkehrseinheiten
sind Akteure:
Prozesse

Bearbeitungseinheiten
sind passiv

beide Sichten sind prozeß-orientiert

Prinzipielle Struktur beider Sichten etwas präziser:



machine-oriented:

jede Maschine

LOOP

```

{ nimm Material aus Puffer };
{ bearbeite, d.h. "verbrauche Zeit" };
{ steck' Material in nächsten Puffer };
  
```

END LOOP

material-oriented:

jede Materialeinheit (Kunde)

...

```

{ bemächte Dich der nächsten Maschine
  und "belege" sie bzgl. anderer Kunden };
  
```

```

{ benutze sie, d.h. "verbrauche Zeit" };
  
```

```

{ gib Maschine frei }
  
```

```

{ bemächte Dich der nächsten Maschine ... }
  
```

...

Sichten schon in Abschn. 2 (mit SIMULA) durchgespielt,
 dazu noch flexiblere Variante, die
 sowohl Material als auch Maschinen (-Verhalten)
 als Prozesse / Prozeßmuster formulierte
 die "interagierten"

Jetzt (als Beispiele)

weitere (in Sicht durchaus ähnliche)

Szenariosichten / Szenariosprachen

GPSS, SLAM, DEMOS, HIT

In Diskussion von Modellwelten / Simulations-"Paradigmen" über die Jahre gewisse Beispiele immer wieder verwandt

Eines davon (seit CSL/ECSL wiederkehrend):

Beispiel 7.0.1: Hafen und Schiffe (in einfachster Version)

- mit gewissen zeitlichen Abständen fahren Schiffe zum Entladen einen Hafen an
- zur Entladung wird ein Pier (jetty) benötigt, zum Einlaufen sind zwei Schlepper (tugs) erforderlich, zum Auslaufen ein Schlepper
- Einlaufen, Entladen, Auslaufen dauern spezifische Zeiten
- der Hafen weist begrenzte Anzahl von jetties auf, verfügt über begrenzte Anzahl von tugs

In allen zu diskutierenden Paradigmen werden (zusätzlich zur programmsprachlichen Spezifikation) graphische Spezifikations-Darstellungen eingeführt,

- zunächst nur als Mittel zur "manuellen" Modellbildung
- erst in jüngerer Vergangenheit (und immer noch nicht in allen Fällen) zur interaktiv-graphischen Erstellung Modell-Spezifikation

Von den Beispielparadigmen weisen lediglich DEMOS + HIT (für heute) annehmbare sprachliche Form auf, erinnern GPSS und SLAM an die "Steinzeit", (Anfang ihrer Entstehungsgeschichte !)

7.1 GPSS

Sprache + System zur Simulation von
verkehrsnetz-strukturierten Modellen

Geschichte reicht in die "late fifties" zurück,
Entwicklungsschritte GPS / GPSS / GPSS II / III / 360 / V ...
IBM-geprägt

Literatur (u.a.): Gree72, Schr74, Gord75, BoKP76,
Schm78, Schr91

Trotz Alter (und Steinzeit-Erscheinungsbild)
so akzeptiert, daß immer noch (für einschlägige Probleme)
weit verbreitete / genutzte Simulationssprache

Heute in unterschiedlichen Versionen vorliegend

- mit z.T. großen Unterschieden,
aber selbem "Weltblick": i.w. material-oriented
 - Einheiten (hier: **transactions**)
bewegen sich über
Ressourcen (hier: **equipment entities**)
- Modellspezifikation besteht i.w. aus transactions,
Spezifikationen der diversen Verkehrs-"Muster"
(Prozeßmuster)
- Generierungsgesetze für Verkehrs-"Objekte"
(Prozeßobjekte)
eines bestimmten Musters in transaction selbst enthalten
- benötigte equipment entities werden
in transaction "genannt", nicht eigens deklariert

GPSS als Sprache stark
vom Lochkarten- / Flußdiagramm- Zeitalter geprägt

Entwurf von transactions empfohlen
in Form

- bestimmten Typs von Flußdiagrammen
- bestehend aus (beträchtlicher Vielfalt an)
"Kästchen": **blocks**
und "Pfeilen": Kontrollfluß-Richtung

Flußdiagramm ursprünglich auch gedacht als
"Abbild Lochkartenstapel": ein block, eine Lochkarte

wir werden sehen, daß diese GPSS-Diagramme
(von heute aus gesehen)
als initiale Form graphischer Modellspezifikation
auffaßbar sind

Jede GPSS-Version ist definiert durch

- ihre **entities** (Einheiten)
- deren **attributes** (Eigenschafts-Konstanten, -Variablen)

Jede entity(-Typ!) besitzt (per default)
charakteristische attributes,

- im Verlauf der Simulation automatisch aktualisiert,
- mit Zugriffsmöglichkeit auf deren (momentane) Werte

Programmierer-definierte attributes nur in Sonderfällen

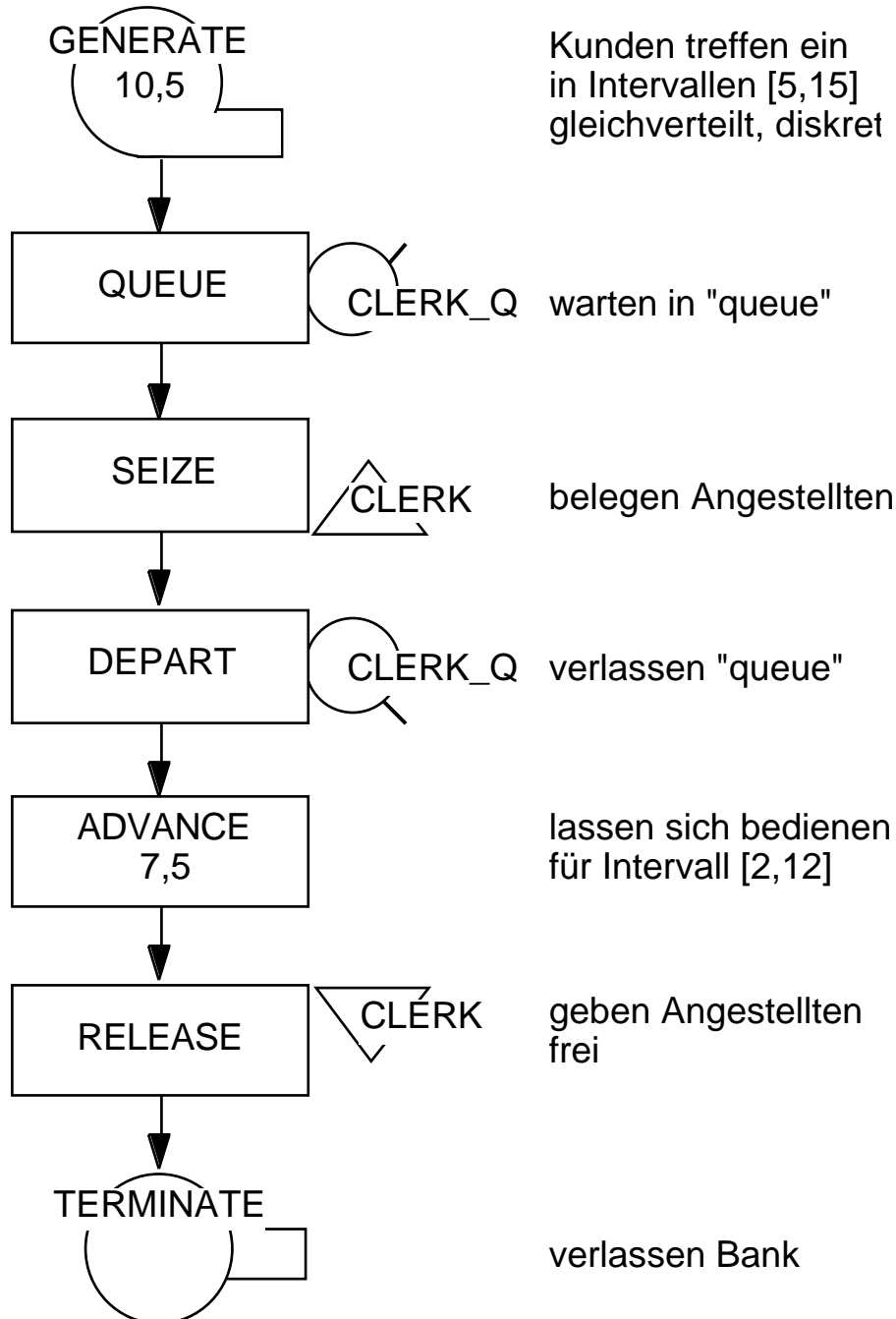
Identifikation von entities über "Nummern",
von attributes über Standardnamen

