Simulation diskreter Systeme

Peter Junglas 12.09.2019

Inhaltsverzeichnis

Einführung in Begriffe und Methoden

- Begriffsklärungen:

  Simulation
  - Experimentieren mit Modellen von interessierenden Systemen

  hier nur **dynamische Systeme** (zeitabhängig)

  beschrieben durch **Zustandsgrößen**
  - zeitabhängige, voneinander unabhängige Größen, die die zeitliche Entwicklung eines Systems festlegen

- Zeitverhalten dynamischer Systeme:

  kontinuierlich
  - Werte ändern sich grundsätzlich zu jedem Zeitpunkt
  - z.B. Fahrzeugschwingungen
  - häufig durch DGLs beschrieben

  diskret
  - Werte ändern sich nur zu endlich vielen Zeitpunkten
  - z.B. Zahl der Studenten in einem Kurs
  - etwa durch Entwicklungsgleichungen beschrieben

  hybrid
  - Mischform aus kontinuierlich und diskret
  - grundsätzlich kontinuierliche Zustandsgrößen ändern sich gelegentlich auf unstetige Weise
  - z.B. hüpfender Ball oder Haft-/Gleitreibung

  hier nur diskrete Systeme (bzw. hybride Systeme in Kap. 9)

- **Ereignis** (Event):

  sprunghafte Änderung (mindestens) einer Zustandsgröße
  - findet zu einem Zeitpunkt statt → braucht keine Zeit

  Zeitpunkte t_n der Änderungen
  - periodenorientiert = in festen Zeitabständen (Taktsteuerung einer CPU)
  - durch Systemverhalten gegeben (Ablaufsteuerung einer Fertigungsmaschine)
  - zufällig (Ankunft eines Fahrzeugs an einer Kreuzung)

  Zustandsgrößen können durch Folgen statt durch Funktionen beschrieben werden

  $x_n := x(t_n) \qquad n = 1, 2, 3, ...$

- Einige Anwendungsbereiche:

  Protokolle zur Netzwerk-Kommunikation
  - Wie kann man eine große Datei sicher über das Internet verschicken?

  Design/Analyse von Fertigungsanlagen
  - Wie groß müssen Zwischenlager vor einer Maschine sein, um eine bestimmte Auslastung zu garantieren?

  Logistische Versorgungsketten (Lagerhaltung, Distribution)
  - Ab welchem Lagerbestand soll ein Distributor Produkte nachbestellen?

  Transportsysteme (Autobahnen, Flughäfen etc.)
  - Wie müssen Ampeln geschaltet werden, um einen möglichst hohen Verkehrsfluss zu erreichen?

  Geschäftsprozesse in einem Unternehmen
  - Welche Schritte sind bei der Bearbeitung einer Beschwerde nötig?

- Ablauf einer Simulationsstudie:
  - Formulieren des Problems und Planung
  - Sammeln von Daten
  - Erstellen von Modellen
  - Überprüfen der Modelle
  - Planung und Durchführung von Experimenten
  - Analyse der Ergebnisse
  - Dokumentation und Umsetzung
- Überprüfen der Modelle:
  - Verifizierung
    - prüfen auf Korrektheit eines Modells
    - Formulierung des Modells ist richtig
    - keine logischen Fehler
  - Validierung
    - prüfen auf Gültigkeit eines Modells
    - Struktur und Ergebnisse sind dem realen System hinreichend ähnlich
  - viele Modelle enthalten zufällige Größen
    - → Stochastik wichtig zur Modellierung, Validierung und Analyse der Ergebnisse
- Modellierungsmethoden:
  - große Zahl sehr unterschiedlicher Verfahren
  - abstrakt-mathematische Modelle
    - Markov-Ketten, Bediensysteme, DEVS
  - Modellierungssprachen
    - GPSS
  - graphische Verfahren
    - abstrakt (Endliche Automaten, Stategraphen, Petri-Netze)
    - mit allgemeinen Komponenten (prozess-basiert, transaktions-basiert, agenten-basiert)
    - konkret für spezielle Anwendungen (Logistik, Geschäftsprozesse)
- Software:
  - große Auswahl an (überwiegend kommerziellen) Paketen
  - Matlab/Simulink von Mathworks
    - hauptsächlich kontinuierlich
    - Zusatzpaket Stateflow für Stategraphen
    - Zusatzpaket SimEvents für transaktions-basierte Modellierung
  - Dymola von Dassault Systèmes
    - Modelica-basiert, hauptsächlich kontinuierlich
    - Zusatzpakete für Zustandsautomaten, Petrinetze, DEVS-Modellierung
  - Arena von Rockwell Automation
    - prozess-basiert
    - im Ranking von [1] auf Platz 1
  - AnyLogic von The AnyLogic Company
    - kombiniert mehrere Ansätze
    - agenten-basiert, prozess-basiert und System Dynamics (kontinuierlich)

PlantSimulation von Siemens PLM Software

- speziell für Produktionsprozesse und logistische Abläufe
- materialfluss-basiert (ähnlich prozess-basiert)

- Einordnung der Veranstaltung:

  großes Spektrum an Vorgehensweisen

  - mathematisch/theoretisch [L3]
  - praktisch mit konkretem Programm [L7]
  - systematisch mit Durchführungsmodellen [L9]

  Methodik hier

  - ausgehend von konkreten Beispielen
  - Schwerpunkt auf Modellierungsverfahren
  - andere Aspekte überwiegend im Beispiel-Kontext

Periodenorientierte Systeme

- Modellierung einer Schule
- Simulation mit Zufallsgrößen
- Mathematische Beschreibung

Modellierung einer Schule

- Grundprinzip:

  Zustandsgrößen x(t) ändern sich in festen Zeitabständen t_i (getaktet)

  $t_n = n\,\Delta t$, n = 0, 1, 2, ...

  - Größe des Zeitschritts (**Sample time**) Δt

  Verhalten der Zustandsgrößen $x_n \equiv x(n)$ nicht durch DGLs beschrieben, sondern z. B. durch Entwicklungsgleichungen

  $x(n+1) = f(n, x(n))$, n = 0, 1, 2, ...

  - Startwert x(0) gegeben

  häufig hängt f nicht vom aktuellen Zeitschritt n ab (zeitunabhängig)

- Beispiel "Schülerzahlen einer Oberstufe":

  Zahl der Schüler x_{11}, x_{12}, x_{13} in Klassen 11, 12, 13

  ändern sich in jedem Jahr durch
  - Zugang x_{in} in der 11. Klasse
  - Zahl der Wiederholer, Anteil W_{11}, W_{12} etc.
  - Zahl der Abbrecher, Anteil A_{11} etc.
  - Zahl der versetzten Schüler vom letzten Jahr

  insgesamt beschrieben durch die Gleichungen

  $$\begin{aligned} x_{11}(k+1) &= x_{in}(k) + W_{11}\,x_{11}(k) \\ x_{12}(k+1) &= (1 - W_{11} - A_{11})\,x_{11}(k) + W_{12}\,x_{12}(k) \\ x_{13}(k+1) &= (1 - W_{12} - A_{12})\,x_{12}(k) + W_{13}\,x_{13}(k) \end{aligned}$$

  Zahl der Abiturienten gegeben durch

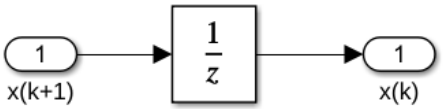
  $$x_{abi}(k+1) = (1 - W_{13} - A_{13})\,x_{13}(k)$$

  genaue Bedeutung
  - x_{11}(k) = Klassenstärke zu Beginn des Schuljahres k

  Fragestellungen
  - Wie groß sind die (voraussichtlichen) Klassengrößen?
  - Wieviele fertige Abiturienten gibt es jedes Jahr?

- Modellierung in Simulink:

  zentraler Block: `Discrete/Unit Delay`
  - verzögert Wert um einen (diskreten) Simulationsschritt
  - übernimmt Rolle von Integralblock ("1/s") bei kontinuierlichen Systemen

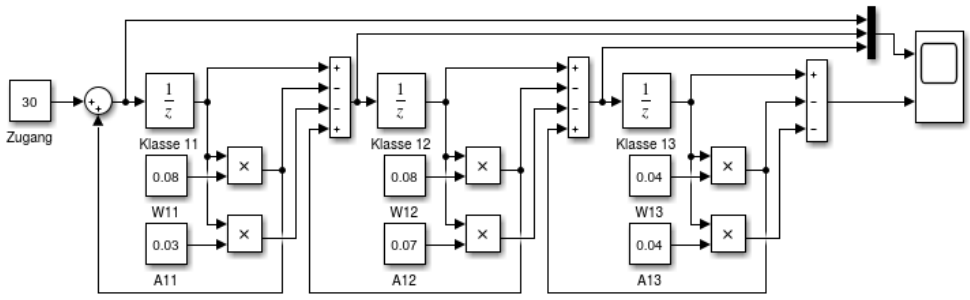
  

  Parameter von `Unit Delay`
  - `Initial condition` = Startwert x(0)
  - `Sample Time` = Schrittweite
  - besser: `Sample Time` = -1 und Wert im Solver vorgeben

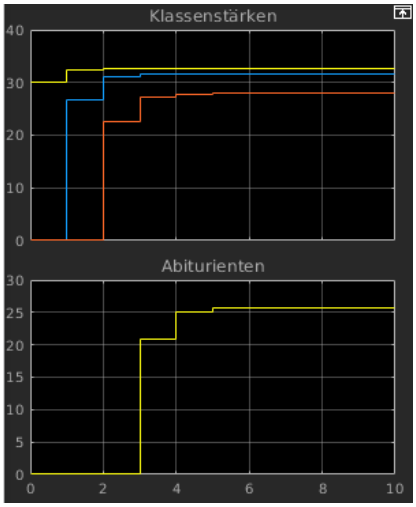
  Simulationsparameter
  - `Solver type: Fixed-step`

- `Solver: discrete (no continuous states)`
- `Fixed-step size: 1` ("ein Jahr")

Komplettmodell `schule1`



Anzeige der Klassenstärken $x_N(k)$ und der Zahl der Abiturienten $x_{abi}(k)$

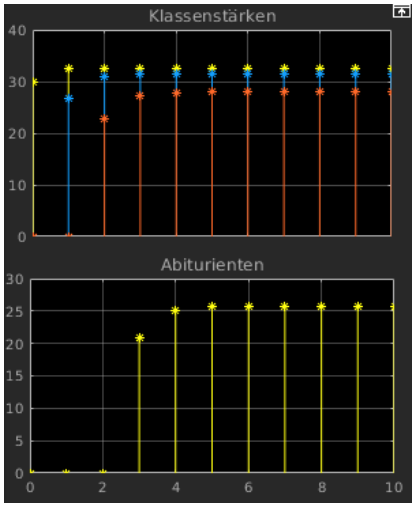


- Darstellung der Ergebnisse:

`Style = Auto` (hier = `Stairs`)

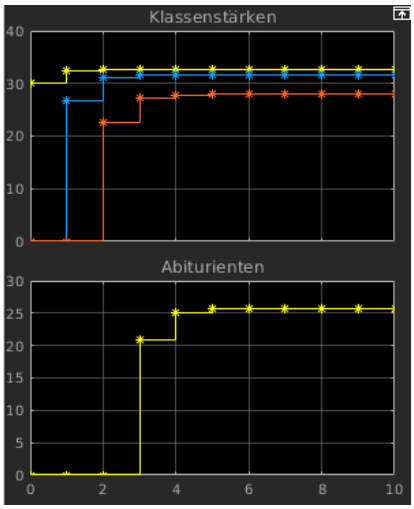
- liefert durchgängige Kurven, x(k) hat konstanten Wert für k $\in$ (n, n+1)
- welcher Wert gilt an der Sprungstelle?

`Style = Stem`



- präziser, x(k) nur definiert für k ganzzahlig
- oft unübersichtlich

`Style = Stairs + Marker`



- häufig guter Kompromiss

sinnvolle Darstellung schützt vor Fehlinterpretation der Ergebnisse!

- Übersichtlichkeit durch Hierarchien von Komponenten:

  Problem

  - `schule1` ist komplex und unübersichtlich
  - Blockstruktur wiederholt sich für jede Schulklasse

  Lösung: eigener Block `KlasseB` zur Beschreibung einer Klasse

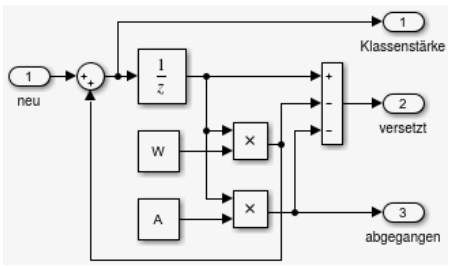
  Eingang

  - I = Zahl neuer Schüler (aus unterer Klasse)

  Ausgänge

  - K = Klassenstärke (zu Beginn des Schuljahres)
  - V = Zahl der versetzten Schüler (der vorigen Klasse!)
  - A = Zahl der abbrechenden Schüler (der vorigen Klasse!)

  fertiges Submodell



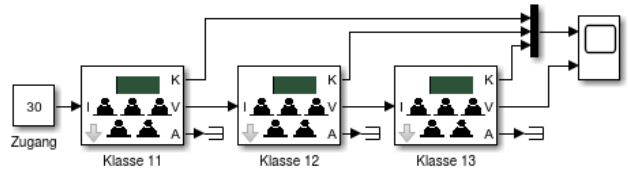
  Maske erstellen mit

  - Parameter für Wiederholer- und Abbrecherrate sowie Anfangsstärke
  - Icon und Beschriftung der Anschlüsse
  - Hilfstext

  eigene Block-Library `discSimLib` für neuen Block (und weitere)

  - Gesamtmodell enthält nur einen Verweis
  - Änderung des Bibliotheksblocks aktualisiert alle Instanzen

  Gesamtmodell `schule2`



Zahl der Abgänger wird nicht verwendet

- erzeugt Warnungen im Matlab-Fenster
- unterbinden mit `Terminator`-Block

- Mathematische Beschreibung des Blocks `KlasseB`:

  definiere Zustandsgröße z(k) als Ausgang des `Unit Delay`-Blocks zur Zeit k

  Parameter W, A durch Maske gegeben

  Eingangswert I(k) zur Zeit k

  Berechnung des nächsten Zustandswerts

  $$z(k+1) = I(k) + W\,z(k)$$

  Berechnung der Ausgangswerte

  $$\begin{aligned} K(k) &= I(k) + W\,z(k) \\ V(k) &= (1 - W - A)\,z(k) \\ A(k) &= A\,z(k) \end{aligned}$$

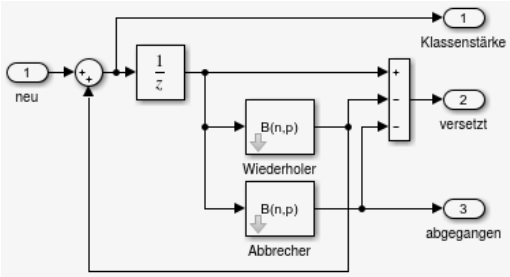
Simulation mit Zufallsgrößen

- Einbau stochastischer Größen:

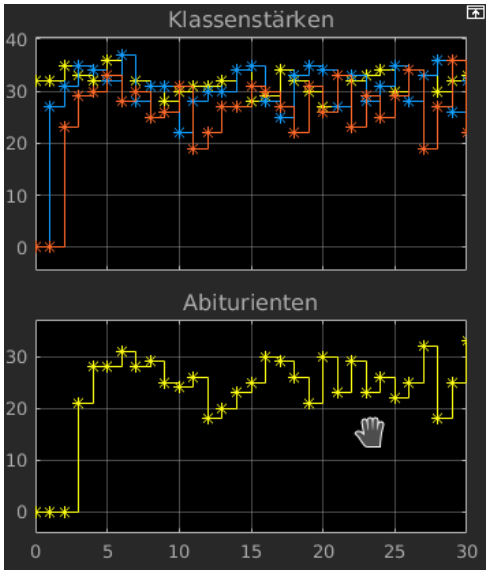
  zufällige Anzahl von Anmeldungen
  - "normalverteilt", aber ganzzahlig
  - Block Anmeldungen ersetzt Konstante

  zufällige Anzahl von Wiederholern und Abbrechern
  - binomialverteilt B(n, p), n = Klassengröße, p = mittlere Quote

  Block `KlasseC` hat dann ganzzahlige Klassengrößen



  Ergebnis von `schule3` sieht sinnvoll aus



- Block für B(n,p)-verteilte Zufallszahlen:

  Anzahl n der Wiederholungen als Eingang

  Erfolgswahrscheinlichkeit p als Parameter

  Berechnung einfach mit Matlab-Funktion `binomial.m`

  Parameter `seed` für Reproduzierbarkeit
  - `seed` ≥ 1 initialisiert Zufallszahlengenerator `Rng`
  - `seed` = -1 erzeugt neue Werte bei jedem Lauf (initialisiert mit Uhrzeit)

  Problem
  - mehrere Blöcke setzen die `seed` für den gleichen `Rng`
  - der letzte (in irgendeiner Reihenfolge) setzt sich durch

  Lösung
  - Blöcke haben normalerweise `seed` = 0, bewirkt nichts
  - nur ein Block bekommt anderen `seed`

- vgl. Start-Callback `startBinomial.m` des Binomial-Blocks

genaues Verhalten von `Binomial`

- Zufallszahl B(I,p) mit festem p und I vom Eingang
- Wert I am Eingang wirkt sich *sofort* aus, d. h.

  Binomial(k) = B(I(k), p)

- Analyse des Ergebnisses:

  Beobachtung
  - Einlaufphase, danach gleichmäßiges Verhalten

  Wie beschreibt man die Klassengrößen?
  - stochastische Größen
  - ermitteln: Mittelwert und Standardabweichung

  praktische Durchführung
  - Festlegung der Einlaufphase
  - ein langer Lauf oder viele kürzere Läufe?

  vgl. Aufgabe 1

- Erweitertes Modell `schule4`:

  zusätzlich: Rückstufung zum Halbjahr

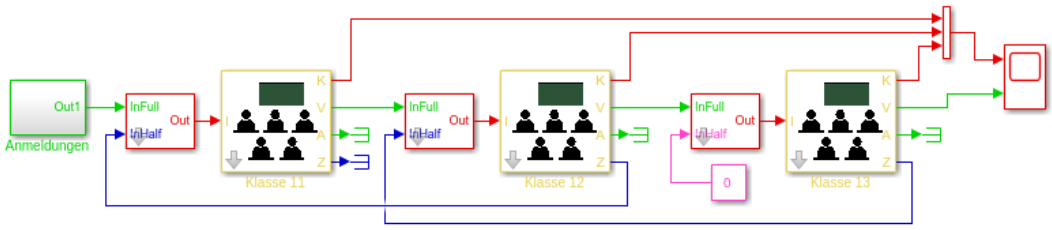
  Anzahl der Zurückgestuften: B(K, Z)
  - Z = mittlere Quote (Parameter)
  - K = Klassenstärke zu Beginn des Jahres

  zum Halbjahr gehen einige (in die untere Klasse) und kommen einige (von der oberen Klasse)
  - `KlasseD` hat einen weiteren Ausgang Z, aber keinen weiteren Eingang
  - Eingang I enthält zu Beginn des Schuljahres und des Halbjahres jeweils verschiedene Werte

  Abbrecher- und Wiederholerzahlen beziehen sich auf die Klassenstärke zu Beginn des 2. Halbjahres
  - diskutierbar, letzlich eine Frage der Definition der Quoten W und A
  - für Abbrecher komplexeres Modell denkbar, das Abbrecher für beide Halbjahre getrennt bestimmt

  Signale mit verschiedenen Samplezeiten → Verhalten wird komplizierter!
  - Angabe der Samplezeiten als $t_S = [\Delta t, t_0] \rightarrow t_n = t_0 + n\,\Delta t$
  - Klassenstärke jedes halbe Jahr, $t_S = [0.5, 0]$
  - Wiederholer und Abbrecher jedes volle Jahr, $t_S = [1, 0]$
  - Zurückgestufte jährlich zum Halbjahr, $t_S = [1, 0.5]$

  Gesamtmodell

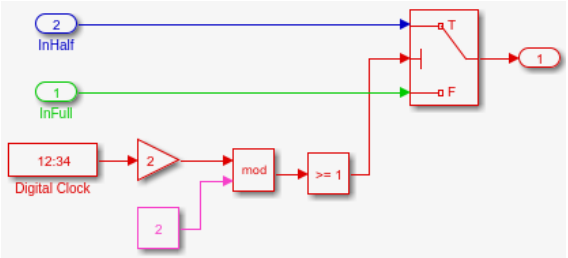


  - farbige Darstellung der Samplezeiten mit `Display/Sample Time/Colors`
  - Legende für die Farben

Color	Description	Value
	Fixed in Minor Step	[0,1]
	Discrete 1	0.5 (period)
	Discrete 2	1 (period)
	Discrete 3	1 (period) 0.5 (offset)
	Constant	Inf
	Hybrid	N/A

Komponente `TimeSwitch`

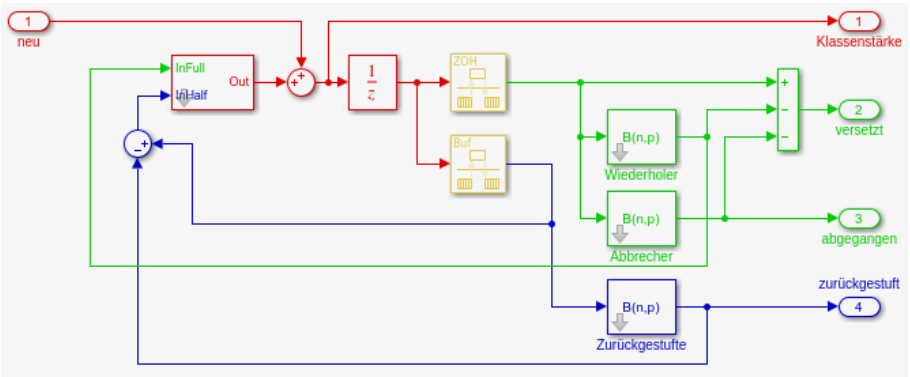
- führt Signale mit t_S = [1, 0] und t_S = [1, 0.5] zusammen zu einem Signal mit t_S = [0.5, 0]



- `DigitalClock` mit `Sample Time` = 0.5

- Implementierung von `KlasseD`:

  Aufbau



  Block `Unit Delay` hat `Sample Time = [0.5, 0]`

  Block `RateTransition`

  - legt Sample Time des Ausgangs fest
  - oben [1,0] → merkt sich den Wert bei t = n für 1 s
  - unten [1,0.5] → merkt sich den Wert bei t = n+0.5 für 1 s
  - unten Parameter `Ensure deterministic data transfer` ausgeschaltet!

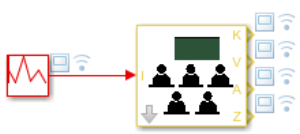
  Eingang des `Unit Delay`

  - volles Jahr: neue vom Eingang (= Versetzte) + Wiederholer
  - Halbjahr: Zurückgestufte vom Eingang + (Klassenstärke - Zurückgestufte)

- Testen von `KlasseD`:

  komplexes Modell, einzeln genau testen!

  einfaches Testmodell `testKlasseD`



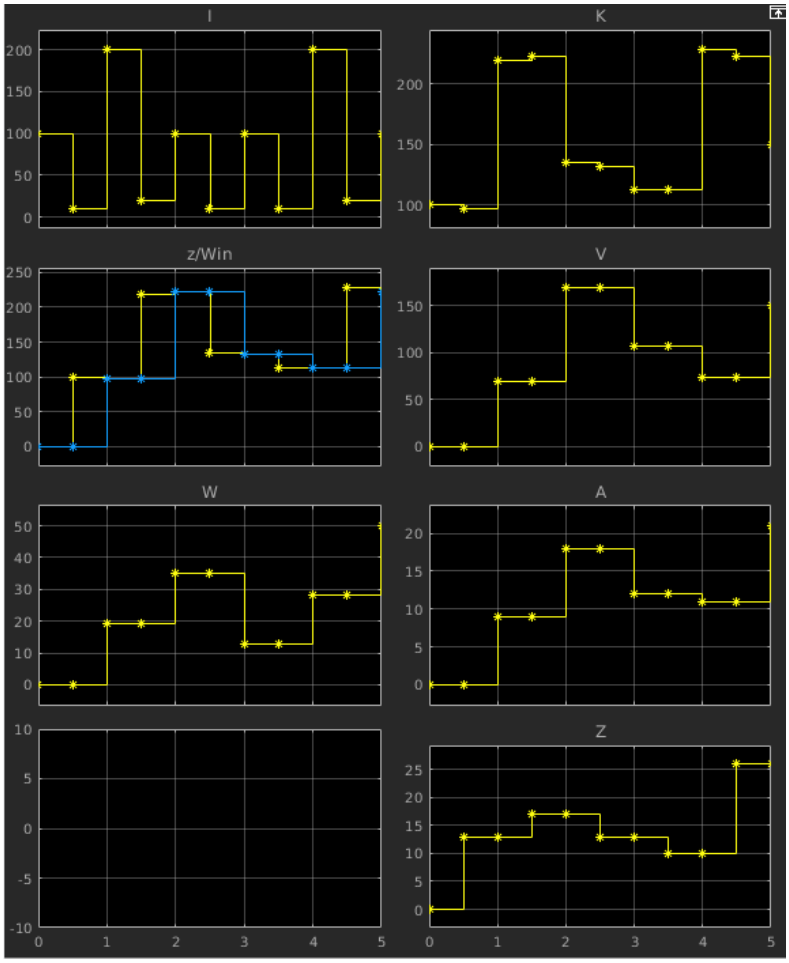
  - Eingangswerte für I (mit Taktrate = 0.5, Start bei 0)

[100 10 200 20 100 10]

Parameter von `KlasseD`

- W = 0.2, A = 0.1, Z = 0.1
- ermöglicht überschlägige Plausibilitätsprüfung der Zufallszahlen

Darstellung relevanter Signale



zusätzlich zu den Anschlüssen sind dargestellt

- W: Zahl der Wiederholer
- z/Win: Ausgang des Unit Delay/Eingang des Wiederholer-Blocks

erscheint plausibel

genauer: manuelle Prüfung der Werte

t	I	K	Z	W	A	V
0.0	100	100	-	0	0	0
0.5	10	97	13	-	-	-
1.0	200	219	-	19	9	69
1.5	20	222	17	-	-	-
2.0	100	135	-	35	18	169
2.5	10	132	13	-	-	-
3.0	100	113	-	13*	12	107

alles ok, lediglich der *-Wert ist etwas klein

- n = 132, p = 0.2, damit

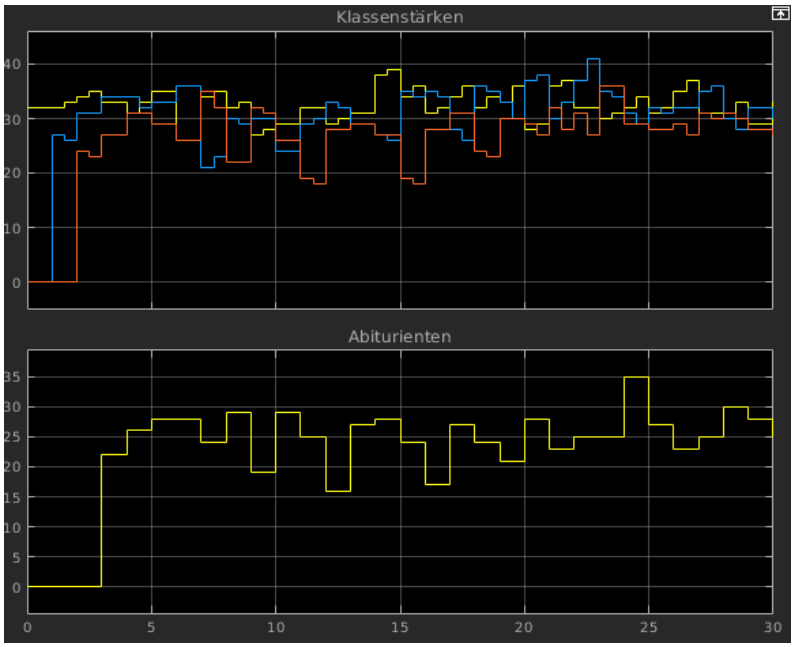
$$E(W) = np = 26.4, \qquad \sigma = \sqrt{np(1-p)} = 4.6$$

- Abweichung knapp 3σ

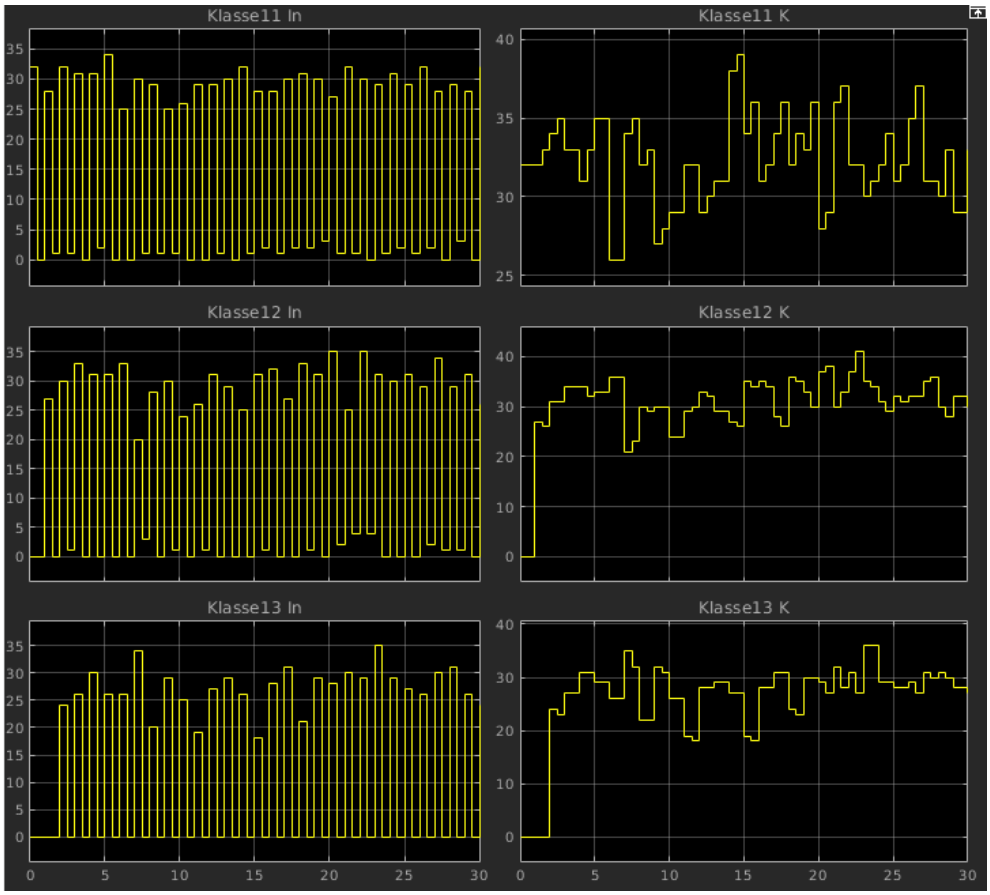
- Überprüfung: Eingang Win(t = 3) = 132, ok

- Ergebnis von `schule4`:

  Gesamtergebnis



  detailliert zur Prüfung



  sieht sinnvoll aus

  Modell grob verifiziert

  - Validierung: Vergleich der benutzten Zufallsverteilungen mit realen Werten

  schließlich "Arbeitsläufe" und statistische Analyse der Ergebnisse

Mathematische Beschreibung

- Zustandsraum-Darstellung:

  verbreitete Beschreibungsweise diskreter Systeme und ihrer Komponenten

  Grundidee
  - System bekommt Eingangswerte v(k)
  - ändert daraufhin seinen inneren Zustand z(k)
  - und produziert Ausgangswerte w(k)

  v, z, w sind i. A. Vektoren (mehrerere Komponenten)

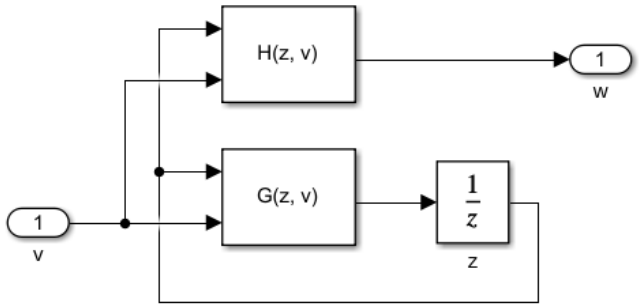
  konkret

  $$\begin{aligned} z(k+1) &= G(z(k), v(k)) \\ w(k) &= H(z(k), v(k)) \end{aligned}$$

  - zusammen mit Startwert

    $z(0) = z_0$

  in Simulink leicht zu implementieren

  Beschreibung komplexer Systeme
  - Zustandsraum-Darstellung für alle Komponenten
  - Gesamtsystem definiert über Verbindungen zwischen den Komponenten sowie externen Ein- und Ausgängen ("Simulink"-artig)
  - hierarchisch: auch Komponenten können durch Aufbau aus Subkomponenten beschrieben werden
  - alternativ: Zustandsraumdarstellung für Komplettsystem, oft umständlich

- Beispiel `KlasseB`:

  Variablen
  - z = Ausgang des `Unit Delay`-Blocks
  - v = Eingangswert I
  - w = Vektor der Ausgangswerte [K, V, A]

  Funktionen

  $$\begin{aligned} G(z, v) &= v + Wz \\ H(z, v) &= \begin{pmatrix} v + Wz \\ (1 - W - A)z \\ Az \end{pmatrix} \end{aligned}$$

  - mit Parametern W, A

- Zustandsraum-Darstellung bei expliziter Zeitabhängigkeit:

  Problem: z oder w hängen direkt von der Zeit n ab

  Idee: Zustand enthält die Zeit

  Beispiel Entwicklungsgleichung

x(n+1) = f(n, x(n))

- keine Eingangs- und Ausgangswerte
- z und G definiert durch

$$z = \begin{pmatrix} t \\ x \end{pmatrix} =: \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

$$G(z) = \begin{pmatrix} z_1 + 1 \\ f(z_1, z_2) \end{pmatrix}$$

- mit Anfangswert

$$z(0) = \begin{pmatrix} 0 \\ x(0) \end{pmatrix}$$

- Zustandsraum-Darstellung bei Abhängigkeit von früheren Zustandswerten:

  Problem: neuer Zustandswert hängt von früheren Zustandswerten ab (nicht nur dem letzten)

  Idee: Zustand enthält ältere Werte explizit

  Beispiel Fibonacci-Folge

$$\begin{aligned} x(n+1) &= x(n) + x(n-1) \\ x(0) &= 1,\ x(1) = 1 \end{aligned}$$

  - keine Eingangs- und Ausgangswerte
  - z und G definiert durch

$$z(k) = \begin{pmatrix} x(k) \\ x(k+1) \end{pmatrix} =: \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$$

$$G(z) = \begin{pmatrix} z_2 \\ z_1 + z_2 \end{pmatrix}$$

  - mit Anfangswert

$$z(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- Stochastische Zustandsraum-Darstellung [2]:

  `KlasseC` enthält Zufallsgrößen, deren Verteilung B(z, W) von z abhängt

  Variablen wie bei `KlasseB`
  - z = Ausgang des Unit Delay-Blocks
  - v = Eingangswert I
  - w = Vektor der Ausgangswerte [K, V, A]

  Funktionen mit Hilfe von B(n,p)

$$\begin{aligned} G(z, v) &= v + B(z, W) \\ H(z, v) &= \begin{pmatrix} v + B(z, W) \\ z - B(z, W) - B(z, A) \\ B(z, A) \end{pmatrix} \end{aligned}$$

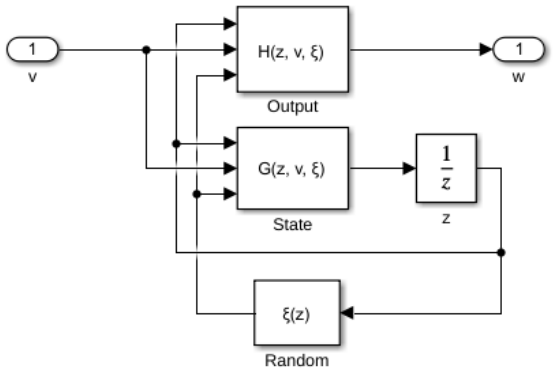
  Problem
  - B(z, W) kommt dreimal vor, ist aber immer dieselbe Zahl (analog B(z, A))
  - bei Implementation mit 3 Blöcken ergeben sich verschiedene Werte, Block darf also nur einmal im Modell vorkommen
  - besser: Symbol für einen konkreten Zufallswert kommt auch in der Beschreibung nur einmal vor

  Erweiterung: **stochastische Zustandsraum-Darstellung**

- neu: Zufallsvektor ξ(z), dessen Verteilung vom Wert des Zustandsvektors z abhängt
- Gleichungen entsprechend erweitert

$$\begin{aligned} z(k+1) &= G(z(k), v(k), \xi(z(k))) \\ w(k) &= H(z(k), v(k), \xi(z(k))) \end{aligned}$$

Implementation in Simulink



konkret für `KlasseC`

$$\begin{aligned} \xi(z) &\sim \begin{pmatrix} B(z,W) \\ B(z,A) \end{pmatrix} \\ G(z,v,\xi) &= v + \xi_1 \\ H(z,v,\xi) &= \begin{pmatrix} v + \xi_1 \\ z - \xi_1 - \xi_2 \\ \xi_2 \end{pmatrix} \end{aligned}$$

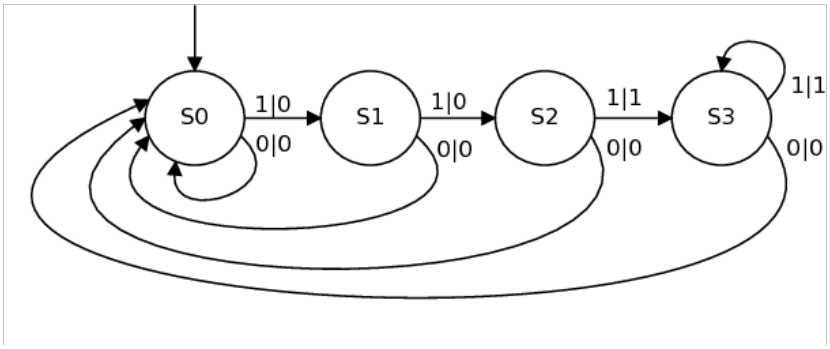
- Aufgaben:
  - Aufgabe 1
  - Aufgabe 2
  - Aufgabe 3

Endliche Automaten

- Erkennung eines Eingangssignals:

  Aufgabenstellung
  - automatische Fertigungskontrolle
  - kein Problem bei bis zu 2 Fehlern
  - Fehlersignal (manuelles Eingreifen nötig) bei 3 aufeinanderfolgenden Fehlern

  Darstellung als endlicher Automat mit vier Zuständen
  - Eingangsfolge aus 0 ("ok") und 1 ("Fehler")
  - Ausgabe 1 bei drei aufeinanderfolgenden Fehlern

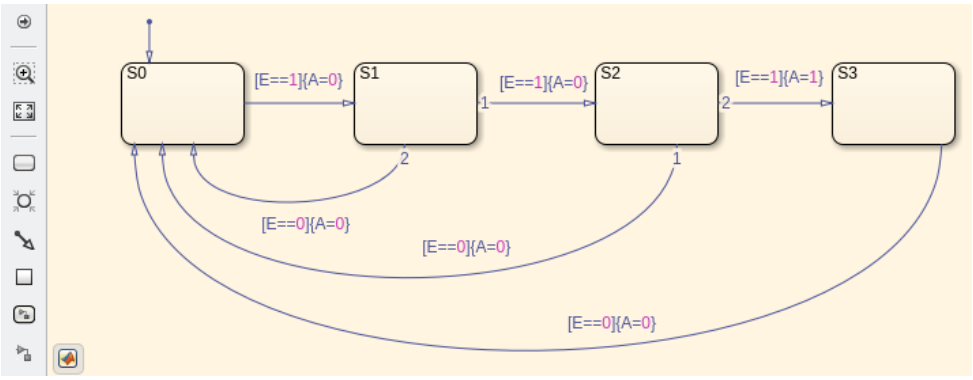


  - definiere: Was geschieht bei Eingang 1 in Zustand S_3?