Im Bereich der Arithmetik (incl. "Zeit")

- zunächst strikte Ganzzahligkeit
- bei neueren Versionen z.T. aufgegeben

"Gefühl gewinnen" über Beispiele

Beispiel 7.1.1: Bankschalter im GPSS-Diagramm



Erinnerung an Abschn. 2,

prinzipielle "Zeitmanipulations"-Überlegungen:

- WAIT_UNTIL, PAUSE_FOR direkt enthalten
- potentielle Ineffizienz des WAIT_UNTIL beseitigt durch starke Reduktion der Wartebedingungen

GPSS-entities:

basic:	blocks	Kästchen, Karten	(s.Bsp.)
	transactions	Prozeßmuster	(s.Bsp.)
equipment:	facilities	Bediener	(s.Bsp)
		SEIZE, RELEASE	
	storages	Speicher	(s.Folgebsp.)
		ENTER, LEAVE	
	logic switches		
computational:	boolean variables		(Vorsicht:
	arithmetic variables		haben mit
	functions		heutigen
			Begriffen
			nur bedingt
			zu tun)
reference:	...		
statistical:	queues	(Vorsicht: GPSS "queue"	
		ist nicht unsere Warteschlange,	
		dient nur Erhebung v. Statistiken)	
	frequency tables		
	...		

Pluspunkte:

einfache Notierbarkeit einfacherer Modelle,
automatische Statistiken,
gute debugging-Hilfen,
wenn Compiler vorhanden, hinreichend schnell
(GPSS oft lediglich interpretiert !)

Alles andere (wie gesagt): Steinzeit

Ein Blick auf den GPSS-Code zu Bsp. 7.1.1
(Codierung traditionell in Festformat !!):

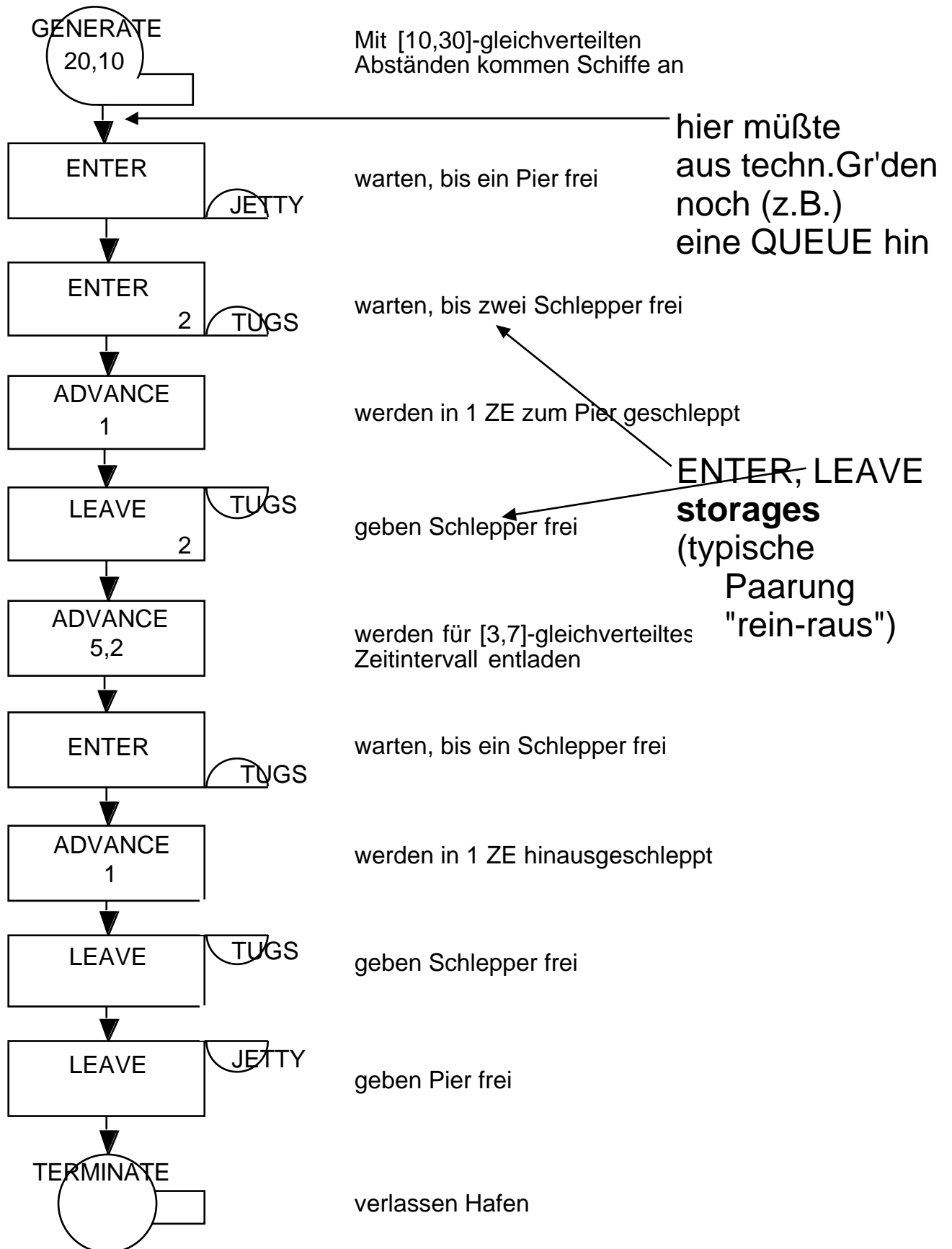
SIMULATE		(u.U.) erste "Karte"
GENERATE	10,5	Beginn Transaktion
QUEUE	1	in queue #1, oben: CLERK_Q, (implizite Deklaration)
SEIZE	1	die facility #1, oben: CLERK
DEPART	1	aus queue #1, (nachdem facility #1 "betreten")
ADVANCE	7,5	der Zeitverbrauch
RELEASE	1	die facility #1
TERMINATE	1	Ende transaction (mit "Herunterzählen Globalzähler")
START	500	Hinweis auf "beginne Simulation" (mit Initialisierung Globalzähler)
END		(u.U.) letztes "statement"

Automatische Ergebnisse:

(echter input / output eher schwierig, Umweg FORTRAN)

- Zeitangaben
- für jeden block Zahl Einheiten in block,
 Zahl eingetretener Einheiten
 jeweils zu / bis Ende Simulation
- für jede facility Auslastung, Zahl Eintritte,
 mittlere Verweilzeit,...
- für jede queue maximaler / mittlerer Inhalt,
 Eintrittszahl, Zahl "0-Zeit" Verweilender
- Füllung spezifizierter Tabellen
 (Histogramme + deskriptive Statistiken)
- ...

Beispiel 7.1.2: Hafen-Bsp. 7.0.1 im GPSS-Diagramm



7.2 SLAM II

Sprache + System zur Simulation (Prit84)

- von ereignisorientierten Modellen,
 - in event-scheduling-Sicht,
 - in process-interaction-Sicht,
 - in Mischung dieser Sichten
- von Modellen mit zeitkontinuierlicher Zustandsänderung,
 - auf Basis von Differenzgleichungen,
 - auf Basis von Differentialgleichungen,
 - unter Mischung der Gleichungstypen
- von Modellen, die ereignisorientierte und zeitkontinuierliche Zustandsänderungen gemischt enthalten
- hier lediglich skizziert:
ereignisorientierte Modellierung
in process-interaction Sicht

Bei Spezifikation Simulator spielt wieder Netzwerk aus spezifischen Knoten- und Kanten-Typen
incl. graphischer Darstellung
und sprachlicher Formulierung
wesentliche Rolle

Netzwerkdenken "erinnert" an GPSS, jedoch
Netz deutlich mehr als "graphische Beschreibung" gesehen
(und weniger als Flußdiagramm / Kartenstapel)

SLAM (mit Vorläufern GASP ... GASP IV, Q-GERT)
ebenfalls lange Geschichte seit "late sixties"
(u.a. KiPr69, Prit74, PrYo75)

Modellspezifikation

(Graph + zugehöriges, detaillierteres "Programm")
aufgebaut aus

in Graph:

- **nodes** Knoten verschiedener Typen (Formen)
 normalerweise mit Beschriftung
- **activities,** Kanten, u.U. beschriftet
branches
- **network** Netz aus obigen Knoten, Kanten
- **blocks** Kästchen,
 die nicht ins Netz eingebunden sind

Modelldynamik (Verkehr) entsteht durch

- **entities** die von Anfang existieren
 oder erzeugt werden,

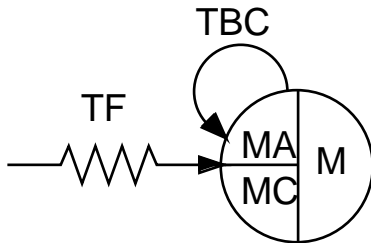
 die sich im Netz
 (über Knoten, entlang Kanten)
 bewegen,

 die (wieder) verschwinden können,

 die je eine Reihe von **attributes**
 (Gedächtnis) ATTRIB(i)
 besitzen können

Erste (von vielen) nodes:

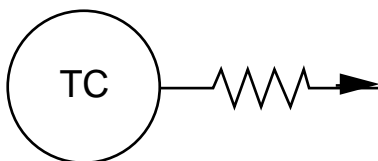
- CREATE (Erzeugung / Ankunft von entities)
- graphisch:



- TBC Zwischen-Generierungs-Zeit entities
 TF Zeit der Generierung der ersten entity
 MA Attributnummer für Notierung Generierungszeit
 MC maximale Zahl durch node generierter entities
 M maximale Zahl bei Ankunft "bedienter" branches
 (potentielle Vervielfältigung entities)

- sprachlich:
 CREATE, TBC, TF, MA, MC, M;
 mit defaults: ,0,"nicht notiert", ,

- TERMINATE (Vernichtung / Abgang von entities)
- graphisch:

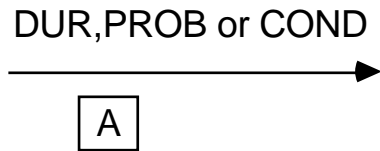


- TC Anzahl vernichteter entities,
 bei der die Simulation abgebrochen wird

- sprachlich:
 TERMINATE, TC
 mit defaults:

Erste activity (von zwei activities):

- regular activity (Zeitverbrauch)
- graphisch:



DUR Zeitintervall, das bei "Durchgang" verstreicht
 PROB ..., für bedingtes "branching"
 COND ..., für bedingtes "branching"
 A Nummer (statt: Name) dieser activity
 (als Referenz für automatische Statistiken)

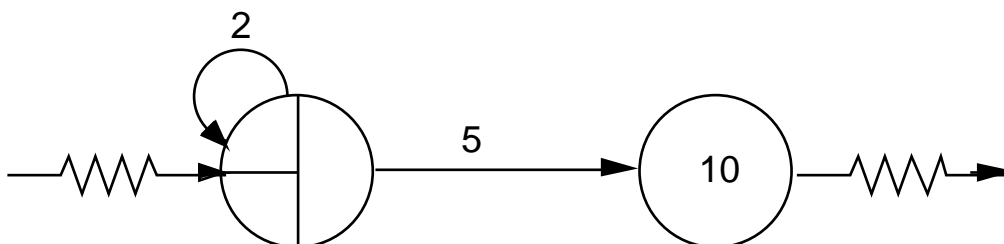
- sprachlich:

ACTIVITY/A,DUR,PROB or COND,NLBL;ID;
 mit defaults:

/"no statistics",0,"always","sequential";"blank"

NLBL ..., für Fortsetzungs-node
 ID ..., für Namensgebung

Erstes "Netz":

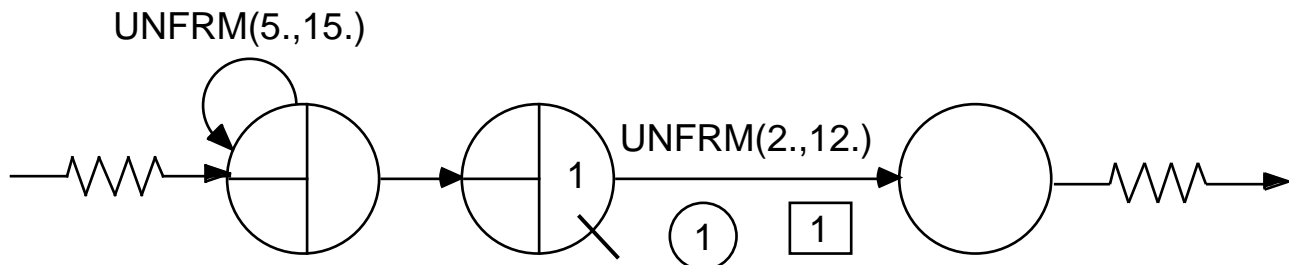


Von Zeitpunkt 0 ab
 werden im Abstand von 2 ZE (einzelne) entities erzeugt,
 die 5 ZE brauchen, um ihr "Lebensende" zu erreichen;
 wenn die 10-te dieser entities ihr Ende erreicht,
 wird Simulation beendet.