  Notation
  - Pfeil auf Startzustand
  - Pfeil x|y von Sa zu Sb ⇔
    wenn im Zustand Sa Eingabe = x, dann Ausgabe y und Übergang in Sb

- Beispiel `fehlerdetektor1A`:

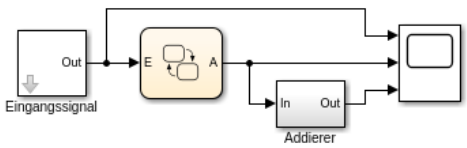
  Aufbau mit Stateflow



  - Pfeile zu gleichem Zustand weglassen

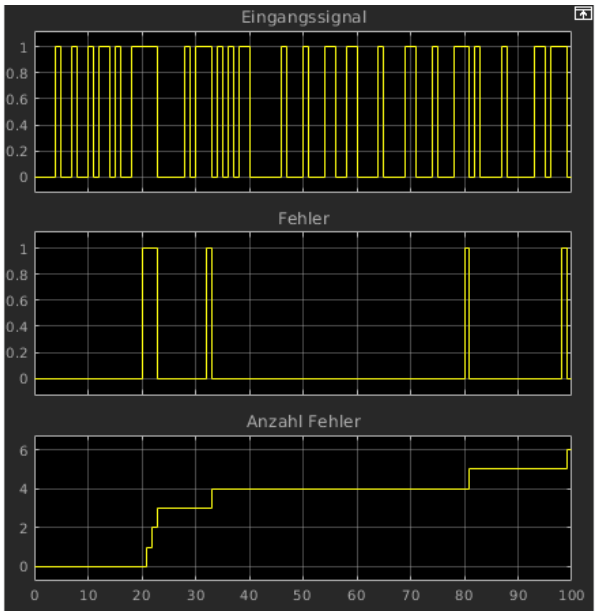
  unter `View/Symbols` in der Chart die Variablen E bzw. A als `Input Data` bzw. `Output Data` festlegen

  Gesamtmodell



  - Eingang erzeugt zufällig Einsen mit Wahrscheinlichkeit 0.3

  Ergebnis



Animation der State Chart mit `Simulation/Stateflow Animation/Medium`

- Definition **deterministischer E/A-Automat**:

  Automat $\mathcal{A}$ gegeben durch 6 Größen $\mathcal{A}$ = ($\mathcal{Z}$, $\mathcal{V}$, $\mathcal{W}$, G, H, z_0)

  mit
  - Zustandsmenge $\mathcal{Z}$
  - Eingabealphabet $\mathcal{V}$
  - Ausgabealphabet $\mathcal{W}$
  - Zustandsübergangsfunktion G: $\mathcal{Z} \times \mathcal{V} \rightarrow \mathcal{Z}$
  - Ausgabefunktion H: $\mathcal{Z} \times \mathcal{V} \rightarrow \mathcal{W}$
  - Anfangszustand $z_0 \in \mathcal{Z}$

- Warum mathematischer Formalismus:

  erzwingt vollständige Klarheit des Verhaltens

  ermöglicht mathematische Analysen, z.B.
  - Sind alle Zustände erreichbar?
  - Erfüllt der Automat vorgegebene Anforderungen?
  - Gibt es einen einfacheren Automaten mit gleichem Verhalten?

  ausgiebige Literatur, Einstieg etwa in [L4] und [L3]

- Zustandsübergangsfunktion G als partielle Funktion:

  G oft nicht für alle Paare aus Zustand/Eingabe definiert

  was tun bei undefiniertem Paar Z/E
  - Automat "bleibt stehen", formal: geht in Endzustand "Fehler"
  - Automat ignoriert Eingabe, formal: bleibt im aktuellen Zustand, gibt ε ("nichts") aus

  Stateflow zeigt anderes Verhalten
  - Automat bleibt im aktuellen Zustand, gibt vorigen Ausgabewert aus

- Andere endliche Automaten:

  Standard-Automat
  - Zustandswechsel bei Eintreffen von Ereignissen (Events)
  - keine Ein- und Ausgaben
  - zusätzlich: Menge von Endzuständen

  wird auch zur Definition formaler Sprachen verwendet

- Anwendung z. B. im Compilerbau

Spezialfall autonomer Automat

- keine Ereignisse
- wechselt "von selbst" zwischen den Zuständen

- Mealy- und Moore-Automat:

  Zustandsraumdarstellung grundsätzlich wie oben

  $$\begin{aligned} z(k+1) &= G(z(k), v(k)) \\ w(k) &= H(z(k), v(k)) \end{aligned}$$

  - mit Startwert z(0) = z_0

  mit dieser Darstellung auch **Mealy-Automat** genannt

  Spezialfall: H hängt nicht von der Eingabe ab (**Moore-Automat**)

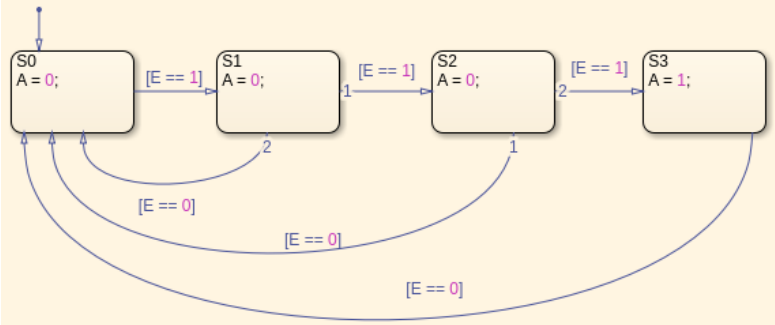
  w(k) = H(z(k))

  andere Sicht

  - Moore = Output im Zustand
  - Mealy = Output an Transition

  Beispiel `fehlerdetektor1B`

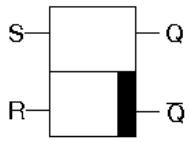
  - Ausgabewert im Zustand einstellen



  - unter `Chart/Properties/State Machine` Wert `Type = Moore` einstellen
  - Ausgabe wie vorher, aber um einen Takt verzögert

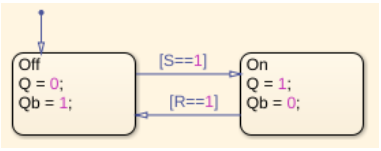
- RS-Flipflop:

  grundlegender Speicherbaustein

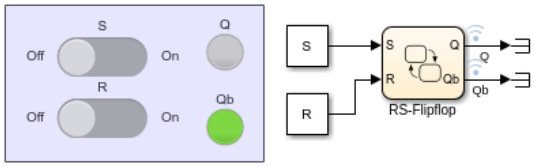
  

  Funktion
  - S = 1 → 1 wird gespeichert → Q = 1, Qb = 0
  - R = 1 → 0 wird gespeichert → Q = 0, Qb = 1

  als Zustandsautomat

  ausprobieren mit interaktiven Ein-/Ausgabe-Elementen (`Dashboard`-Bibliothek)

  R = S = 1 → schaltet hinundher mit jedem Solvertakt

  Tipp zur Benutzung von Dashboard-Eingabe-Elementen
  - Verbindungsvariable in Modell `InitFcn` definieren (`S = 0`)
  - Variablenname `S` im `Constant`-Block eingeben
  - im Dashboard-Block `Constant`-Block anklicken → Variable `S` auswählen

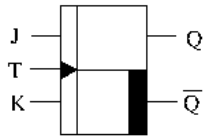
- Getriggertes JK-Flipflop:

  Problem: Laufzeiten von Signalen beim Umschalten

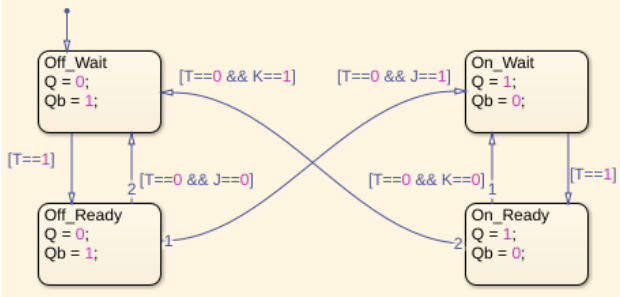
  Lösung: Taktgeber zur Synchronisierung

  Änderungen nur bei Wechsel am Takteingang
  - bei steigender Taktflanke = bei Wechsel von 0 auf 1 (positiv getriggert)
  - bei fallender Taktflanke = bei Wechsel von 1 auf 0 (negativ getriggert)
  - hier immer negativ getriggert

  3 Eingänge
  - Takteingang T, J (≙ S), K (≙ R)
  - J = K = 1 → Wechsel des Ausgangs mit jedem Takt

  als Zustandsautomat



ausprobieren mit interaktiven Ein-/Ausgabe-Elementen

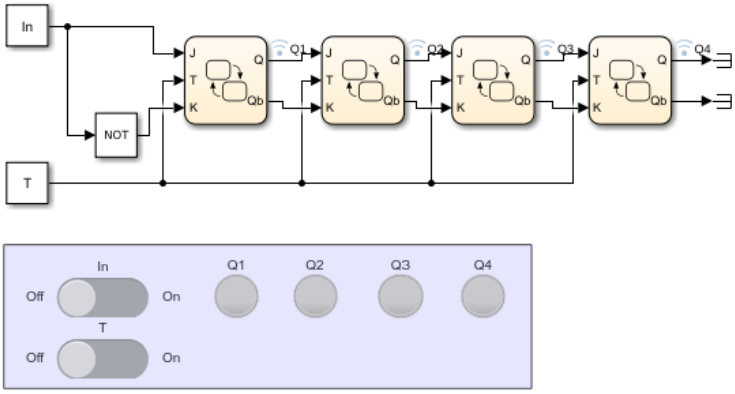
- Anwendung Schieberegister:

  Kette von Flipflops

  übergibt bei Taktimpuls Signal an das nächste Flipflop

  Wandler zwischen seriellen und parallelen Signalen

  Modell `schieberegisterA`



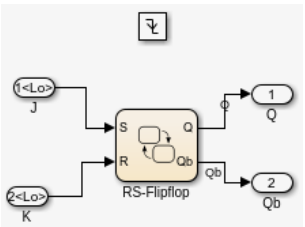
  Problem
  - Signal läuft bei einem Taktwechsel durch
  - Ursache: "Gleichzeitigkeit der Signale" bzw. "direct feedthrough"

- Getriggertes System in Simulink:

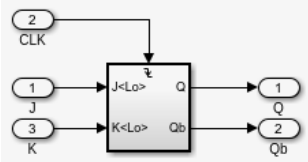
  enthält den `Trigger`-Block aus der `Ports & Subsystems`-Bibliothek
  - wird nur bei Auslösen des Triggers ausgeführt, Ausgänge sonst konstant
  - Trigger kann auf steigende oder fallende Flanke (oder beides) reagieren
  - Triggereingang ist oben (oder unten), nicht links (oder rechts)

  `Inport` in getriggertem Subsystem
  - hat normalerweise aktuellen (neuen) Wert beim Triggern
  - muss hier alten Wert haben → `Latch input` aktivieren

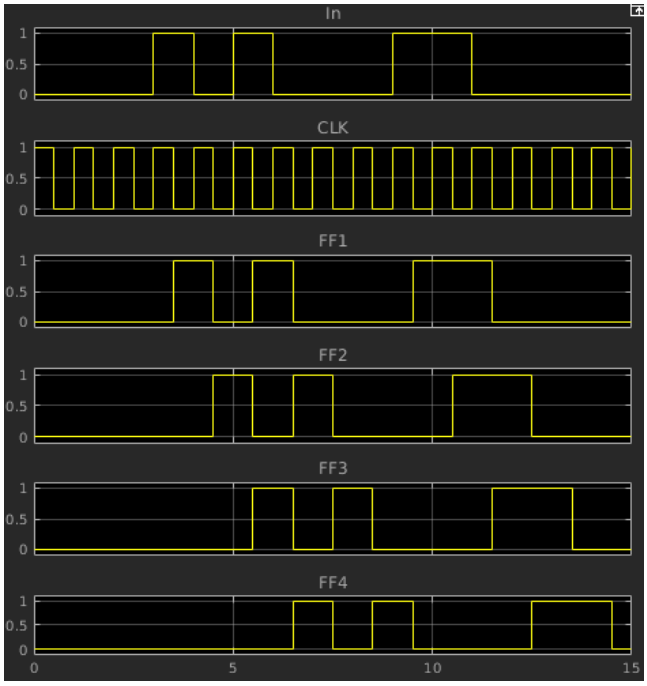
  Triggerung extra → intern genügt RS-Flipflop



  Trick, um Takteingang an die Seite zu bekommen
  - Flipflop in weiteres Subsystem verpacken



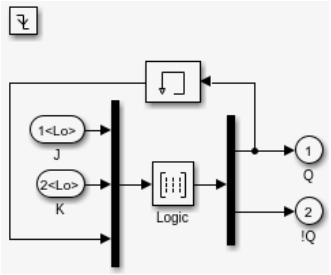
Modell `schieberegisterB` funktioniert



- Implementierung ohne Stateflow:

  RS-Flipflop schneller ohne Stateflow

  ersetze inneres Modell durch



  Logik-Block ersetzt komplexe logische Schaltung

  - Wahrheitstabelle incl. Rückkopplung

J	K	Q_n	Q_{n+1}	$!Q_{n+1}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

  - Parameter `Truth table = [0 1; 1 0; 0 1; 0 1; 1 0; 1 0; 1 0; 0 1]`

Steuerung eines Lastenaufzugs

- Betriebsweise:

  Aufzug transportiert Last von unten nach oben oder umgekehrt

  Anwender drückt Anforderungsknopf (oben oder unten) → Aufzug fährt in entsprechende Etage, Tür wird geöffnet

  Benutzer belädt Aufzug und drückt Startknopf → Kabine fährt zum Ziel

  dort angekommen wird sie entladen, Benutzer drückt Freigabeknopf

- Grundsätzliche Funktionsweise der Steuerung:

  Sensoren erkennen Standort der Kabine (oben/unten)

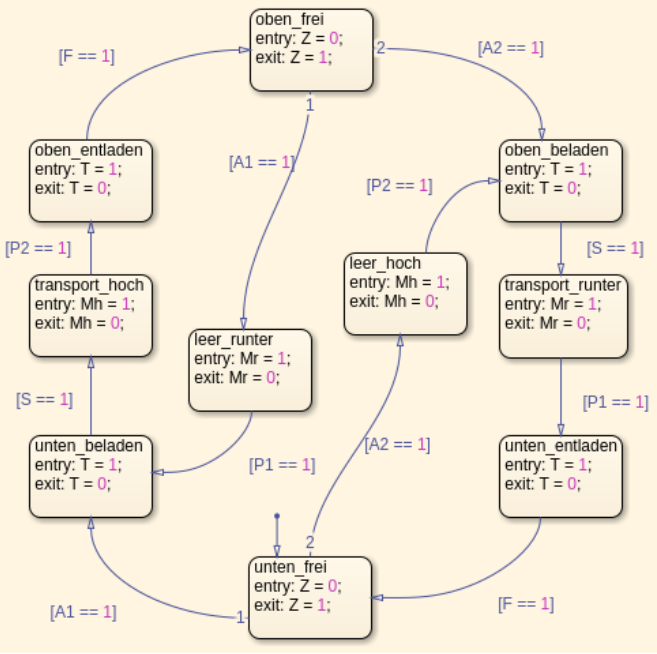
  Anzeige zeigt Status (frei/besetzt)

  Steuerung hat 6 Eingänge
  - 2 Anforderungstasten A1, A2 (unten/oben)
  - Starttaste S und Freigabetaste F
  - zwei Sensoren P1, P2 zur Positionsbestimmung

  Steuerung erzeugt 4 Ausgangssignale
  - Mh, Mr (Motor hoch/runter)
  - T (Öffnen/Schließen der Tür)
  - Z (Zustand des Fahrstuhls)

- Implementierung als Zustandsautomat:

  Verhalten beschrieben durch Zustandsdiagramm



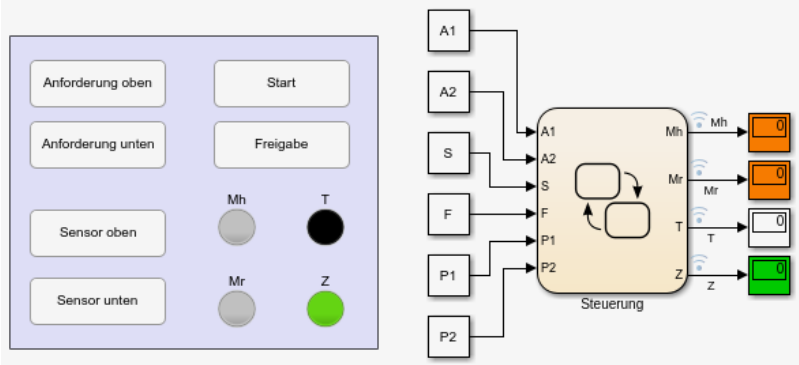
  Ausgabewerte durch Zustand definiert (Moore-Automat)

  zur Vereinfachung
  - bei Betreten eines Zustands (`entry`) gesetzt
  - bei Verlassen eines Zustands (`exit`) zurückgesetzt

  Vorteil: Ausgabewerte müssen nicht in jedem Zustand definiert werden

  `State Machine Type` = `Classic` (nicht `Moore`!)

  ausprobieren mit `fahrstuhlFS1`



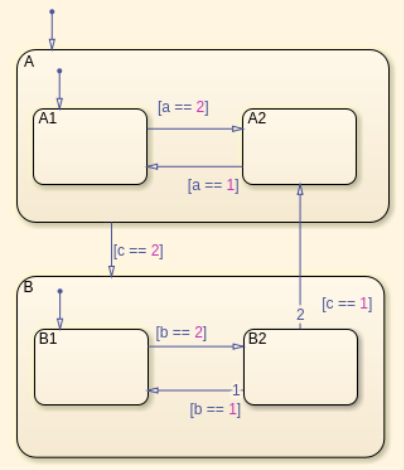
- Aufgaben:
  - Aufgabe 4
  - Aufgabe 5
  - Aufgabe 6

- Statechart [3]:

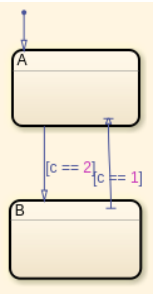
  starke Erweiterung des endlichen Automaten

  viele Implementierungen, u.a. in Matlab (Stateflow) und Modelica (StateGraph)
  - jeweils leichte Abweichungen und Erweiterungen
  - hier Stateflow-Notation
  - geht wesentlich über Statecharts hinaus, aber sehr komplex und oft nicht klar definiert (vgl. [4] und [5])

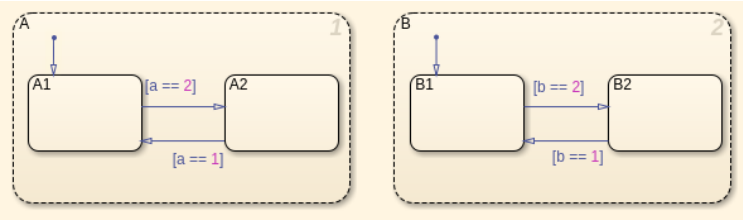
  Hierarchien von Zuständen
  - Zustand kann mehrere Unterzustände enthalten



  - System im Zustand A ↔ entweder A1 oder A2 aktiv (`Decomposition/Exclusive`)
  - B wird aktiv → B1 wird aktiv (Default-Zustand)
  - direkte Auswahl des Unterzustands von Start und Ziel möglich
  - Modell ohne interne Zustandsdetails (`Group & Subchart/Subchart`, `Format/Content Preview/off`)

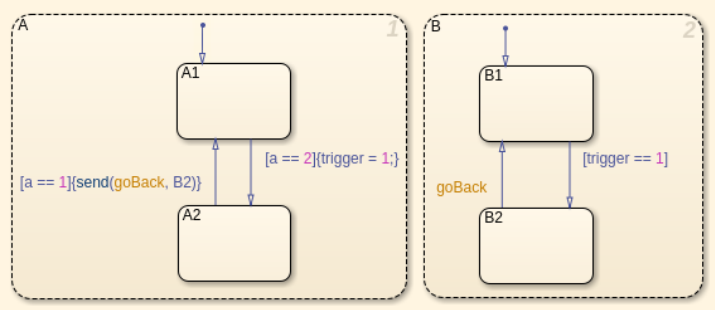


  Parallele Zustände
  - Gesamtzustand = Produkt von Teilzuständen



  - System befindet sich in A und B gleichzeitig (`Decomposition/Parallel`)
  - Zustand beim Start konkret A1/B1

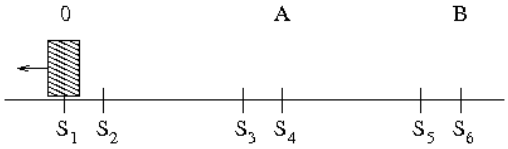
  Aktionen und Aktivitäten
  - Übergang kann Aktivitäten auslösen, auch in anderen Teilzuständen



- Im Zustand A1/B1 werde a = 2 ⇒ (A1 → A2) ⇒ `trigger` wird 1 ⇒ (B1 → B2)
- Nebeneffekte möglich, da `trigger` global
- besser mit lokalem Event, etwa in A2/B2: `send(goBack, B2)` ⇒ (B2 → B1)
- Event erzeugen mit `Chart/Add Other Elements/Local Event...`
- Aktivitäten können auch ausgelöst werden bei Betreten (`entry`) oder Verlassen (`exit`) eines Zustands

- Beispiel Robotersteuerung:

  Roboter soll auf Startsignal hin Objekt von A nach B bringen



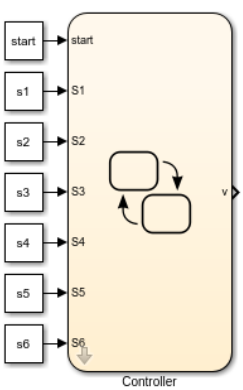
  Motorverhalten beschrieben durch
  - drei Geschwindigkeiten MV: 0, langsam, schnell
  - zwei Richtungen MR: nach links, nach rechts

  Sensoren S_1 .. S_6 zeigen Erreichen einer Position an

  Ablauf
  - Startposition ist 0, MV = 0, MR = L
  - Startsignal → Roboter fährt nach A, zunächst schnell, dann ab S_3 langsam
  - bleibt bei S_4 stehen, greift das Objekt (als Zeitverzögerung modelliert)
  - fährt nach B, zunächst schnell, ab S_5 langsam
  - bleibt bei S_6 stehen, setzt Objekt ab (Zeitverzögerung)
  - fährt zurück nach 0, erst schnell, langsam ab S_2
  - bleibt bei S_1 stehen

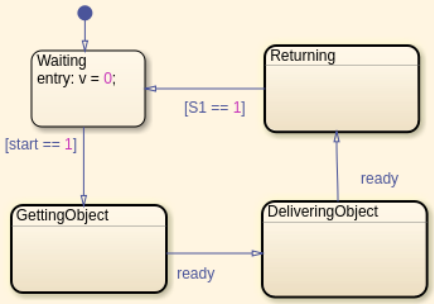
- Implementierung in Stateflow (`robcontrol1A`):

  Chart-Block im Modell öffnen
  - Eingänge `start`, `S1` ... `S6` definieren (`Chart/Add Inputs & Outputs/Data Input From Simulink`)
  - analog Ausgang `v`
  - Elemente für interaktive Steuerung hinzufügen

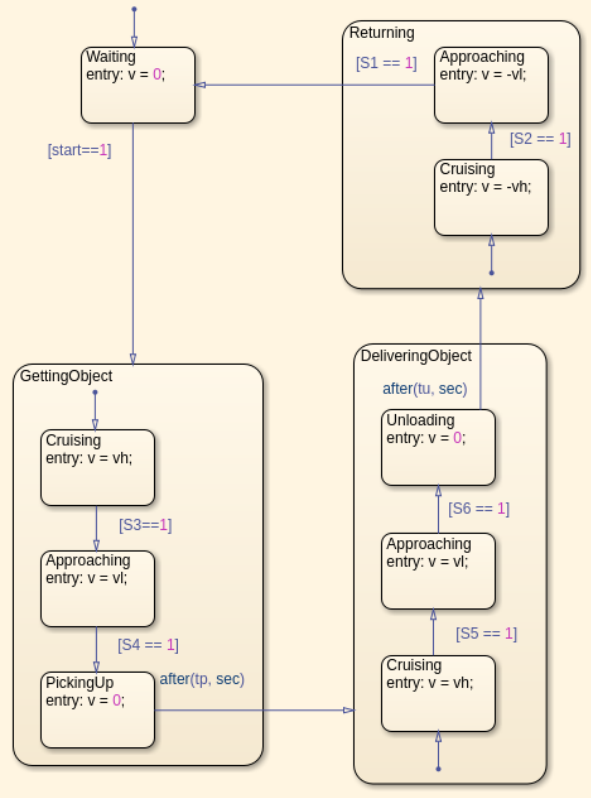


  Aufbau des Diagramms

- zur Übersichtlichkeit Hierarchiestufen nutzen
- Grobstruktur



- Feinstruktur



Definieren der Geschwindigkeit jeweils bei Betreten eines Zustands (`entry`)

Transitionen

- meistens bei Aktivieren des Startknopfs oder eines Sensors
- Aufnehmen (`PickingUp`) und Absetzen (`Unloading`) nach fester Zeit im Zustand
- mit Stateflow-Funktion `after(n, sec)`

Masken-Parameter (`vl`, `vh`, `tp`, `tu`) müssen in Chart aktiviert werden

- in Chart Variablen anzeigen lassen (`View/Symbols`)
- für alle Masken-Parameter `TYPE = Parameter Data` setzen

- Testlauf manuell:

  Hauptebene und Chart in zwei Fenstern nebeneinander

  Simulation starten → aktiver Zustand `Waiting` wird markiert

  Taster `Start` anklicken → `Getting Object` und `Cruising` werden markiert

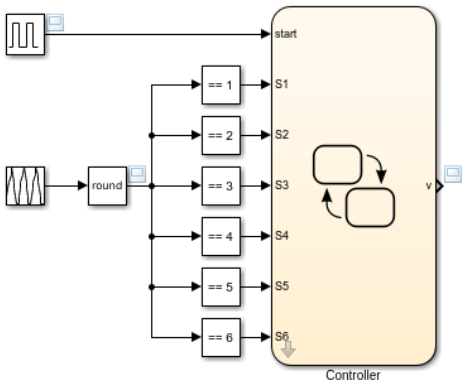
  durch Anklicken der Sensor-Taster Ablauf nachvollziehen

- Testlauf automatisiert (`robcontrol1B`):

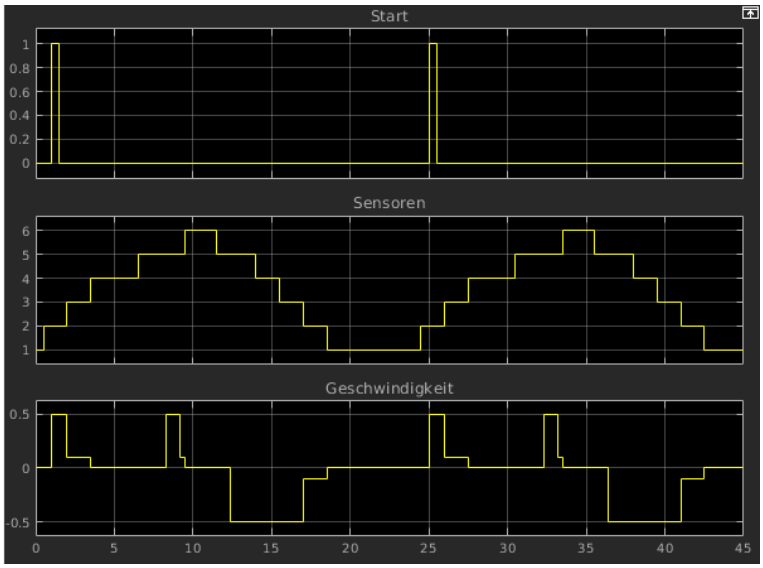
  Signalgeber für Startimpulse

  Signalgeber, der in geeigneten Zeitabständen von 1 bis 6 hoch und runter zählt

  - Wert = i → Sensor i ist aktiv



Ergebnis zeigt den gewünschten Verlauf der Geschwindigkeit



- Modellierung des Gesamtsystems (`robcontrol2`):

  Hybridmodell mit diskreter Steuerung und kontinuierlicher Außenwelt (Roboter/Sensoren)

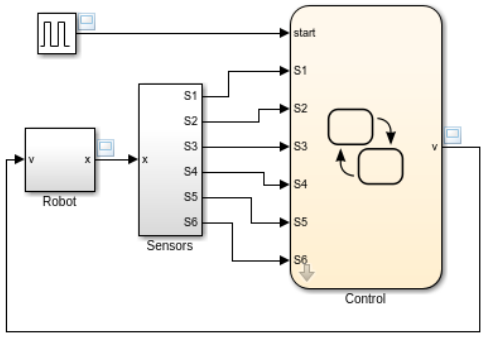
  Roboter

  - Eingang: `v`
  - Ausgang: `x`
  - Parameter: Anfangsort `x0`
  - Implementierung: einfacher Integrator

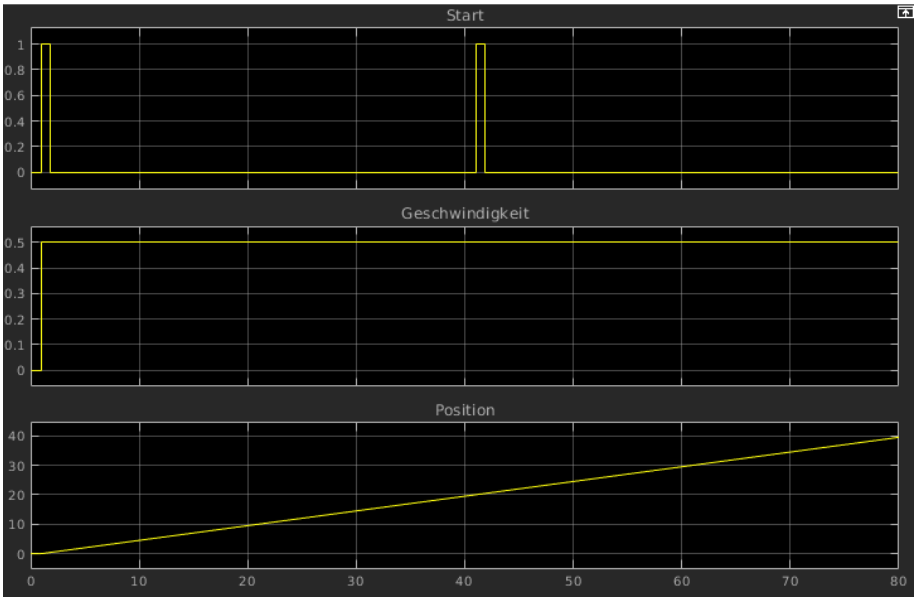
  Sensoren

  - Eingang: `x`
  - Ausgänge `S1` ... `S6` (Typ `boolean`)
  - Parameter: Orte `x1` ... `x6` der Sensoren, Sensorbreite `b`
  - Implementierung: 6 Intervalltester

  Gesamtmodell



  Ergebnis seltsam



- Fehler in Simulink (Version 2019a):

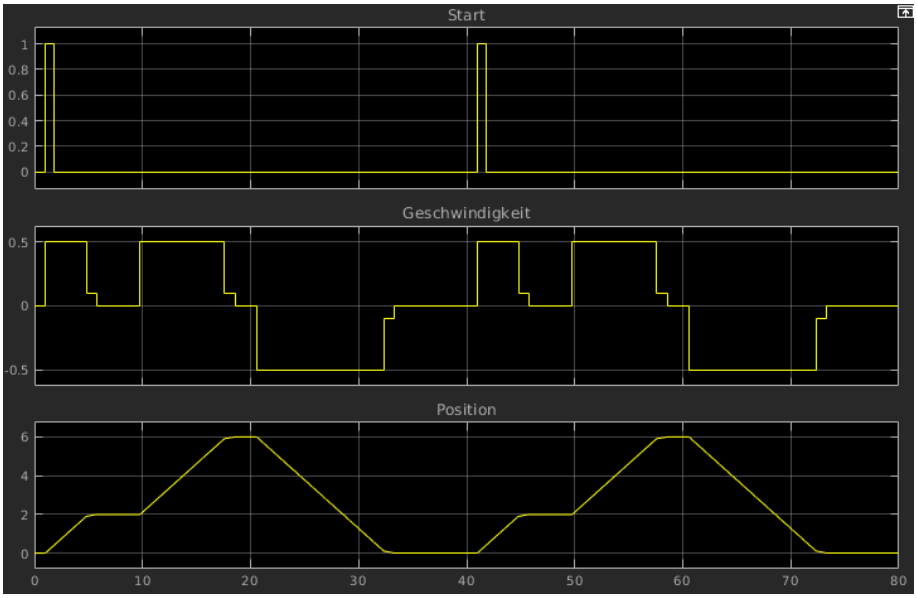
  Solver überspringt Sensorreaktion → Roboter fährt einfach weiter

  einfacher Fix: Solverparameter `Max step size` von `auto` auf `0.01`

  eigentliche Fehlerursache
  - im Block `Interval Test` ist `Zero Detection` ausgeschaltet!

  besserer Fix
  - eigener Block `Interval tester` mit `Zero Detection`
  - `Max step size` bleibt auf `auto`

  Modell funktioniert



- Hinzufügen eines Greifers:

  Funktion
  - kann greifen oder loslassen
  - fährt zum Aufnehmen herunter, zum Transport hoch

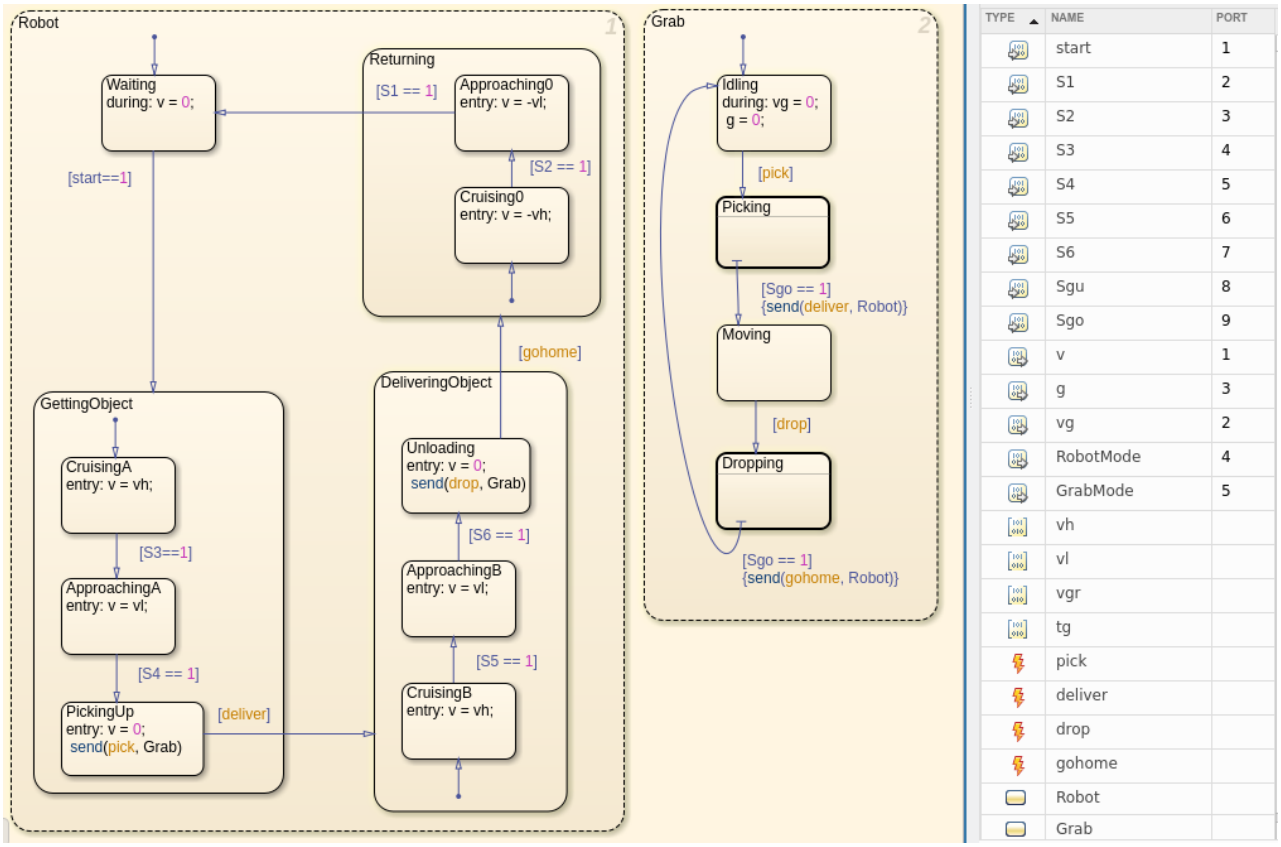
  Anschlüsse
  - Eingang Motor (+vg nach oben, -vg nach unten)
  - Eingang Greifer (g = 0, 1)
  - Ausgänge zweier Sensoren (Sgu = unten, Sgo = oben)

- Modellierung des Greifers (`robcontrol3`):

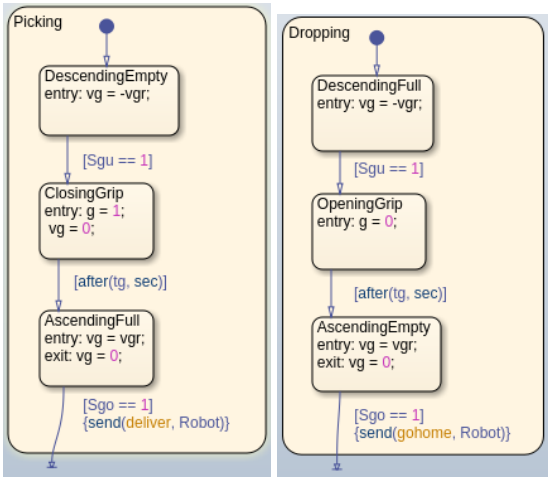
könnte über Subzustände bei `PickingUp` und `Unloading` eingefügt werden

übersichtlicher als paralleles Subsystem `Grab`

Grundaufbau



- Greifer wechselt zwischen vier Zuständen
- Aufnehmen (`Picking`) und Ablegen (`Dropping`) jeweils Subzustände



Kopplung zwischen `Robot` und `Grab`

- `PickingUp` wird betreten

  → lokales Event `pick` wird an `Grab` geschickt

  → Transition von `Idling` zu `Picking`

- `Picking` wird verlassen

  → lokales Event `deliver` wird an `Robot` geschickt

  → Transition von `PickingUp` zu `DeliveringObject/CruisingB`

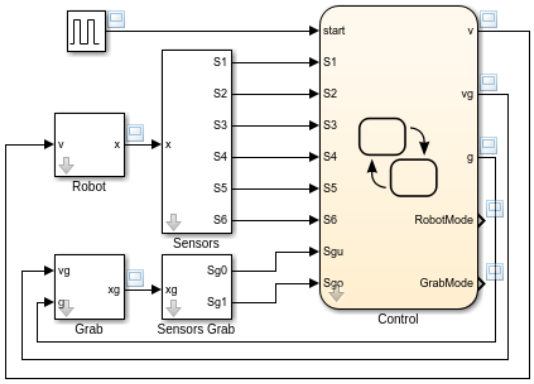
- analog bei `Unloading` und `Dropping`

Symboltabelle zeigt übersichtlich alle verwendeten Symbole an (Ein-/Ausgänge, Parameter, Events, Zustandsnamen)

- Implementierungsdetails:

  Transition von `Picking.AscendingFull` nach `Moving` (u.ä.)
  - in `Picking` über die Grenze hinausziehen → Pfeil endet mit Querstrich (**Slit**)
  - wird in `Grab` als rotes Dreieck angezeigt
  - von dort aus Transition zu `Moving` ziehen

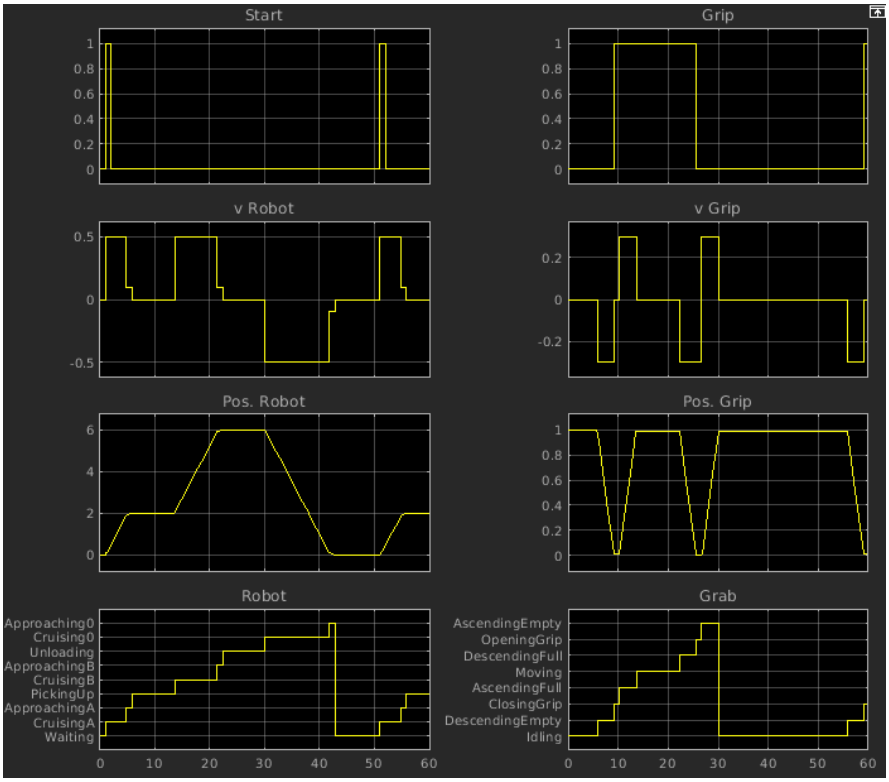
  Gesamtmodell



  - primitive Modelle für `Grab` (Integrator mit Saturation) und dessen Sensoren

  Ausgabe der angenommenen Zustände über die Zeit als Hilfe beim Debuggen
  - Kontextmenü in `Robot/Properties...`
  - `Create output for monitoring/Leaf state activity`
  - Voraussetzung: Zustände in `Robot` haben verschiedene Namen
  - analog für `Grab`

  Ausgabe zeigt komplettes Systemverhalten



- Aufgaben:

  Aufgabe 7

  Aufgabe 8

Petri-Netze

- Definition von Petri-Netzen
- Simulation von Petri-Netzen
- Anwendung: Simulation des Straßenverkehrs

Definition von Petri-Netzen

- Graphische Modellierung mit Petri-Netzen:

  eingeführt von Carl Petri 1962 (mathematisch)

  erweitert Zustandsautomaten um Prozesse = zeitliche Abläufe

  ermöglicht u.a. Beschreibung paralleler Prozesse

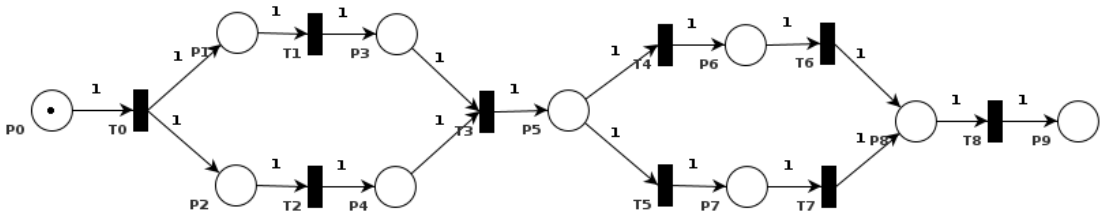
  einfaches Grundmodell und viele Erweiterungen

  umfangreiche mathematische Analysen, z. B.
  - Erreichbarkeit von Zuständen
  - Beschränktheit von Plätzen
  - Verklemmungsfreiheit

  Anwendungen u.a.
  - parallele Prozesse in der Informatik
  - Fertigungstechnik
  - Logistik
  - Geschäftsprozesse
  - theoretische Biologie

- Grundaufbau eines Petri-Netzes:

  Graph mit
  - zwei verschiedenen Arten von Knoten (**Stellen** und **Transitionen**)
  - gerichteten Kanten von Stellen zu Transitionen und umgekehrt



  Stellen (Places)
  - rund dargestellt
  - können eine oder mehrere Marken (**Token**) enthalten
  - beschreiben Teil-Zustände bzw. Vor-/Nach-Bedingungen von Prozessen

  Transitionen
  - schwarze Rechtecke (quadratisch bis sehr schlank)
  - beschreiben Prozesse
  - **Prästellen** einer Transition t = Stellen mit Kanten zu t
  - **Poststellen** einer Transition t = Stellen mit Kanten von t

- Verhalten eines einfachen Petri-Netzes:

  Grundtyp: Stellen enthalten keine oder eine Marke

  Transition t heißt **aktiviert** (enabled) ↔
  - alle Prästellen von t enthalten eine Marke
  - UND alle Poststellen von t sind leer

  aktivierte Transition t kann schalten →
  - Marken aller Prästellen von t werden vernichtet
  - alle Poststellen von t erhalten eine Marke
  - anschaulich: Marken wandern (incl. Vernichtung/Erzeugung) durch Transitionen

- Wann schalten aktive Transitionen:

  sobald (externe) Bedingung erfüllt ist (**cold transition**)

  automatisch (**hot transition**)

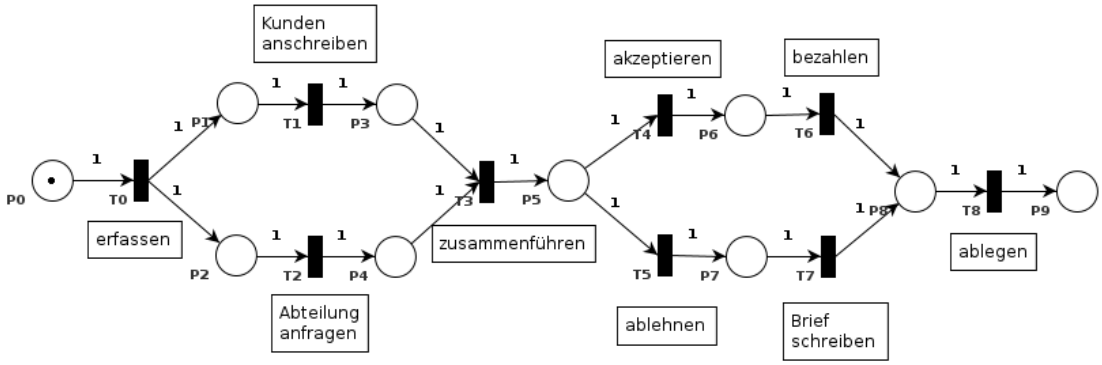
  bei mehreren aktiven Transitionen
  - eine wird (zufällig oder gezielt) ausgewählt
  - mehrere können gleichzeitig schalten, falls möglich

  im Beispiel: T4/T5 können *nicht* gleichzeitig schalten

  Zeit zum Schalten
  - instantan (beim Grundtyp)
  - vorgegebene Schaltzeit einer Transition
  - zufällige Schaltzeit einer Transition

- Beispiel "Bearbeitung einer Beschwerde":

  Petri-Netz von oben, Transitionen entsprechen konkreten Prozessen



  Beschwerde kommt an (Token in P0)

  Beschwerde wird erfasst (T0 schaltet)

  Kunde wird angeschrieben und Abteilung angefragt (T1 und T2 schalten)

  beide Antworten werden zusammengeführt (T3 schaltet)

  Beschwerde wird akzeptiert (T4 schaltet)
  - von den zwei aktivierten Transaktionen T4, T5 wurde T4 ausgewählt

  Kunde wird bezahlt (T6 schaltet)

  Vorgang wird abgelegt (T8 schaltet)

- Notation von Simulationsläufen:

  Zustand Z des Petrinetzes = Belegung der Stellen
  - bei einfachen Netzen reicht Liste der Stellen mit Marken

  Darstellung einer Transition

  $$Z_1 \xrightarrow{t} Z_2$$

  damit sequenzieller Ablaufplan

  $$Z_1 \xrightarrow{t_1} Z_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} Z_n$$

  bei unabhängigen Prozessen (im Beispiel T1/T2) Reihenfolge wählen

  Beispiellauf damit etwa

  $$0 \xrightarrow{T_0} 12 \xrightarrow{T_1} 23 \xrightarrow{T_2} 34 \xrightarrow{T_3} 5 \xrightarrow{T_4} 6 \xrightarrow{T_6} 8 \xrightarrow{T_8} 9$$

Unabhängigkeit von T1/T2 hier nicht darstellbar

genauere Notation: verteilter Ablaufplan (distributed run) [L6]

- Mathematisches Modell eines einfachen Petri-Netzes:

  gegeben durch 4 Größen (P, T, F, M_0)

  - P = Menge der Stellen
  - T = Menge der Transitionen (mit $P \cap T = \varnothing$)
  - $F \subseteq (P \times T) \cup (T \times P)$ = Menge der Kanten
  - M_0: $P \rightarrow \{0,1\}$ = Anfangsbelegung der Stellen (Markierung)

  für Transition t ist

  - $\bullet t = \{p \mid (p, t) \in F\}$ (Prästellen von t)
  - $t\bullet = \{p \mid (t, p) \in F\}$ (Poststellen von t)

  Markierung M: $P \rightarrow \{0,1\}$ = aktuelle Belegung der Stellen

  Transition t ist aktiviert, wenn

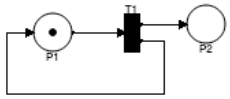
  - $M(p) = 1$ für alle $p \in \bullet t$
  - $M(p) = 0$ für alle $p \in t\bullet$

  Schaltvorgang einer aktivierten Transition t ändert Markierung M zu M' gemäß

$$M'(p) = \begin{cases} 0 & | \quad p \in \bullet t \\ 1 & | \quad p \in t\bullet \\ M(p) & | \quad \text{sonst} \end{cases}$$

  Anmerkung

  - Was ist, wenn p zu •t und zu t• gehört?



  - Dann kann t nicht aktiviert sein!

- Verallgemeinerungen:

  mehr als ein Token pro Platz möglich

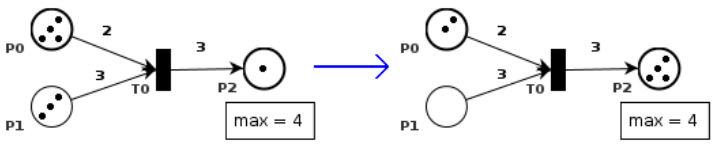
  - maximale Anzahl angebbar

  mehr als ein Token fließt ab bei Transition

  - Zahl bei Kante angebbar

  Transition t aktiviert ↔

  - alle Prästellen von t enthalten genügend Marken (gemäß Kanten zu t)
  - UND alle Poststellen von t haben noch Platz für Marken (gemäß Kanten von t)



  Transitionen brauchen Zeit zum Schalten

  - pro Transition festgelegt
  - alternativ stochastisch

  Marken haben Zusatzeigenschaften ("Farben")

  - hier nicht behandelt

Simulation von Petri-Netzen

- Programme zum Simulieren von Petri-Netzen:

  etliche Werkzeuge vorhanden, vgl. Petri Nets Tool Database

  PIPE5 für schöne Bilder und einfache Analysen

  Modelica-Bibliothek PNlib für komplexere Simulationen und hierarchisch aufgebaute Modelle

- Aufbau des Beispiels "Beschwerde" mit Dymola/PNlib:

  Stellen mit `PD` (Discrete Place)
  - Parameter für Zahl der Eingänge, Ausgänge und Start Tokens angeben
  - Parameter `minTokens` auf 0 (default)
  - Parameter `maxTokens` auf 1

  Transitionen mit `TD` (Discrete Transition)
  - Parameter `nIn` und `nOut` setzen
  - Parameter `delay` hier auf 1 (default), nicht 0 (sonst sofort fertig)
  - Parameter `firingCon` für "cold transitions" verwenden (z. B. `t > 8`)

  Verbindungen
  - direkt bei 1 Eingang → 1 Ausgang
  - sonst Indizes angeben

  Block `Settings` hinzufügen für
  - globale Einstellungen zur Darstellung
  - globale Seed für Zufallszahlen

- Verschönerung durch eigene Blöcke in DiscSimLib:

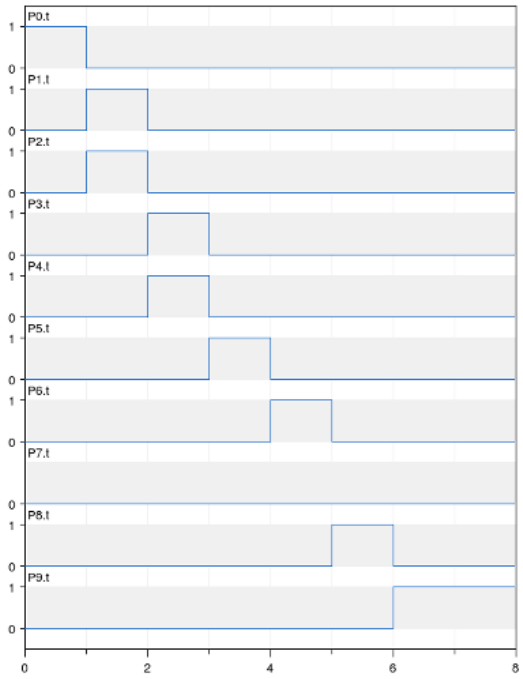
  Leitungsführung in PNlib sehr unübersichtlich
  - Vektoranschlüsse → alle Eingänge/Ausgängen landen/starten an einem Punkt
  - Animation des Diagramms wichtig zur Ergebnisdarstellung

  einfache Hilfsblöcke in `DiscSimLib/PetriNets` für Stellen und Transitionen
  - mehrere Ein-/Ausgänge statt Vektoren
  - Varianten für verschiedene Geometrien
  - nur wenige, hier wichtige Parameter

- Simulationsergebnisse:

  Simulation laufen lassen und Diagram-Fenster anzeigen
  - Animation zeigt wechselnde Belegungszustände der Stellen (s.o.)

  alternativ Belegungen (`P0.t` etc.) als Kurven anzeigen lassen



  - T1, T2 schalten gleichzeitig (vgl. P3, P4)

- Erweiterungen:

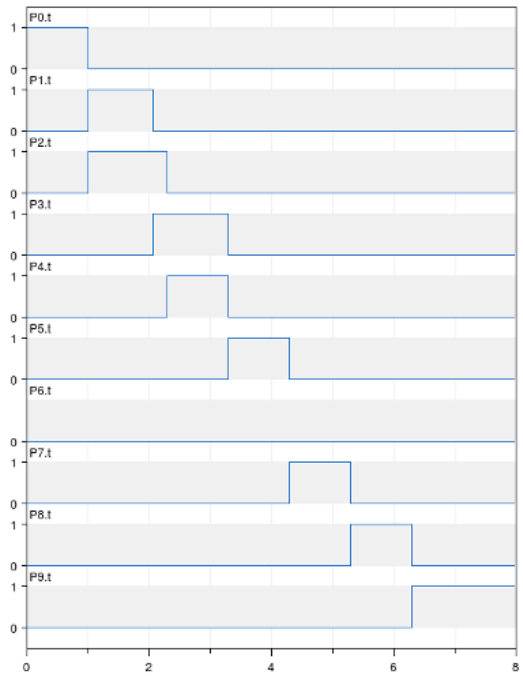
  zufällige Delays bei T1, T2

  - Transitionsblock vom Typ `TDS` ("stochastic") statt `TD`
  - Auswahl zwischen verschiedenen Verteilungen incl. Parametern und Seed

  Entscheidung zwischen T4 und T5 in P5

  - default: nach Priorität (hier: 1. Ausgang zuerst)
  - alternativ: über Wahrscheinlichkeit

  Ergebnis



- Hierarchie von Petri-Netzen:

  komplexe Netze übersichtlicher durch Subsysteme

  ersetze Teilsystem, das nur Stellen als Ein- und Ausgänge enthält, durch Stelle

  ersetze Teilsystem, das nur Transitionen als Ein- und Ausgänge enthält, durch Transition

konkret im Beispiel

- linke Verzweigung → Stelle `GetInfos`

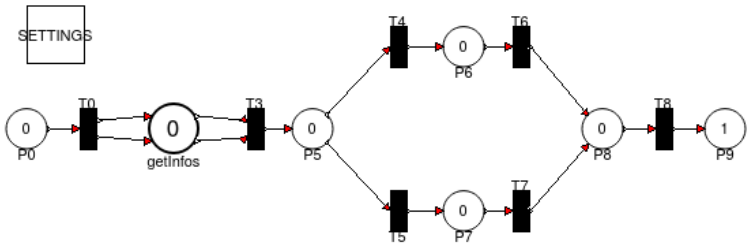
in Modelica-Umgebung kein Problem

- Subsystem einführen
- im Subsystem mehrere Ein-/Ausgänge einführen und neu verbinden
- dynamisches Icon hinzufügen, z.B.

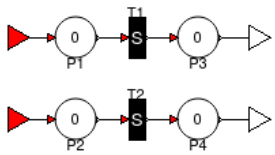
```
textString = DynamicSelect("0",
  if settings.animateMarking then String(tAll) else " ")
```

hierarchisches Modell

- Gesamtmodell



- Subsystem `GetInfos`



Anwendung: Simulation des Straßenverkehrs

- Modellierung von Verkehrflüssen:

  große Breite an interessierenden Systemen
  - eine Ampelkreuzung
  - Haupt-Verkehrwege in einer Stadt
  - Autobahnnetz in Europa

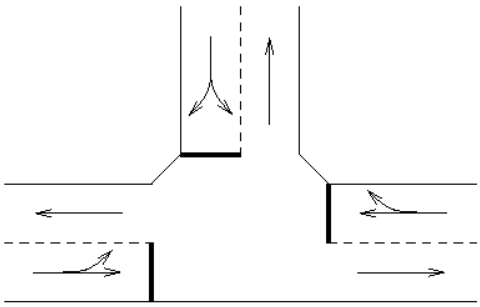
  vielfältige Problemstellungen
  - Ampel oder Kreisel?
  - geschickte Wahl von Ampelschaltungen
  - Verkehrsleitsysteme
  - Planung neuer Autobahn

  sehr verschiedene Modellierungsverfahren
  - kontinuierlich (Flüsse = Fahrzeuge pro Stunde o.ä.)
  - diskret (bis zu Einzelfahrzeugen)
  - hybrid

  Ansatz hier (nach [6],[7])
  - diskrete Modellierung mit Petrinetzen
  - ein Token = 1 Fahrzeug
  - Submodelle als komplexe Transitionen oder komplexe Stellen

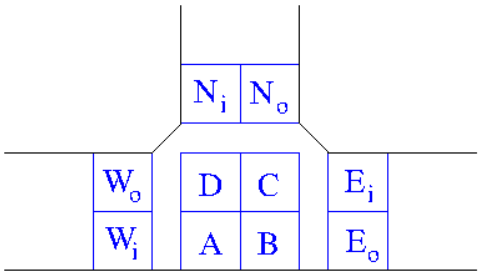
- Beispiel Durchgangsstraße mit Abzweigung:

  Straßenplan



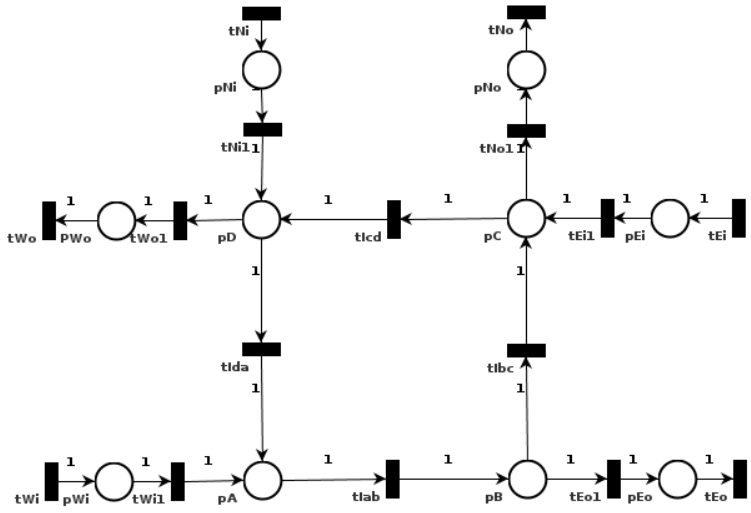
  - aus allen drei Richtungen können Fahrzeuge kommen
  - Fahrzeuge fahren geradeaus oder biegen ab

  Grundmodell



  - Modell enthält Platz für je ein ankommendes und auslaufendes Fahrzeug pro Spur
  - Bereiche A, B, C, D, enthalten jeweils höchstens ein Fahrzeug

  Aufbau des Modells

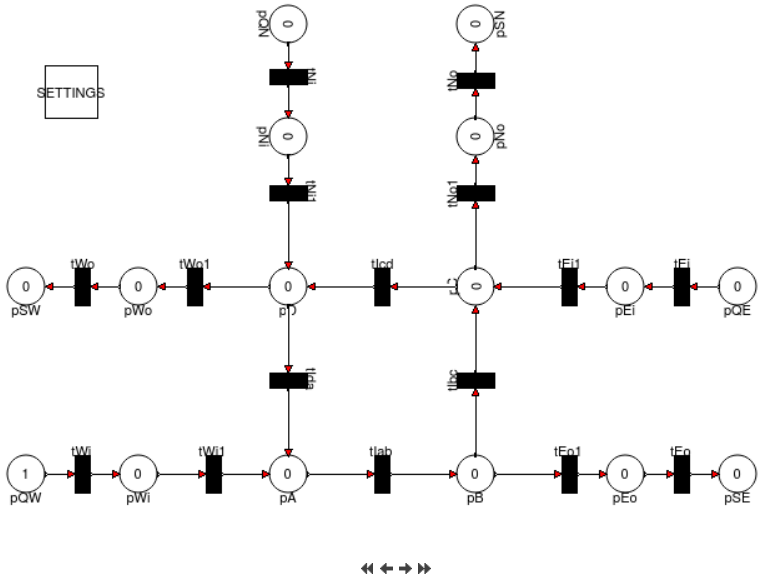


- Eingangstransition und -stelle für jede einfahrende Fahrbahn (tWi/pWi, tEi/pEi, tNi/pNi)
- Ausgangstransition und -stelle für jede ausfahrende Fahrbahn (tWo/pWo, tEo/pEo, tNo/pNo)
- Stellen pA, pB, pC, pD für die Plätze auf der Kreuzung
- Transitionen tWi1, tEi1, tNi1 zur Einfahrt auf die Kreuzung
- Transitionen tWo1, tEo1, tNo1 zur Ausfahrt aus der Kreuzung
- Transitionen tlab, tlbc, tlcd, tlda für Bewegungen auf der Kreuzung

- Tests des Grundverhaltens:

  einfache Stellen als Quellen/Senken für einfahrende/ausfahrende Fahrzeuge

  einzelnes Fahrzeug kommt von W und will links abbiegen



  - klappt, ist aber Zufall (`globalSeed` = 0)!
  - Auswahl der Transitionen ist stochastisch → Fahrzeug könnte auch geradeaus fahren oder sogar auf ABCD im Kreis fahren
  - Setze `Settings.globalSeed` = 2 → Fahrzeug wendet

  ein Fahrzeug kommt von E, eins von N

  - `globalSeed` = 0: beide Wagen biegen rechts ab
  - `globalSeed` = 1: Wagen von E biegt rechts ab, Wagen von N biegt links ab
  - Achtung: Wagen von E biegt *immer* rechts ab, da Geradeausfahrt blockiert ist
  - `globalSeed` = 4: Wagen von N wendet
  - `globalSeed` = 15: Wagen von N fährt einmal im Kreis und biegt dann links ab

  viele weitergehende Fragestellungen, etwa

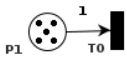
  - Untersuchung größerer Anzahlen (Ströme = Fahrzeuge/Zeit),
  - wie "rechts vor links" implementieren

    - wie bekommt man Wegvorgaben der Fahrzeuge
- Fahrzeugquellen und -senken:

  Submodelle, die Fahrzeuge erzeugen bzw. vernichten

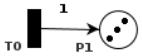
  nötig für vollständiges Modell

  einfache Quelle

    - hat Reservoir von N Fahrzeugen
    - alle t Sekunden kommt ein Fahrzeug heraus
    - alternativ stochastisch (z.B. exponentiell verteilt)

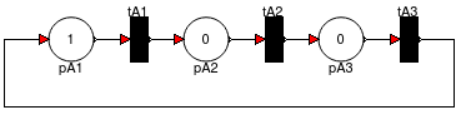
  einfache Senke

    - alle ankommenden Fahrzeugen landen sofort in riesigem Reservoir
- Abzweigung mit Ampelsteuerung:

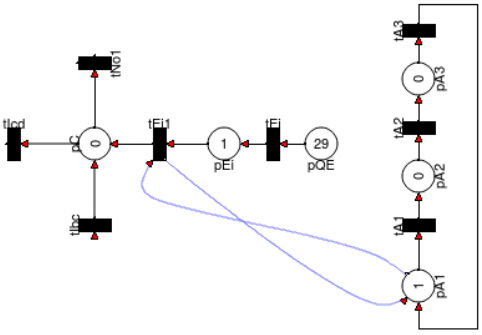
  Ampelschaltung hat drei Phasen
    - Phase 1: W und E grün, N rot (freie Fahrt Hauptstraße)
    - Phase 2: W grün, N und E rot (ermöglicht Linksabbiegen von W)
    - Phase 3: N grün, W und E rot (freie Fahrt Abzweigung)

  Implementierung der Ampel
    - drei Stellen pA1/pA2/pA3 für die drei Phasen
    - zyklisch Transitionen dazwischen

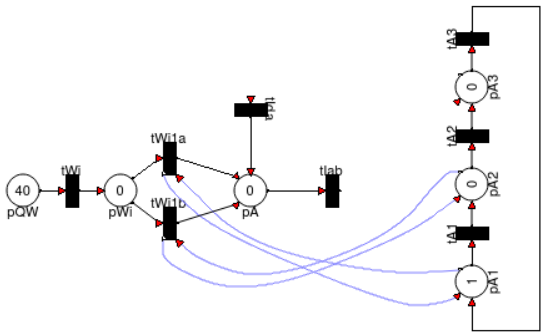
  

    - Transitionen definieren Länge der Phasen
    - → schaltet mit vorgegebenen Zeiten zyklisch zwischen den Transitionen

  Implementierung der Steuerung
    - Transition tEi1 schaltet nur in Phase 1
    - braucht also Token in pA1
    - jedes durchlaufende Fahrzeug muss das Token an pA1 zurückgeben
    - damit das klappt, muss `maxToken` von pA1 > 1 sein

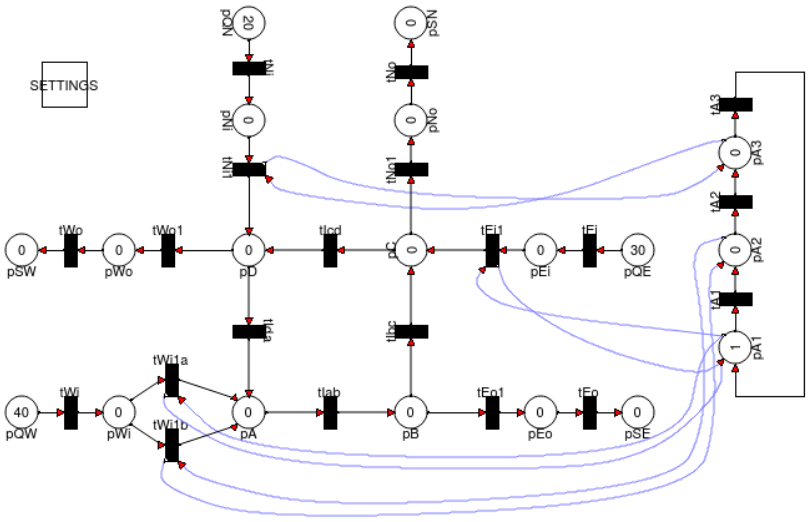
  

  Problem bei Einfahrt von W
    - grün in Phasen 1 und 2
    - zwei Ampel-Eingänge an tWi1 → UND-Verknüpfung
    - ODER-Verknüpfung durch zwei parallele Transitionen

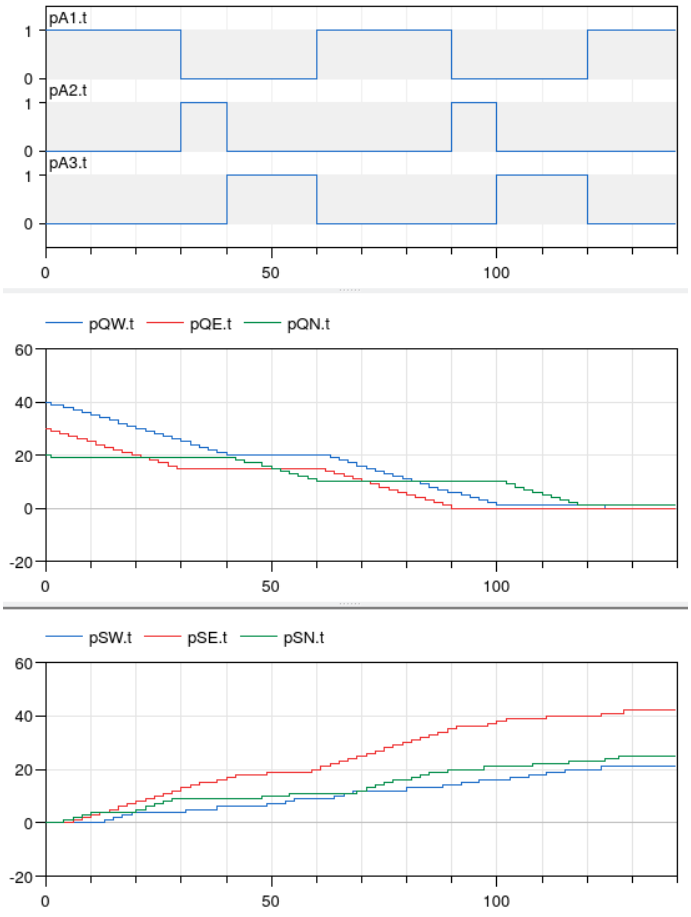


Gesamtmodell `Abzweigung2A`

- unübersichtlich, Farbgebung hilft



Ergebnis



- Gesamtergebnis sieht qualitativ gut aus
- Bewegung der Einzelfahrzeug wie üblich (incl. Kreisfahrten)

- durch zusätzliche Steuerung während der Ampelphasen kontrollierbar (vgl. Aufgabe 10)
- z.B.: Transition tIbc in Phase 3 sperren → Wagen von Norden können nur direkt nach W oder E fahren

- Modellierung einer Straße:

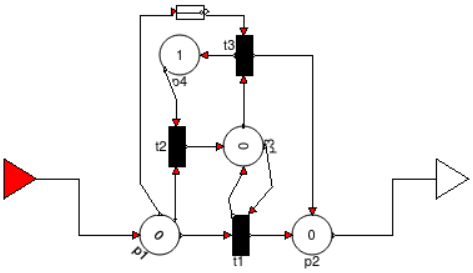
  Submodell für komplexe Stelle

  viele Einzelschritte nötig bei Modellierung der räumlichen Fahrzeugverteilung

  hier vereinfacht
  - ein Platz mit großer Kapazität n_S (komplette Straße)
  - Durchlaufen der Straße braucht Übergangszeit t_S
  - t_S folgt aus Länge der Straße und mittlerer Geschwindigkeit

  einfache Transition mit t_S ungeeignet, erzeugt Abstand t_S zwischen je zwei Wagen!

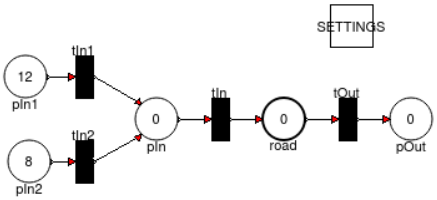
  besseres Modell (vgl. [8])



  - erstes Token in p1 startet Transition der Zeit t_S in t2
  - weitere bis zu n_S Token in p1 gesammelt
  - nach Zeit t_S schaltet t2 → t1 wird frei und schaltet ein Token pro Zeitschritt
  - t3 hat **Inhibitor-Kante** → schaltet nur, wenn p1 leer ist
  - also: p1 leer → t3 schaltet → p4 ist wieder besetzt → Wartezeit beginnt erneut

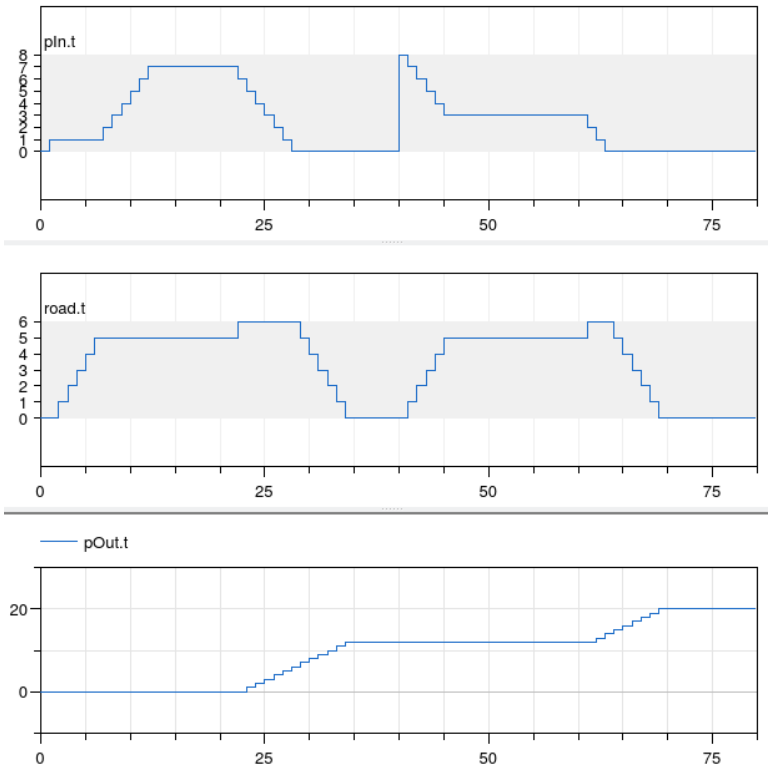
- Testmodell mit zwei Kolonnen:

  Aufbau



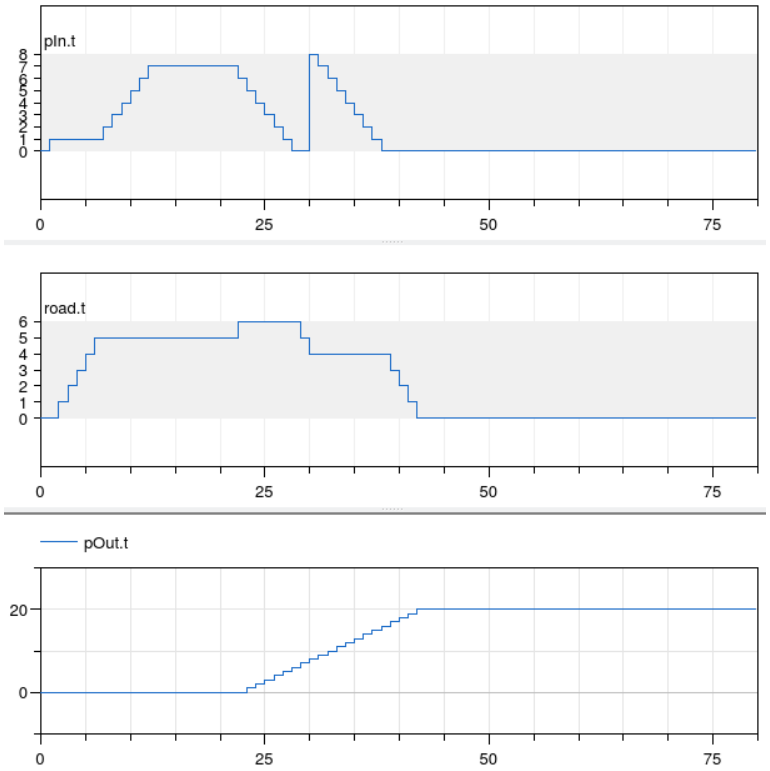
  - Komponente `road` mit $n_S = 6$, $t_S = 20$
  - 1. Kolonne aus pIn1 wandert durch die Straße
  - tIn2 hat `delay` = 40, `arcWeight` = 8 (in und out)
  - → bei t = 40 wandern alle 8 Tokens von pIn2 nach pIn (2. Kolonne)

  Ergebnis



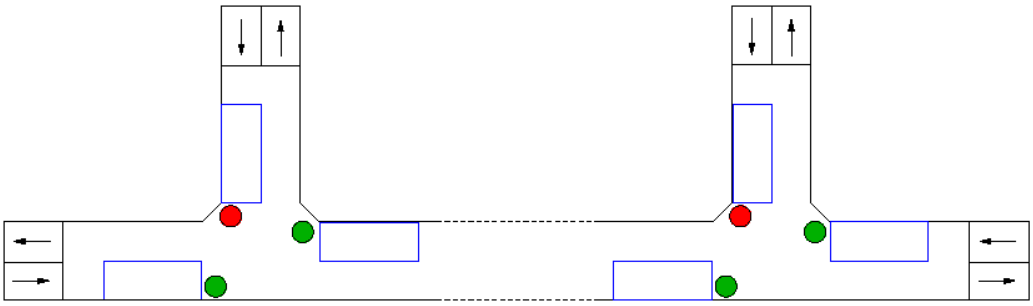
nur realistisch bei getrennten Kolonnen

- Überlapp bei `delay` = 30 in tIn2
- 2. Kolonne sollte frühestens nach 50 s aus der Straße kommen

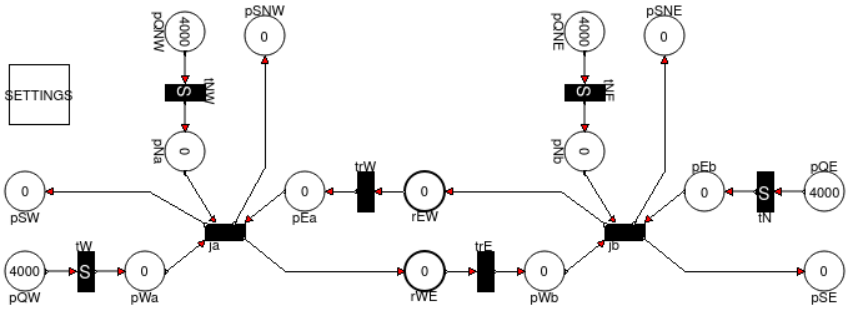


- Untersuchung von Verkehrsflüssen:

  Beispiel Hauptstraße mit zwei Seitenstraßen



Gesamtmodell `Strassennetz`

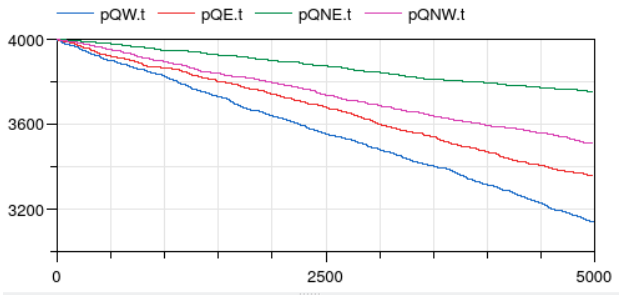


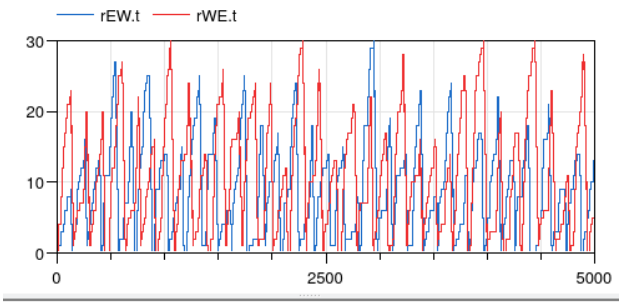
Parameter

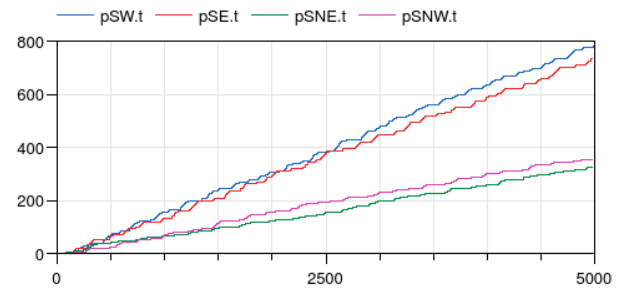
- Eingangsflüsse [Fahrzeuge/min] $Q_W = 10$, $Q_E = 8$, $Q_{NW} = 6$, $Q_{NE} = 3$
- unterschiedliche Ausgangs-Wahrscheinlichkeiten in den Kreuzungen (vgl. Aufgabe 9)
- Ampelzeiten jeweils 150s/30s/60s für die drei Phasen
- Straßen: jeweils Kapazität von 30 und Delay von 2 min

Ergebnisse

- Gesamtströme



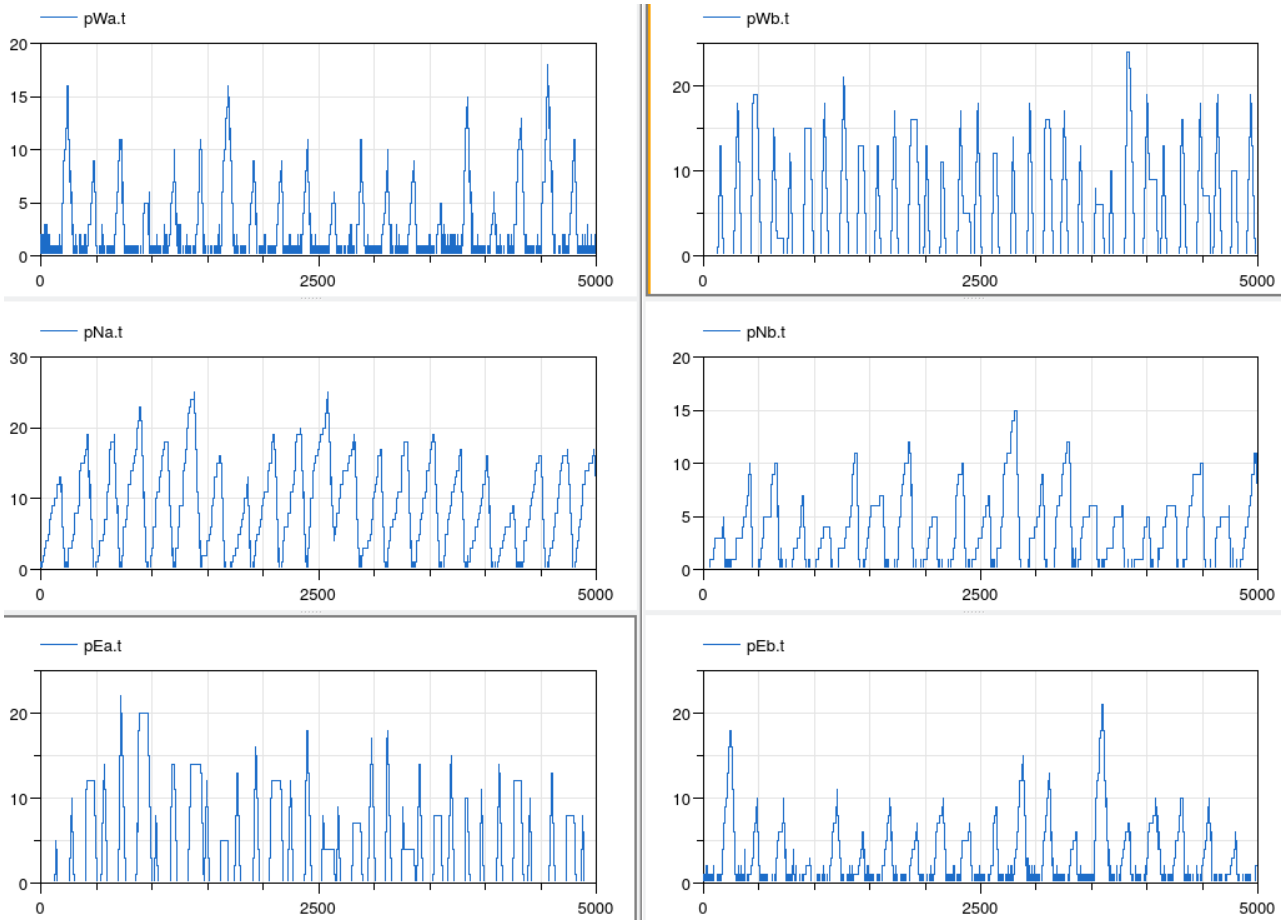




- Straßen knapp an der Kapazitätsgrenze (vor allem Richtung E)
- Übersicht der Fahrzeugströme

Richtung	Vorgabe in	Anzahl in	Fahrzeuge/min in	Anzahl out	Fahrzeuge/min out
W	10	862	10.34	786	9.43
E	8	644	7.73	736	8.83
NW	6	489	5.87	353	4.24
NE	3	251	3.01	325	3.90

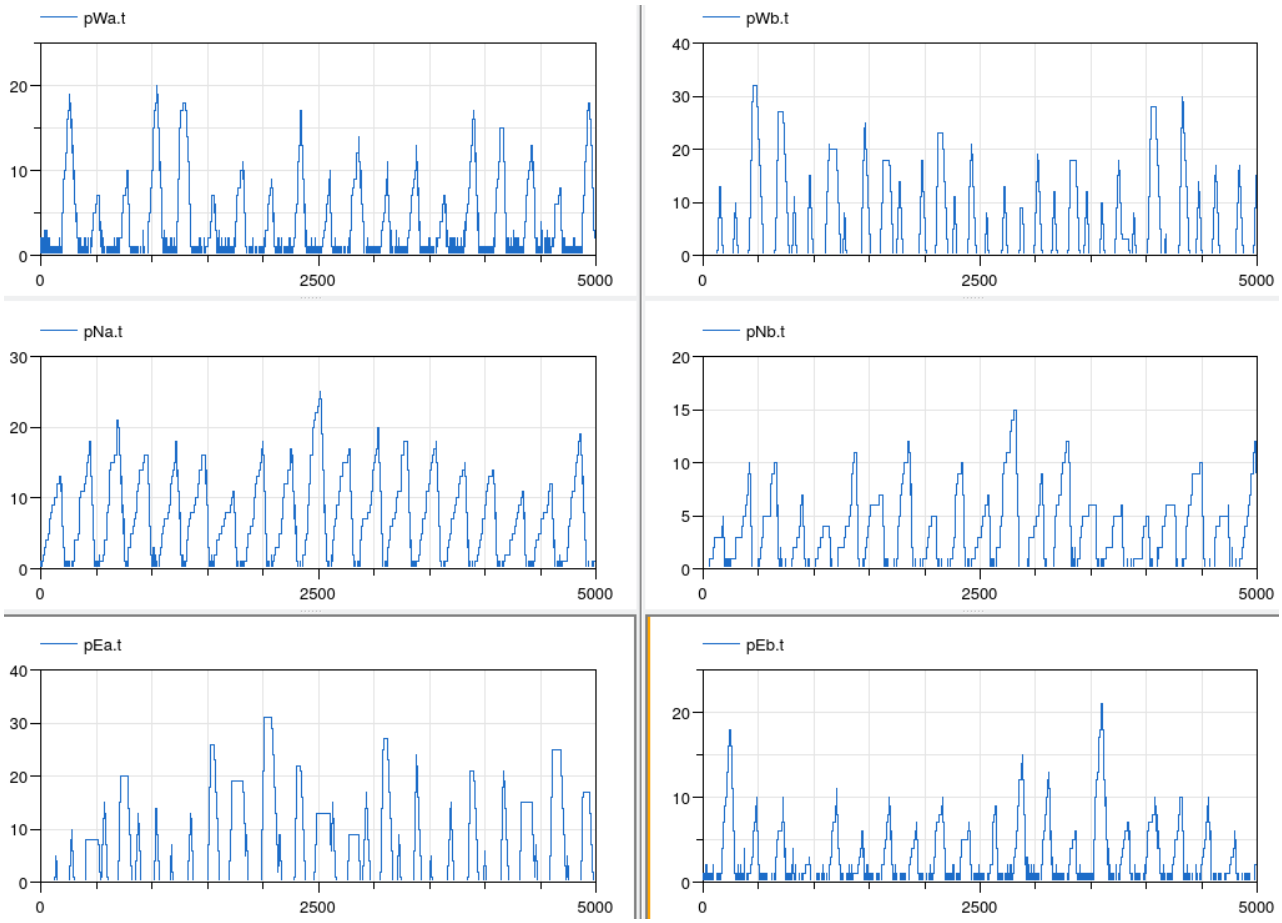
Problemstelle: Schlangen vor den Ampeln



- Schlange aus Richtung NW besonders lang, wird nicht immer abgebaut

Lösungsidee: Ampelphase 3 (links) von 60 s auf 80 s verlängern

Ergebnisse



- Schlange aus NW im Schnitt kürzer, wird immer abgebaut
- Schlangen aus W länger
- Schlange aus W an rechter Ampel deutlich länger

zur genaueren Analyse mehr Kennzahlen nötig, z. B.