somit

Beispiel 7.2.1: Bankschalter in SLAM-Fassung

graphisch (Netz):



sprachlich (Programm):

```

NETWORK;
  CREATE,UNFRM(5.,15.);
  QUEUE(1);
  ACTIVITY(1)/1,UNFRM(2.,12.);
  TERMINATE;
ENDNETWORK;

```

Nächstes wichtiges Konzept

(nachdem "zeitliche Verzögerung"
"zeitliche Belegung eines servers"
durch regular / service activity bedacht)

ist Berücksichtigung "passiver Ressourcen",
"räumliche Belegung"

bei Betriebsmitteln, die in begrenzter Stückzahl vorhanden
(GPSS: storages)

dazu in SLAM: RESOURCES
mit "Deklaration", graph. RESOURCE block
sowie nodes AWAIT (anfragend, belegend)
und FREE (freigebend)

- RESOURCE block

- graphisch:



RLBL "resource label": Name

(zusätzlich automatisch: Nummer)

CAP initial vorhandene Anzahl "vergebbarer" Einheiten

IFL(s) "file"-Nummer(n), aus denen potentiell wartende entities bei Freiwerden von Einheiten "bedient"

- sprachlich:

RESOURCE/RLBL(CAP),IFLs;

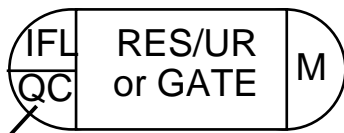
mit defaults: /"error"(1),"error";

oder RESOURCE/RNUM,RLBL(CAP),IFLs;

(mit expliziter RES Nummer: RNUM)

- AWAIT node (auch für nicht bespr. GATEs verwandt)

- graphisch:



RES RLBL (Name/Nummer) der RES, von der
UR Einheiten benötigt

IFL "file"-Nummer für entities, die warten müssen

QC räumliche queue-Kapazität

M maximale Zahl "beschickter" Ausgangs-branches

- sprachlich:

AWAIT(IFL/QC),RES/UR,...,M;

mit defaults:

("gemäß RESOURCE-block"/),"error"/1,..., ;

- FREE node

- graphisch:



RES RLBL (Name/Nummer) der RES, von der
 UF Einheiten freigegeben
 M maximale Zahl "beschickter" Ausgangs-branches

- sprachlich:

FREE,RES/UF,M;

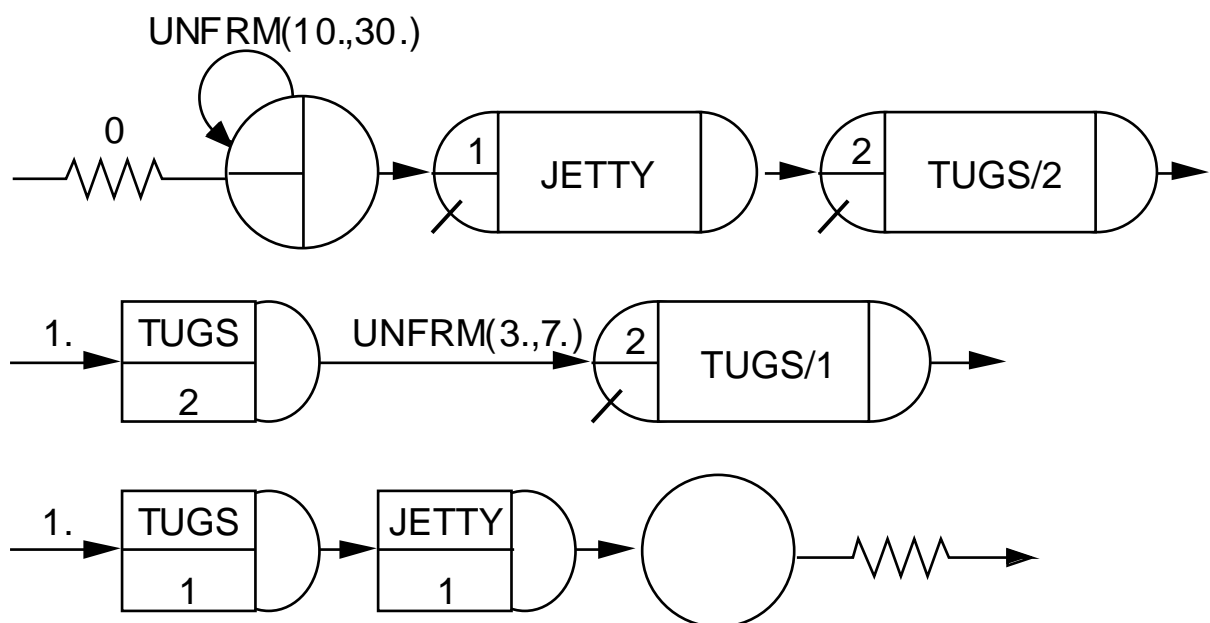
mit defaults:

"error"/1, ;

somit

Beispiel 7.2.2: Hafen-Bsp. 7.0.1 im SLAM-Diagramm

JETTY	2	1
TUGS	3	2



Viele weitere Facetten:

- weitere node-Typen
- weitere block-Typen
- spezielle "globale Variablen"
- weitere ZV-Generatoren
- Abbruch-Optionen
- automatische und beeinflussbare Statistiken
- debugging-Hilfen
- control statements
- ...

Sprach-Erscheinungsbild "alt":

- FORTRAN-Anlehnung
- festformatig
- ...

Neuerdings:

network subset durch interaktive Graphik unterstützt
(nicht "alles" aus Graphik "nochmal" zu codieren)
vgl. TESS, Stan85

7.3 DEMOS

Modellierungstechnische Konzepte

Modellwelten

Modellierungs-Paradigmen

von GPSS bzw. SLAM (als Beispiele) zweifellos "geschickt",
zugehörige sprachliche Spezifikationsformen eher
"altertümlich", abstoßend

Bei Versuchen, Szenario-spezifische Paradigmen

- zwar verfügbar zu machen,
- ohne aber auf modernere Sprachkonzepte zu verzichten,
insbesondere SIMULA als Gast- (host-) Sprache geeignet
(u.a.: SIMON, Hill75; DEMOS, Birt79),

da Übernahme von Konzepten einer
(in SIMULA programmierten) "Umgebungs"-CLASS
inhärenter Sprachbestandteil

Zur Erinnerung (vgl. Abschn. 3.5):

```
simset BEGIN
    {SIMULA-"Programm"}
END
```

machte Listenmanipulationskonzepte
für "Programm" verfügbar

Ähnlich auch selbstgeschriebene "Umgebung"en
möglich und per

```
EXTERNAL CLASS umgebung;
umgebung BEGIN
    {SIMULA-"Programm"}
END
```

diesbezügliche Konzepte in "Programm" nutzbar

Solche Entwicklungen waren SIMON, SIMON75 und (hier kurz zu beschreiben) DEMOS (1979).

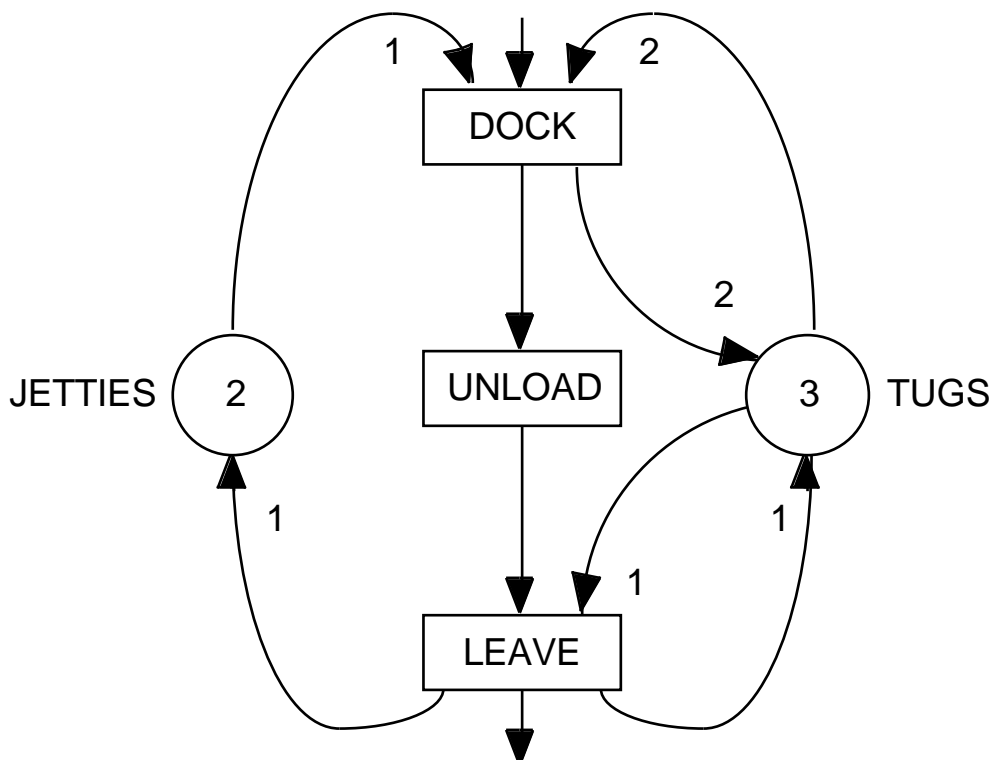
Auch bei DEMOS (wie zuvor bei GPSS, SLAM) graphische Darstellungen für Modellbeschreibung genutzt (hier ganz explizit als "Spezifikation")

Konkret: (eine Form von) **activity diagrams**
(existieren in and. Bereichen in versch. Formen)

Prinzip activity diagrams ("klingt uns sehr vertraut"):

dynamische Einheiten (entities)
verfolgen einen Plan / ein Muster über Aktivitäten;
zur Ausführung von Aktivitäten
passive Einheiten / Ressourcen (resources) benötigt,
die nur in begrenzter Anzahl vorhanden,
so daß u.U. auf sie zu warten ist

Als Einführung bekanntes Hafen-Beispiel:



Beispiel 7.3.1: Hafen-Bsp. im activity diagram

Deutlich: wieder die gleiche "Sicht"
 Transaktionen,
 Folge von Aktivitäten,
 Benutzung von Ressourcen

Und: auch bei DEMOS ist tatsächliche Nutzung
 von Graphik zur Modellspezifikation
 erst neuerdings in Einführung
 (z.B.: PIT, ESPRIT-Projekt IMSE)

(7.3.2) (erster Blick auf) DEMOS-Konzepte:

```

SIMSET CLASS demos;
  BEGIN
    {"entities" können deklariert, erzeugt werden;}
    CLASS entity(title); VALUE title; TEXT title;
    BEGIN
      {diese können u.a. "später gestartet" werden;}
      PROCEDURE schedule (t); REAL t; ... ;
      ...
    END entity;
    {"resources" gewisser Vielfachheit
    können deklariert, erzeugt werden;}
    CLASS res(title,avail);
      VALUE title; TEXT title; INTEGER avail;
    BEGIN
      {diese können u.a.
      "verlangt" + "zurückgegeben" werden;}
      PROCEDURE acquire(n); INTEGER n; ... ;
      PROCEDURE release(n); INTEGER n; ... ;
      ...
    END resource;
    {es kann "für Zeit" gewartet werden;}
    PROCEDURE hold(t); REAL t; ... ;
    {Zeit kann erfragt werden;}
    REAL PROCEDURE time; ... ;
    ...
  END demos;
  
```

(7.3.3) Hafen-Bsp. 7.0.1 in DEMOS
 (erste Version: deterministische Zeiten
 drei Schiffe/Boote)

```
EXTERNAL CLASS DEMOS;
DEMOS BEGIN
```

```
REF(RES) tugs, jetties;
```

```
ENTITY CLASS boat;
BEGIN
```

```
  jetties.ACQUIRE(1); tugs.ACQUIRE(2);
  HOLD(1.0);          {dock}
  tugs.RELEASE(2);
  HOLD(5.0);          {unload}
  tugs.ACQUIRE(1);
  HOLD(1.0);          {leave}
  tugs.RELEASE(1); jetties.RELEASE(1);
END boat;
```

```
tugs:-NEW RES("tugs",3);
jetties:-NEW RES("jetties",2);
NEW boat("boat").SCHEDULE(0.0);
NEW boat("boat").SCHEDULE(20.0);
NEW boat("boat").SCHEDULE(40.0);
HOLD(100.0);
END
```

Benennung der einzelnen Objekte
 durch automatische sequentielle Numerierung
 Wieder (z.T. automatische) report-Generierung

Erweiterung gewünscht: Zeiten als ZV,
 Generierung Realisierungen

DEMOS stellt ZV-Generatoren
in Form bestimmter Klassen bereit

Oberklassen	Klassen
RDIST für REAL-wertige ZV	NORMAL(a,b) NEGEXP(a) UNIFORM(a,b) ERLANG(a,b) EMPIRICAL(n)
IDIST für INTEGER-wertige ZV	RANDINT(a,b) POISSON(a)
BDIST für BOOLEAN ZV	DRAW(p)

Handhabung wie folgt:

- Generator-Namen werden deklariert
z.B. REF(RDIST) next, discharge
- zugehörige Generatoren werden instantiiert
z.B. next:–NEW UNIFORM("next boat",10.0,30.0);
discharge:–NEW UNIFORM("discharge",3.0,7.0);
- ZV-Realisierungen werden "gezogen"
z.B. next.SAMPLE
discharge.SAMPLE

RDIST, IDIST, BDIST sind ihrerseits Unterklassen einer
CLASS dist(title); VALUE title; TEXT title;
BEGIN INTEGER u; ... ; u:="suitable seed"; END dist