- mittlere Längen der einzelnen Schlangen
- mittlere Wartezeiten der Fahrzeuge

- Aufgaben:

  Aufgabe 9

  Aufgabe 10

Warteschlangen und Bedienstationen

- Standardbeispiele:

  Kunden stellen sich an Kassenschlangen an

  Zwischenprodukte werden vor einer Fertigungsmaschine zwischengelagert, bis sie weiterverarbeitet werden

  Aufträge warten im Eingangskorb auf Bearbeitung

  Datenpakete werden bis zur Weiterleitung im Router zwischengespeichert

  abstrakt
  - Entitäten (**Entities**) warten in Warteschlangen (**Queues**), bis sie von Bedienstationen (**Server**) verarbeitet werden

- Bestandteile:

  Bedienstation (Server)
  - bearbeitet eine Entität
  - auch mehrere parallel möglich (N-Server)
  - Bearbeitungszeit fest oder stochastisch

  Warteschlange (Queue)
  - Wartebereich für Entitäten
  - unbeschränkt oder feste Kapazität
  - verschiedene Queuedisziplinen (FIFO, LIFO, Prio)

  Ankunftsprozess (Generator)
  - erzeugt ankommende Entitäten
  - Zeitabstände fest oder stochastisch

- Kendall-Notation:

  übliches Klassifizierungs-Schema von Bediensystemen

  enthält sechs Parameter A|B|s|c|n|Q

  Bedeutung

Parameter	Bedeutung	wichtige Werte
A	Art des Ankunftsprozesses	D (deterministisch) M (Markov), G (allgemein)
B	Art des Bedienprozesses	D (deterministisch) M (Markov), G (allgemein)
s	Zahl der Bediener	1, 2, .., ∞
c	Größe des Warteraums	1, 2, .., ∞
n	max. Anzahl von Kunden	1, 2, .., ∞
Q	Queue-Politik	FIFO, LIFO, Prio, Random

  häufig c = ∞, n = ∞, Q = FIFO
  - werden in Notation weggelassen
  - wichtigstes Beispiel: M|M|1

- Basismodell `singleserver1A`:

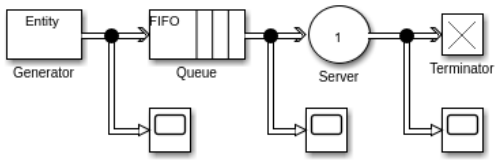
  einfaches Bediensystem mit festen Zeiten (D|D|1)

  Parameter
  - Ankunftsprozess (Generator) mit festem Zeitabstand $t_G = 1$
  - unbeschränkte FIFO-Queue
  - Server mit Bedienzeit $t_S = 1.5$

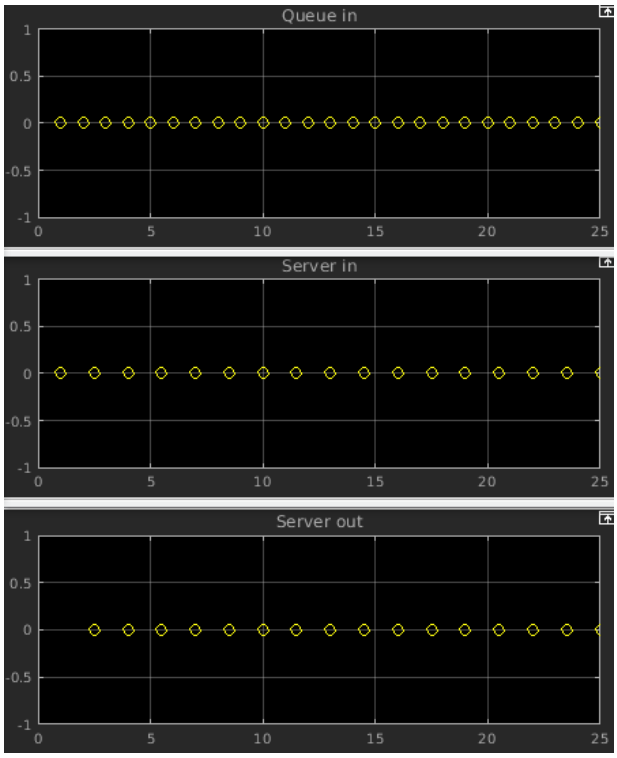
  Modellierung mit SimEvents

- Zusatzpaket zu Matlab/Simulink für DES (**Discrete event simulation**)
- Leitungen transportieren Entitäten, keine Zahlen

Grundaufbau



Oscis zeigen durchlaufende Entitäten an



- Anzeige statistischer Daten:

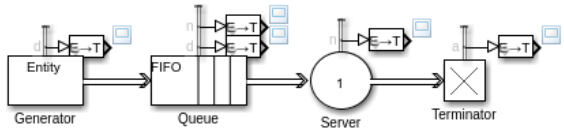
  Übersicht über Entities auch durch Anzeige von Anzahlen

  SimEvent-Blöcke enthalten Parameter-Seite `Statistics`

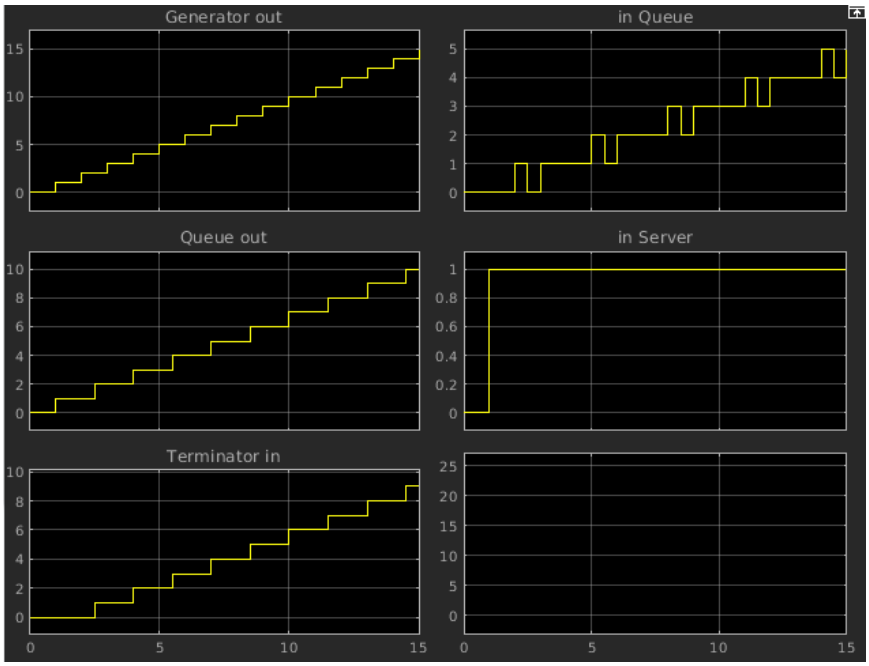
  darunter diverse Anzahlen, z. B.
  - d = Zahl der Entitäten, die einen Block verlassen haben
  - n = Zahl der Entitäten innerhalb eines Blocks
  - a = Zahl der Entitäten, die einen Block betreten haben

  zur bequemen Anzeige in Viewer Umwandlung der Signale von **event-driven** in **time-driven** nötig
  - Trick: automatisch durch `Gain`-Block mit Faktor 1
  - zur Verschönerung: `Gain`-Block in Subsystem versteckt

  Modell `singleserver1B`



  Ergebnisse



- Manuelles Bestimmen des Verhaltens:

  Liste aller Ereignisse (Events) mit Zeiten erstellen
  - für kurze Zeiten bzw. wenige Entities
  - möglichst übersichtlich aufschreiben
  - wichtig zur Fehlersuche in Modellen!

  Vorgehensweise z. B.
  - Tabelle mit Zeit und Belegung der Komponenten
  - zunächst alle vorher schon bekannten Events aufschreiben
  - dann oben anfangen und jeweils neue zukünftige Events einfügen
  - Entitäten durchnummerieren E1, E2, ...

  fertige Liste für `singleserver1B` (bis t = 10)

```
   t     Gen   Que          Srv   raus    Bemerkungen
 ------------------------------------------------
  0.0          -            -             Startzeit
  1.0   E1     ->           E1            E1 läuft bis zum Server,
                                          fertig bei t=2.5 (eintragen!)
  2.0   E2     E2           E1
  2.5          -            E2    E1      fertig bei t=4
  3.0   E3     E3           E2
  4.0          E3           -     E2      *
  4.0          -            E3            *, fertig bei t=5.5
  4.0   E4     E4           E3            *
  5.0   E5     E5/E4        E3            Reihenfolge in Queue beachten!
  5.5          E5           E4    E3      fertig bei t=7
  6.0   E6     E6/E5        E4
  7.0          E6           E5    E4      fertig bei t=8.5
  7.0   E7     E7/E6        E5
  8.0   E8     E8/E7/E6     E5
  8.5          E8/E7        E6    E5      fertig bei t=10
  9.0   E9     E9/E8/E7     E6
 10.0          E9/E8        E7    E6      fertig bei t=11.5
 10.0   E10    E10/E9/E8    E7
```

  Anmerkung *
  - Reihenfolge (E2 verlässt Srv/E3 rückt nach/E4 verlässt Gen) unklar
  - Wahl: erst raus aus Server, dann Nachrücken, dann Erzeugung

- andere Wahlen führen (hier!) zu gleichen Ergebnissen

Vergleich mit Ergebnissen von `singleserver1B`

- passt zu `Generator out`, `Queue out`, `Terminator in`
- passt auch zu `in Queue` und `in Server`

- Kennzahlen eines Bediensystems:

  für Generator
  - mittlere Ankunftsrate λ

  für Queue
  - mittlere Länge der Warteschlange L_Q

  für Server
  - mittlere Bedienrate μ
  - Auslastung A := Aktivzeit/Gesamtzeit

  für Gesamtsystem
  - mittlere Zahl von Entitäten im System L

  für Entities
  - mittlere Wartezeit in Queue W_Q
  - mittlere Verweilzeit (Wartezeit + Bedienzeit) W

  Satz von Little (im Gleichgewicht)
  - $L = \lambda W$
  - $L_Q = \lambda W_Q$

- Manuelles Bestimmen der Kennzahlen:

  mittlere Queuelänge L_Q aus momentaner Queuelänge q(t) durch

  $$L_Q = \frac{1}{T}\int_0^T q(t)\,dt$$

  - manuell durch Tabelle

Queuelänge q	**Dauer Δt**	**q*Δt**
0	2.5	0.0
1	3.0	3.0
2	3.0	6.0
3	1.5	4.5
gesamt		13.5

  - also $L_Q = 1.35$

  Server-Auslastung A analog aus momentaner Serverbelegung s(t) durch

  $$A = \frac{1}{T}\int_0^T s(t)\,dt$$

  - hier einfach, Ergebnis A = 0.9

  mittlere Wartezeit W_Q ist der Mittelwert der Wartezeiten t_i der Entitäten E_i, also (bis zu T = 10)

Entität i	1	2	3	4	5	6	7	8	9
Wartezeit t_i	0.0	0.5	1.0	1.5	2.0	2.5	3.0	2.0	1.0

  - zählt man nur Entitäten nach Verlassen der Queue → $W_Q = 1.5$
  - zählt man alle Entitäten → $W_Q = 1.5$ (zufällig identisch!)

  Check: Satz von Little

- $\lambda = 1 \rightarrow W_Q = L_Q/\lambda = 1.35$
- passt nicht, wir haben ja auch kein Gleichgewicht
- schlimmer noch: es gibt hier keins!

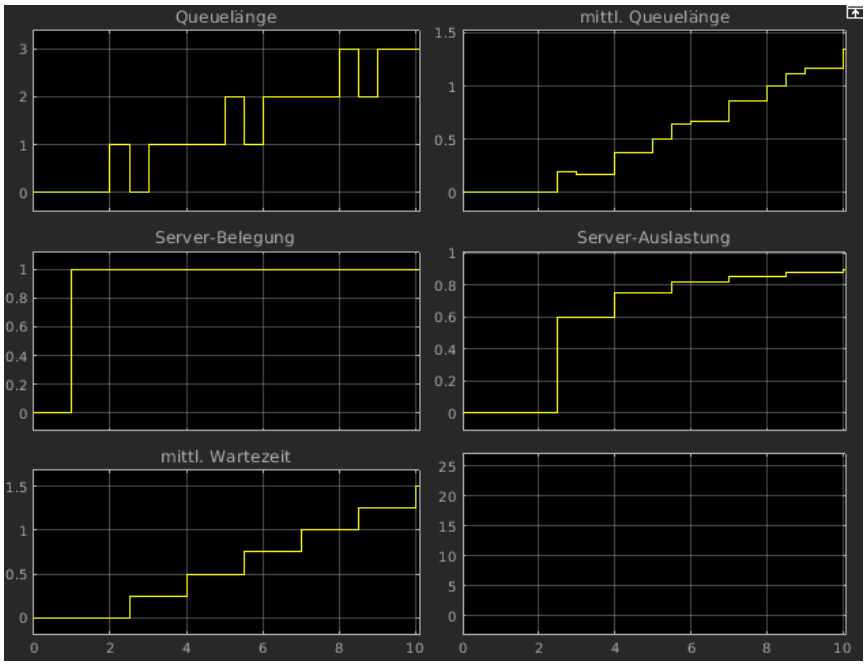
- Kennzahlen aus Simulation (`singleserver1C`):

  Daten direkt aus `Statistics` der Blöcke

  Queue
  - n = Queuelänge
  - l = mittlere Queuelänge
  - w = mittlere Wartezeit in der Queue

  Server
  - n = Zahl der bedienten Entitäten = Aktivität
  - util = Auslastung

  Ergebnisse

  passt mit manueller Auswertung zusammen

  Plot der mittleren Wartezeit → nur fertige Entitäten werden gezählt

Stochastische Bediensysteme

- Stochastische Modelle:

  Zeiten (Ankunft, Bedienung) häufig stochastisch

  typische Verteilungen für Zeitabstand beim Ankunftsprozess
  - Exponential, Erlang, empirisch

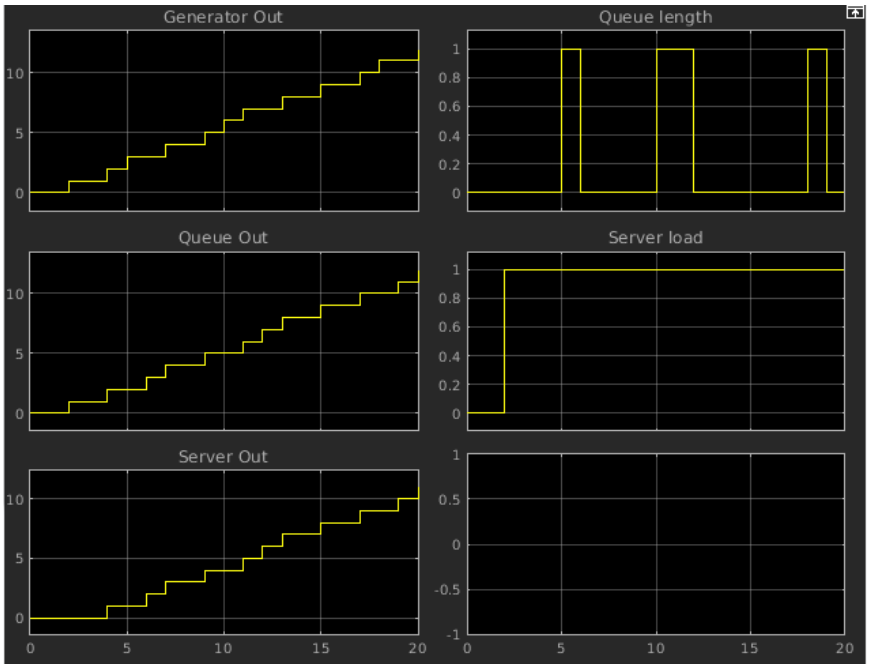
  typische Verteilungen für Bedienzeiten
  - Exponential, Weibull, Normal, empirisch

- Einfaches stochastisches Modell:

  Ankunftszeiten und Bedienzeiten jeweils 1s oder 2s mit gleicher Wahrscheinlichkeit

  Modell `singleserver2A`
  - `Generator/Entity generation/Time source: Matlab action`
  - `Intergeneration time action:  dt = randi(2);`
  - `Server/Main/Service time source: Matlab action`
  - `Service time action:  dt = randi(2);`

  Ergebnis



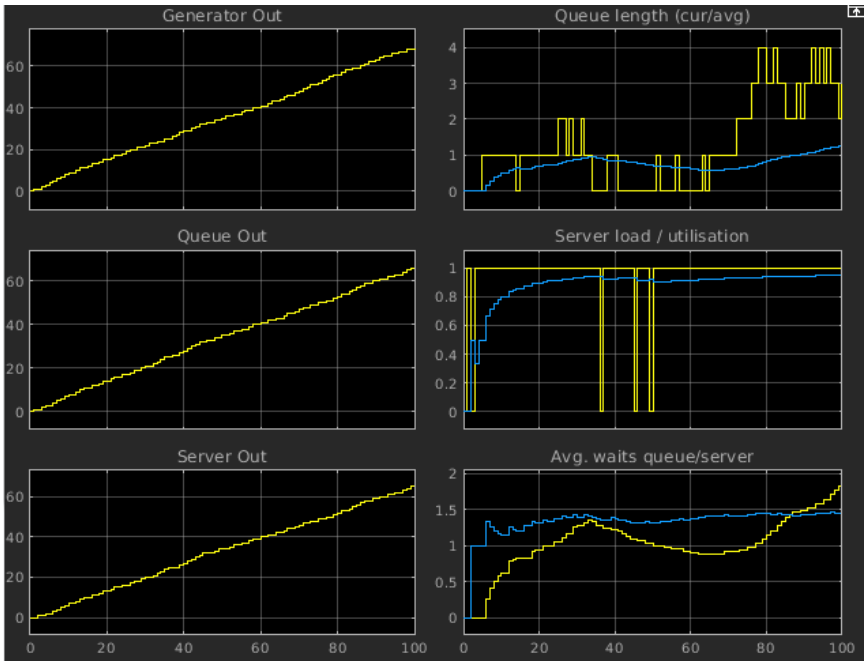
  Analyse
  - wirkt auf den ersten Blick ok: Zeiten zwischen Events wechseln zufällig
  - seltsam: `Server load` = 1 konstant, bei Zufall müsste Server auch mal warten
  - genauer hinsehen: Zeitintervalle bei `Generator Out` und `Server Out` gleich!

  Ausgabe der Zufallszeiten mit `fprintf` (landet im Warnungs-Fenster)
  - Ergebnis: Werte in Generator und Server sind identisch
  - Ursache: Event actions werden nach C übersetzt, bekommen jeweils eigenen `rand`-Aufruf

  Abhilfe: jeweils mit verschiedenen Seeds initialisieren

```
seed = 12;
persistent isSeedSet;
if isempty(isSeedSet)
  rng(seed);
  isSeedSet = true;
end
dt = randi(2);
```

Ergebnis für längere Zeit



- Standardmodell M|M|1 (`singleserver3A`):

  Zeit zwischen zwei ankommenden Entitäten ~ Ex(λ)

  Bedienzeit ~ Ex(μ)

  zur Erinnerung

  - Ankunftsprozess hat die Markov-Eigenschaft

    ⇔ Zeitintervalle zwischen zwei Ankünften sind exponentiell verteilt

  Modell in SimEvents analog `singleserver2B`

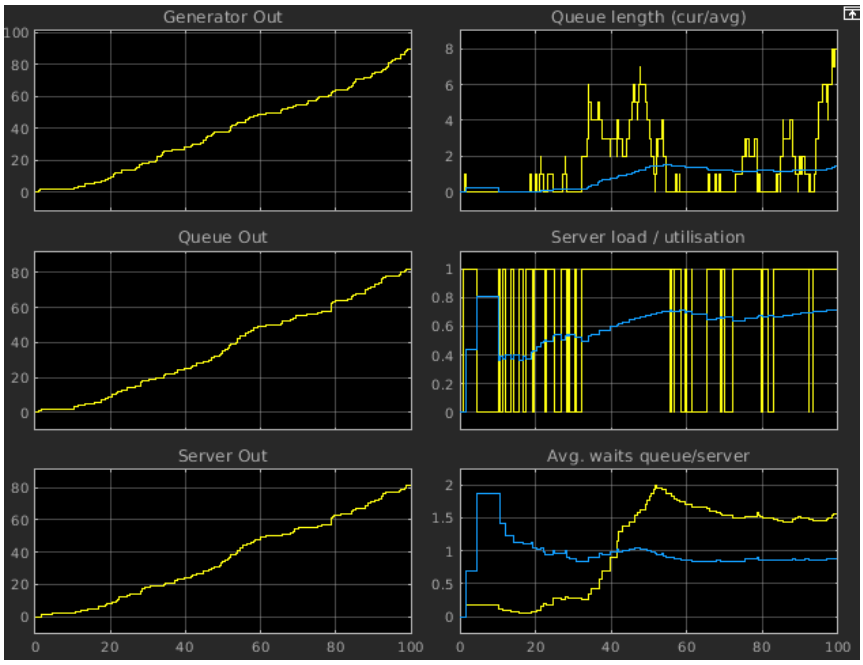
  - Berechnung der Intergeneration time

    ```
    dt = -(1/lambda)*log(rand());
    ```

  - Berechnung der Service time

    ```
    dt = -(1/mu)*log(rand());
    ```

  Ergebnisse für λ = 1, μ = 1.2



- Ergebnisse der Warteschlangentheorie:

M|M|1 mit λ ≥ μ haben keine Gleichgewichtsverteilung

M|M|1 mit λ < μ haben eine Gleichgewichtsverteilung, es gilt

$$A = \frac{\lambda}{\mu} =: \rho$$

$$L_Q = \frac{\rho^2}{1-\rho} \qquad L = \frac{\rho}{1-\rho}$$

$$W_Q = \frac{\rho}{\mu-\lambda} \qquad W = \frac{1}{\mu-\lambda}$$

M|G|1 mit i.i.d-verteilten Servicezeiten mit Erwartungswert 1/μ (mit μ > λ) und Varianz σ^2

$$A = \rho \equiv \frac{\lambda}{\mu}$$

$$L_Q = \frac{\rho^2(1+\sigma^2\mu^2)}{2(1-\rho)} \qquad L = \rho + \frac{\rho^2(1+\sigma^2\mu^2)}{2(1-\rho)}$$

$$W_Q = \frac{\lambda(\frac{1}{\mu^2}+\sigma^2)}{2(1-\rho)} \qquad W = \frac{1}{\mu} + \frac{\lambda(\frac{1}{\mu^2}+\sigma^2)}{2(1-\rho)}$$

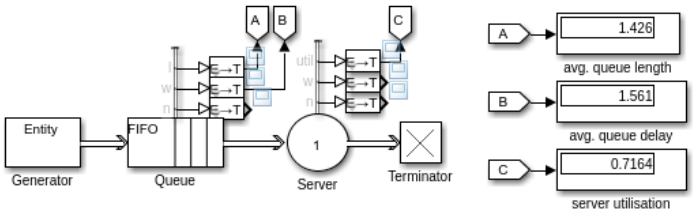
- Beweis in [L3]

- Auswertung der Simulationsergebnisse (`singleserver3B`):

  theoretische Ergebnisse bei λ = 1, μ = 1.2

  - L_Q = 4.1667
  - W_Q = 4.1667
  - A = 0.8333

  am einfachsten Endergebnisse im Modell direkt anzeigen



  Ergebniswerte sind viel zu klein!

  bessere Ergebnisse durch längere Laufzeit

T	L_Q	W_Q	A
100	1.426	1.561	0.7164
1000	3.879	3.948	0.8286
10000	4.921	4.941	0.8471
1e5	4.207	4.228	0.8312
1e6	4.291	4.289	0.8359
1e7	4.185	4.186	0.8333

  - zur Beschleunigung: kein `Osci` und `Decimation=1e5` bei `Displays`

  Analyse

  - Konvergenz *sehr* langsam
  - Ursache: große Schwankungen, z. B. für T = 10000

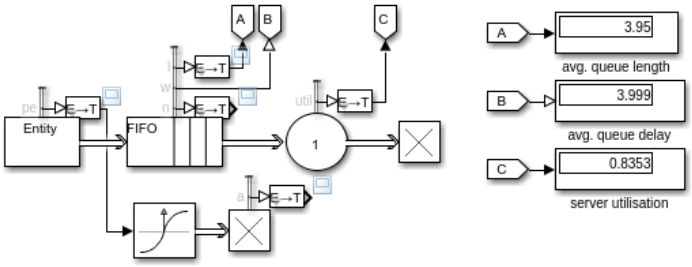
  - Queuelänge wächst bis 33, ist aber auch immer wieder 0

- Queue mit beschränkter Kapazität:

  wie viel Platz braucht man für die Warteschlange?

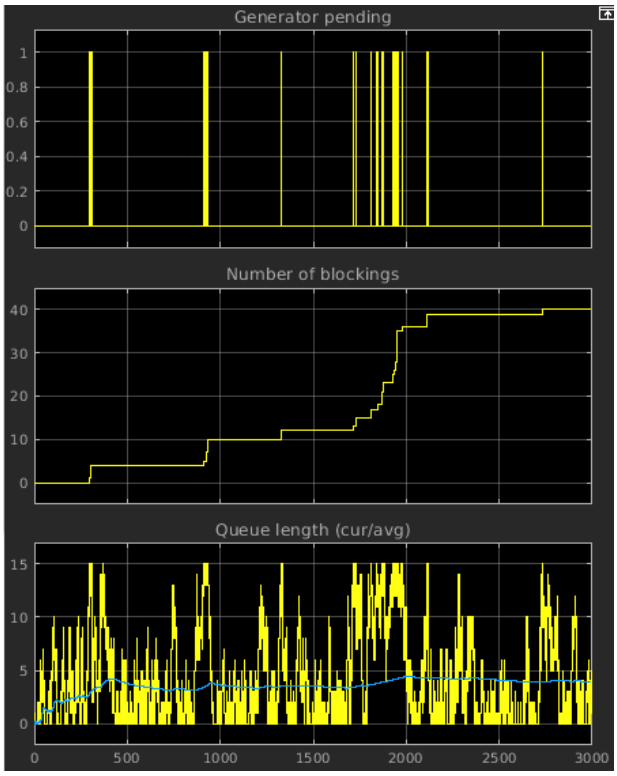
  konkret
  - Bei gegebener Kapazität K der Warteschlange treten Blockaden auf
  - Wie groß muss K sein, damit die Zahl B der Blockaden/Zeitintervall kleiner ist als 1e-3?

  Untersuchung im M|M|1|c-Modell `queuecapacity1A` mit K = 15
  - bei Blockade bleibt neue Entity im Generator (**pending**)
  - Erzeugungsprozess wird gestoppt, bis Generator wieder frei



  Trick zum Zählen der Blockaden
  - `Hit Crossing`-Block erzeugt Message (einfache Entität)
  - `Terminate`-Block zählt und vernichtet sie

  Ergebnis

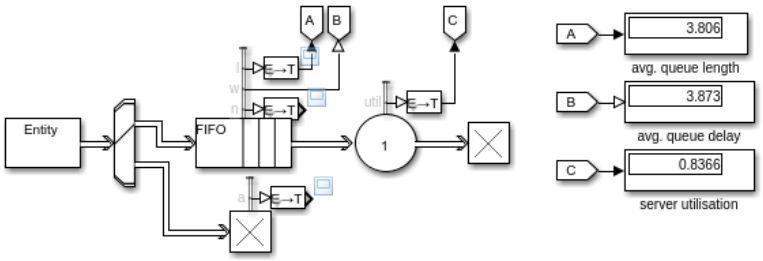


- Blockaden treten gehäuft auf (nämlich bei voller Queue)

- Verbessertes Modell `queuecapacity1B`:

  Problem: Ankunftsprozess wird verändert

  Alternative: geblockte Entities umleiten



  - Verhalten im Detail anders, da Generator ungestört durch Blocking

  Bestimmung von K

  - K setzen und für große Zeit (T = 10^5) simulieren
  - B ablesen
  - K ändern und iterieren

  halb-automatisiert mit Matlabskript `computeQueueCapacity`

  - Ergebnis ist zufällig, Variable ist ganzzahlig
  - quick and dirty mit `fzero`, Abbrechen nach einigen Ausgaben →

```
computeQueueCapacity(1)
run: K =  15,  B = 0.008090
run: K =  35,  B = 0.000180
run: K =  33,  B = 0.000240
run: K =  24,  B = 0.001490
run: K =  27,  B = 0.001010
run: K =  28,  B = 0.000790
run: K =  27,  B = 0.001010
```

  - prüfen mit langer Simulation

```
computeQueueCapacity(2, 26)
run: K =  26,  B = 0.001194
run: K =  27,  B = 0.000986
run: K =  28,  B = 0.000839
```

Ergebnis: 27 Plätze in der Queue sollten genügen - bei $L_Q = 4.1667$!

- Aufgaben:

  Aufgabe 11

  Aufgabe 12

Prozess-basierte Systeme

- Ereignisdiskrete Simulation
- Entwurf von Fertigungsanlagen
- Protokolle zur Netzwerk-Kommunikation
- Logistische Versorgungsketten

Ereignisdiskrete Simulation

- Prinzip der ereignisdiskreten Simulation (DES = **Discrete Event Simulation**):

  Events haben Typ und Zeitpunkt

  alle Events stehen zeitlich geordnet in einer Liste
  - Zeitsteuerung springt jeweils zum nächsten Zeitpunkt in der Eventliste

  zu jedem Typ von Events gibt es eine entsprechende Ereignisroutine
  - ändert Zustand des Systems
  - erzeugt in der Regel neue Events und fügt sie in die Liste ein
  - kann die Simulation beenden

  prinzipielle Ausführung

```
setInitialState()
setInitialEvents()
t = 0
while true          % beenden durch Eventroutine
  nextEvent = getNextEvent(liste)
  t = nextEvent.t
  runRoutine(nextEvent)
end
```

  bei gleichzeitigen Events
  - haben Reihenfolge in der Liste
  - häufig durch Kausalität gegeben ("verlasse Queue", "betrete Server")
  - sonst durch Prioritäten ("verlasse Server" vor "verlasse Generator")
  - oft nur implizit geregelt - wenn überhaupt!

- Ereignisroutinen beim D/D/1-Beispiel:

  Initialisierung

```
tG = 1, tS = 1.5, tEnd = 10, serverFree = true
```

  Eventtypen
  - Entry (neue Entität wird erzeugt)
  - Leave (Entität verlässt Server)
  - Stop (Zeit ist um)

  Routine für Entry

```
scheduleEntry()        % setze nächstes Entry-Event in Liste
if serverFree
  serverFree = false
  scheduleLeave()      % setze nächstes Leave-Event in Liste
else
  addToQueue(entity)
end
```

  Routine für Leave

```
destroy(entity)
if isQueueEmpty()
  serverFree = true
else
  removeFromQueue(entity)
  serverFree = false
  scheduleLeave()
```

```
end
```

Routine für Stop

```
if t == tEnd
  exit
end
```

- Ablauf im Beispiel D/D/1:

  per Hand durchführen

  Ergebnis (**Event-Tabelle**)

```
  t  nQ SFree list
0.0   0    1   1.0E/10.0S/
1.0   0    0   2.0E/2.5L/10.0S/
2.0   1    0   2.5L/3.0E/10.0S/
2.5   0    0   3.0E/4.0L/10.0S/
3.0   1    0   4.0L/4.0E/10.0S/
4.0   0    0   4.0E/5.5L/10.0S/
4.0   1    0   5.0E/5.5L/10.0S/
5.0   2    0   5.5L/6.0E/10.0S/
5.5   1    0   6.0E/7.0L/10.0S/
6.0   2    0   7.0L/7.0E/10.0S/
7.0   1    0   7.0E/8.5L/10.0S/
7.0   2    0   8.0E/8.5L/10.0S/
8.0   3    0   8.5L/9.0E/10.0S/
8.5   2    0   9.0E/10.0S/10.0L/
9.0   3    0   10.0S/10.0L/10.0E/
10.0  3    0   10.0L/10.0E/
```

  Vergleich mit Tabelle von oben
  - grundsätzlich identisch
  - Entities haben keine Identität, nur Anzahl in Queue
  - gleichzeitige Events bei 4.0 wie oben
  - gleichzeitige Events bei 10.0 anders: STOP kommt zuerst

- Problem bei DES:

  globale Sicht (von außen auf das ganze Modell)
  - globale Ereignisliste
  - global agierende Eventroutinen

  bei komplexen Systemen schwierig zu erstellen bzw. nachzuvollziehen

  andere Modellierungsverfahren
  - aus der Sicht der Akteure (**prozess-orientiert**)
  - aus der Sicht der Entitäten (**transaktions-orientiert**)
  - aus der Sicht der Komponenten (**komponenten-orientiert**)

  basieren intern auf DES

- Prozess-orientierte Simulation:

  Ereignisse werden Akteuren (abstrahiert als Prozesse) zugeordnet

  Aktivität = Menge von Operationen eines Akteurs während eines Zeitintervalls
  - z.B. "Kunde wartet" (in der Queue) oder "Kunde zahlt" (an der Kasse)
  - z.B. "Kasse bedient" oder "Kasse wartet"
  - hat jeweils Start- und End-Event

  Prozess = Folge von Aktivitäten eines Akteurs

- End-Event einer Aktivität ist Start-Event der folgenden Aktivität

z. B. Prozess Kunde mit Aktivitäten
- Kunde kommt - Kunde wartet - Kunde bezahlt - Kunde geht

z. B. Prozess Kasse mit Aktivitäten
- Kasse wartet - Kasse bedient - Kasse wartet - ... (bis Simulationsende)

mögliche Zustände eines Prozesses
- aktiv: eine seiner Aktivitäten wird ausgeführt
- wartend: zukünftige Aktivitäten sind geplant
- passiv: keine zukünftigen Aktivitäten sind geplant

Prozess kann anderen passiven Prozess aktivieren
- z.B. bei Ende von "Kasse bedient" wird nächster Kunde aktiv

Vorteile
- besser nachvollziehbar als direktes DES
- graphische Unterstützung vorhanden

Nachteile
- komplexe Simulationsroutine muss zwischen Prozessen umschalten
- Prozess-Routinen komplexer als Event-Routinen
- häufig deutlich größere Zahl von Ereignissen

- Spezielle Sichtweisen der prozess-orientierten Simulation:

  transaktions-orientiert
  - Akteure sind die Entitäten
  - sie laufen von Komponente zu Komponente
  - haben oft zusätzliche Eigenschaften (u.a. eine Id)
  - Ablauf des D/D/1-Modells mit Entitäten in der **Entitäten-Tabelle** (s.o.)

  komponenten-orientiert
  - Akteure sind die Komponenten
  - sie verarbeiten einlaufende Entitäten und reichen sie weiter
  - Akteure neben Queues oder Servern auch Verzweigungen, Schalter etc.

  oft eher eine Frage der Sichtweise als des Programmablaufs

  Uneindeutigkeiten bei gleichzeitigen Events
  - transaktions-orientiert: welche Entity läuft zuerst weiter?
  - komponenten-orientiert: welche Komponente arbeitet zuerst?
  - in konkreten Simulationsprogrammen oft schwierig aufzulösen bzw. überhaupt zu sehen

- Aufgaben

  Aufgabe 13

  Aufgabe 14

Entwurf von Fertigungsanlagen

- Simulation von Fertigungsanlagen:

  wichtige Fragestellungen
  - Wie wirkt sich der Ausfall einer konkreten Maschine aus?
  - Wie groß ist der Anteil defekter Produkte?
  - Wie groß ist die Auslastung?

  Modellierung
  - Bearbeitungsstationen abstrahiert als Server
  - Grundmodell: komplexe Ketten von Queues und Servern

  Erweiterung durch spezielle Bausteine, z. B.
  - Transportsysteme
  - Bearbeitungsstationen mit Ausfall- oder Wartungszeiten
  - Bearbeitungsstationen mit komplexerem Verhalten

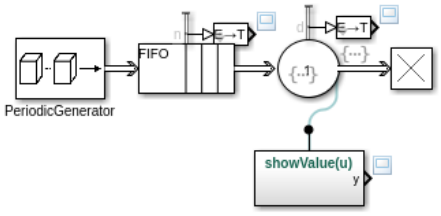
  spezialisierte Software
  - enthält viele konkrete Komponenten
  - berücksichtigt Platzbedarf
  - ermöglicht 3d-Animationen und "virtuelle Fabrik"

  Beispiele i. F.
  - einige spezialisierte Komponenten
  - einfache Fertigungskette

- Bearbeitungsstation mit defekten Werkstücken:

  Erweiterung des Servermodells
  - Anteil p von Werkstücken ist defekt
  - implementiert durch Entity-Attribut `isOk` (1 = ok, 2 = defekt)

  Modell `reworking1A`



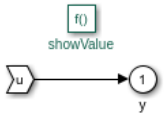
  - Ankunftsintervall tG = 1
  - Queuekapazität nQ = 3
  - Servicezeit logarithmisch mit Mittelwert tS = 0.8

  Server setzt `isOk` in `Service complete`-Funktion

```
pD = 0.3;           % probability of defect
if rand() < pD
  entity.isOk = 2;  % defect
else
  entity.isOk = 1;  % ok
end
```

  - `Exit`-Funktion darf keine Attribute ändern!

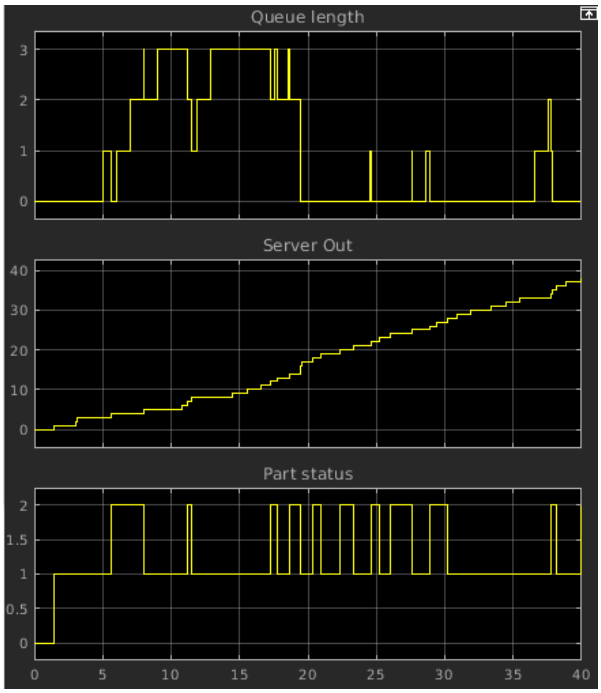
  Anzeige des Attributs `isOk`
  - mit `Simulink Function showValue`



- und `Exit`-Funktion des Servers

```
showValue(entity.isOk);
```

Ergebnis



- Ausgabe unschön, da nicht Zeitpunkten zugeordnet

- Hilfsblock `PeriodicGenerator`:

  erzeugt vorgegebene Anzahl von Entities mit fester Periode

  - Trick: Zeit `dt` wird unendlich nach letzter Entity

```
persistent nr;
if isempty(nr)
  nr = 1;
end
if nr <= N     % N is mask parameter
  dt = T;      % T is mask parameter
else
  dt = inf;
end
nr = nr + 1;
```

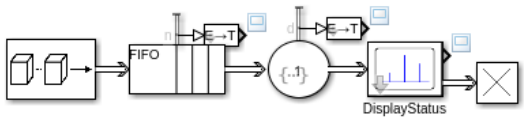
  definiert Attribut `id` zur Kennzeichnung der Entities in `Generate`-Funktion

```
persistent next_id
if isempty(next_id)
  next_id = 1;
end
entity.id = next_id;
next_id = next_id + 1;
```

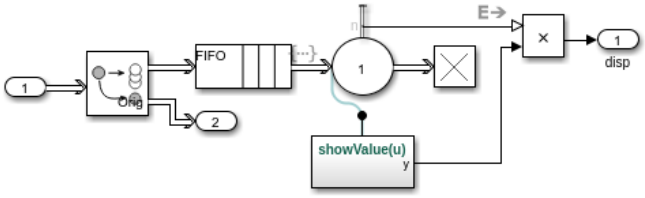
  definiert i.F. benutzte Attribute (u.a. `isOk`)

- Verbesserung der Ausgabe:

  Modell `reworking1B` enthält eigene `DisplayStatus`-Komponente



Funktion von `DisplayStatus`

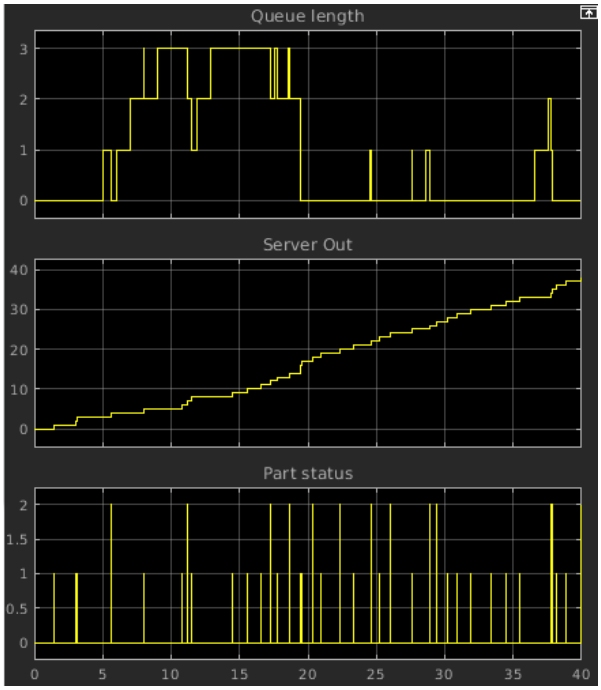


- durchlaufende Entities werden repliziert
- Kopien dienen zur Ausgabesteuerung
- Server hat sehr kurze Servicezeit (w = 0.001)
- gibt Belegung n (1 oder 0) über Statistikausgang aus
- gibt Attribut `isOk` über Entry-Funktion `showValue(entity.isOk);` und Simulink Function aus
- Ausgang `disp` enthält kurze Peaks der Höhe `isOk`

Problem bei Entities mit kleinerem Abstand als w

- Queue hält sie auf → Peak wird entsprechend dicker

Ergebnis

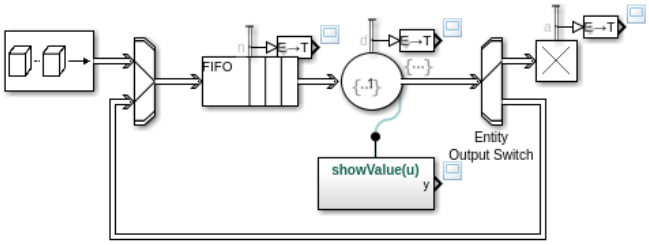


Vorsicht

- `Entry Replicator` hat einen internen Speicherplatz, falls Ausgang geblockt

- Nachbearbeitung eines Werkstücks:

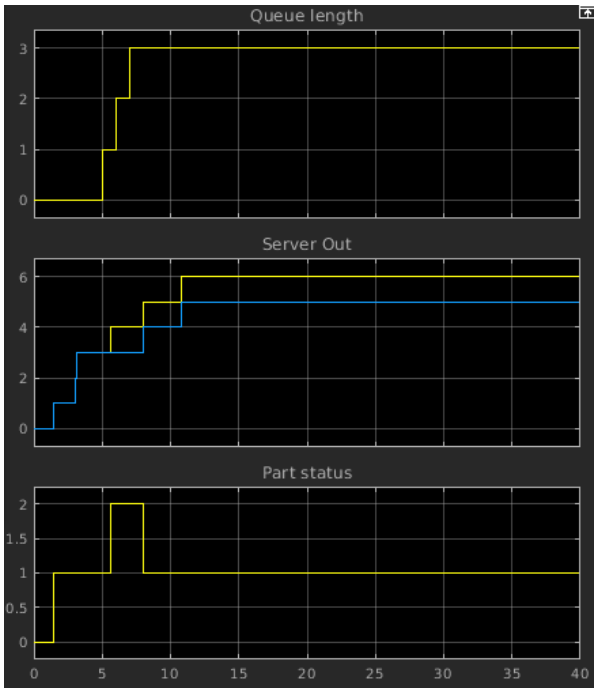
  defektes Werkstück wird zur Wiederverarbeitung zurückgeführt (`reworking2A`)



`Entity Output Switch` wählt Ausgang nach `entity.isOk`

- dafür Werte 1, 2 erforderlich

Ergebnis (zunächst mit einfacher Ausgabe)



- Server blockiert, keine Ausgabe nach t = 12
- Ursache: Queue ist voll, defektes Werkstück kann Server nicht verlassen
- **Verklemmung** (**Deadlock**) - gefährlich in der Realität

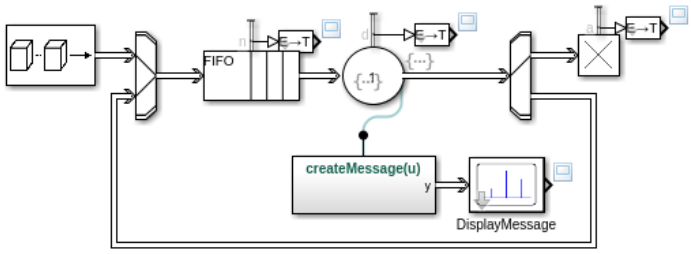
Modell `reworking2B` mit `DisplayStatus` für bessere Ausgabe

- zeigt keine Verklemmung!
- Ursache: defektes Werkstück wandert in den Replicator → keine Verklemmung

Vorsicht mit unerwarteten Seiteneffekten von Komponenten!

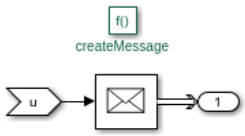
- Bessere Ausgabe ohne Replicator:

  Modell `reworking2C`

  Trick

  - `Simulink Function` erzeugt eine Message

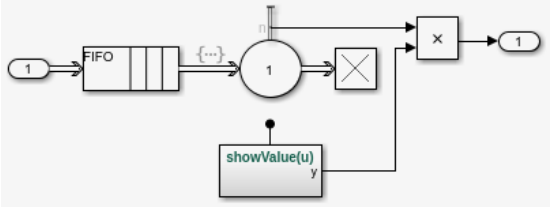
    

  - ersetzt Replicator

  Message

  - einfache Entity ohne Attribute
  - hat einen internen Zahlenwert (direkt als `entity`)

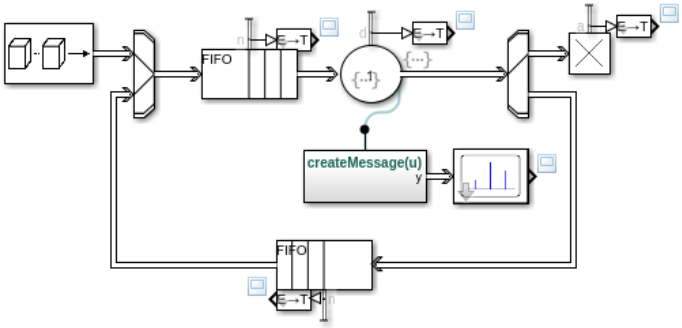
  Anzeige mit `DisplayMessage` i. W. wie vorher



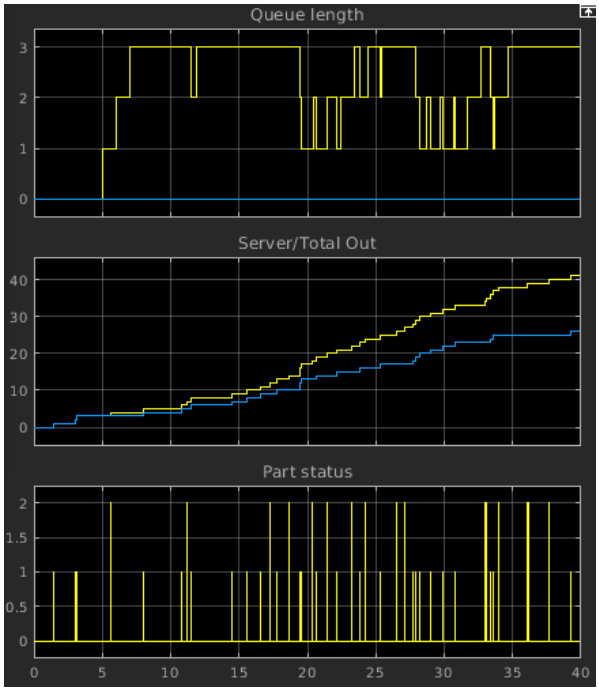
- Rückführung eines Werkstücks ohne Verklemmung:

  Lösung klar: zusätzlicher Platz (`Queue1` mit `nQ = 1`) für das zurückzuführende Werkstück

  fertiges Modell `reworking2D`



  Ergebnis



  Queuelänge der zusätzlichen `Queue1` immer Null (blaue Linie oben)
  - klar: das Wechseln Server → Queue1 → Queue ist zeitlos

  Was passiert, wenn gleichzeitig neues Werkstück eintrifft?
  - Verklemmung, wenn neues Vorrang hat
  - Vorrang des unteres Wegs durch Priorität regeln

- Förderband:

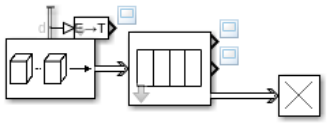
  Transportsysteme sind wichtige Komponenten der Fertigungssimulation

  hier einfaches Modell eines Förderbands (`Conveyor`)
  - Parameter Transitzeit tT und Kapazität nC
  - Statistik-Ausgänge n und d
  - wesentlich komplexeres Modell in SimEvents-Bibliothek

  Implementierung

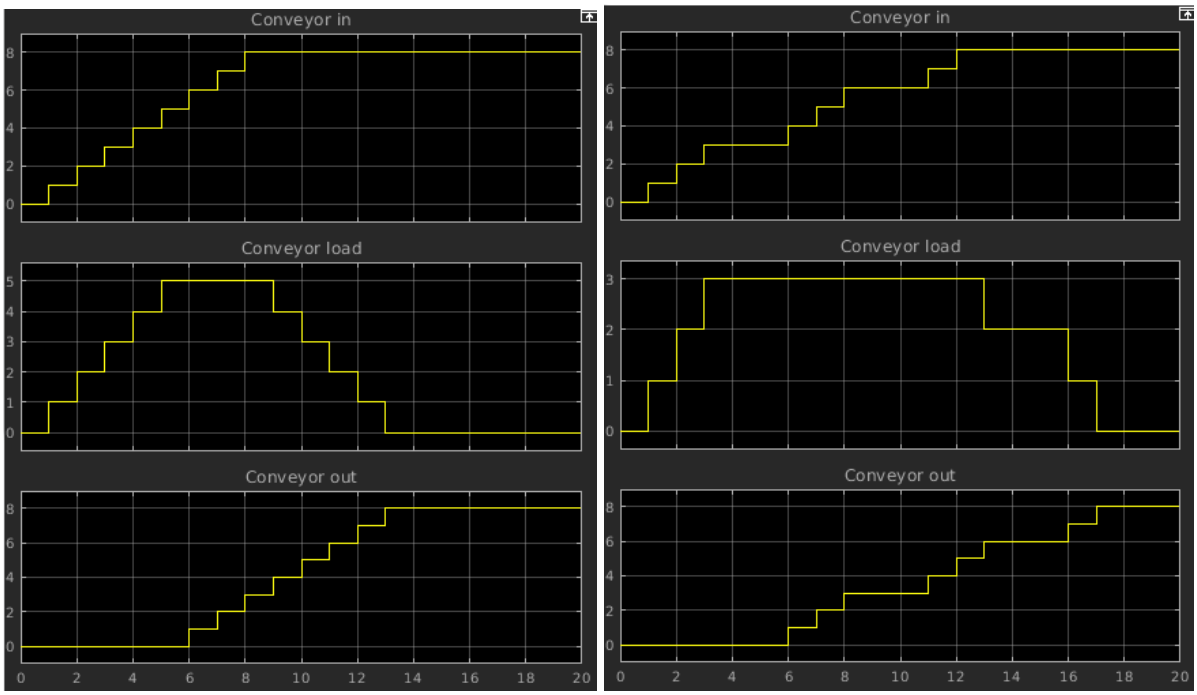
- Server mit Kapazität nC und Service time tT

einfaches Testmodell `testConveyor`



- Generator mit Periode tG = 1 und Anzahl nG = 8
- *keine* Queue

Ergebnisse für tT = 5 und nC = 5 bzw. nC = 3



was macht `Entity Generator`, wenn Ausgang blockiert?

- hält Entity `pending`
- nächste Periode startet, wenn Entity den Block verlässt

mit Entitäten-Tabelle manuell überprüfen für nC = 3

```
t   Gen   Srv       Out   Kommentar
 0   -     -
 1   E1    E1
 2   E2    E2/1
 3   E3    E3/2/1
 4   -     E3/2/1          *1
 6   E4    E4/3/2    E1    *2
 7   E5    E5/4/3    E2
 8   E6    E6/5/4    E3
11   E7    E7/6/5    E4
12   E8    E8/7/6    E5
13         E8/7      E6
16         E8        E7
17         -         E8
```

Kommentare

- *1 Server ist voll, Generator hält an
- *2 Reihenfolge ist hier wichtig: erst wird der Server geleert, dann kommt die neue Entität vom Generator
- wäre es anders herum, käme erst bei t=7 wieder eine Entität!

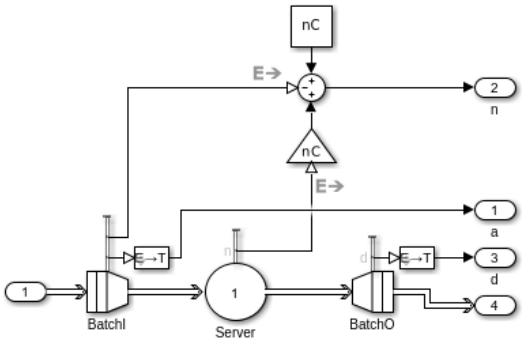
Tabelle stimmt mit den drei Kurven überein

- Brennofen (`OvenA`):

  generisches Modell einer Bearbeitungsstation, die mehrere Werkstücke auf einmal bearbeitet
  - Parameter Backzeit tM und Kapazität nC
  - Statistik-Ausgänge a, n und d

  Ablauf in Phasen
  - Beladen, bis nC erreicht ist
  - Verarbeiten, Dauer tM
  - Entladen aller Teile gleichzeitig (hier instantan)

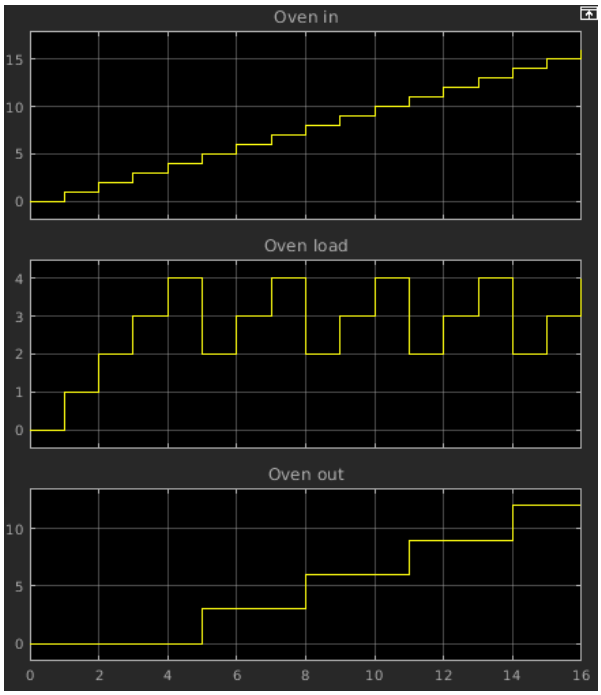
  Implementierung



  - `Entity Batch Creator` erzeugt Batch (Array von Entities) der Größe nC
  - speichert zwischen (externes Backblech)
  - gibt Anzahl noch fehlender Entities aus
  - `Entity Batch Splitter` erzeugt aus Batch wieder einzelne Entities

  TestModell `testOvenA`
  - Generator mit Periode tG = 1
  - Oven mit nC = 3, tM = 2

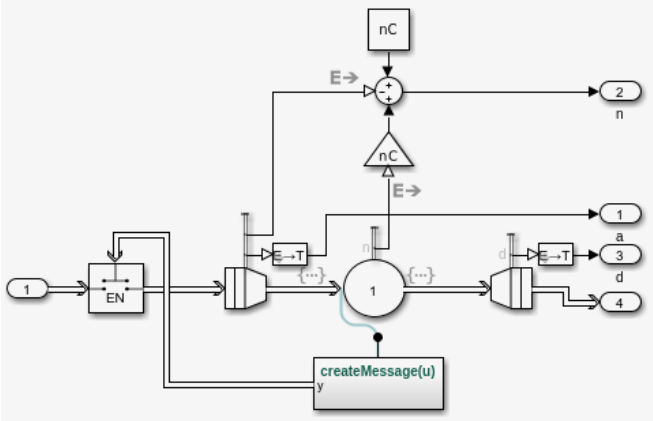
  Ergebnis



- Erweiterung des Brennofens (`Oven`):

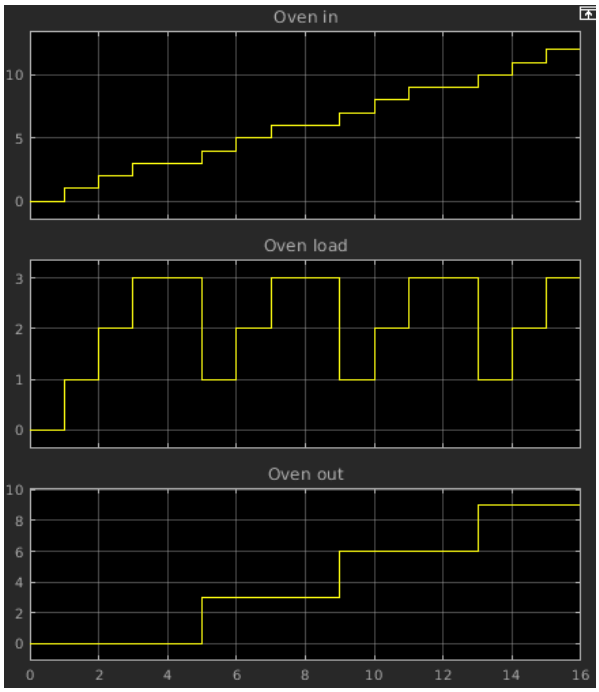
  Ansammeln direkt im Ofen (kein externes Backblech)

  Implementierung



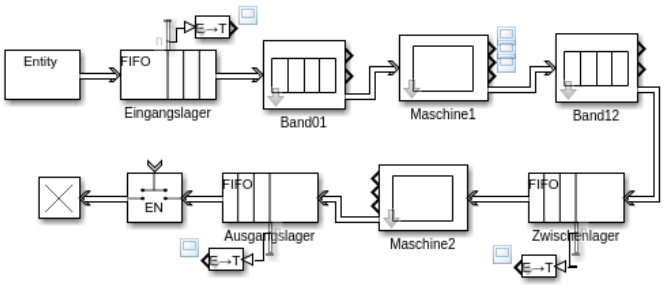
- `Entity Gate` am Eingang öffnet/schließt bei Message mit positivem/negativem Wert
- am Anfang geöffnet
- `Entry`-Funktion des Servers: `createMessage(-1)`
- `Exit`-Funktion des Servers: `createMessage(1)`

Ergebnis von `testOvenB`



- Fertigungskette:

  einfaches Modell `fertigungsketteA`



- `Entity Gate` am Ausgang geschlossen → Ausgangslager sammelt Endprodukte an

Parameter

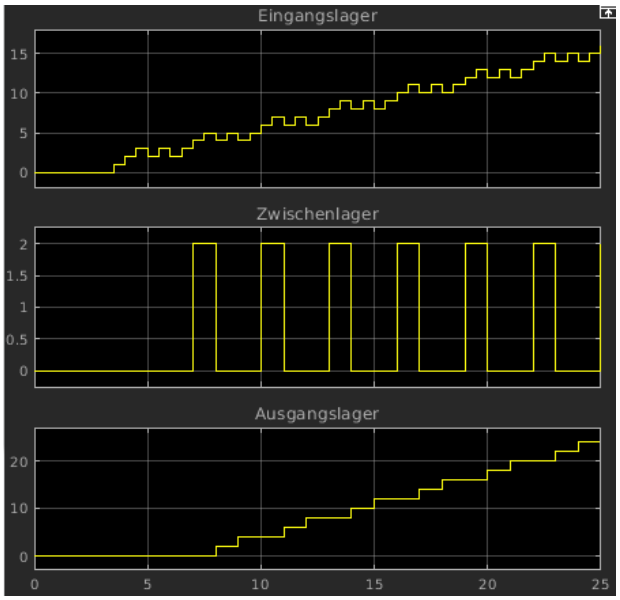
- `Generator`: tG = 0.5
- `Band01`: tT = 1, nC = 2
- `Maschine1`: tM = 2, nC = 4

- `Band12`: tT = 2, nC = 4
- `Maschine2`: tT = 1, nC = 2

Erwartung

- alle Komponenten schaffen durchschnittlich 2 Teile/Zeiteinheit
- Durchsatz (= Ankunftsrate am Ausgangslager) sollte also 2 Teile/Zeiteinheit sein
- alle Lager sollten endliche Dauergrößen haben

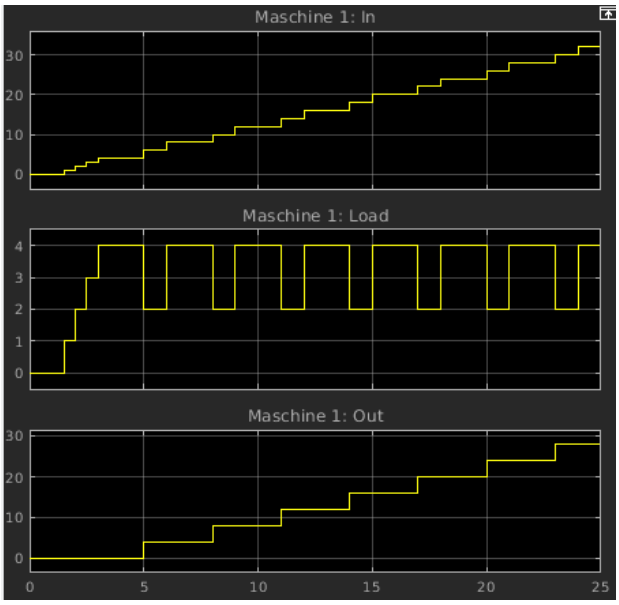
Ergebnis



- durchschnittlicher Durchlauf: 4/3 Teile/Zeiteinheit
- Eingangslager wächst an

- Verbesserung des Durchsatzes:

  Problem bei `Maschine1`

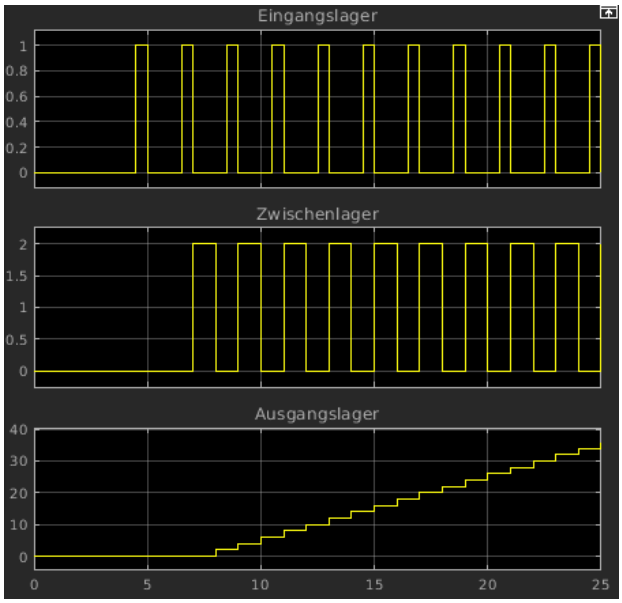


  - Maschine macht Pausen
  - Ursache: Beladung dauert 2 Schritte

  Abhilfe

  - `Band01` muss schneller liefern: tT = 1, nC = 4

  Ergebnis (`fertigungsketteB`)



    - alles ok

- Aufgaben

    Aufgabe 15

    Aufgabe 16

Protokolle zur Netzwerk-Kommunikation

- Fragestellung:

  Wie kann man eine große Datei sicher über das Internet verschicken?

  Grundprinzip
  - Datei in Blöcke zerlegen
  - Blöcke verschicken (ggf. auf verschiedenen Wegen)
  - beim Empfänger einsammeln und auf Vollständigkeit prüfen
  - ggf. fehlende Blöcke nachfordern

- Grundsätzlicher Modell-Aufbau:

  Generator erzeugt große Datei

  Zerleger macht daraus viele Einzelblöcke und schickt sie weiter

  Router schickt Blöcke weiter, wählt einen von mehreren Wegen

  Empfänger sammelt Blöcke ein und erzeugt komplette Nachricht

- Stark vereinfacht, es fehlt z. B.:

  dynamische Wahl des Weges durch komplexes Netzwerk

  Verlorengehen von Blöcken
  - → keine Nachforderung fehlender Blöcke

  volle Komplexität in den Protokollen TCP und IP

- Interessierende Größen:

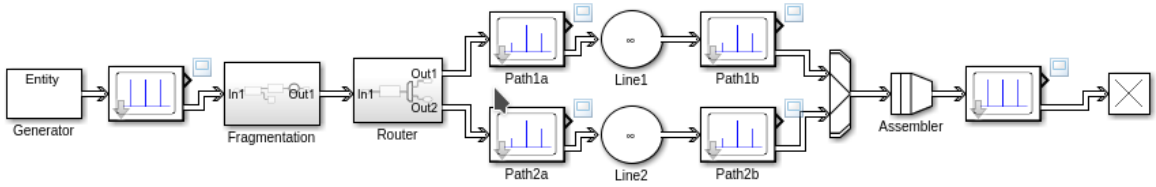
  Durchsatz (MB/s) bei kleinen/großen Dateien

  Zeit Sender - Empfänger

  Verhalten bei mehreren Dateien, z. B.
  - zwei große Dateien parallel
  - eine große Datei + viele kleine

- Grundbeispiel `sendmessage1A`:

  eine Nachricht auf zwei Wegen



  Generator
  - erzeugt eine große Nachricht
  - ≙ eine Entität mit Attributen `messageId`, `size` (in Blöcken), `destination`, `blockId` (für später)
  - Maske mit Parametern für `messageId`, `size`, `destination`, Erzeugungszeit `t0`

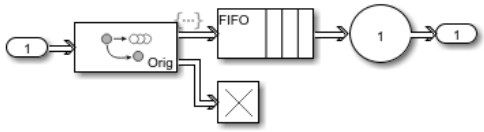
  Berechnung der `Intergeneration time`

```
persistent index;
if isempty(index)
  index = 1;
end
times = [t0, inf];
dt = times(index);
```

```
index = index + 1;
```

Fragmentation

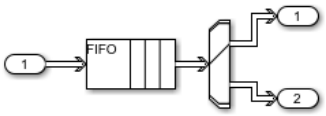
- zerlegt Nachricht in Blöcke



- `Entity Replicator` erzeugt `size` Kopien
- Server mit fester Zeit 0.1s → Blöcke kommen in kurzen Zeitabständen
- Queue als Zwischenspeicher, setzt `blockId` in `Entry action`

Router

- verteilt Blöcke auf zwei Wege



- verwendet Verteilmethode `Equiprobable`
- hat eigenen Zwischenspeicher

Leitung (Line)

- Server mit unendlicher Kapazität
- feste, aber pro Line unterschiedliche Servicezeit (= Laufzeit)

Anzeige der `blockId`s (Path1a etc.)

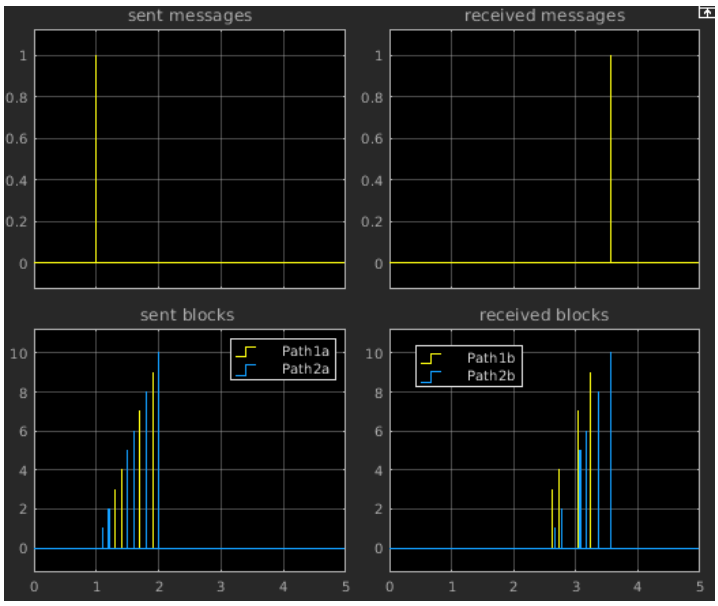
- Display-Variante mit Masken-Parameter `attrib` zur Auswahl des angezeigten Attributs
- Eigenschaft `Evaluate` von `attrib` ist `off`
- Masken-Initialisierung setzt `Entry action` des Servers

```
varName = ['entity.', strtrim(attrib)];
block = [gcb, '/EntityServer'];
set_param(block, 'PostArrivalFunction', ['showValue(' varName ');']);
```

Assembler

- Standard-Komponente `Entity Batcher`
- sammelt feste Zahl von Entitäten in einer einzelnen Batch-Entität
- Batch-Entität intern Array der Einzel-Entitäten

Ergebnis



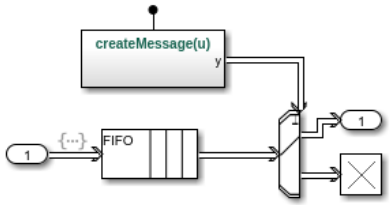
- Dynamischer Assembler (`sendmessage1B`):

  Problem: `Entity Batcher` verwendet feste Zahl von Blocks, nicht aus Message

  besser: Länge N der Message aus Message-Attribut

  - Idee: N-1 ankommende Blöcke vernichten, N-ten durchlassen
  - letzter Block repräsentiert dann komplette Message
  - wir sind nicht an der genauen Nachbildung der Message interessiert, sondern nur am Routing-/Zeitverhalten

  Implementierung

  - Queue steuert `Output Switch` über `Entry Action`

    ```
    persistent count;
    if isempty(count)
      count = 1;
    end
    if count == entity.size
      port = 1;
    else
      port = 2;
    end
    count = count + 1;
    createMessage(port);
    ```

  - Queue sammelt nicht, aber es gibt keinen "Durchlauf"-Block mit `Entry` oder `Exit Action`
  - `Simulink Function` `createMessage` erzeugt Message (wie oben)

  Verhalten identisch zu vorher bei n = 10, klappt auch bei n = 20

  Problem bei n = 30: nur 25 Blöcke werden erzeugt

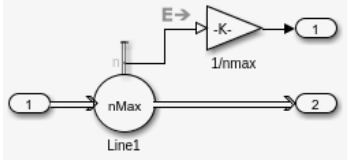
  - Ursache: FIFO bei Fragmentation zu klein

  `sendmessage1C` hat großen FIFO → alles ok

- Erweiterung des Leitungsmodells:

  Komponente `Line`

  - Durchlaufzeiten exponentiell verteilt
  - maximale Kapazität als Parameter
  - zusätzlich Ausgabe der momenten Auslastung

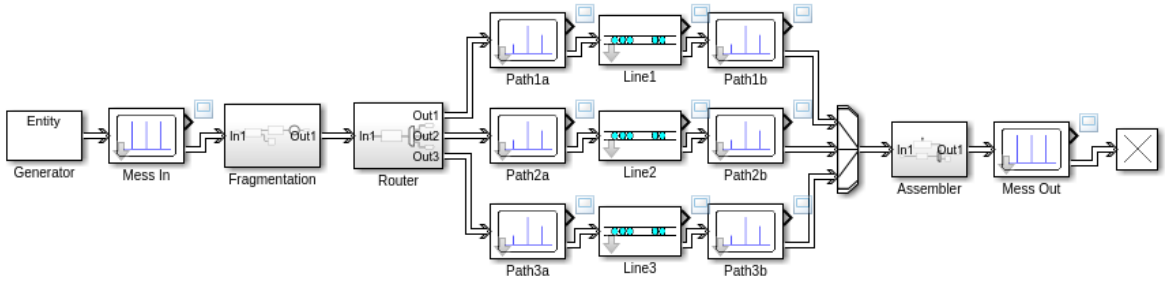
  Problem: viele Blöcke mit Zufallswerten

  - verwenden alle verschiedene Zufalls-Streams (wegen Übersetzung nach C++)
  - brauchen verschiedene Seeds
  - häufiger Trick: eine generelle Seed, andere Blöcke addieren Konstanten
  - bei vielen Blöcken unübersichtlich und fehleranfällig

  bessere Lösung

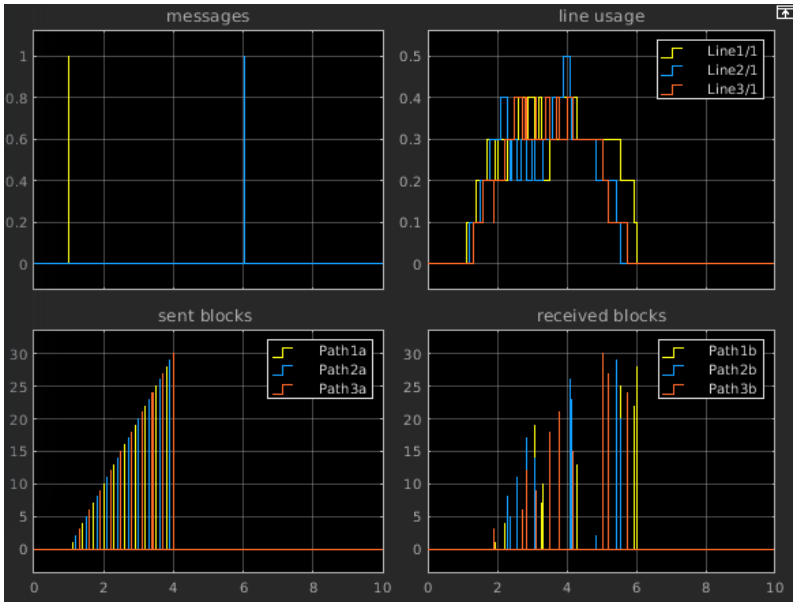
- ein globaler Stream im Modell
- in Action-Funktionen Matlabs `rand`-Funktion verwenden: `coder.extrinsic('rand');`
- modell-globale Seed setzen per `InitFcn()`

Gesamtmodell `sendmessage2A`



Router verwendet Methode `round-robin`

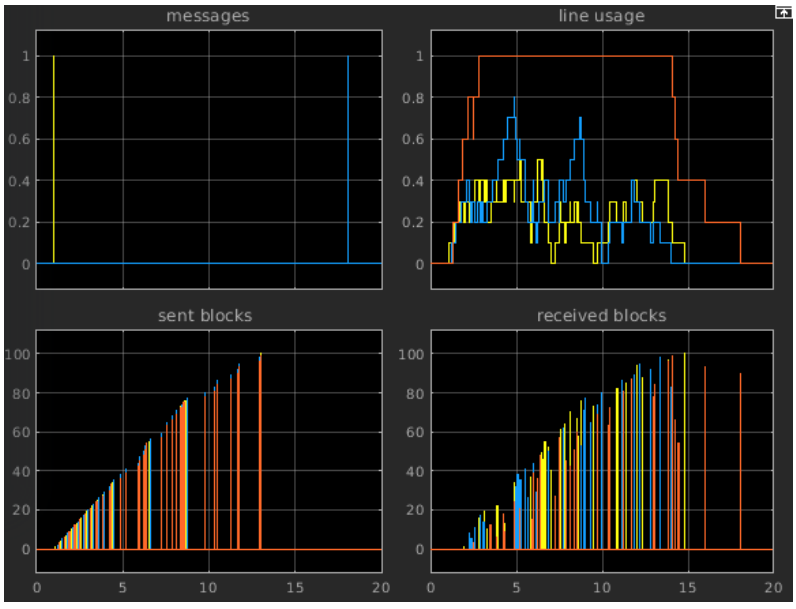
Ergebnis



- Schicken einer längeren Nachricht:

  andere Parameter (`sendmessage2B`)
  - unterer Weg langsamer, kleinere Kapazität

  Ergebnis

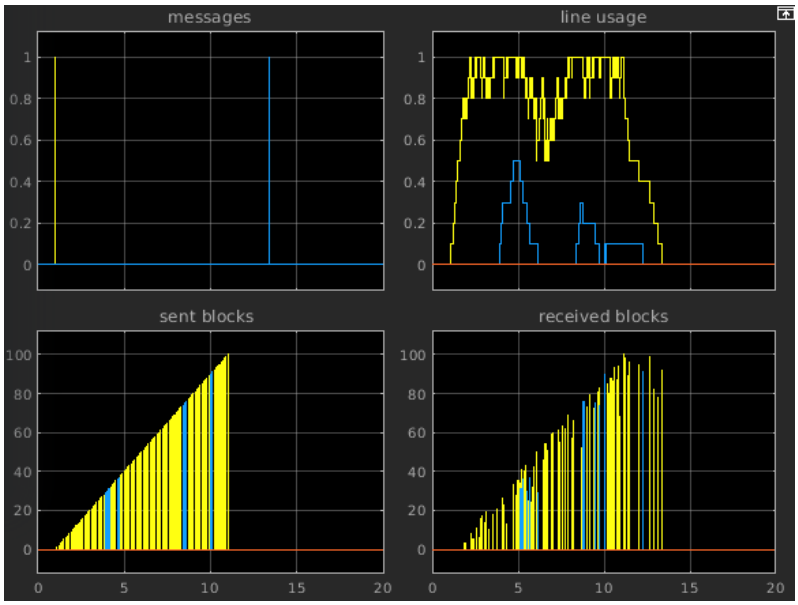


Analyse

- untere (rote) Leitung ausgelastet, blockiert
- → alle anderen Leitungen warten (wegen round-robin)

Übertragungszeit 17.1 s

Routing umstellen auf `First port that is not blocked` (`sendmessage2C`)



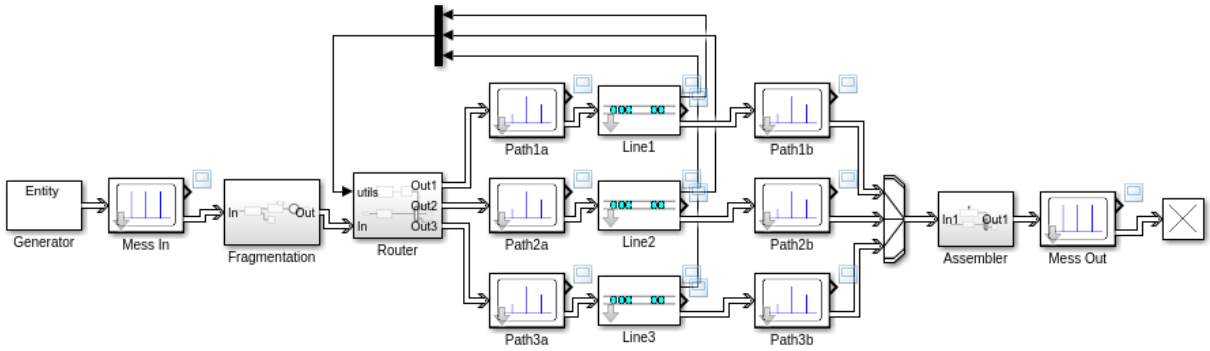
- Blöcke verwenden `Line1`, bis die voll ist, dann auch `Line2`
- besser: Lines gleichmäßig auslasten

Übertragungszeit 12.4 s

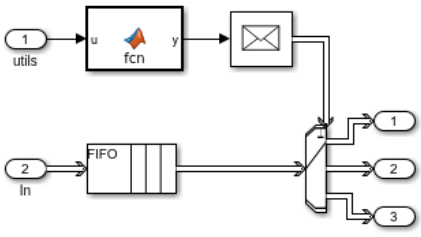
- Verbessertes Routing (`sendmessage2D`):

  adaptiver Routing-Algorithmus
  - Auslastungen der Lines werden an Router gemeldet
  - Router wählt immer Leitung mit kleinster Auslastung

  Gesamtmodell
  - aktuelle Auslastungen der Lines gehen zum Router



  Router

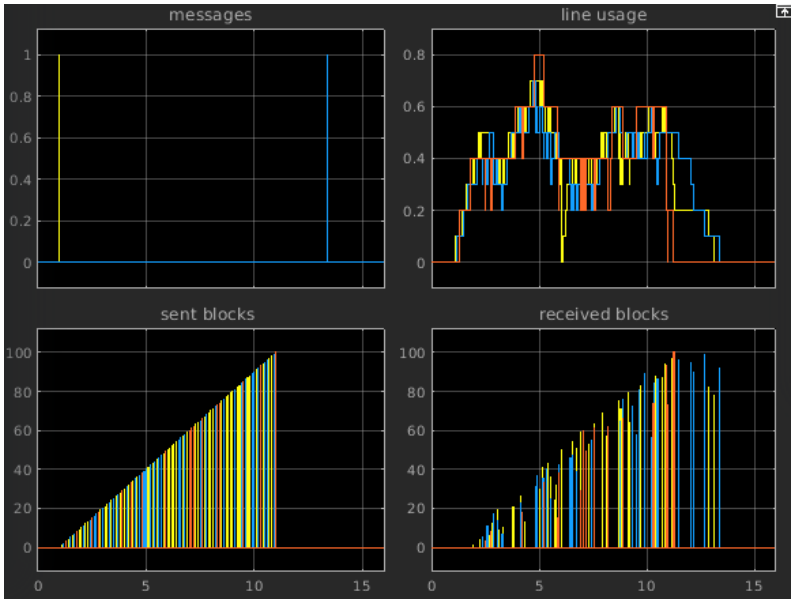


  - Matlab-Funktion liefert Index des kleinsten Werts

```
[~, y] = min(u);
```

- erzeugt Message zur Steuerung des `Output Switch`

Ergebnis

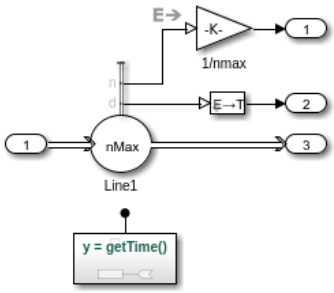


- Übertragungszeit wie vorher, da kein Engpass
- Leitungen gleichmäßig ausgelastet

- Störung in einer Leitung (`sendmessage2E`):

  ab t > 10s Transportzeit auf `Line1` 5x so lang

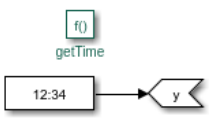
  Implementierung



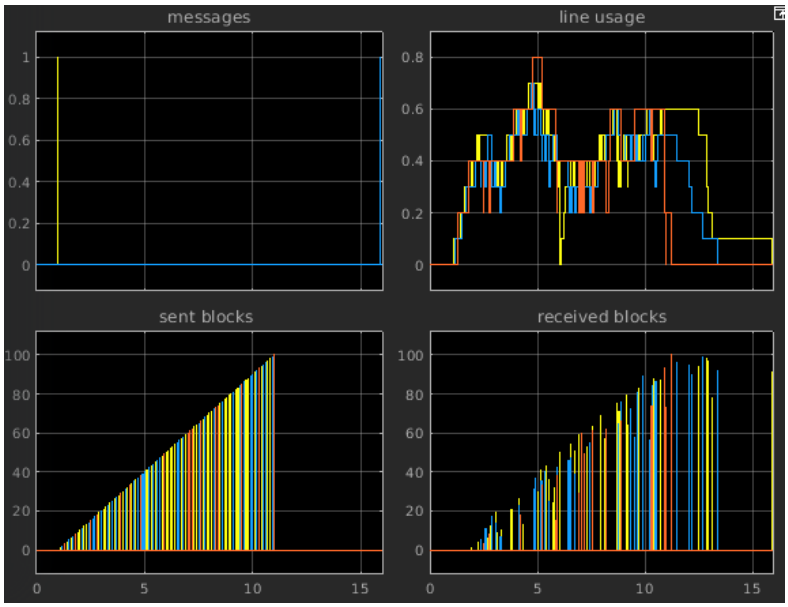
- Berechnung der `Service Time`

```
coder.extrinsic('rand');
dt = 1.0;   % tell coder type of result
if getTime() < 10
  dt = -delay*log(rand());
else
  dt = -5*delay*log(rand());
end
```

- verwendet Simulink-Funktion `getTime()`



Ergebnis



  - Übertragung dauert etwas länger
  - Auslastungen bleiben gleichmäßig

- Aufgaben:

  Aufgabe 17

Logistische Versorgungsketten

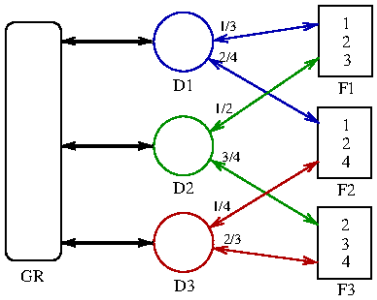
- Beispiel einer Lieferkette:

  vereinfachte Version des Argesim-Benchmarks C14 [9]

  drei Firmen stellen je drei von vier verschiedenen Produkten her

  Distributoren beziehen Produkte von je zwei Fabriken

  Großhändler bestellen Produkte bei den Distributoren



  Fragestellung z. B.
  - Nach welcher Strategie soll ein Distributor Produkte nachbestellen?
  - Wie groß sollten die Lager der Fabriken bzw. Zwischenhändler sein?
  - Wie lange dauert der Vorgang von der Bestellung eines Großhändlers bis zur Auslieferung des Produkts?