Diese setzt je ZZ-"Strom" (je "title")
selbständig einen Saatwert,
bei bis zu 500 Strömen der Länge 120000
"hinreichend verschieden" von anderen Saatwerten
(um "Überlappungen" ZZ-Folgen zu vermeiden)
Saatwerte überschreibbar
durch Wertzuweisung an (z.B) next.U, discharge.U

(7.3.4) Hafen-Bsp. 7.0.1 in DEMOS
(Standard-Version)

EXTERNAL CLASS DEMOS;
DEMOS BEGIN

REF(RES) tugs, jetties;
REF(RDIST) next, discharge;

ENTITY CLASS boat;
BEGIN
 { jedes Boot sorgt für "nächstes": }
 NEW boat("boat").
 SCHEDULE(next.SAMPLE);
 jetties.ACQUIRE(1); tugs.ACQUIRE(2);
 HOLD(1.0); {dock}
 tugs.RELEASE(2);
 HOLD(**discharge.SAMPLE**); {unload}
 tugs.ACQUIRE(1);
 HOLD(1.0); {leave}
 tugs.RELEASE(1); jetties.RELEASE(1);
END boat;

tugs:–NEW RES("tugs",3);
jetties:–NEW RES("jetties",2);

next:–NEW UNIFORM("next boat",10.0,30.0);
discharge:–NEW UNIFORM("discharge",3.0,7.0);

TRACE; {"trace"-output}

{ nur noch für "erstes" Boot gesorgt: }
NEW boat("boat").SCHEDULE(0.0);

HOLD(**28.0*24.0**); {28 Tage}

END

DEMOS stellt eine Reihe weiterer Klassen für Spezifikation von "Verkehrsnetz"-Problemen bereit

neben besprochener

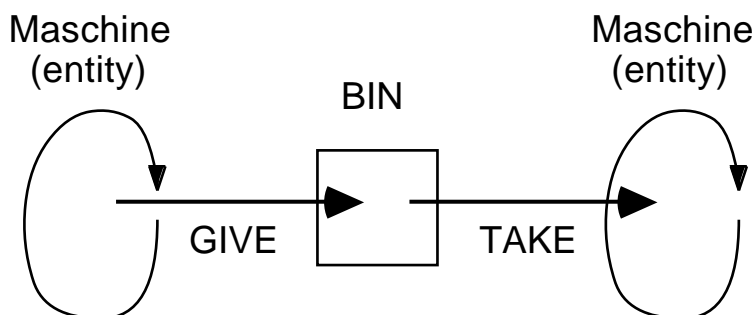
RES mit Prozeduren ACQUIRE, RELEASE

sind wesentlichste

BIN	mit Prozeduren	TAKE, GIVE	u.a.
WAITQ	mit Prozeduren	WAIT, COOPT	u.a.
CONDQ	mit Prozeduren	WAITUNTIL, SIGNAL	u.a.

wie RES die "material-oriented"-Sicht stützt, so BIN die "machine-oriented"-Sicht;

Skizze:



Aktive Kunden **und** Maschinen werden durch WAITQ unterstützt:

für gewisse Zeit **gemeinsamer** Zeitfortschritt beteiligter entities (Prozesse)

Initial gewünschtes WAIT_UNTIL-Konzept durch CONDQ ermöglicht:

allerdings nur "ungefähr", explizites SIGNAL erforderlich

Gewisse Ergebnisse

- automatisch gesammelt
- nach Simulationslauf automatisch ausgegeben

So:

- Statistiken über die div. ZZ/ZV-Ströme
 - Statistiken über BINs
- und Statistiken über einrichtbare "data collection"-Objekte

- Zähler

```
CLASS count(title); ... PROCEDURE update(v); ... ;
```

```
REF(COUNT)c;
```

```
...
```

```
c:-NEW COUNT("id_c");
```

```
...
```

```
c.UPDATE(2)
```

- "Wertestrom-Beobachter"

```
CLASS tally(title); ... PROCEDURE update(v); ... ;
```

```
REF(TALLY)t;
```

```
...
```

```
t:-NEW TALLY("id_t");
```

```
...
```

```
t.UPDATE(7.9)
```

- Histogramme

```
CLASS histogram(title,lb,ub,cells);
... PROCEDURE update(v); ... ;
```

```
REF(HISTOGRAM)h;
```

```
...
```

```
h:-NEW HISTOGRAM("id_h",0.0,10.0,5);
```

```
...
```

```
h.UPDATE(3.718)
```


Modellierungsumgebungen im Vormarsch !

so z.B.

- im Fertigungs- und Materialfluß-Bereich
(Übersicht BaCN96: SIMFACTORY, ProModel, AutoMod, Taylor, Witness, AIM, Arena)
- im Bereich Leistungs- und Zuverlässigkeits-Analyse von Rechen- und Kommunikations-Systemen
(Übersicht THRM94: SHARPE, QNAP, RESQ, NUMAS, MACOM, HIT, DSPNexpress, GreatSPN, UltraSAN, SPNP, ...
Übersicht Beil95: TimeNet, PEPSY-QNS, MOSES, HIT, MACOM, QPN-Tool, PEPP, PAPS, ...)

bei weitem nicht durchgängig alle "Wünsche" erfüllend, zum Teil groß, "mächtig", ... kommerziell: teuer

Allgemeine Erfahrungen:

- Modellierungs-Paradigmen, die im Anwendungsbereich "fremd" sind (nicht konstruktiv verwendet), sind schwer zu "vermitteln"
- Mit tatsächlicher Anwendung, wenn "Einfachheit" der M-Spezifikation erreicht, Tendenz zu (sehr) großen Modellen

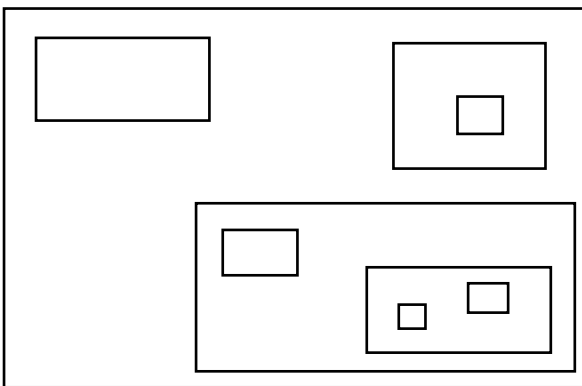
=> Denkweisen des Anwendungsbereichs berücksichtigen, Arbeiten mit großen Modellen erleichtern -> **Struktur !!**

HIT-Paradigma

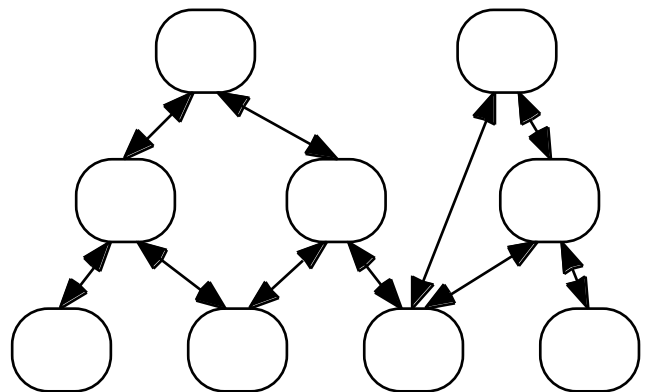
("mehrere" Analyse-Techniken,
s. BeMW89, Bütt93, Sczi93)

- greift gängige Strukturierungsprinzipien auf, so
 - Modularisierung (Abschottung, Autonomie)
 - Hierarchisierung (welche H-Relation ??)