  Überblick über veröffentlichte Lösungen des Benchmarks

- Modellierung der Großhändler:

  Bestellungen erfolgen nach einer Startzeit von tW Stunden

  Zeit zwischen Bestellungen gleichverteilt im Intervall [tMin, tMax] Sekunden, ganzzahlig

  Produkttyp gleichverteilt 1 .. nProd

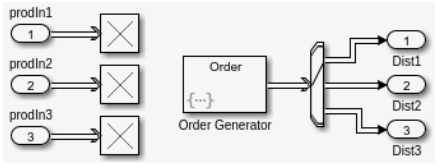
  Distributor gleichverteilt 1 .. nDist

  Versionen für nDist = 3 bzw nDist = 4

  Eingang der Produkte wird nicht weiter modelliert

- Implementierung des Großhändler-Modells:

  Komponente `Wholesaler` mit 3 Ausgängen zu den Distributoren



  Parameter
  - Initial waitting time [h]: `tW`
  - Minimum interarrival time [s]: `tMin`
  - Maximum interarrival time [s]: `tMax`
  - Number of product types: `nProd`

  Entität `Order`
  - Attribute `Product`, `Distributor`
  - Attribute `NItems`, `Factory`, `Port` für spätere Erweiterungen

  Intergeneration time action

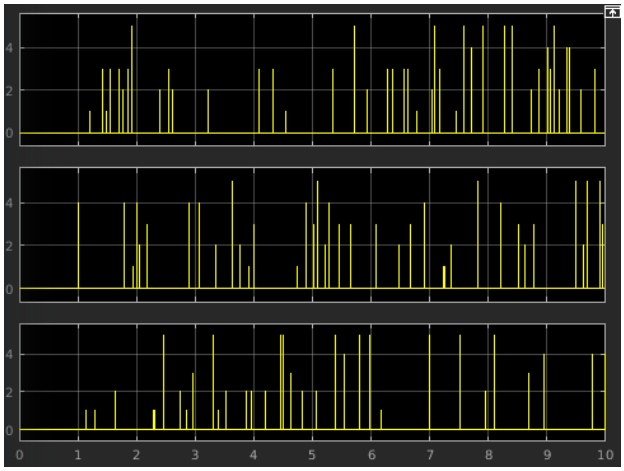
```
coder.extrinsic('randi');
persistent init;
if isempty(init)
  init = true;
end
```

```
% time between orders in hours
res = 1.0;      % define type
if init
  dt = tW;
  init = false;
else
  res = randi(tMax - tMin + 1);
  dt = (res + tMin - 1)/3600;
end
```

Generate action

```
coder.extrinsic('randi');
entity.Product = randi(nProd);
entity.Distributor = randi(3);
```

Ergebnis von `testWholesalers`



- Modellierung einer Fabrik:

  Fabrik produziert rund um die Uhr
  - jeweils nur einige Produkte gemäß Vektor-Parameter `prodTypes` (z.B. [1,2,4])
  - Produktionszeit exponentiell mit `tMean`

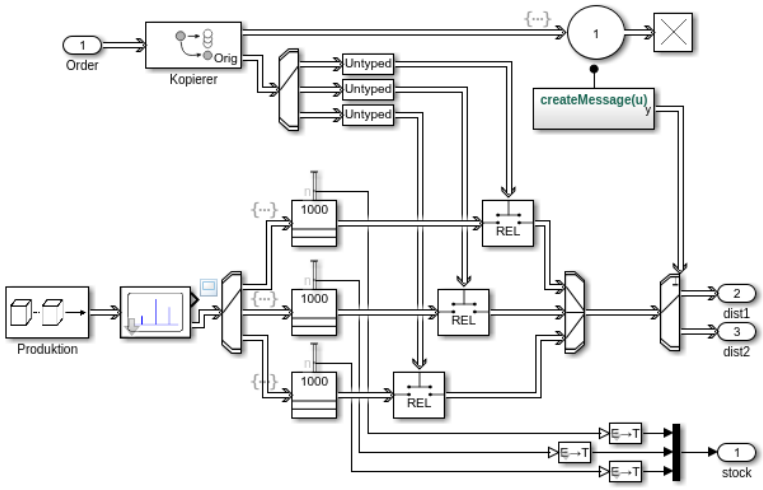
  Fabrik beliefert jeweils zwei Distributoren

  am Eingang kommen Bestellungen mit Produkt-Typ, Distributor und Anzahl
  - genügend Produkte vorhanden → Artikel an Distributor liefern
  - sonst 24 h später erneut probieren

- Entwicklung einer Grundversion:

  Beschränkungen
  - drei feste Produkttypen 1/2/3
  - zwei feste Distributoren 1/2
  - Anzahl = 1 fest
  - keine Prüfung auf Bestand

  Implementierung



Produktion

- jeweils ein Produkt in exponentiell verteilten Zeitabständen
- Produkt-Typ 1/2/3 gleichverteilt

Produkte, nach Typ sortiert, in Speicherbausteinen (`Entity Store`) gesammelt

- Queue statt Store wäre hier genauso gut

Bestellungen werden einmal kopiert

Bestellung eines Typs wird in anonyme Entität (**Message**) verwandelt

- dazu `Entity Generator` mit `Generation method` = `Event-based`
- Message schaltet `Release Gate` frei
- `Release Gate` lässt genau eine Entität (Produkt) durch

aus Kopie der Bestellung wird Distributor abgelesen (in Server)

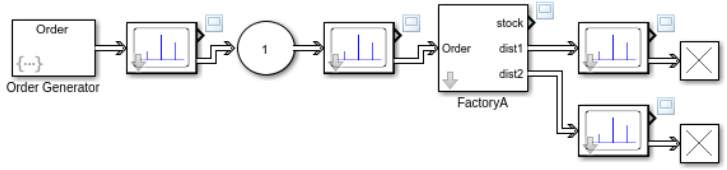
- als Message an `Output Switch` geschickt
- steuert Ausgangs-Port der `Factory`

Vorsicht **Nullserver** (Server mit `tS = 0`)

- werden eingesetzt wegen `Entry/Exit Action`
- können problematisch sein (vgl. [10])
- speichern bei blockiertem Ausgang eine (ggf. mehrere) Entities
- unkritisch, wenn Ausgang nie blockiert ist

- Test der Grundversion:

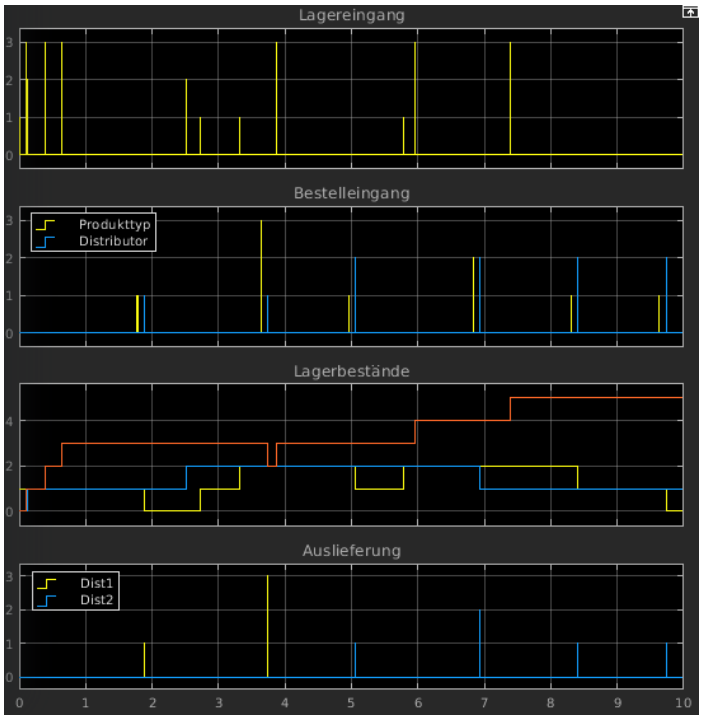
  Modell `testFactory1`



  Trick

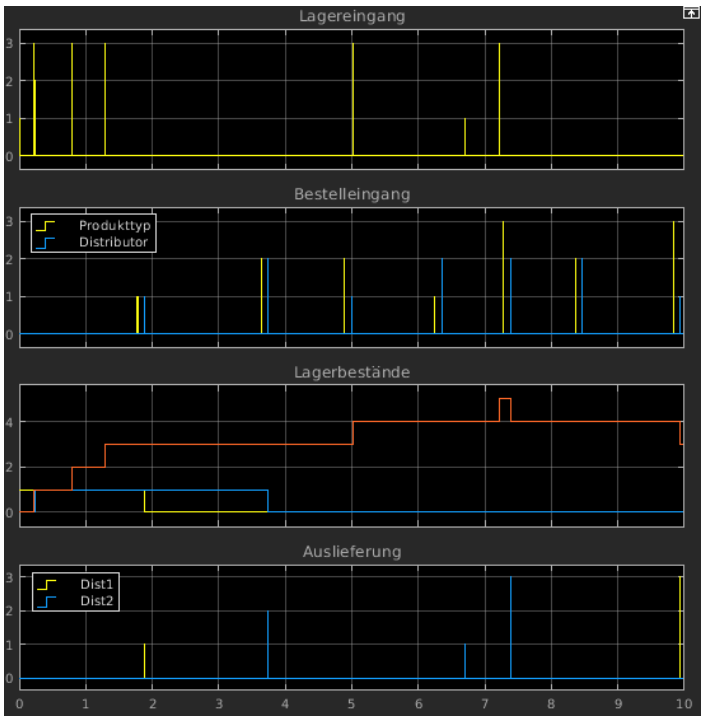
  - Server mit kurzer Verzögerung zwischen zwei Displays
  - → verschiedene Attribute der Entities nebeneinander angezeigt

  Ergebnis bei `tMean = 2000`



- ok: Lagereingang führt zur Vergrößerung des Bestands
- ok: Bestelleingang führt zur Bestandsverringerung und Auslieferung
- Bestand reicht hier immer!

Ergebnis bei `tMean = 4000`



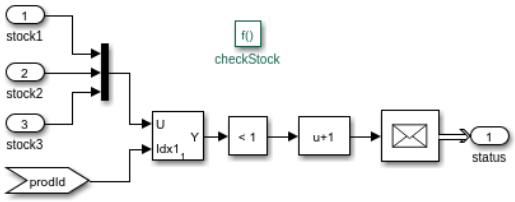
- mehrere Bestellungen ohne Bestand
- bei t = 6.2 Bestellung von Produkt-Typ 1
- `Release Gate` bleibt offen
- bei t = 6.7 passender Wareneingang → wird sofort ausgeliefert

- 1. Erweiterung:

  vor Auslieferung Prüfung auf Bestand

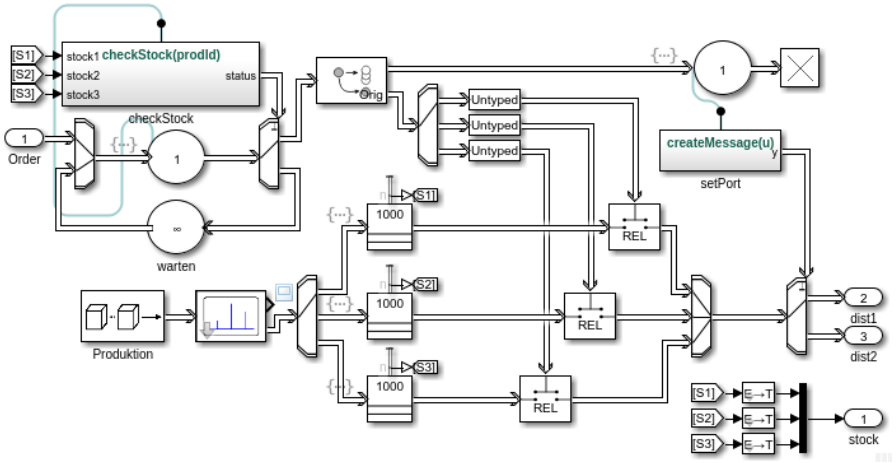
  - ggf. `tD` Stunden später wieder versuchen

  Simulink-Funktion `checkStock(prodId)`



- prüft, ob `stock(prodId) < 1`
- schickt entsprechende Message zum Kontrolleingang des `Output Switches`

Implementierung



- Bestand reicht nicht → Bestellung kommt in Warteschleife

Hinweis

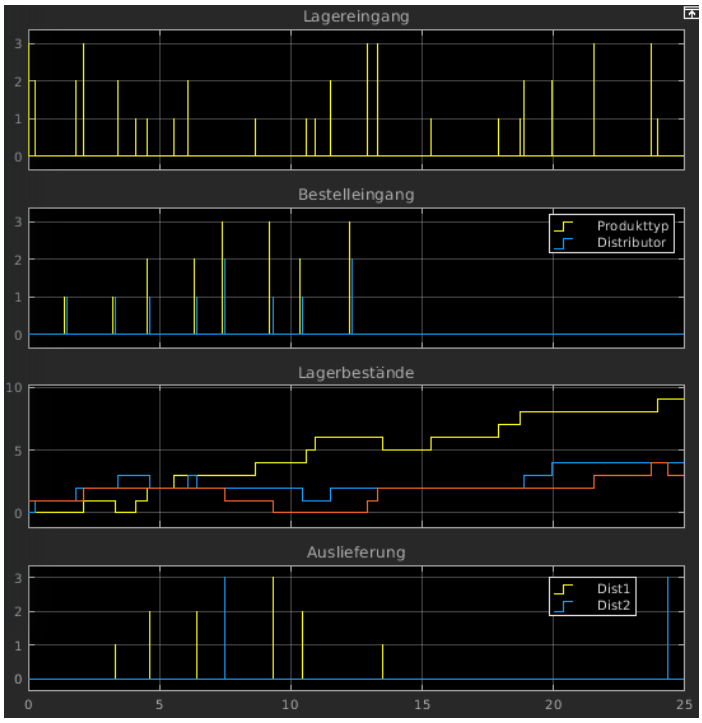
- Bestandsausgänge mit Multiplexer zusammenfassen → Modell deutlich übersichtlicher
- aber: geht nicht, Fehlermeldung in SimEvents

- Test der Erweiterung:

  Bestellungen zeitlich begrenzt

  Wartezeit `tD = 12`

  Ergebnis



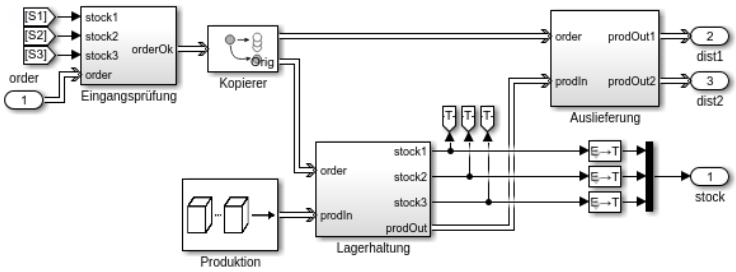
- Bestellung bei t = 1.4 wird ausgeführt um t = 13.4
- Bestellung bei t = 12.3 wird ausgeführt um t = 24.3

- 2. Erweiterung:

  beliebige Produkttypen und Distributoren als Parameter, z. B.

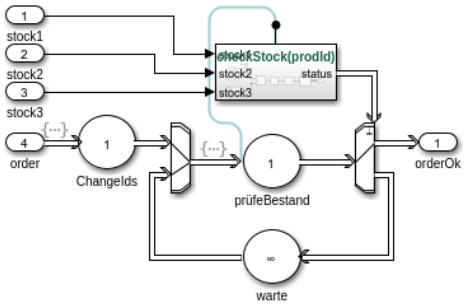
  - `prodTypes = [1,2,4]`
  - `dists = [3,2]`

  zunächst aufräumen durch Subsysteme



  Trick: intern immer Produkt-Typen 1/2/3

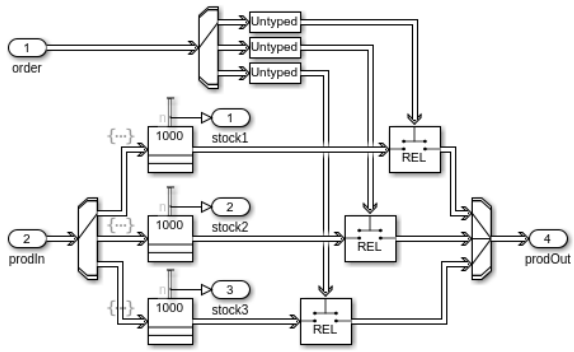
  Produkt-Typ der Bestellung sofort auf 1/2/3 umsetzen



  - dazu `Entry action` in Nullserver `ChangeIds`

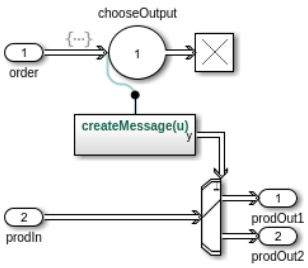
```
idx = find(prodTypes == entity.Product);
entity.Product = idx(1);  % idx could be a vector
```

  Lagerhaltung bleibt i. W. unverändert



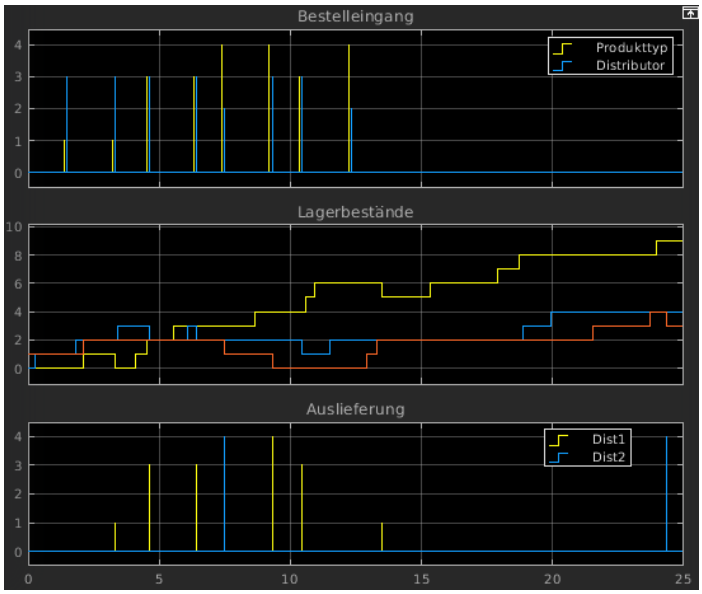
  - Produkt-Typ I bei Erzeugung 1/2/3 (zur einfachen Speicher-Zuordnung)
  - bei Eintritt in Speicher auf `prodTypes(I)` umgesetzt

  Wahl des Ausgangs zum Distributor in Nullserver `chooseOutput`



  - wie bei Produkt-Typ durch Umkehrung von `dists`

  Ergebnis ok



- Endversion `Factory`:

  Bestellungen haben Anzahl eines Produkts als Attribut `NItems`

  `Order Generator`
  - würfelt Werte für `NItems` von 1 bis `maxItems` (Masken-Parameter)

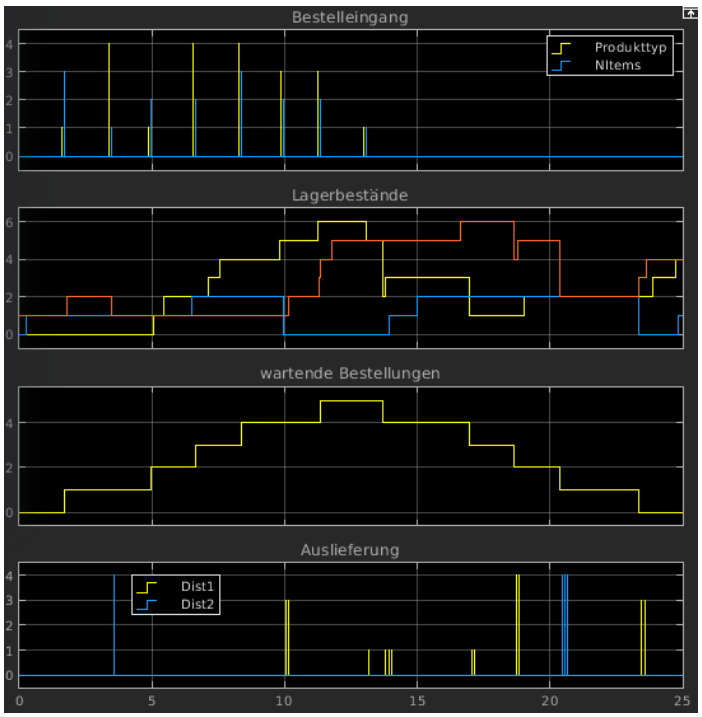
  Eingangsprüfung
  - keine Teillieferungen!
  - `prüfeBestand` ruft `checkStock(entity.Product, entity.NItems)` auf
  - Simulink-Funktion vergleicht `stock` und `entity.NItems`

  Kopierer schickt `NItems` Kopien zur Lagerhaltung

  Anzeige der wiederholten Produkte mit `DisplayIdM`
  - hat am Eingang Queue/Server mit kleiner Verzögerung

  zusätzlicher Ausgang `nWaiting` für Statistik-Zwecke

  Ergebnis ok



- Modell eines Distributors:

  Grundfunktion
  - bekommt Bestellungen von den Großhändlern

- liefert ggf. Produkte an die Großhändler
- bestellt Produkte bei jeweils 2 Firmen
- eigene Lagerhaltung für alle insgesamt 4 Produkte

Anschlüsse

- Eingang für Bestellung von Großhändlern
- Ausgang für Produkte zu den Großhändlern
- 2 Eingänge für Produkte von Fabriken
- 2 Ausgänge für Bestellungen an die Fabriken
- Ausgang des Lagerbestands

Parameter

- Distributor-Id: `distId`
- Liste, welche Produkte bei welcher Fabrik bestellt werden: `factory` (z. B. [1,2,1,2])
- Parameter `tI`, `nInit`, `nDaily` für die Strategie (s.u.)

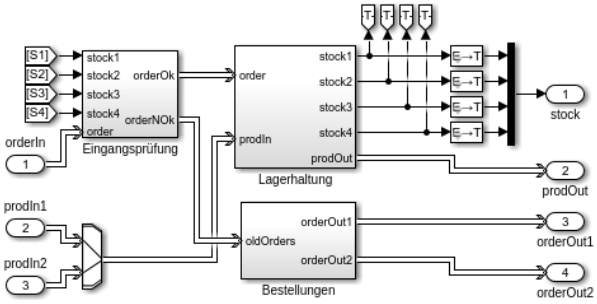
Strategie für Fabrikbestellungen je nach Lagerstand und Auftragslage

einfache Strategie (`DistributorA`)

- Lagerfüllphase: nach Startzeit `tI` je `nInit` Produkte jeder Sorte bestellen
- alle 24 h später je `nDaily` Stücke von jedem Produkt bestellen
- Bestellung eines Großhändlers konnte nicht ausgeführt werden
- ⇒ wird bei nächster Fabrikbestellung hinzugefügt

- Implementierung `DistributorA`:

Grundstruktur



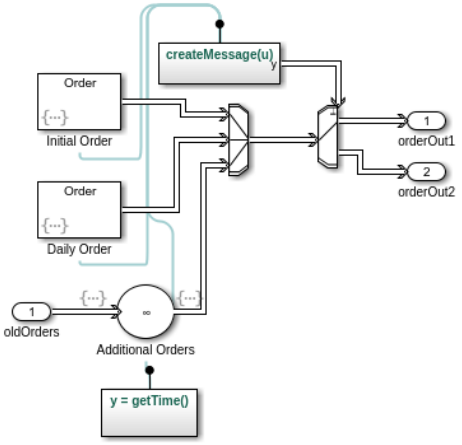
Eingangsprüfung

- wie bei `Factory`, aber ohne Warteschleife
- stornierte Bestellung wird zu Fabrikbestellung

Lagerhaltung

- wie bei `Factory`, aber alle 4 Produkte, keine Index-Spielereien

Bestellungen



`Initial Order`

- erzeugt zur Zeit `tI` Bestellungen für je `nInit` Produkte jeden Typs

- setzt Attribut `Factory` gemäß Parameter `factory(entity.Product)`
- ruft `createMessage(port)` auf → Weiterleitung zur richtigen Fabrik

`Daily Order` analog, aber täglich ab `tI + 24`

`Additional Orders`

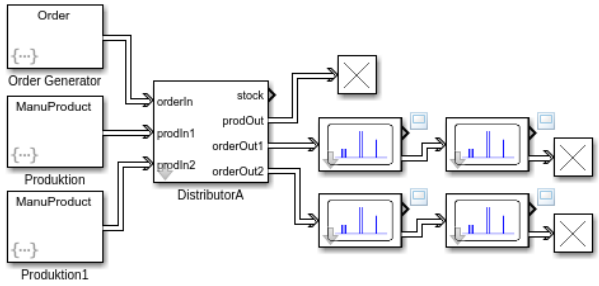
- warten bis Mitternacht

```
dt = 24 - rem(getTime(), 24);
```

- `Entry action` setzt `Factory`
- `Exit action` wählt Ausgang über `createMessage(port)`

- Test der Verteilung von Bestellungen zur Fabrik:

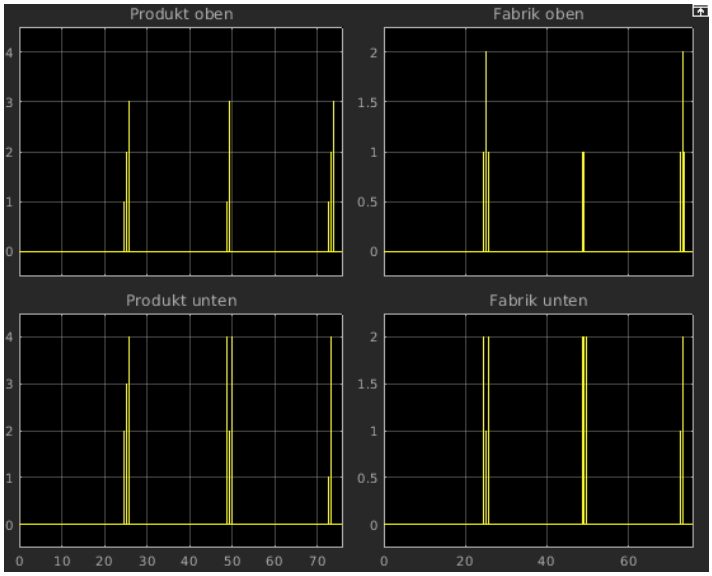
`testDistributor1A`



Erwartung (bei `factory = [1,2,1,2]`)

- oben nur Fabrik 1 und Produkte 1/3
- unten nur Fabrik 2 und Produkte 2/4

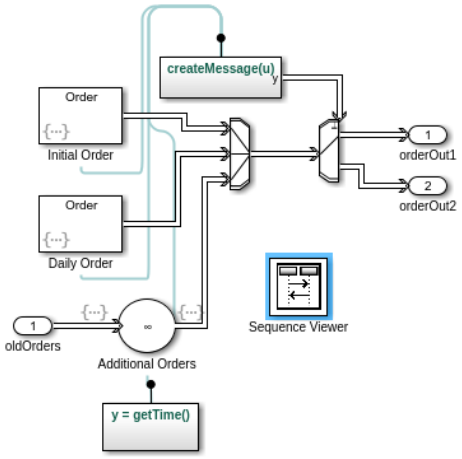
Ergebnis fehlerhaft



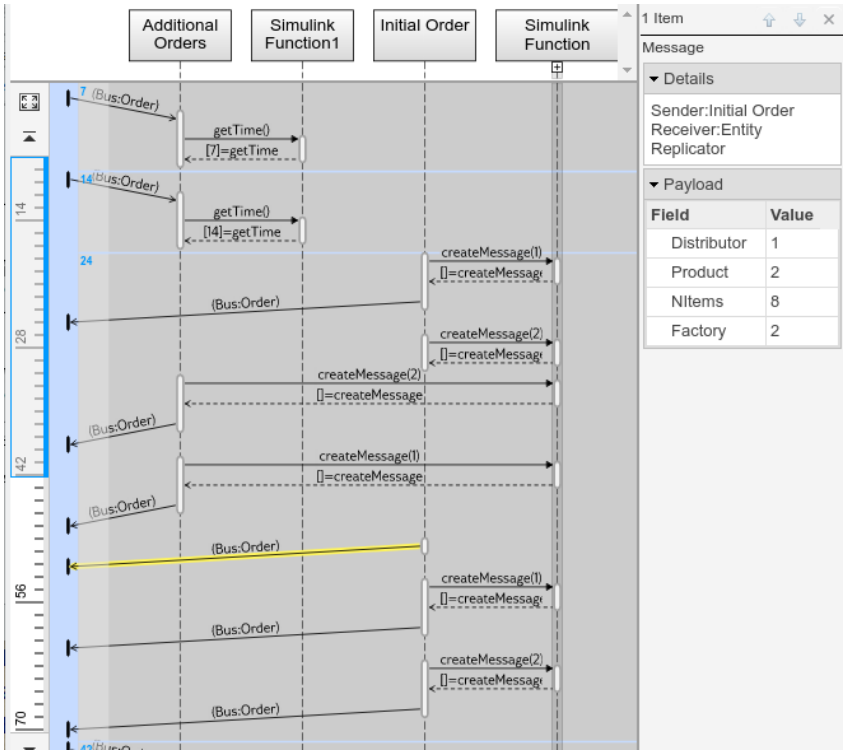
Vermutung

- Problem in Bestellungen
- Timing der `createMessage`-Aufrufe von zwei Blöcken aus durcheinander

`Sequence Viewer` in `Bestellungen` zum Debuggen



- zeigt Nachrichten zwischen Blöcken
- Anklicken einer Event-Linie zeigt Entity und ihre Attribute



Analyse

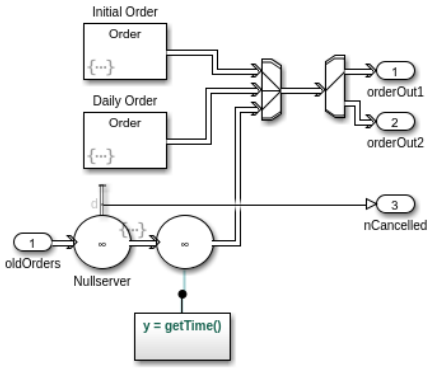
- zwei einlaufende `Additional Orders` (t = 7, t = 14)
- 1. `Initial Order` mit `createMessage(1)` und auslaufender Entity
- 2. `Initial Order` startet mit `createMessage(2)`
- `Additional Orders` drängelt sich dazwischen, am Ende `createMessage(1)`
- Entity der 2. `Initial Order` kommt endlich, jetzt auf falschem Ausgang (1)

Auflösen von Gleichzeitigkeit über mehrere Blöcke in TBM generell ein Problem

- Lösung durch anderes Routing-Schema:

  Idee: verwende Attribut statt Control-Port zum Routing

  - Entity `Order` hat weiteres Attribut `Port`
  - wird von `Entry action` des Servers gesetzt
  - Auswahl des Ports im Switch über `Port`-Attribut statt über Simulink-Funktion



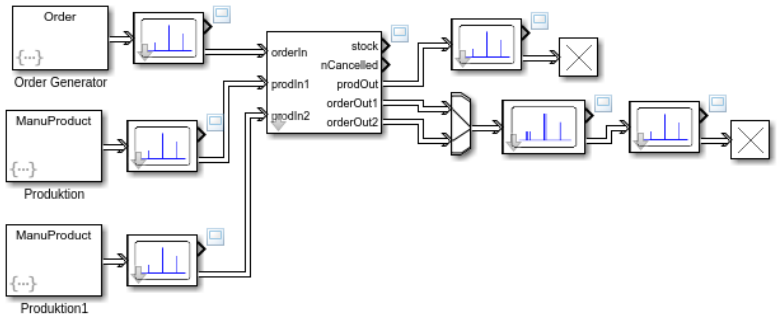
Ergebnis: klappt!

zusätzlicher Ausgang `nCancelled` für Statistik-Zwecke

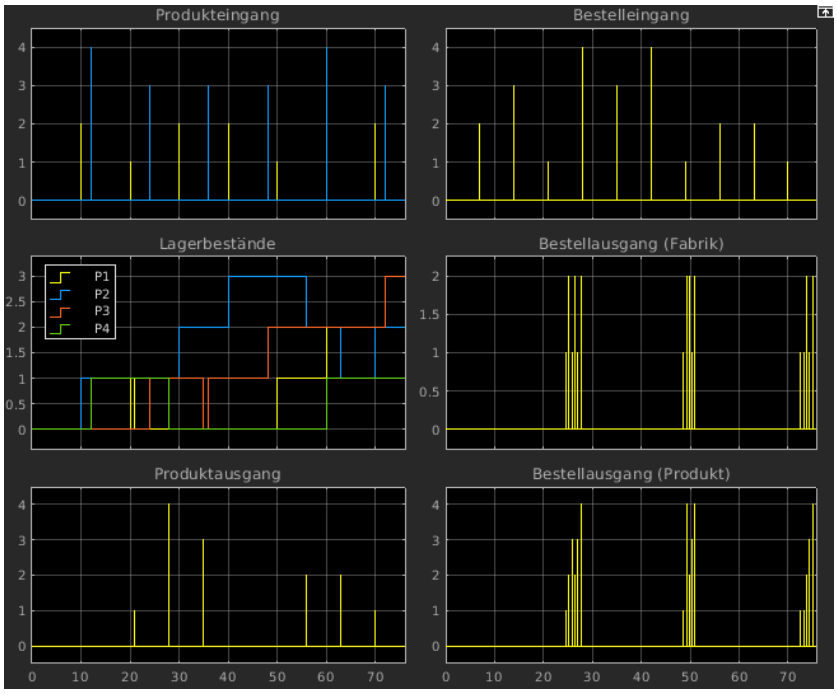
- Zahl der stornierten Bestellungen
- soll direkt bei Eingang (und Abweisung) steigen
- Problem: Server zeigt nur Gesamtzahl der *ausgehenden* Entities, nicht der *eingehenden*
- Trick: Nullserver davor

- Verifikation von `DistributorA`:

  Überprüfung des Gesamtverhaltens mit `testDistributor2`



Ergebnis



manuell Entitäten-Tabelle erstellen

- dazu zunächst Inputs (PIn, OIn) aus Osci-Daten ablesen
- dann zeitlich vorgegebene OOuts eintragen
- danach alle weiteren Größen per Hand ermitteln
- Lagerwerte immer *nach* dem Input/Output

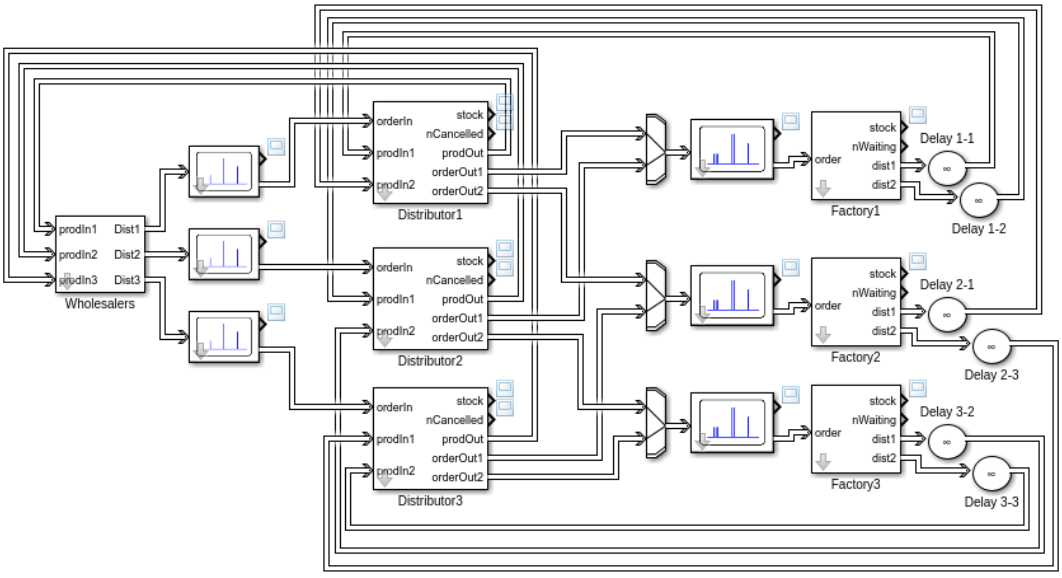
```
 t PIn OIn   Lager    POut  OOut(P)  OOut(F) Bemerkung
-------------------------------------------------------------------
 7      2   0/0/0/0                         delayed bis 24, E1
10  2      0/1/0/0
12  4      0/1/0/1
14      3   0/1/0/1                         delayed bis 24, E2
20  1      1/1/0/1
21      1   0/1/0/1    1
24  3      0/1/1/1
                          1/2/3/4  1/2/1/2
                            2/3      2/1        E1/E2
28      4   0/1/1/0    4
30  2      0/2/1/0
35      3   0/2/0/0    3
36  3      0/2/1/0
40  2      0/3/1/0
42      4   0/3/1/0                         delayed bis 48, E3
48  3      0/3/2/0
                          1/2/3/4  1/2/1/2
                            4        2          E3
49      1   0/3/2/0                         delayed bis 72, E4
50  1      1/3/2/0
56      2   1/2/2/0    2
60  4      1/2/2/1
    1      2/2/2/1
63      2   2/1/2/1    2
70  2      2/2/2/1
        1   1/2/2/1    1
72  3      1/2/3/1
                          1/2/3/4  1/2/1/2
                            1        1          E4
```

Vergleich mit Osci → ok

- Gesamtmodell `supplychain1`:

  Aufbau gemäß obigem Plan

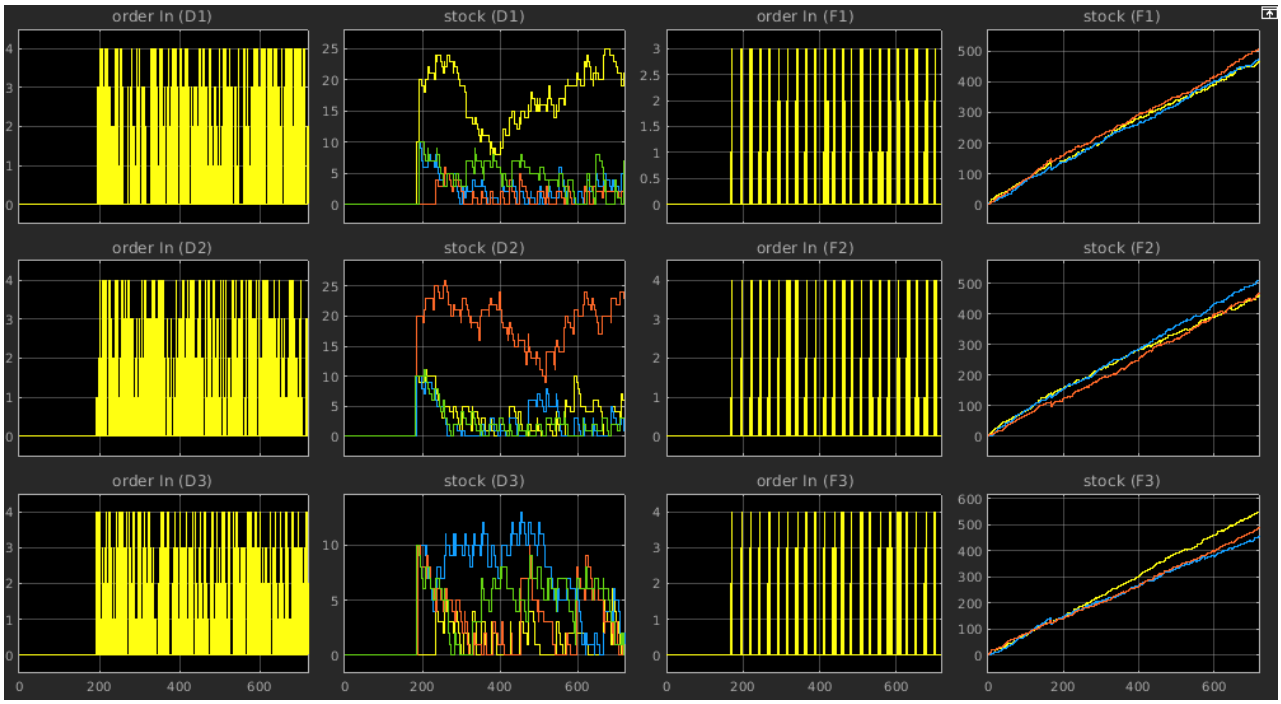
  - zusätzlich Lieferzeiten Fabrik → Distributor berücksichtigen



  Parameter nach [9]

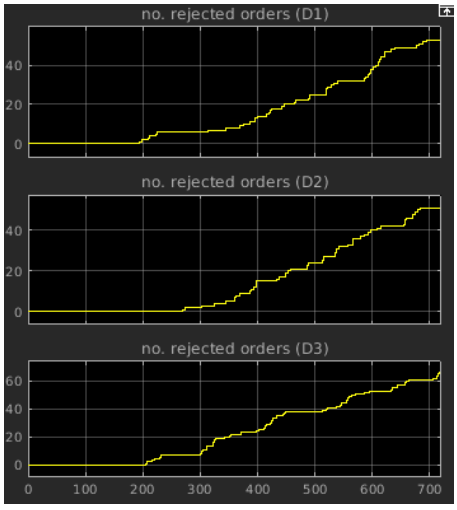
  - Fabrikproduktion reduziert von `tMean` = 600 s auf `tMean` = 1500 s
  - Originalwert viel zu hoch
  - SimEvents stoppt sonst wegen zu vieler gleichzeitiger Events pro Block!

  Ergebnisse



Analyse

- Fabriken haben Überproduktion
- Distributoren bestellen zu wenig nach
- viele stornierte Bestellungen

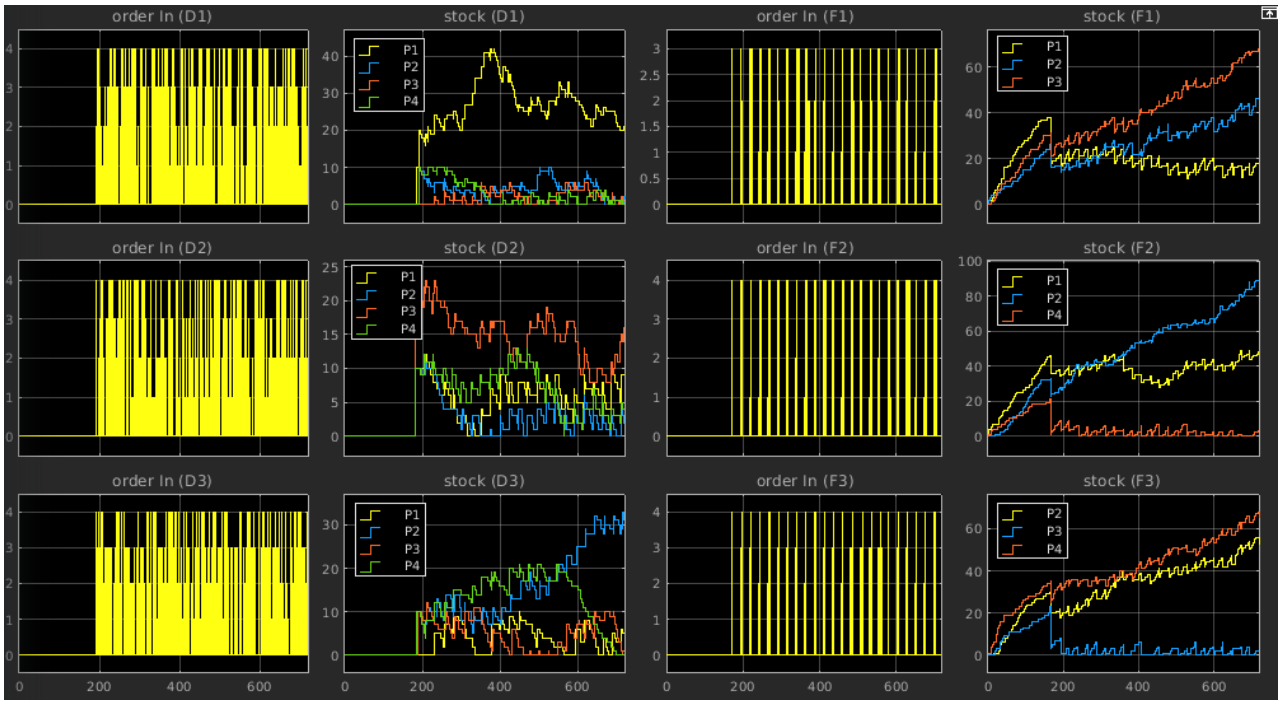


- Verbesserungen der Parameter:

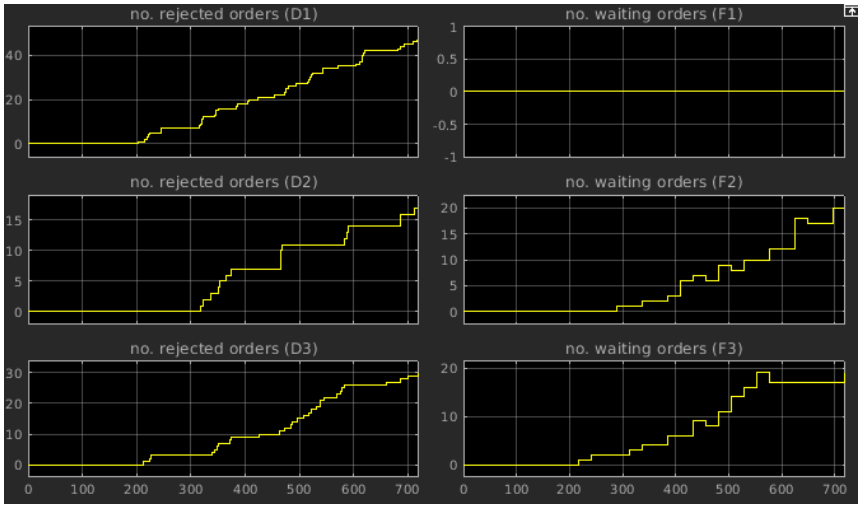
  Abschätzung der Nachfrage

  - mittlere Zeit zwischen Bestellungen: 2100 s = 7/12 h
  - bei gleichmäßiger Verteilung auf Fabriken: `tMean` = 7/4 h = 6300 s
  - bei 4 Produkten und 3 Distributoren: alle 7 h ein Produkt bei einem Distributor
  - bei täglicher Nachbestellung: `nDaily` = 24/7 = 3 oder 4?
  - zum Testen: setze `nDaily = 2/3/4` für D1/D2/D3

  Ergebnisse



- Produktion von Produkten 4/3 in Fabriken F2/F3 nicht ausreichend
- → Distributoren bekommen nicht genug geliefert
- → Zahl der stornierten Bestellungen hoch



- Erweiterungsideen:

  andere Distributorstrategien
  - jeweils soviel, wie von den Großhändlern in den letzten 24 h bestellt wurde
  - Auswahl einer Fabrik nach bisherigen Verfügbarkeiten/Lieferzeiten

  viele Produkt-Typen, gemeinsamer Speicher für alle Typen

  Berechnung von Kosten für Lagerhaltung und Transport

  Großhändler-Modell verfeinern

  Teillieferungen ermöglichen

- Aufgaben:

  Aufgabe 18

  Aufgabe 19

Modell-Verifizierung und -Validierung

- Grundlegende Vorgehensweisen
- Bestimmen der Zufallsverteilungen von Eingangsgrößen
- Statistische Analyse von Simulationsergebnissen

Grundlegende Vorgehensweisen

- Drei Stufen der Nützlichkeit:

  Verifikation: Modell wird korrekt beschrieben

  Validierung: das korrekte Modell wird beschrieben

  Glaubwürdigkeit (Credibility): Auftraggeber akzeptiert Modell zur Entscheidungsfindung

- Verifizierung:

  beschreibt das Programm das (theoretische) Modell?

  viele Grundtechniken wie beim Programmieren
  - Testen von Submodellen/Blöcken
  - Programmcode mit Kollegen durchgehen
  - sehr kurze Läufe manuell durchspielen
  - Läufe mit sehr verschiedenen Parametern → Ergebnisse plausibel?
  - spezielle Parameterwerte mit theoretisch bekannten Ergebnissen vergleichen

- Konkrete Beispiele bei bisherigen Modellen:

  Überprüfen von Beispiel-Ergebnissen
  - fast immer

  Testen von Submodellen
  - RS-Flipflop
  - Straße
  - Conveyor, Oven
  - Wholesaler, Factory, Distributor

  Läufe manuell durchspielen
  - Entitäten-Tabelle (`singleserver1B`, `testConveyor`, `testDistributor2`)
  - Event-Tabelle (D/D/1-Beispiel)

  Vergleich mit theoretisch bekannten Ergebnissen
  - Ergebnisse der Warteschlangentheorie (`singleserver3B`)

- Validierung:

  repräsentiert das Modell das betrachtete System im Rahmen der Fragestellung hinreichend genau?

  notwendig, um aus am Modell gewonnenen Ergebnissen sinnvoll Entscheidungen über das System abzuleiten

  hängt stark von der Fragestellung ab

  ändert sich mit der Weiterentwicklung des realen Systems

  wird häufig zu wenig betrieben → Schlüsse aus Modellverhalten fragwürdig

  niemals vollständig möglich

- Vier Ebenen der Modellvalidierung [11]:

  Verhaltensgültigkeit
  - dynamisches Verhalten des Modells entspricht realem System
  - für relevante Parameterwerte
  - im Rahmen der gewünschten Genauigkeit

  Strukturgültigkeit
  - relevante kausale Beziehungen und Strukturen abgebildet
  - dazu gehören auch die Zustandsgrößen
  - bei Blackbox-Modell irrelevant

empirische Gültigkeit
  - numerische Ergebnisse entsprechen dem System oder sind plausibel

Anwendungsgültigkeit
  - Modell berücksichtigt relevante Systemparameter
  - Modell berechnet die interessierenden Größen

- Detaillierungsgrad eines Modells:

  konkretisiert "hinreichend genau"

  muss der Fragestellung entsprechen

  abhängig von den vorhandenen Systemdaten

  häufig durch Zeit-/Geldknappheit beschränkt

  Faustregel: grob anfangen und nach Bedarf verfeinern

- Validierungs-Techniken:

  Sammeln von Systemdaten
  - Eingangsgrößen/-verteilungen, Parameter, Ergebnisse

  Dokumentation aller Annahmen
  - Projektziele, Vereinfachungen, Beschränkungen

  Ergebnisse von Teilkomponenten und der Gesamtsimulation überprüfen
  - statistische Verfahren nutzen
  - unterschiedliche Daten zum Kalibrieren (Parameterschätzung) und Validieren
  - Turing-Tests (Blindvergleich von Modell- und Systemdaten durch Experten)
  - Sensitivitätsanalysen
  - Animationen

Bestimmen der Zufallsverteilungen von Eingangsgrößen

- Vorgehen:

  zuerst natürlich Daten erheben (möglichst viele)

  Daten direkt benutzen
  - häufig nicht genügend Daten für viele Simulations-Läufe
  - gelegentlich hilfreich zur Validierung

  empirische Verteilung konstruieren
  - einfach bei diskreten Größen mit wenigen Werten
  - bei wenig Daten meist irregulär (z. B. große Abstände zwischen Daten)
  - Probleme mit seltenen Ereignissen (etwa großen Servicezeiten)

  theoretische Verteilungsfunktion anpassen
  - Auswahl aufgrund von Theorie (z.B. Normal- oder Exponentialverteilung)
  - Auswahl aufgrund von Ähnlichkeit (z.B. mit Q-Q-Plots)
  - Parameter aus Daten abschätzen
  - durch statistische Analysen absichern (Konfidenzintervalle, u. U. Tests)

- Bestimmen einer empirischen Verteilung F:

  gegeben seien n Datenpunkte x_i, aufsteigend sortiert
  - Beispiel: `x = [4.8933 4.9359 4.9521 4.9599 4.9795 4.9837 4.9858 4.9968 5.0046 5.0137 5.0202 5.0500]'`

  ganz primitiv: F springt bei jedem Wert um 1/n
  - formal

$$F_0(x) = \frac{1}{n} |\{i \in [1,n] \mid x_i \leq x\}|$$

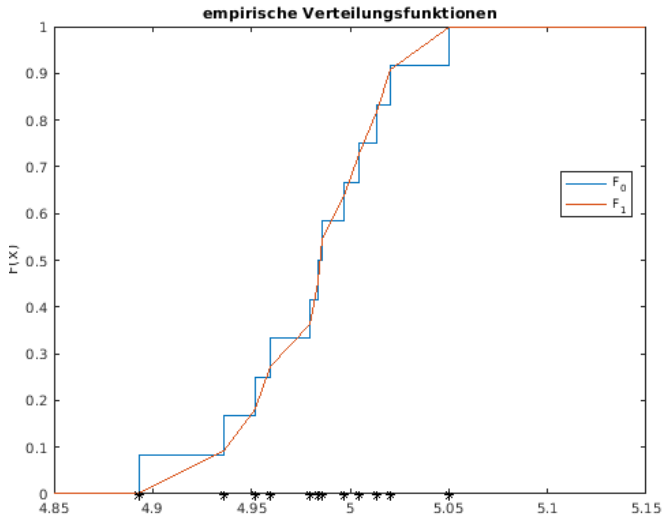
  - Problem: reproduziert nur die vorhandenen Datenwerte

  besser: F steigt in jedem Intervall linear um 1/(n-1)
  - formal

$$F_1(x) = \begin{cases} 0 & | \quad x < x_1 \\ \frac{i-1}{n-1} + \frac{x - x_i}{(n-1)(x_{i+1} - x_i)} & | \quad x_i \leq x < x_{i+1}, \quad i = 1, 2, \ldots, n-1 \\ 1 & | \quad x_n \leq x \end{cases}$$

  - Problem: keine Werte außerhalb des Datenintervalls

  im Beispiel



x

- Erzeugen von gemäß F_1 verteilten Zufallszahlen:

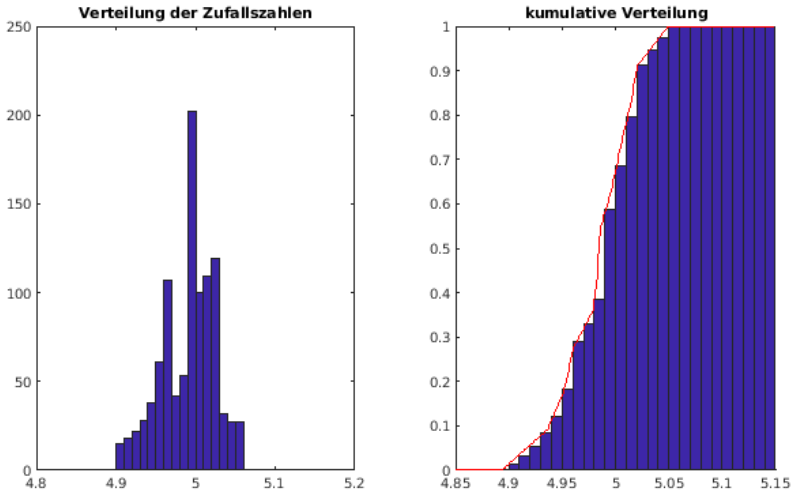
  empirische Verteilung erzeugen

  ```
  n = length(x);
  f = 0:1/(n-1):1;
  pd = makedist('PiecewiseLinear', 'x', x, 'Fx', f);
  ```

  damit 1000 Zufallszahlen

  ```
  xr = random(pd, 1000, 1);
  ```

  Ergebnis



- Bestimmen der Verteilungsfamilie:

  theoretische Überlegungen

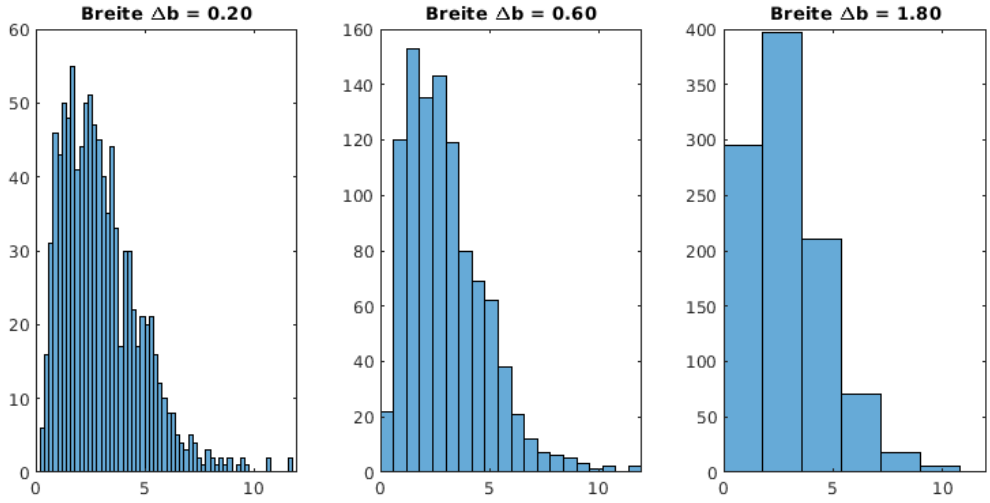
  - viele unabhängige Einflüsse → Normalverteilung $N(\mu, \sigma^2)$
  - Zeitabstände ohne Gedächtnis → Exponentialverteilung $Ex(\lambda)$
  - Summe mehrerer $Ex(\lambda)$-Größen → Erlangverteilung $Erl(n, \lambda)$

  Bereichseinschränkungen

  - positiv → $N(\mu, \sigma^2)$ eher schwierig
  - festes Intervall → z.B. Beta-, Dreiecks-, Gleichverteilung

  Histogramm erstellen

  - sollte näherungsweise der Dichtefunktion entsprechen
  - wichtig: Wahl der Intervallbreite Δb
  - Δb zu groß → Diagramm zu blockartig, wenig Details
  - Δb zu klein → zu große Schwankungen der einzelnen Werte



Boxplot hilfreich für Schiefe (Unsymmetrie der Verteilung)

- Abschätzen der Parameter:

  vermutete Verteilung habe (mehrere) Parameter θ

  gesucht: gute Schätzer für θ

  wichtiges Grundverfahren: Maximum Likelihood Estimator (MLE)
  - Idee: wähle θ, das (gemeinsame) Dichtefunktion zu den Daten maximiert
  - häufig grundsätzlich einfach (Logarithmieren, Ableiten, Null setzen)
  - manchmal Gleichung nur numerisch auflösbar

  für wichtige Verteilungen MLEs (gelegentlich auch bessere Schätzer) bekannt

  einige Beispiele
  - Normalverteilung N(μ, σ^2)

    $$\hat{\mu} = \frac{1}{n}\sum_i x_i =: \overline{x}$$
    $$\hat{\sigma^2} = \frac{1}{n-1}\sum_i (x_i - \overline{x})^2$$
  - Exponentialverteilung Ex(λ)

    $$\hat{\lambda} = \frac{1}{\overline{x}}$$
  - Gammaverteilung γ(r, λ) durch numerisches Lösen von

    $$\frac{\hat{r}}{\hat{\lambda}} = \overline{x}$$
    $$\psi(\hat{r}) - \ln\hat{\lambda} = \frac{1}{n}\sum_i \ln x_i$$

    mit der Digammafunktion

    $$\psi(x) := \frac{\Gamma'(x)}{\Gamma(x)}$$

- Kontrolle der Ergebnisse:

  heuristisch
  - Histogramme von Dichte- oder Verteilungsfunktion mit erwarteter Kurve
  - Q-Q-Plot

  statistische Tests
  - Chi-Quadrat-Anpassungstest
  - Kolmogorow-Smirnow-Test

Statistische Analyse von Simulationsergebnissen

- Problemstellung:

  Modell mit stochastischen Prozessen
  - → Simulationsergebnisse sind zufällig
  - → Ergebnisse müssen statistisch analysiert werden

  Standardtests oft nicht anwendbar
  - Ergebnisse zeitlich korreliert
  - Verteilungen nicht stationär
  - Nullhypothese "Modelldaten ≙ Systemdaten" ist sicher falsch (bei hinreichender Genauigkeit)

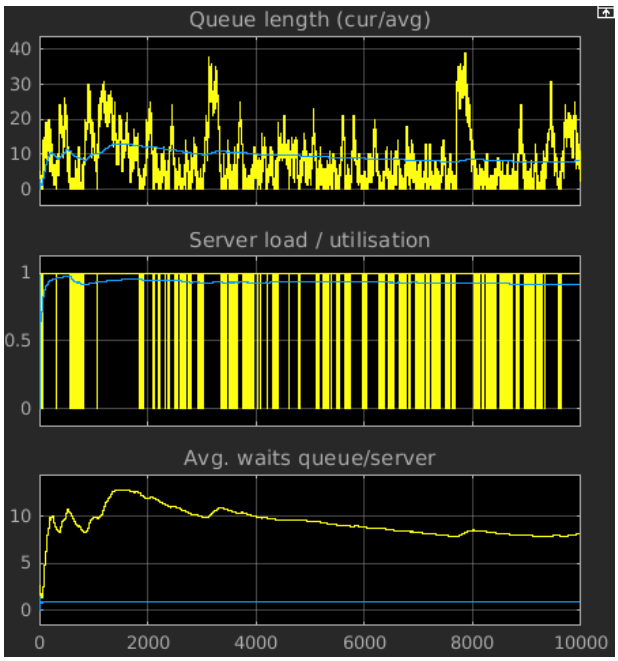
- Standardbeispiel M|G|1-Queuing-System:

  Ankunftszeiten ~ Ex(λ)

  Servicezeiten ~ N(1/μ, σ^2)

  Ausgaben
  - aktuelle Queuelänge nQ(t_i) mit Zeiten t_i
  - mittlere Queuelänge L_Q
  - mittlere Wartezeit in der Queue W_Q
  - mittlere Serverauslastung A

  Beispiellauf mit Parametern λ = 1, μ = 1.2, σ = 0.9, t_{End} = 10000



  Ergebnisse von 6 Beispielläufen mit aufeinanderfolgenden Seeds

Lauf	L_Q	W_Q	A
1	8.0348	8.0814	0.9201
2	9.1228	9.0291	0.9287
3	7.3117	7.3779	0.9129
4	5.6341	5.8311	0.8785
5	10.2697	10.2059	0.9338
6	15.8034	15.5841	0.9428

- Abschätzen der mittleren Queuelänge - naiv (und falsch):

  betrachten Queuelänge X_i zu Zeiten t_i = i (i = 1 .. n)

- aus Modellergebnissen interpoliert (mit `'previous'`)

mit üblichen Standardformeln aus den X_i berechnet

- Mittelwert $L_Q = 8.0380$
- Varianz $S^2 = 61.5559$
- zum Nachrechnen: $t_{End} = 10000$, seed = 4

Problem: L_Q erscheint zu klein (angesichts der Tabelle)

Ursache: Korrelation der Werte innerhalb eines Laufs

genauer: Korrelationskoeffizient ρ jeweils aufeinanderfolgender Werte X_i und X_{i+1} abschätzen

$$\begin{aligned} S^2 &:= \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2 \\ C &:= \frac{1}{n-1}\sum_{i=1}^{n-1}(X_i - \overline{X})(X_{i+1} - \overline{X}) \\ \rho &= \frac{C}{S^2} \end{aligned}$$

Ergebnis

- ρ = 0.9859
- Werte **sehr** stark korreliert
- klar, z.B. Queue voll → aufeinander folgende X_i alle groß

- Ensemblemittelung:

  k komplette Läufe mit verschiedenen Seeds

  - → Daten X_{ij}, i = 1..k, j = 1..n

  daraus Mittelwerte Y_i pro Lauf bestimmen

  $$Y_i = \frac{1}{n}\sum_{j=1}^{n} X_{ij}, \quad i = 1 \ldots k$$

  Gesamtmittel = Mittel über Läufe (Ensemblemittel)

  $$\overline{Y} = \frac{1}{k}\sum_{i=1}^{k} Y_i$$

  Y_i sind i.i.d → Standardverfahren funktionieren

  - Schätzer für Varianz

    $$S^2 = \frac{1}{k-1}\sum_{i=1}^{k}(Y_i - \overline{Y})^2$$

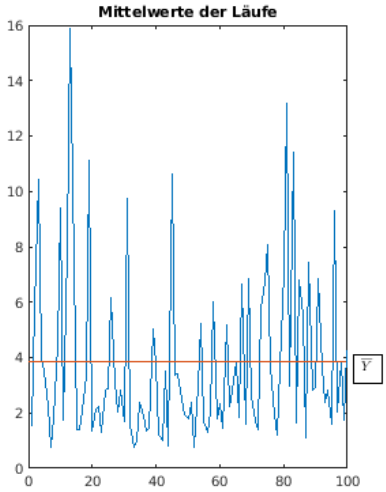
  - Schätzer für (1-α)-Konfidenzintervall

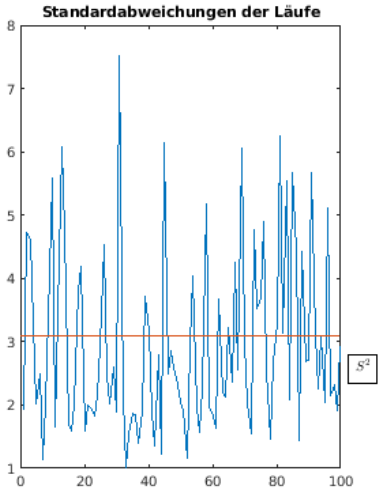
    $$\overline{Y} \pm t_{k-1,1-\alpha/2}\sqrt{\frac{S^2}{k}}$$

- Ensemblemittelung im Beispiel:

  k = 100 Läufe bis $t_{End} = 100$ (statt $t_{End} = 10000$)

  Verteilung der Y_i und der Standardabweichungen pro Lauf





Ergebnisse

- $\overline{Y}$ = 3.8414
- S^2 = 9.5769
- Konfidenzintervall: [3.2274, 4.4554] für 1-α = 95%

Ergebnis seltsam (vgl. Tabelle)

- Mittelwert anscheinend viel zu klein
- Konfidenzintervall enthält keins von 6 Ergebnissen

Ursache

- Läufe zu kurz
- größere Queuelängen erst nach längerer Zeit
- Gleichgewicht (falls vorhanden) noch nicht erreicht

Abhilfe

- "Aufwärmzeit" abwarten, erst danach Werte nehmen
- Problem: Wie lange dauert die?

- Ermitteln der Aufwärmzeit t_A:

  grundsätzlich

  - häufig: Verteilungen für $X(t_i)$ haben Limes für $t \to \infty$ (**Steady State**)
  - gesucht: Laufzeit t, ab der Steady State ungefähr erreicht ist
  - schwierig, da X_i auch im Steady-State stark schwanken

  graphisches Verfahren nach Welch [12]

  1. mache k Läufe der Länge n → Daten X_{ij}, i = 1..k, j = 1..n
     - z.B. k = 10, n groß
  2. mittle Werte gleicher Zeit über die Läufe

     $$\overline{X}_j := \frac{1}{k} \sum_{i=1}^{k} X_{ij}, \quad j = 1 \ldots n$$

     - verringert Varianz um k bei gleichem Erwartungswert
  3. mittle Werte über Zeitfenster der Breite w

     $$\overline{X}_j(w) := \begin{cases} \frac{1}{2w+1} \sum_{s=-w}^{w} \overline{X}_{j+s} & | \quad j = w+1, \ldots, l-w \\ \frac{1}{2j-1} \sum_{s=-(j-1)}^{j-1} \overline{X}_{j+s} & | \quad j = 1, \ldots, w \end{cases}$$

     - glättet hochfrequente Anteile weg

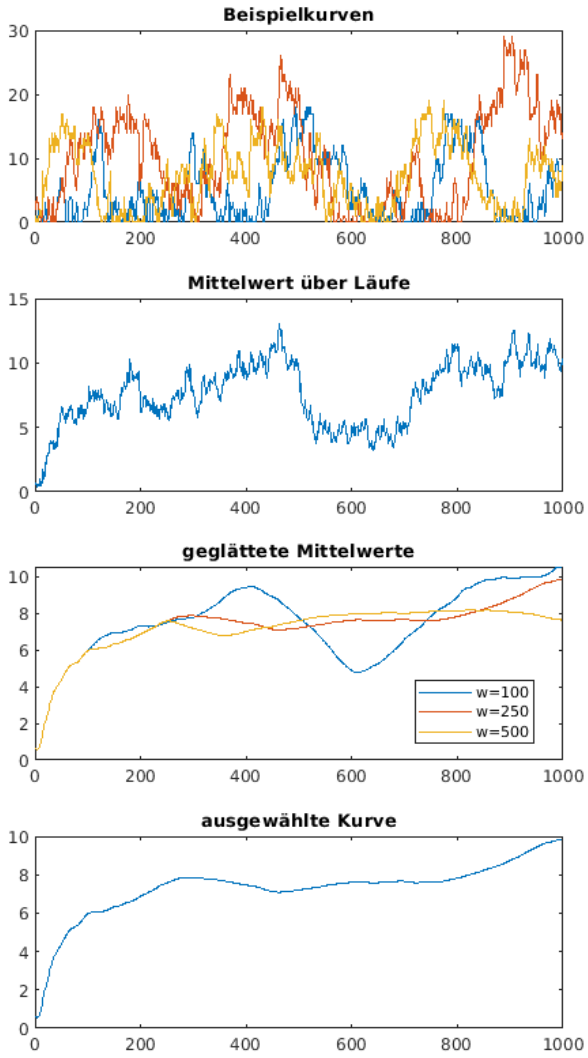
- plotte $\overline{X}_j(w)$ für verschiedene w
- wähle w so klein, dass Kurven "hinreichend glatt"
- im Zweifelsfall k vergrößern

4. plotte $\overline{X}_j(w)$ und wähle t_A, ab dem Grenzwert etwa erreicht

- Vorgehensweise im Beispiel (**Replication/Deletion-Verfahren**):

  Aufwärmzeit t_A nach Welsh ermitteln

  - k = 10 Läufe bis t_{End} = 1000



  - geglättete Kurven mit w = 250 und w = 500 sehr ähnlich
  - Schwankungen bleiben groß
  - Aufwärmbereich grob gut erkennbar
  - wähle t_A = 500 (eher etwas zu groß)

  Achtung Matlab

  - gleitendes Mittel mit `movmean` benutzt am Rand einseitige Intervalle
  - damit wird der (entscheidende) Anstieg am Anfang weggemittelt
  - also: eigene Routine `modmean2`, die korrekte Formel verwendet

  anschließend mit neuen Läufen X_{ij} bestimmen (**Replication**)

  - jeweils Werte vor t_A streichen (**Deletion**)
  - mit den restlichen Werten Mittel Y_i über Läufe bestimmen
  - Y_i mit Standardverfahren auswerten

  Ergebnisse bei k = 10 und t_{End} = 1500

  - $\overline{Y}$ = 10.7314

- S^2 = 56.1750
- Konfidenzintervall: [5.3698, 16.0930]

gut verträglich mit Tabelle

Ergebnisse bei k = 100 und tEnd = 1500

- $\overline{Y}$ = 9.4257
- S^2 = 29.7782
- Konfidenzintervall: [8.3430, 10.5085]
- mehr Daten → deutlich höhere Genauigkeit

- Weitergehende Methoden:

  bisher nur an der Oberfläche gekratzt

  verschiedene Verfahren für endlichen (festen) Zeitrahmen, Steady-State-Systeme oder steady-state-zyklische Systeme

  komplexe Methoden zum Vergleich zweier Systemkonfigurationen

  ausgefeilte Verfahren zur Veringerung der Varianz

  guter Einstieg: [Law, Kap 9-11]

- Aufgaben:

  Aufgabe 20

  Aufgabe 21

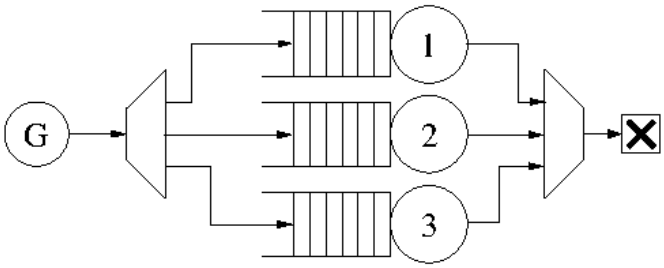
Abstrakte Modellierung ereignisgesteuerter Systeme

- Problemstellung
- Definition von PDEVS
- Arbeiten mit MatlabDEVS

Problemstellung

- Spezifikation eines Queueing-Systems:

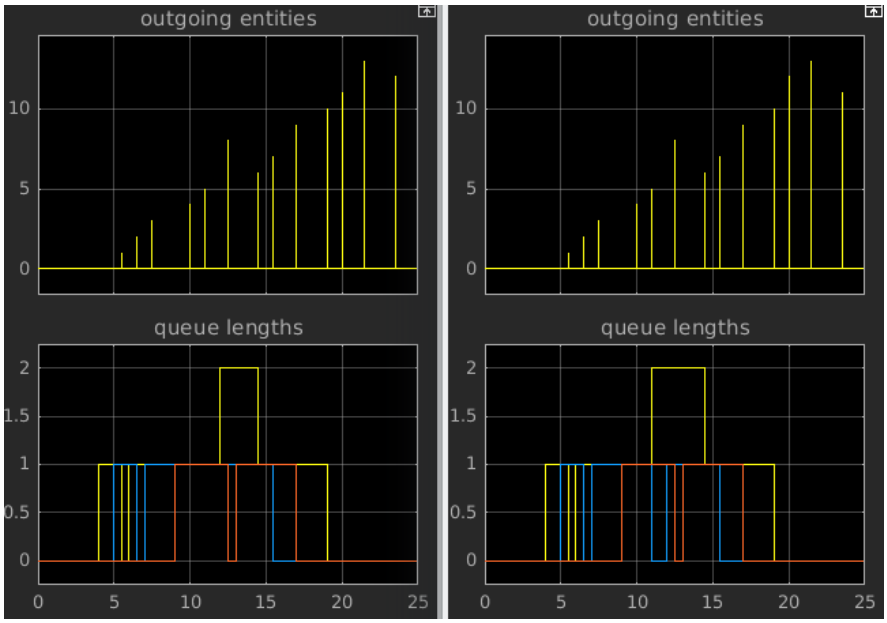
  Beispiel `fifo3`



  - Generator erzeugt $n_E = 13$ Entities mit id's 1,2 .. n_E im Zeitabstand $t_G = 1$
  - ankommende Entity sucht sich die kürzeste Queue (incl. Serverbelegung)
  - bei mehreren kürzesten wird die mit der kleinsten Nummer gewählt
  - Server bearbeiten jeweils eine Entity mit fester Bedienzeit $t_S = 4.5$
  - fertige Entities verlassen das System

  gewünschte Ergebnisse
  - id's der ausgehenden Entities über die Zeit des Verlassens
  - Länge der drei Queues über die Zeit

  2 verschiedene Implementationen (`fifo3A`, `fifo3B`)
  - Ergebnisse



  - welche sind richtig?

- Verhalten per Hand bestimmen:

  wie gehabt mit Entitäten-Tabelle

t	G	Q1	S1	Q2	S2	Q3	S3	raus
1	E1		E1					
2	E2				E2			
3	E3						E3	
4	E4	E4						
5	E5			E5				
5.5		–	E4					E1
6	E6	E6						
6.5				–	E5			E2
7	E7			E7				
7.5							–	E3
8	E8						E8	
9	E9					E9		
10		–	E6					E4
	E10	E10						
11				–	E7			E5
	E11			E11				
12	E12	E12/10						
12.5						–	E9	E8
13	E13					E13		
14.5		E12	E10					E6
15.5				–	E11			E7
17						–	E13	E9
19		–	E12					E10
20					–			E11
21.5							–	E13
23.5			–					E12

bei t = 10 gibt es zwei verschiedene Reihenfolgen

- A: E4 verlässt Server / E6 rückt nach / E10 betritt Queue
- B: E10 betritt Queue / E4 verlässt Server / E6 rückt nach
- ergibt letzlich keinen Unterschied

bei t = 11 analog, aber diesmal mit Konsequenzen!

obiges Bild zeigt Version A = Ergebnis von `fifo3A`

Version B

t	G	Q1	S1	Q2	S2	Q3	S3	raus
1	E1		E1					
2	E2				E2			
3	E3						E3	
4	E4	E4						
5	E5			E5				
5.5		–	E4					E1
6	E6	E6						
6.5				–	E5			E2
7	E7			E7				
7.5							–	E3
8	E8						E8	
9	E9					E9		
10	E10	E10/6						
		E10	E6					E4
11	E11	E11/10						
				–	E7			E5
12	E12			E12				
12.5						–	E9	E8
13	E13					E13		
14.5		E11	E10					E6
15.5				–	E12			E7
17						–	E13	E9
19		–	E11					E10
20					–			E12
21.5							–	E13
23.5			–					E11

was ist richtig?

- Spezifikation eines Systems:

  oft sprachlich, nicht präzise genug

  häufig Details/Spezialfälle nicht beachtet

  besonders schwierig: gleichzeitige Events im Kontext vieler Komponenten

  was passiert bei echter Gleichzeitigkeit (Mehr-Prozessor-CPU)?

- Formale Spezifikation mit PDEVS:

  PDEVS = Parallel Discrete EVent System [13], [L8, Kap.4]

wichtige Eigenschaften

- exakt definiert
- geeignet zur Beschreibung beliebiger diskreter Systeme
- erlaubt mathematische Analyse von Modellen

beschreibt Komponenten und ihre Verknüpfungen

- einzelne Komponente: **Atomic PDEVS**
- System aus Komponenten: **Coupled PDEVS**
- hierarchischer Aufbau wie üblich

Erweiterung von (Classical) DEVS

- vereinfacht Beschreibung bei gleichzeitigen Events
- erlaubt Parallelität der Ausführung

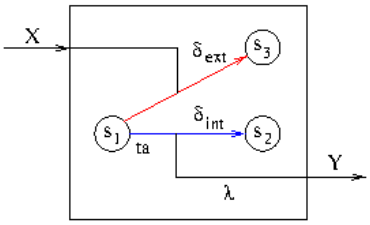
Definition von PDEVS

- Grundidee von Atomic PDEVS:

  beschreibt Komponente durch ihr Verhalten

  Bestandteile
  - Komponente hat Eingänge mit Werten aus Menge X
  - Komponente hat Ausgänge mit Werten aus Menge Y
  - Komponente ist in einem von vielen möglichen Zuständen aus der Menge S
  - Zustände haben eine Lebenszeit ta(s) (auch 0 oder ∞ möglich)

  Verhalten

  - System sei im Zustand s_1
  - kein Input → Zustand geht nach der Zeit $ta(s_1)$ in Folgezustand $s_2 = \delta_{int}(s_1)$ über
  - Input x nach Zeit $e < ta(s_1)$ → Zustand geht in Folgezustand $s_3 = \delta_{ext}(s_1, e, x)$ über
  - Input genau bei anstehendem Zustandswechsel, also nach $ta(s_1)$ → neuer Zustand mit $\delta_{con}(s_1, e, x)$

  Ausgabe
  - Ausgabewert $\lambda(s_1)$, nur vom Zustand abhängig (Moore, nicht Mealy!)
  - direkt **vor** dem Wechsel des Zustands
  - also vor dem Aufruf von δ_{int} bzw. von δ_{con}

- Formale Definition von Atomic PDEVS:

  Modell M gegeben durch $M = (X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$ mit

$X = \{(p, v) \mid p \in IPorts, v \in X_p\}$	Menge der Eingangsports und Werte
$Y = \{(p, v) \mid p \in OPorts, v \in Y_p\}$	Menge der Ausgangsports und Werte
S	Menge der Zustände
$ta : S \to [0, \infty]$	Zeitfortschaltungsfunktion
$\delta_{int} : S \to S$	interne Übergangsfunktion
$Q = \{(s, e) \mid s \in S, 0 \le e < ta(s)\}$	Zustand s und Zeit e seit letzter Transition
$\delta_{ext} : Q \times X^b \to S$	externe Übergangsfunktion
$\delta_{con} : S \times X^b \to S$	konfluente Übergangsfunktion
$\lambda : S \to Y^b$	Ausgabefunktion

  IPorts bezeichnet die Eingabeports
  - am einfachsten IPorts = {1, 2, .., n_{in}} bei n_{in} Eingabeports
  - analog OPorts

  Bedeutung von X^b
  - eine Eingabe enthält Port und (möglicherweise zusammengesetzten) Wert
  - es können mehrere Eingaben gleichzeitig an den Eingängen ankommen
  - darunter sogar mehrmals identische Werte
  - alle gleichzeitig eingetroffenen Port/Wert-Paare im **Bag** X^b (= Menge mit Wiederholungen) gesammelt

Eingabe-Beispiel mit $n_{in} = 2$

- x = {{ (1,5), (2,3), (1,3), (2,3) }}
- an Port 1 kommen gleichzeitig Werte 5 und 3 an
- an Port 2 kommt Wert 3 zweimal gleichzeitig an
- Reihenfolge der Werte ist irrelevant (wie bei Menge)

- Beispiel Generator:

  einfaches Generatormodell von `singleserver1B`
  - liefert Entities (nur mit id) in festen Zeitabständen
  - Ausgang für Anzahl überflüssig, da mit Entity-id identisch

  Anschlüsse
  - keine Eingänge, ein Ausgang A
  - A liefert in festen Zeitabständen t_G die Werte 1, 2, 3, ...
  - beginnt bei t_G

  Implementierung in PDEVS
  - Zustand s speichert letzten ausgegebenen Wert, beginnt bei 0
  - $ta(s) = t_G$
  - $\lambda(s) = s+1$ (kommt **vor** dem Zustandswechsel!)
  - $\delta_{int}(s) = s+1$
  - kein δ_{ext}, kein δ_{con}

- Beispiel Terminator:

  einfaches Terminatormodell von `singleserver1B`
  - Eingang bekommt Entities
  - kein Ausgang

  Implementierung in PDEVS
  - ein Eingang, kein Ausgang
  - Zustand s speichert Anzahl der eingegangenen Werte, beginnt bei 0
  - $ta(s) = \infty$
  - kein δ_{int}, kein λ, kein δ_{con}
  - $\delta_{ext}(s, e, \{\{(1,x_1), (1,x_2) \dots (1,x_n)\}\}) = s+n$

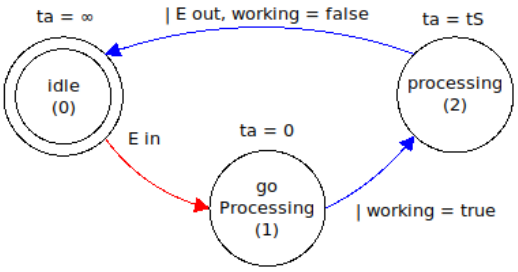
- Beispiel Server:

  einfaches Servermodell mit Prozesszeit t_S

  Anschlüsse
  - Eingang in für Entities
  - Ausgang out für Entities
  - Ausgang working (true/false) als Info zu vorgeschaltetem Block (Queue)

  Verhalten (Prinzip)
  - startet in Zustand "idle"
  - Entity am Eingang → wechselt in "processing"
  - gibt nach Zeit t_S Entity aus und geht wieder in "idle"

  Problem
  - bei Übergang in "processing" muss working = 1 ausgegeben werden
  - ist aber externes Event, daher keine Ausgabe möglich
  - Trick: Zwischenzustand "goProcessing" mit ta = 0



- Implementierung des Servers in PDEVS:

  Eingang "in", Ausgänge "out", "working"

  Zustand
  - phase: hat Werte 0|1|2 ("idle"|"goProcessing"|"processing")
  - entity: speichert die Entität (id = 1, 2, ..., 0 bei leer)

  ta(s), λ(s), δ_{int} wie im Bild

  δ_{ext}
  - bei "idle" Entity speichern und in "goProcessing"
  - bei "goProcessing"/"processing": Fehler (Eingang ist blockiert)

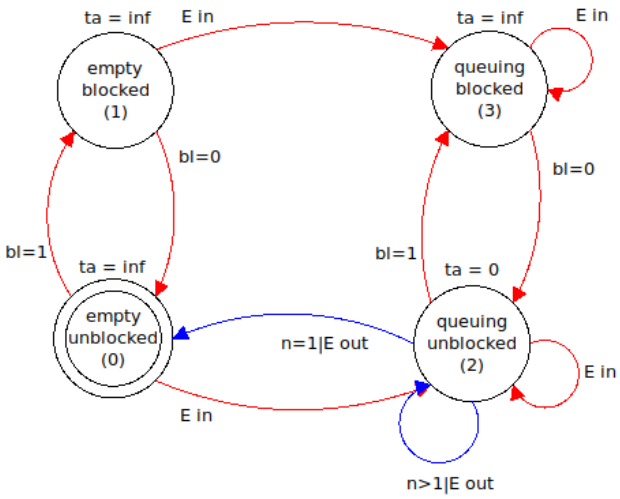
  δ_{con}
  - erst δ_{int}, dann δ_{ext}
  - in "goProcessing": Fehler
  - in "processing": alte Entität raus, neue rein
  - Achtung: ist komplizierter (s.u.)

- Beispiel Queue:

  einfaches Queuemodell ohne Kapazitätsbeschränkung

  Anschlüsse
  - Eingang in für Entities
  - Eingang bl für Blocking-Status des Ausgangs
  - Ausgang out für Entities

  Verhalten



- Implementierung der Queue in PDEVS:

  Eingänge "in", "bl", Ausgang "out"

  Zustand
  - phase: hat Werte 0|1|2|3 ("empty unblocked"|"empty blocked"|"queueing unblocked"|"queueing

blocked")
- queue: speichert den Vektor der Entitäten (ids als Werte)
- n: speichert Länge der Queue (zur einfachen Anzeige)

Zustandsbeschreibung ist redundant
- verschiedene Phasen für empty/queueing nicht nötig, da schon aus Länge von queue klar
- erleichtert aber Verständnis und vereinfacht Programmierung

ta(s), λ(s), δ_{int} wie im Bild

δ_{ext} wie im Bild
- bei mehreren bl-Werten letzten nehmen (sollte nicht sein)
- bei mehreren in-Werten alle anhängen
- bei in- und bl-Werten: beides bearbeiten und entsprechend weiterschalten

δ_{con}
- erst δ_{int}, dann δ_{ext}
- Achtung: Bug, s.u.!

- Coupled PDEVS:

  Grundidee
  - Komponente besteht aus mehreren Atomic-PDEVS-Blöcken
  - hat selbst (externe) Ein- und Ausgänge
  - Verbindungen der externen Anschlüsse mit internen Komponenten definiert
  - Verbindungen der internen Komponenten untereinander definiert
  - verboten: Ausgang einer Komponente direkt an ihren Eingang

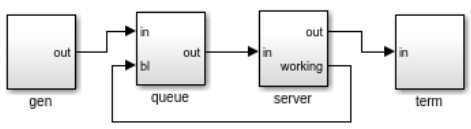
  Eigenschaften
  - Ablauf genau definiert [L8, Kap. 14.4], [16]
  - Coupled-PDEVS-Block kann als Atomic-PDEVS-Block definiert werden
  - → beliebige Hierarchiestufen aus Atomic- und Coupled-PDEVS-Komponenten möglich

  formale Definition
  - Modell M gegeben durch N = (X, Y, D, M_D, C_{EI}, C_{EO}, C_I} mit

$X = \{(p, v) \mid p \in IPorts, v \in Xp\}$	Menge der Eingangsports und Werte
$Y = \{(p, v) \mid p \in OPorts, v \in Yp\}$	Menge der Ausgangsports und Werte
D	Menge der Komponentennamen
$M_D = \{M_d \mid d \in D\}$	Menge der Atomic-PDEVS-Komponenten
$C_{EI} \subseteq \{(i_N, d, i) \mid i_N \in iPorts, d \in D, i \in IPorts_d\}$	Menge der Verbindungen von externen Inputs zu Inputs interner Komponenten
$C_{EO} \subseteq \{(d, o, o_N) \mid o_N \in OPorts, d \in D, o \in OPorts_d\}$	Menge der Verbindungen von Outputs interner Komponenten zu externen Outputs
$C_I \subseteq \{(d_1, o, d_2, i) \mid d_1,d_2 \in D, o \in OPorts_{d1}, i \in IPorts_{d2}\}$	Menge der Verbindungen zwischen internen Komponenten

- Gesamtmodell `singleserver1B` als Coupled PDEVS:

  graphisch



  keine Eingänge, keine Ausgänge (**root model**)

  D = {"gen", "queue", "server", "term"}

$M_D = \{M_{Generator}, M_{Queue}, M_{Server}, M_{Terminator}\}$

C_{EI}, C_{EO} leer (mangels externer Anschlüsse)

C_I = ('gen', 'out', 'queue', 'in'), ('queue', 'out', 'server', 'in'),
('server', 'out', 'term', 'in'), ('server', 'working', 'queue', 'bl')}

- Weiterentwicklung RPDEVS [14,15]:

  Problem
  - betrachte Mealy-Komponente M_1 mit nachfolgender Komponente M_2
  - M_1 benötigt internen Zwischenzustand
  - ihr Output kommt später als der Output von M_2
  - → Ausgang von M_2 reagiert verspätet

  Lösungsidee: λ hängt auch von Inputwerten ab → RPDEVS (Revised PDEVS)

  detaillierte Umsetzung
  - macht Simulator deutlich komplizierter
  - macht Modellierung deutlich einfacher

  ist das die Lösung?

- MatlabDEVS:

  Implementierung von PDEVS in Matlab
  - entwickelt an der HS Wismar [17,18]
  - aktuelle Version mit hybrider Modellierung [19]
  - objektorientiert implementiert

  stellt Templates als Ausgangspunkt zur Verfügung
  - `am_discrete_template.m` für AtomicPDEVS-Komponente
  - `cm_discrete_template.m` für CoupledPDEVS-Komponente

  Vorgehensweise
  - Erstellen von Klassendateien für die AtomicPDEVS-Komponenten
  - Erstellen von Klassendateien für die CoupledPDEVS-Komponenten (incl. root model)
  - Erstellen eines Run-Skripts `run_xxx.m` für die Simulation
  - dazu: Erstellen eines Auswerte-Skripts `analyse_xxx.m` zur Ergebnis-Darstellung

- Erstellen eines AtomicPDEVS:

  Klasse ableiten von `atomic`

  implementiere Konstruktor
  - definiert Parametergrößen
  - definiert x-, y-Anschlüsse
  - definiert und initialisiert Zustandsgröße s

  implementiere Methoden
  - deltaintfun
  - deltaextfun
  - deltaconffun
  - lambdafun
  - tafun

- Implementieren des Generators:

  als Klasse `am_generator`, abgeleitet von `atomic`

  ```
  classdef am_generator < atomic
  ```

  Deklaration des Konstruktors

  ```
  function obj = am_generator(name, tG, n0, debug)
  ```

  - mit Parametern

`name`	Name des Objekts
`tG`	Zeitintervall zwischen neuen Entitäten
`n0`	id der ersten Entität
`debug`	für optionale Debugging-Ausgaben

  Implementierung des Konstruktors

```
if nargin == 4
  x = {};           % no inputs
  y = {'out'};      % output to emit entities
  s = {'counter'};  %
  sysparams = struct('tG', tG, 'n0', n0, 'debug', debug);
else
  error('mistake at constructor method for class am_generator');
```

```
  end
  obj = obj@atomic(name, x, y, s, 0, sysparams); % call parent
  obj.s.counter = obj.sysparams.n0; % initialize the state
```

Implementierung der internen Übergangsfunktion

```
function deltaintfun(obj)
  obj.s.counter = obj.s.counter + 1;
end
```

Implementierung der Ausgabefunktion

```
function lambdafun(obj)
  % generate an entity with given id
  obj.y.out = {obj.s.counter + 1};

  if obj.sysparams.debug
    fprintf('%4.1f %8s OUT, out=%2d\n', ...
            obj.tnext, obj.name, cell2mat(obj.y.out))
  end
end
```

Implementierung der Zeitfortschaltungsfunktion

```
function ta = tafun(obj)
  ta = obj.sysparams.tG;
end
```

`deltaextfun`, `deltaconffun` leer

komplette Klassendatei `am_generator.m`

- Implementierung der anderen Komponenten:

  komplette Klassendateien
  - `am_terminator.m`
  - `am_serverA.m`
  - `am_queueA.m`

  Hilfsfunktionen in `am_queueA`
  - `readInputBag` sammelt Eingangswerte ein
  - `showState` zeigt Debug-Informationen

- Erstellen eines CoupledPDEVS:

  Klasse abgeleitet von `coupled`

  enthält nur Konstruktor
  - Name und Parameter als Argumente
  - Komponenten initialisieren und hinzufügen
  - Ein- und Ausgänge definieren, z.B.