Enthaltensein



Benutzung / Aufruf



=> objekt-basiertes,
blockstrukturiertes Denkschema ("Informatik")

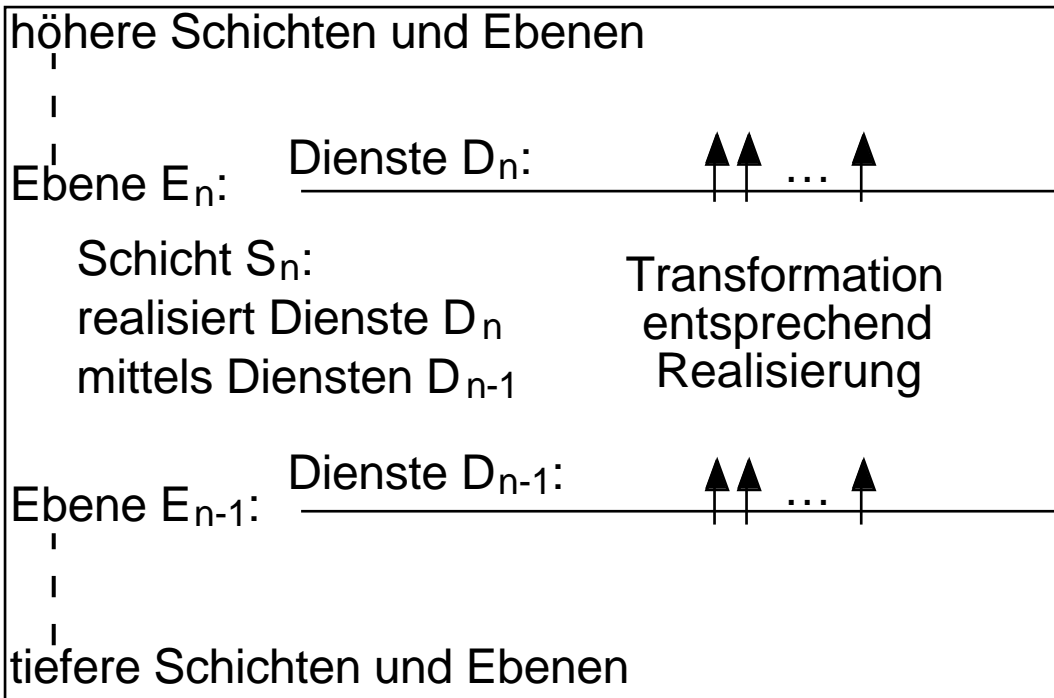
- greift Prozeßorientierung auf, so

- Prozeßmuster: (Halb-)Ordnung über Aktivitäten
- Aktivität: lokale Aktion oder Benutzung Ressource
- Prozeß: instantiiertes P-Muster

=> durchgängige Last / Maschine-Sicht /-Interpretation
("Informatik")

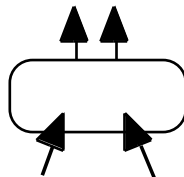
virtuelle Maschinen, Schichten / Ebenen,
Objekte / Komponenten mit Methoden -Import / -Export

- Schichten / Ebenen



globale Sicht: Hierarchie virtueller Maschinen
 oberste Schicht: "Umgebungsdynamik"
 unterste Schicht: "Standard-Komp'n"
 Raum-, Zeit-Verbrauch

- + Komponenten

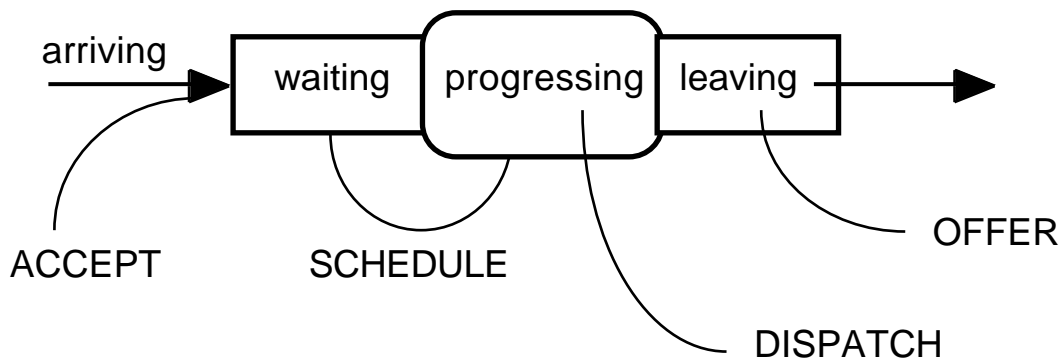


globale Sicht: Aufgabe systemweiter Ebenen
 lokale Sicht: benutzte vs bereitgestellte Dienste

HIT-(Modell-)Spezifikation

- per spezifischer HLL : Hi-SLANG
- gemischt graphisch geführt,
Menü-geführt,
sprachlich : HITGRAPHIC

- Komponenten sind autonome Objekte
parametrisierbare Steuerung durch
"Kontrollprozeduren" CONTROL PROCEDURES



4 Typen: alle mit sinnvollen "defaults"
in HITGRAPHIC aus Sortiment wählbar
in HI-SLANG programmierbar

- Analysemöglichkeiten (je nach Modell)
 - simulativ
 - analytisch-algebraisch
 - analytisch-numerisch
 - Aggregierungsoptionen
- simulative Analyse über (Typen von)

"streams"	+ vorbereitete	"standard streams"
- EVENT	z.B.:	TURNAROUNDTIME
- STATE	z.B.:	POPULATION
- COUNT	z.B.:	THROUGHPUT

 auch: "selbstgeschriebene"
mit je angepaßten statistischen Verfahren

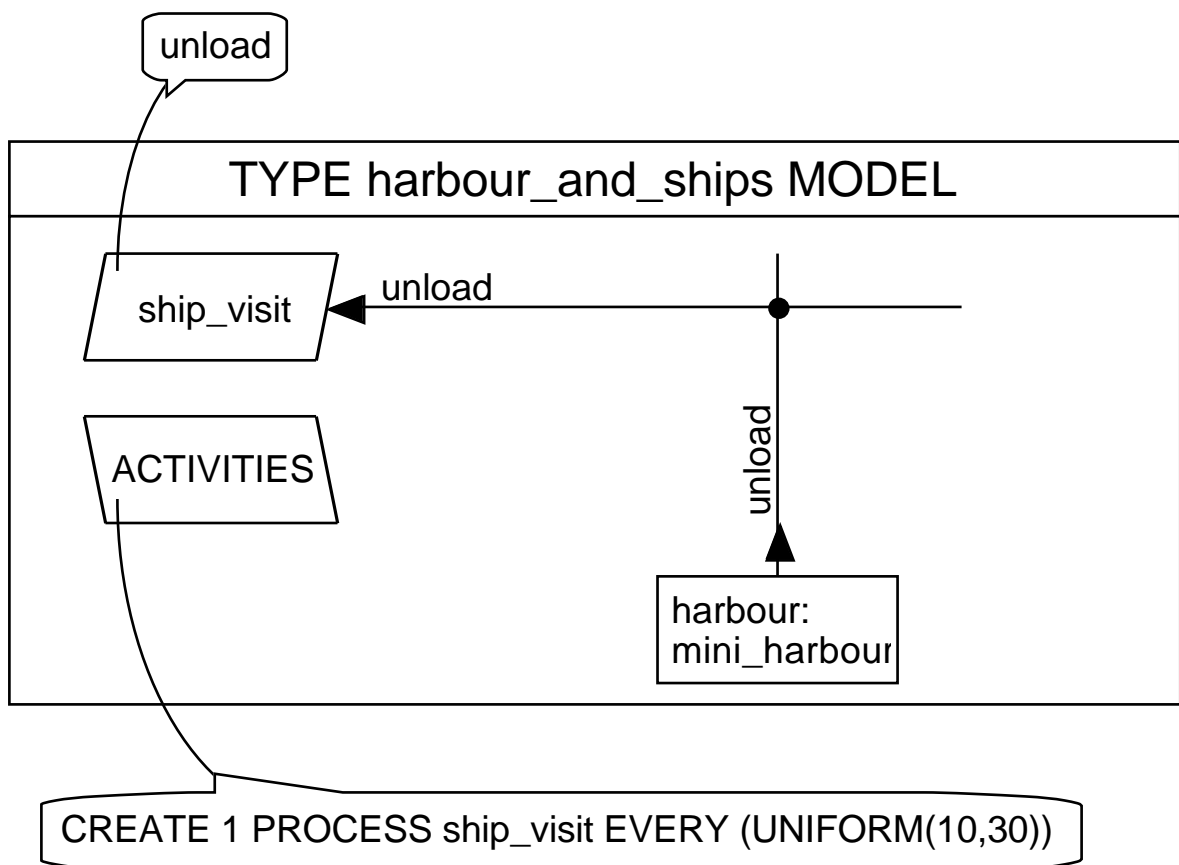
als EVALUATION OBJECTs an Komponenten geheftet,
in EXPERIMENT (zu MODEL Objekt) zusammengefaßt

- + Experimentierumgebung
- + Anschluß an statistische Pakete
- + ...

HIT-Beispiel:

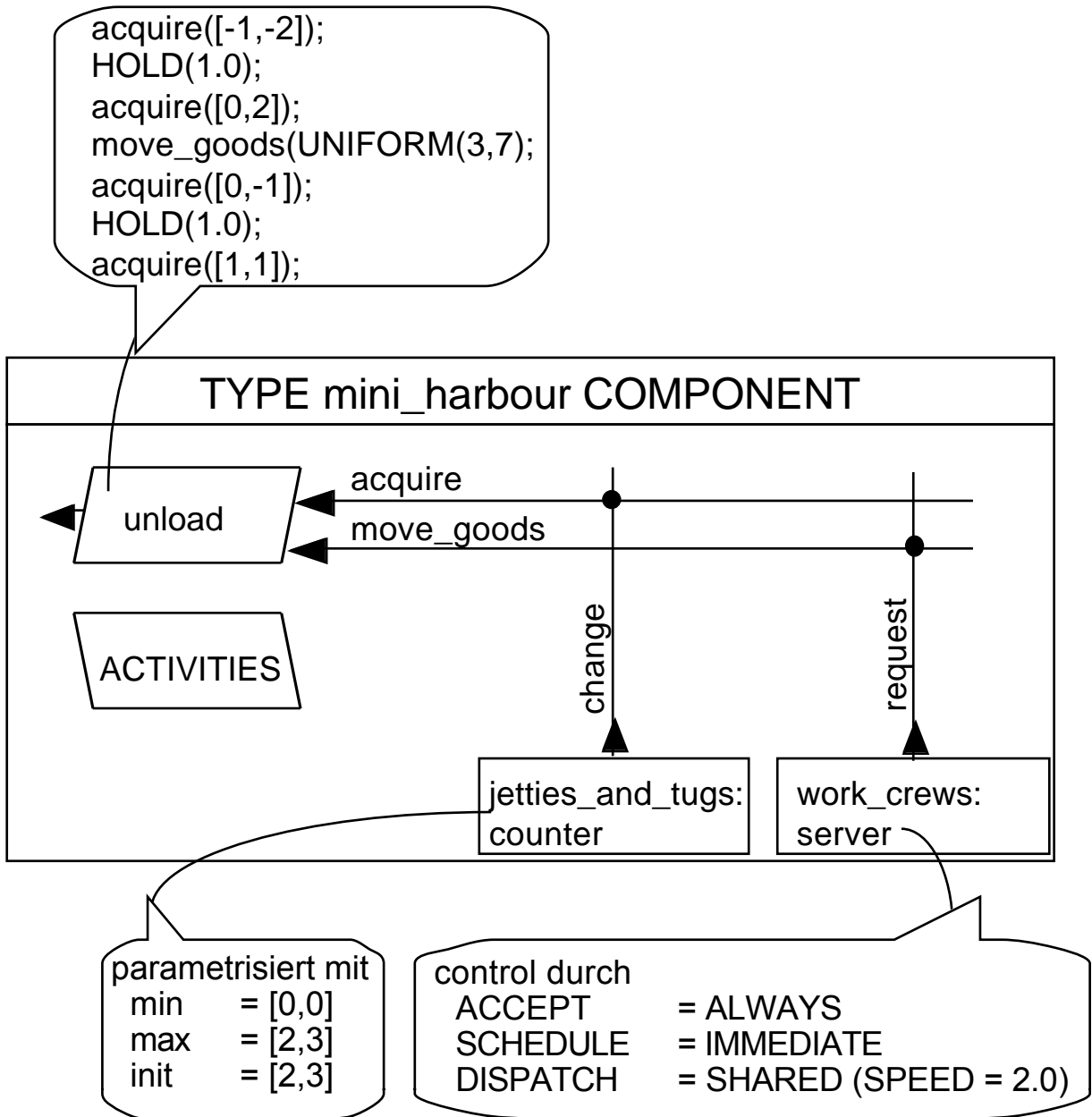
(obwohl untypisch) "Schiffe + Häfen":

- gewählt: hierarchisches Modell
- oberste Schicht: Modell



"Schiffe laufen in gewissen Abständen Hafen an
+ verlangen Abfertigung"

- zweite Schicht: Komponententyp mini_harbour
 nutzbarer Dienst unload



- Dynamik unload (bewußt) "à la DEMOS"
- aber mit Anreicherungen
 so: jetties und tugs gemeinsam verwaltet
 und 2 work crews vorhanden
 => Aushilfe: falls nur 1 Schiff, doppelte
 "Geschwindigkeit Entladung"