    ```
    set_x_ports(obj, {'in1', 'in2'});
    set_y_ports(obj, {'out1'});
    ```

  - Verbindungen als 4xN-Cell-Matrix bei N Verbindungen (intern und extern zusammen)
  - Verbindungszeile: Startobjekt, Ausgang, Zielobjekt, Eingang
  - bei externen Anschluss: Name des Objekts = `'parent'`

  komplette Klassendatei für Root Model `model_singleserverA.m`

- Ausführung:

  am einfachsten mit kleinem Skript

am einfachsten mit kleinem Skript

- setzt Parameterwerte
- erzeugt root model (Konstruktor)
- markiert alle Zustandsgrößen zur Beobachtung (`set_observe`)
- führt das Modell aus (`r_c_discrete`)
- zeigt Ergebnisse an (eigene Hilfsfunktion `analyse_MODEL`)

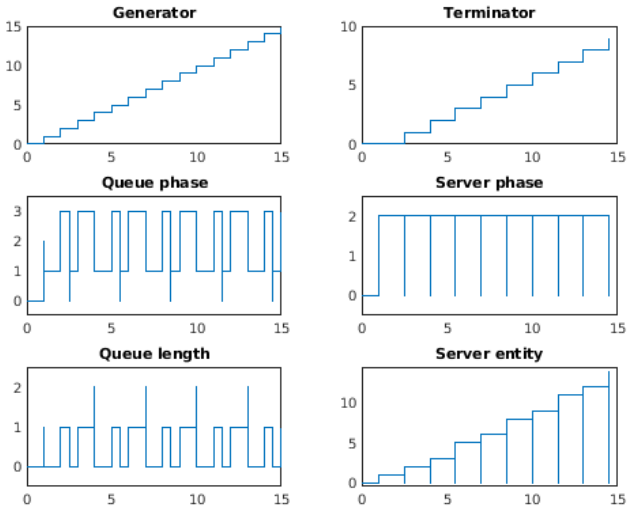
komplettes Beispielskript `run_singleserver.m`

Ausgabeskript

- verwendet Standard-Ausgabe- und Plot-Routinen
- Zugriff auf Zustandsgrößen über `root_model.components.KOMPONENTE.observed`
- 1. Spalte enthält Zeitwerte, 2. Spalte enthält Zustandswerte
- alles jeweils in Strukturen und Cellarrays verpackt
- Umwandeln in einfache Vektoren über Hilfsvariablen und []

komplettes Beispielskript `analyse_singleserver.m`

Ergebnis



- Vergleich mit oben: Ergebnis falsch ab t=4!
- außerdem Warnungen `dExt: wrong phase 2 in server`

- Fehlersuche:

  Erwartung vor t=4 (vgl. manuelle Analyse in `singleserver1B`)
  - vorher: E2 im Server, E3 in der Queue
  - bei t=4: E2 wird fertig, E3 geht in den Server, E4 von Generator in Queue

  setze `debug = true` in `run_singleserver` und starte

  zur Übersicht Ausgaben zusammen mit Zustandsgrößen in Spreadsheet
  - danach durchgehen und Kommentare hinzufügen (Zustandsdiagramme daneben legen!)

	t	Description	gen		queue			server		Comment
				phase	queue	n	phase	entity	wk	
5	1	gen OUT, out= 1								
6	1	queue entering ext, being in phase 0								
7		phase=0 n=0 queue=								
49		phase=3 n=1 queue= 3		3	3	1	2	2		
50	4	gen OUT, out= 4								
51	4	server OUT, working= 0 out= 2							0	cf Z56
52	4	queue entering ext, being in phase 3								cf Z50/51
53		phase=3 n=1 queue= 3								ok
54		queue leaving ext, going to phase 2								ok
55		phase=2 n=2 queue= 3 4		2	3,4	2				ok
56	4	server int, going to phase 0					0	0		cf Z51
57	4	queue OUT, out= 3								cf Z59
58	4	server ext, going to phase 1					1	3		cf Z57
59	4	queue entering int, being in phase 2								cf Z57
60		phase=2 n=2 queue= 3 4								ok
61		queue leaving int, going to phase 2								ok
62		phase=2 n=1 queue= 4		2	4	1				ok
63	4	queue OUT, out= 4								kommt vor Z64!!
64	4	server OUT, working= 1 out=							1	sollte vor Z63 sein!
65	4	queue con, in phase 2								bl-In und 2→2
66	4	queue entering int, being in phase 2								cf Z65
67		phase=2 n=1 queue= 4								
68		queue leaving int, going to phase 0								
69		phase=0 n=0 queue=								
70	4	queue entering ext, being in phase 0								
71		phase=0 n=0 queue=								
72		queue leaving ext, going to phase 1								
73		phase=1 n=0 queue=								
74	4	server con, in phase 1								
75	4	server int, going to phase 2								
76		dExt: wrong phase 2 in server								
77	4	server ext, going to phase 2								
78	5	gen OUT, out= 5								

zeilenweise Analyse

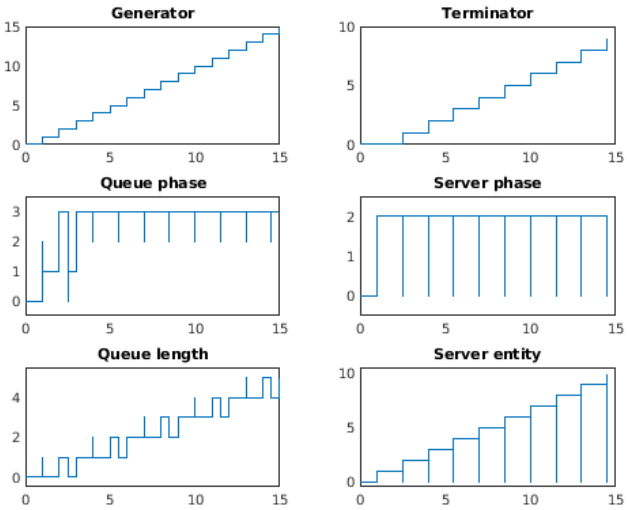
Z49	(vor t=4) wie erwartet
Z50	`gen` gibt E4 aus
Z51	`server` gibt E2 und working=0 aus, Zustandsänderung danach in Z56
Z52-55	`queue` sammelt E4 ein und geht in unblocked
Z56	`server` geht nach idle (vgl. Z51)
Z57	`queue` gibt E3 aus (vgl. Z59)
Z58	`server` empfängt E3, geht in goProcessing
Z59-62	Zustandsänderung zur Ausgabe (vgl. Z57)
Z63	`queue` gibt schon E4 raus!
Z64	jetzt kommt erst das Blocking vom Server!
Z65	bl-Input und Wechsel 2→2 sind confluent!
Z66-73	`queue` macht erst int (also E raus), dann ext (also bl)
	muss anders herum!

- Fehlerkorrektur:

  Lösung

```
function deltaconffun(obj,gt)
  [bl, ~] = readInputBag(obj);
  if bl == true  % go to phase 3, no entity output!
    deltaextfun(obj,gt);
  else
    deltaintfun(obj);
    deltaextfun(obj,gt);
  end
end
```

  Ergebnis



- Plot identisch mit oben
- aber neue Warnungen: `dExt: wrong phase 2 in server`

erneute Analyse mit Debugging-Output →

- Entity am Eingang in Phase 2 des Servers
- Ursache: `working` kam bei Queue an *nach* Ausgabe einer Entity (vgl. Z63f)
- → Server ruft `deltaconffun` in Phase 1

Lösung: Entity ignorieren

```
function deltaconffun(obj,gt)
  if obj.s.phase == 1   % blocking came to late -> ignore input
    deltaintfun(obj);
  else  % first current entity leaves, then new one enters
    deltaintfun(obj);
    deltaextfun(obj,gt);
  end
end
```

alles ok: Plot wie vorher, keine Warnungen mehr

korrigierte Klassen `am_server.m`, `am_queue.m`, `model_singleserver.m`

- Aufgaben:

  Aufgabe 22

  Aufgabe 23

Hybride Systeme

- Systeme mit State-Events
- Systeme mit veränderlicher Struktur
- Diskrete Systeme mit kontinuierlichen Subsystemen

Systeme mit State-Events

- Eigenschaften hybrider Systeme:

  haben kontinuierliche Zustandsgrößen sowie diskrete Events (und evtl. diskrete Zustandsgrößen)

  große Vielfalt
  - kontinuierliche Systeme mit Unstetigkeiten (Bsp. Hüpfender Ball)
  - diskrete Umschaltung zwischen kontinuierlichen Systemen (Bsp. Fadenpendel mit Freiflug-Phase)
  - diskrete Systeme mit kontinuierlichen Subsystemen (Bsp. Muffelofen)

  Simulation schwierig (vgl. Argesim-Benchmark C21 [20])
  - meist Kombination verschiedener Methoden
  - Ansätze häufig ausgehend von DGLs oder ausgehend von Ereignissteuerung

- Mathematische Beschreibungsmethoden:

  sehr verschiedene Ansätze je nach Ausgangspunkt
  - bisher kein allgemein verbreitetes universelles Hybrid-Modell

  Hybrid DAEs [21]
  - Ausgangspunkt kontinuierlich (DAEs)
  - Einführen verschiedener Klassen von Ereignissen [20]
  - Details je nach möglichen Event-Typen unterschiedlich

  DEV&DESS [L8]
  - Kombination von DEVS und DESS ("Differential Equation System Specification")
  - DESS = einfache Zustandsraum-Beschreibung von DGLs incl. Eingangs- und Ausgangsgrößen
  - Kopplung zwischen DEVS- und DESS-Größen

  DEVS mit QSS ("Quantized State Systems") [L8]
  - Idee: DGLs müssen zur Simulation sowieso diskretisiert werden
  - statt der Zeit werden hier die Zustände der DGLs diskretisiert
  - damit wird DGL-Anteil durch DEVS beschrieben
  - entsprechende QSS-DGL-Solver wurden entwickelt

- Beispiel "hüpfender Ball":

  System
  - Ball fällt senkrecht herunter
  - prallt am Boden ab, verliert dabei Energie

  Modell
  - DGL zur Beschreibung des Falls
  - Abprall als instantan vereinfacht
  - Höhe wird 0 → v ändert sein Vorzeichen, sein Betrag nimmt um festen Faktor ab

  grundsätzlich kontinuierliche Zustandsgrößen, aber zusätzliche Events
  - Zeitpunkt der Events nicht vorher bekannt
  - Zustände ändern sich bei Event (ggf. unstetig)
  - State-Change Event SE-X in Nomenklatur von [20]

- DGL-Solver mit Events:

  Solver bekommt Funktion h(t,y) zur Event-Definition (zusätzlich zu f(t,y))

  Event = "h wird 0"

  Solver führt Schritt durch und prüft, ob Vorzeichen von h wechselt

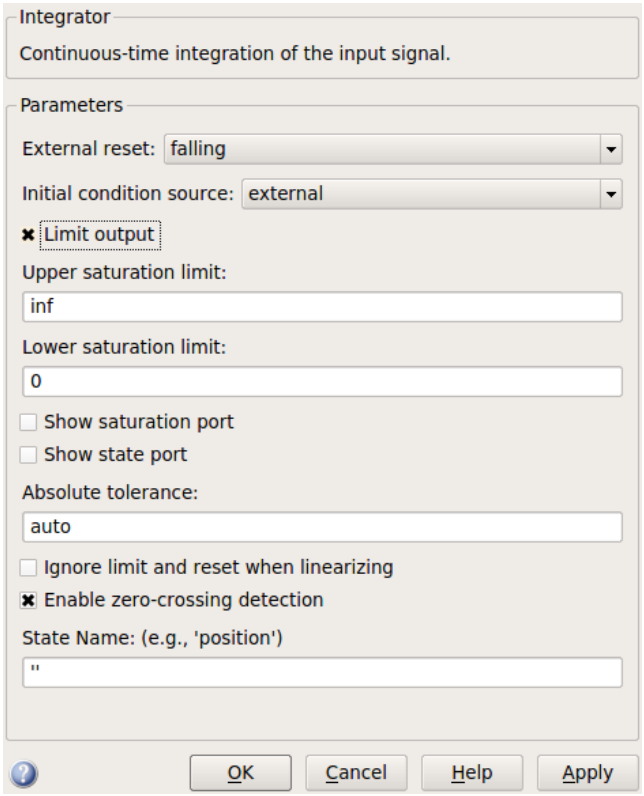
  nein → nächster Schritt wird berechnet

  falls ja

- genauer Zeitpunkt t_E des Events wird bestimmt (z. B. durch Newton-Verfahren)
- Solver löst DGL bis t_E und stoppt
- Zustandsgrößen können geändert werden
- Solver wird von t_E an neu gestartet

- Implementierung in Simulink mit Integrator-Block:

  Parameter



  Beschränkung des Ergebnis-Bereichs (`saturation limits`)

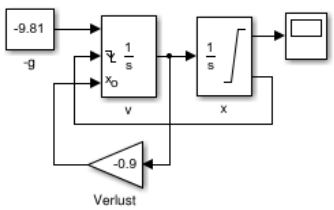
  optionales Signal bei Überschreiten und Verlassen der Grenzwerte (`saturation port`)

  Eingangssignal zum Zurücksetzen auf Anfangswerte (`external reset`)

  Eingangssignal zur externen Festlegung der Anfangsbedingung (`initial condition source` = `external`)

- Modell `huepfballA`:

  Aufbau



  Funktionsweise
  - x-Integrator ist eingeschränkt auf [0, Inf]
  - `saturation port` liefert negatives Triggersignal bei Erreichen von x = 0
  - v-Integrator wird dadurch auf Anfangswert zurückgesetzt
  - Anfangswert ist -0.9*aktueller Wert

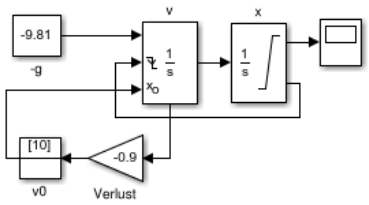
  Ergebnis:
  - hüpft nicht, sondern bleibt liegen
  - Ursache: algebraischer Schleife bei v

- Modell `huepfballB`:

  Abhilfe: state port = Integrator-Ausgang zur Zeit t, vor dem Reset

  durchbricht algebraische Schleife

  Aufbau

  zusätzliches v_0 durch `Initial Condition`-Block

  Ergebnis: funktioniert

  interessanter Effekt

  - `zero-crossing detection` bei beiden Integratoren ausschalten
  - Ursache: Solver bekommt keine Event-Funktion mehr

- Zenon-Effekt:

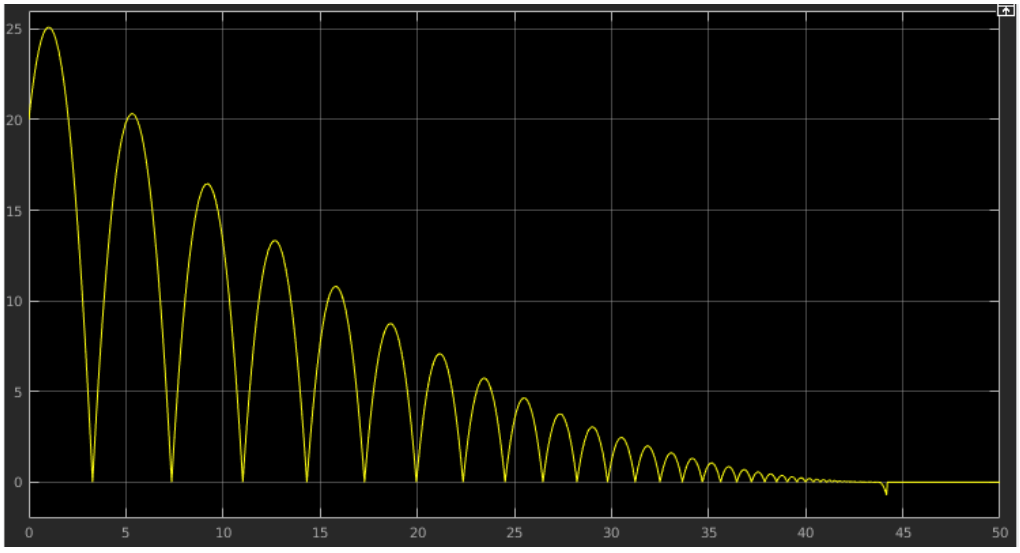
  Modell `huepfballB` bis t=50 laufen lassen → Fehlermeldung

  ```
  At time 43.997112102528291, simulation hits (1000) consecutive zero
  crossings.
  ```

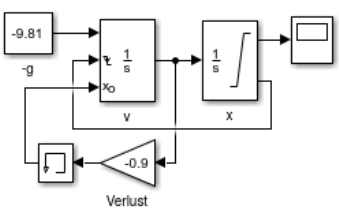
  Ursache: unendliche viele Hüpfer in endlicher Zeit (**Zenon-Effekt**)

  Abhilfe: `Solver-Parameter Zero-crossing options/Algorithm` auf `Adaptive`

  Ergebnis von Modell `huepfballC`

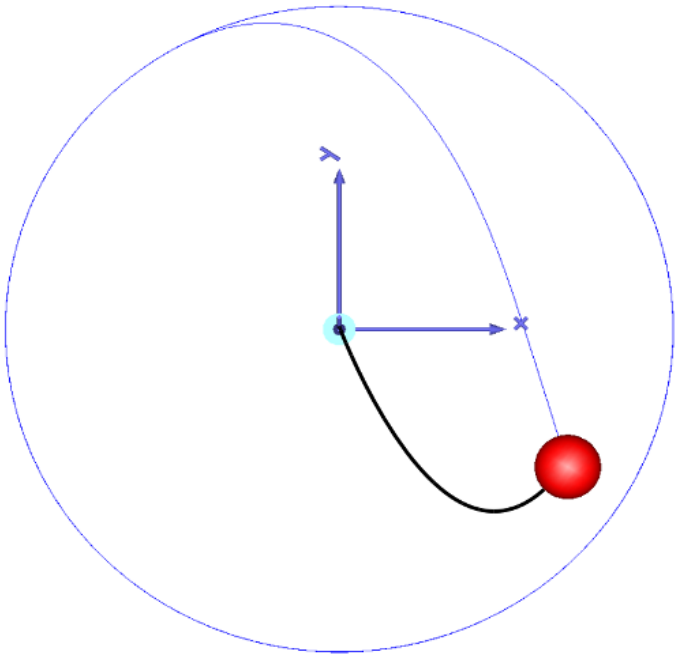
  alternativ `Memory`-Block statt state port → klappt ohne "Zacke" auch bei `Algorithm` auf `Non-adaptive` (`huepfballD`)

  genauere Untersuchung nötig (vgl. [20])

- Beispiel "frei fallendes Fadenpendel" [20]:

  System
  - Punktmasse mit Luftwiderstand an (masselosem) Seil fester Länge
  - Bewegung wechselt zwischen Pendel- und Freifall-Phase je nach Richtung der Kraft auf die Masse
  - Bewegung bleibe in einer Ebene



  Modell-DGLs
  - Pendel-Phase

$$s_1 = \begin{pmatrix} \varphi \\ \omega \end{pmatrix}, \qquad \dot{s}_1 = \begin{pmatrix} \omega \\ -\frac{k}{m}\omega + \frac{g}{l}\sin\varphi \end{pmatrix}$$

  - Frei-Fall-Phase

$$s_2 = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}, \qquad \dot{s}_2 = \begin{pmatrix} v_x \\ v_y \\ -\frac{k}{m}v_x \\ -\frac{k}{m}v_y - g \end{pmatrix}$$

  - Achtung: φ = 0 am *höchsten* Punkt

  Umschalten Pendel → Frei-Fall
  - Seilkraft h_1 wird negativ

$$h_1 = -mg\cos\varphi + ml\omega^2$$

  Umschalten Frei-Fall → Pendel
  - Punktmasse erreicht Abstand l (von innen)
  - Funktion h_2 wird negativ

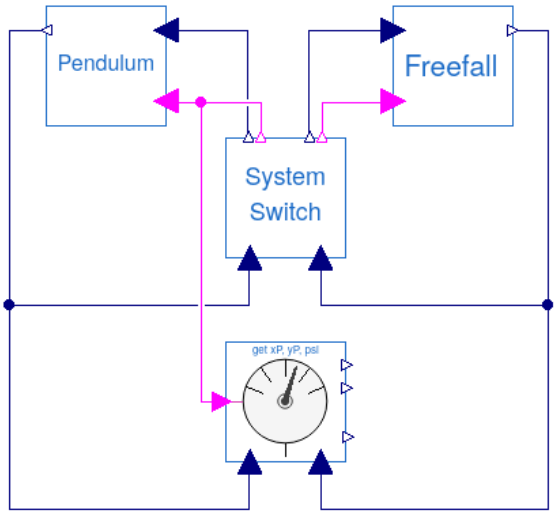
$$h_2 = l^2 - x^2 - y^2$$

  Beim Umschalten ändert sich die Struktur, insbesondere die Dimension des Zustandsraums

  Structure-Change event SE-S in Nomenklatur von [20]

- Implementierung in Modelica [22, 23]:

  Gesamtmodell `FreeFallPendulum`



zwei Blöcke für die beiden System-Konfigurationen mit

- Eingang `active` zum Aktivieren
- Eingang `newState` für Anfangswert beim Aktivieren
- Ausgang `stateOut` für aktuelle Zustandwerte

Block `SystemSwitch`

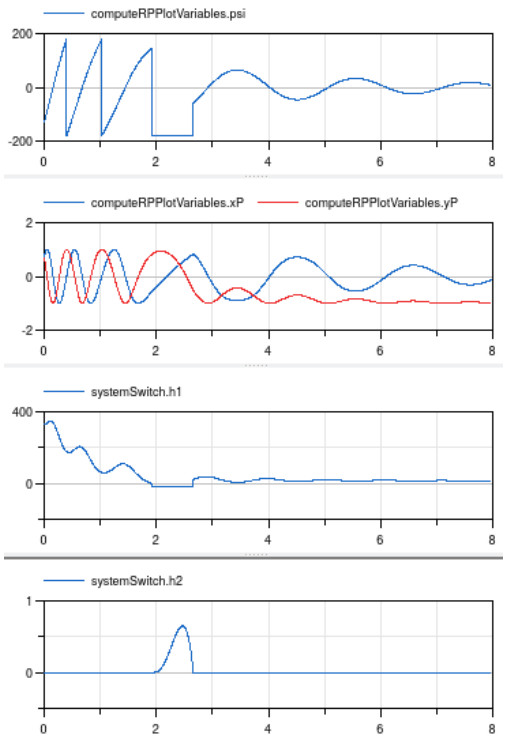
- bekommt aktuelle Zustandwerte (inaktives System liefert "0-Werte")
- aktiviert damit abwechselnd die beiden Systeme
- liefert jeweils Anfangswerte beim Aktivieren

unterer Block berechnet nützliche Ausgabewerte

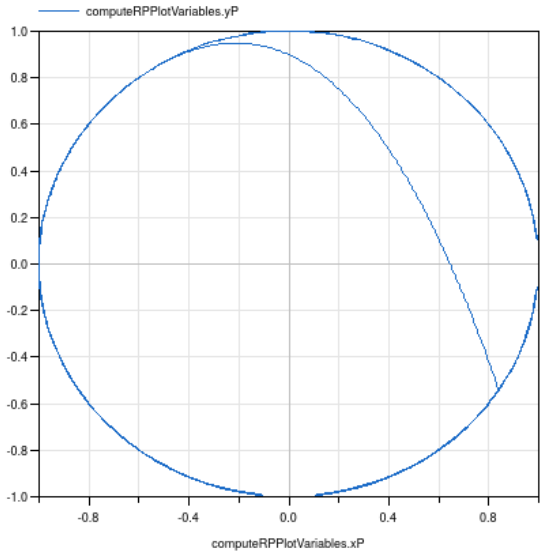
- Winkel von unten
- x/y-Werte auch in Pendel-Phase

Struktur geeignet für beliebige Systeme mit Strukturwechsel

Ergebnisse



- x-y-Plot



- Implementierung der Teil-Systeme:

  erben von Basis-System `SwitchableSystem`
  - definiert Dimension des Zustandsraums, Ein- und Ausgänge
  - enthält Code zum Einschalten

  ```
  when active then
    reinit(state, pre(newState));
  end when;
  stateOut = if active then state else sOff;
  ```

  definieren Systemparameter

  enthalten Zustandgleichungen, z.B. beim Pendel

  ```
  if active then
    der(state) = {state[2], -(k/m)*state[2] + (g/l)*sin(state[1])};
  else
    der(state) = zeros(N);
  end if;
  ```

  `when` und `if` in Modelica erzeugen automatisch Events

- Implementierung des Umschalters:

  erbt von `PartialSystemSwitch`
  - definiert Dimension der Zustandsräume, Ein- und Ausgänge
  - enthält Code zum Umschalten

  ```
  when h1 < 0 then
    active1 = false;
  elsewhen h2 < 0 then
    active1 = true;
  end when;
  active2 = not active1;
  ```

  berechnet Werte für h1 und h2

  berechnet neue Zustandswerte beim Umschalten, z.B. s_2 aus s_1

  ```
  new2[1] = l*sin(state1[1]);
  new2[2] = l*cos(state1[1]);
  new2[3] = l*state1[2]*cos(state1[1]);
  new2[4] = -l*state1[2]*sin(state1[1]);
  ```

- Umschalten zwischen Gleichungen in Modelica:

  streng genommen nicht möglich
  - Modell enthält zu jedem Zeitpunkt Zustandvariable für beide Teilsysteme
  - alle Variablen brauchen immer Gleichungen

  aber
  - inaktiver Teil hat triviale Gleichungen
  - Modelica-Compiler optimiert triviale Gleichungen weg

- Beispiel "Hybrider Härteofen" [24]:

  industrieller Härteofen zur Wärmebehandlung von Werkstücken aus Stahl

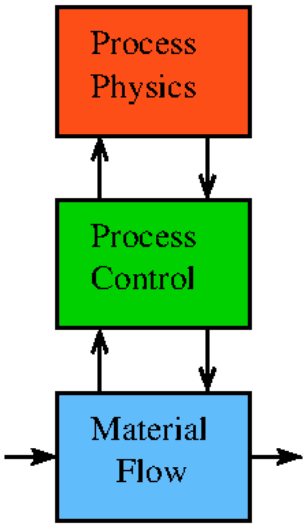
  Funktionsweise
  - ankommende Werkstücke werden gesammelt, bis eine Ofenladung erreicht ist
  - Werkstücke durchlaufen mehrere Wärmebehandlungen aus Aufheizen und Halten
  - verlassen Ofen und durchlaufen ggf. weitere Fertigungsschritte
  - Abkühlungsphase des Ofens am Ende

  besonderes Interesse am Energieverbrauch
  - → Wärmeströme zwischen Werkstücken, Ofen und Umgebung untersuchen

  komplexe Kopplung aus diskretem und kontinuierlichem Systemverhalten

- Beschreibung als Schichtenmodell:

  Trennung unterschiedlicher Systemaspekte

  Materialfluss-Schicht (MF)
  - beschreibt Fluss der Werkstücke in den Ofen und hinaus
  - Verbindung des Ofens innerhalb der Produktionskette

  Prozesskontroll-Schicht (PC)
  - beschreibt interne Kontrollabläufe
  - wichtig für Maschinen mit komplexen inneren Abläufen (Phasen)

  Prozessphysik-Schicht (PP)
  - beschreibt relevante physikalische Prozesse
  - z.B. Wärmeströme, chemische Reaktionen

  Kopplung der Schichten
  - MF → PC: Ofenladung an Werkstücken ist angekommen
  - PC → MF: Werkstücke verlassen den Ofen
  - PC → PP: aktuelle Phase (`load`, `heatUp`, s.u.)
  - PP → PC: Ofen-Temperatur

  Verbindung nach außen hier nur über MF-Schicht
  - grundsätzlich auch weitere externe Kopplungen möglich, z.B. auf PP-Level
  - erzeugt deutlich komplexere Struktur des Gesamtsystems (Netzwerk)

- Mathematische Modellierung:

  Modell enthält DGLs und komplexe diskrete Prozesse

Beschreibung möglich mit DEV&DESS

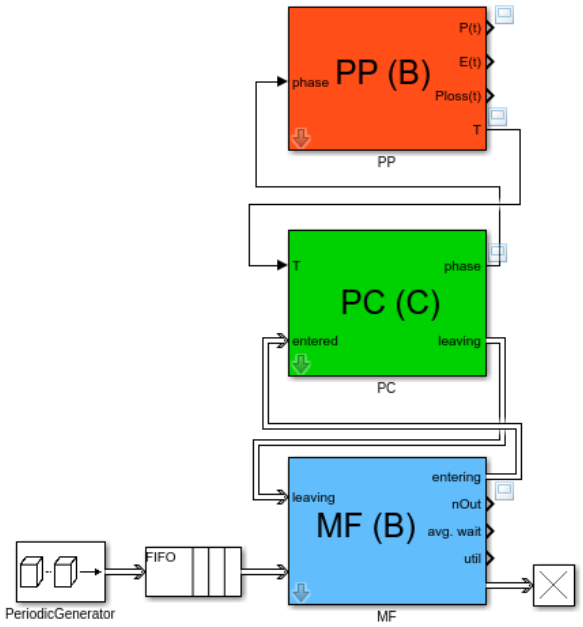
- DESS = DGLs in der PP-Schicht
- DEVS = diskrete Prozesse in MF- und PC-Schicht
- Kopplung der Teilsysteme über Variable (Ein-, Ausgangsgrößen, Zuständsgrößen) der Teilsysteme

mögliche Alternative

- DGLs als QSS
- Gesamtmodell als DEVS

- Implementierung des Gesamtsystems in Simulink/Stateflow/SimEvents:

  Schichtenmodell durch Subsysteme



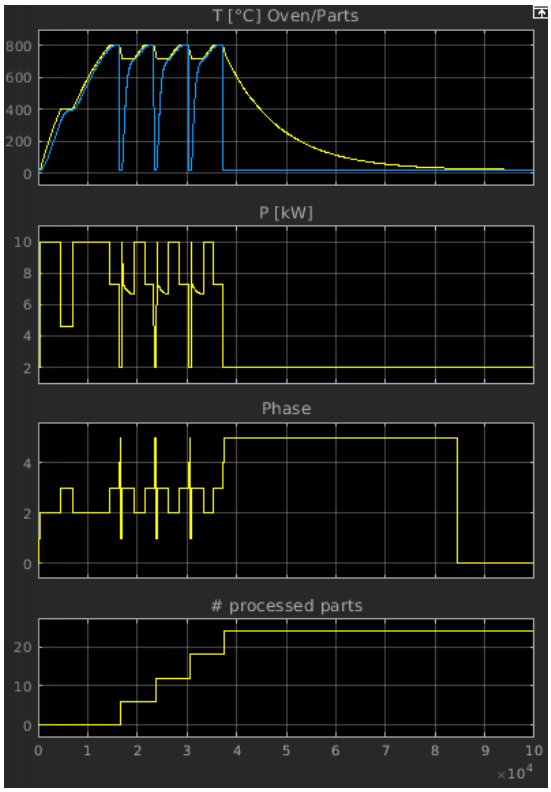
  minimales Gesamtsystem

  - Werkstücke werden in festen Intervallen erzeugt
  - in Queue zwischengespeichert
  - durchlaufen Ofen
  - verlassen System

  Kopplungen

  - MF → PC: SimEvent-Message
  - PC → MF: SimEvent-Message
  - PC → PP: reelle Variable
  - PP → PC: reelle Variable (eigentlich 2-Vektor, incl. Werkstück-Temperatur)

  Ergebnisse

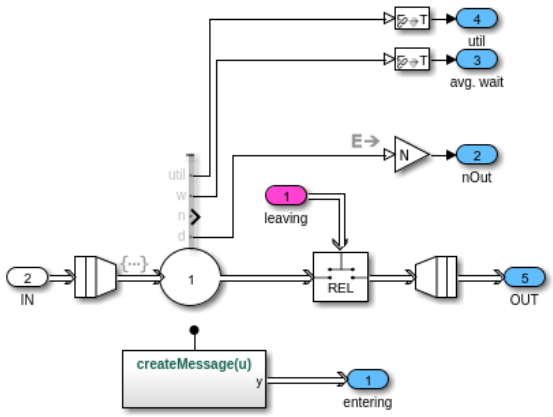


- Modellierung der Materialfluss-Schicht:

  Verhalten

  - eingehende Werkstücke werden auf einem "Blech" bis zur Batchgröße gesammelt
  - ist der Ofen leer, wird ein kompletter Batch in den Ofen gebracht
  - eine Message wird an die PC-Schicht geschickt
  - kommt eine Message von PC, wird der Ofen geleert
  - die Werkstücke werden vereinzelt und weitergegeben

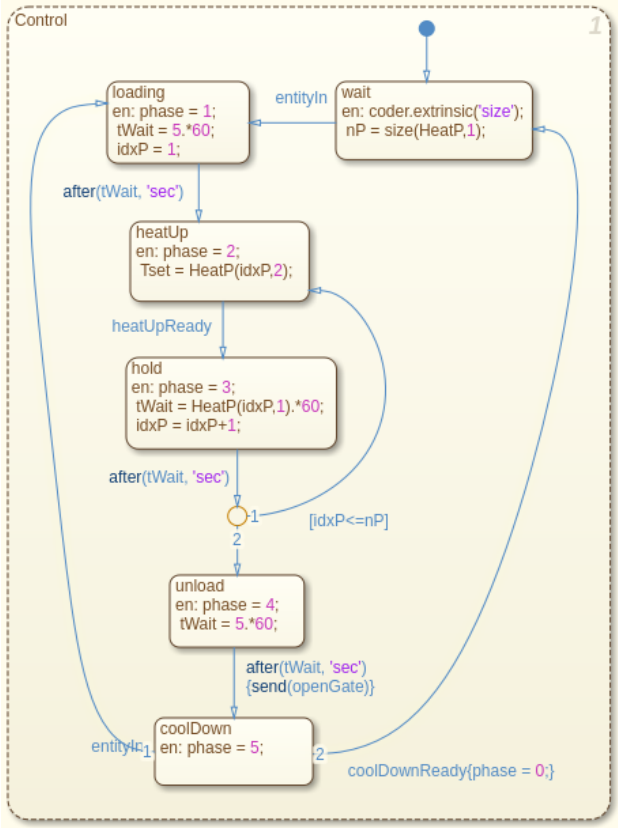
  Implementation in SimEvents



  - Vorsicht: `Entity Batcher` und `Entity Unbatcher` speichern intern!
  - → `Entity Gate` muss direkt hinter dem Server sein

- Modellierung der Prozesskontroll-Schicht:

  Verhalten gegeben durch Zustandsdiagramm

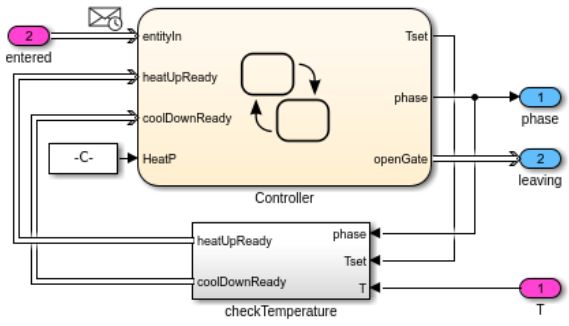


verwendet verschiedene Ein-/Ausgabe- und interne Größen

TYPE	NAME	VALUE	PORT
	entityIn		1
	heatUpReady		2
	coolDownReady		3
	HeatP		4
	Tset		1
	phase		2
	openGate		3
	idxP	1	
	nP		
	tWait		

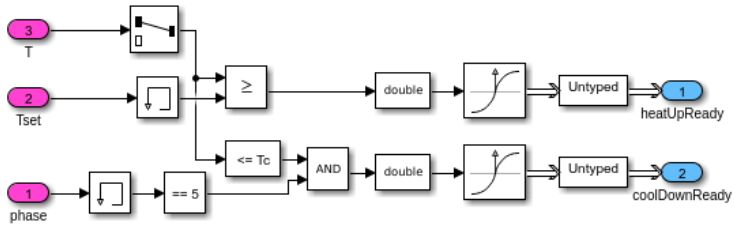
- Eingänge 1-3: SimEvents-Messages
- Eingang 4: Matrix `HeatP` mit Zeilen (t_i, T_i) für Haltedauer und Temperatur
- Ausgänge 1-2: reelle Werte
- Ausgang 3: SimEvents-Message

gesamte Komponente in Simulink



Submodell `checkTemperature`

- vergleicht Ist-Temperatur `T` und Soll-Temperatur `Tset`
- erzeugt Message `heatUpReady` für Statechart
- prüft ggf., ob Abkühltemperatur erreicht ist

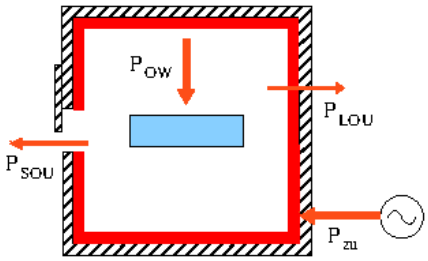


umständliches Erzeugen der Messages

- Hit Crossing-Block erzeugt *keine* Message, sondern komplexe Entity
- muss in einfache Message gewandelt werden
- Standard-Trick: mit `Entity Generator`

- Modellierung der Prozessphysik-Schicht:

  betrachtete Energieströme

  - P_{zu}: zugeführte Heizleistung
  - P_{OW}: Ofen → Werkstück, Konvektion + Strahlung
  - P_{LOU}: Ofen → Umgebung, Leitung
  - P_{SOU}: Ofen → Umgebung, Konvektion + Strahlung (bei geöffneter Tür)

  Berechnung aus den Temperaturen von Ofen, Werkstück und Umgebung

  $$\begin{aligned} P_{LOU} &= k_a(T_o - T_u) \\ P_{SOU} &= \alpha A_{ok}(T_o - T_u) + \varepsilon\sigma A_{os}(T_o^4 - T_u^4) \\ P_{OW} &= \alpha A_{wk}(T_o - T_w) + \varepsilon\sigma A_{ws}(T_o^4 - T_w^4) \end{aligned}$$

  - Parameter aus der Maske
  - Temperaturen in K (wegen T^4)!

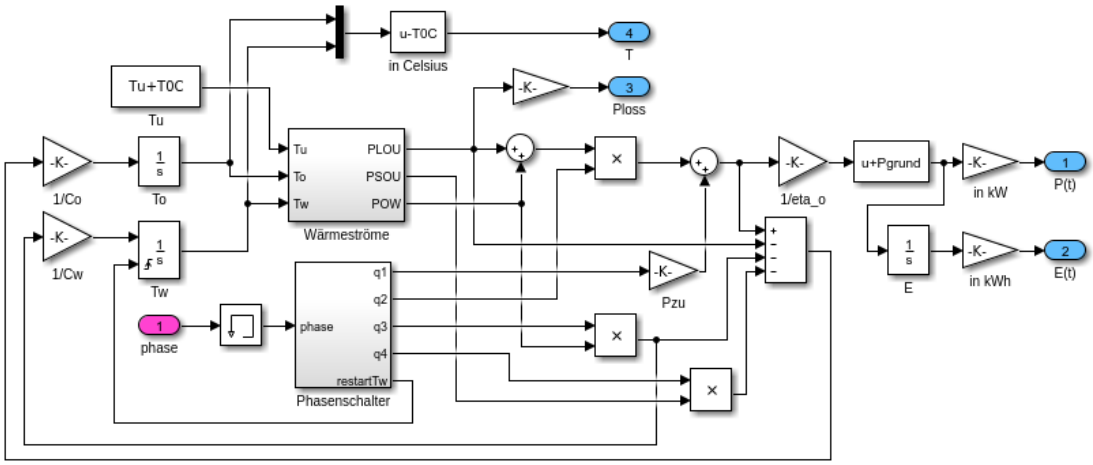
  Berechnung der Temperaturen mit

  $$\begin{aligned} C_o\dot{T}_o &= P_{heat} - P_{loss} \\ C_w\dot{T}_w &= P_{OW} \end{aligned}$$

  P_{heat}, P_{loss}, P_{OW} jeweils abhängig von der Phase, insbesondere

  $$P_{heat} = \begin{cases} P_{zu} & \text{heatUp} \\ P_{loss} & \text{hold} \\ 0 & \text{sonst} \end{cases}$$

  in Simulink



- Ausgänge von Phasenschalter immer 0 oder 1 zu geeigneten Phasen
- trickreich, aber richtig (nachprüfen!)

- Modell-Varianten:

  mehrere Implementierungen der einzelnen Schichten
  - unterschiedliche Detailgrade oder Modellierungsmethoden
  - z.B. vereinfachtes PC-Modell, Physical Modeling für PP

  Zweck
  - nicht immer die komplexesten Teilmodelle im riesigen Gesamtsystem sinnvoll
  - Auswahl der Komponenten nach aktuellem Simulationszweck
  - untersuchen: welche Komponente ist nötig für bestimmte Fragestellung
  - z.B. Energieverbrauch einer Maschine eher klein → PP kann einfacher modelliert werden

  Beschreibung des Gesamtmodells incl. (vieler!) verschiedener Varianten
  - aktuelles Forschungsthema
  - ein spannender Ansatz: SES/MB-Framework [25]

- Aufgaben:

  Aufgabe 24

  Aufgabe 25

Aufgaben

Aufgabe 1

- Bestimmen Sie für das Modell `schule3` die Mittelwerte und Standardabweichungen der Klassen- und Abistärken. Untersuchen Sie dabei den Einfluss verschieden langer Einlaufphasen.
- Testen Sie die Annahme, die Klassen- und Abistärken seien normalverteilt.

Aufgabe 2

- In Hannon/Ruth "Dynamic Modeling" wird ein einfaches Modell für die Ausbreitung einer Infektionskrankheit beschrieben: Man teilt die Bevölkerung in vier Gruppen ein, deren Größe jeweils durch eine Zustandsgröße beschrieben wird:

N	gesund, nicht immun
A	infiziert und ansteckend, aber noch nicht krank
K	krank
I	gesund und immun

- Änderungen in der Gruppenzugehörigkeit geschehen durch Geburt b, Ansteckung a, Erkrankung k, Gesundung g und Tod t; sie werden modelliert durch

  $b = b_0$

  $a = \alpha(A + K)N$

  $k = \beta A$

  $g = \gamma K$

  $t = \delta K$

  mit folgenden Parameterwerten:

Geburtenzahl	$b_0 = 5000$
Ansteckungsrate	$\alpha = 2 \cdot 10^{-6}$
Erkrankungsrate	$\beta = 1$
Gesundungsrate	$\gamma = 0.9$
Sterberate	$\delta = 0.1$

- Geht man von einer Zeitentwicklung in festen Takten (Zeiteinheiten) aus, lauten die Entwicklungsgleichungen

  $N(k + 1) = N(k) + b - a$

  $A(k + 1) = A(k) + a - k$

  $K(k + 1) = K(k) + k - g - t$

  $I(k + 1) = I(k) + g$

a. Implementieren Sie das Modell und studieren Sie die Entwicklung für die Anfangswerte

   $N(0) = 1 \cdot 10^6$, $A(0) = 1$, $K(0) = 0$, $I(0) = 0$.

b. Ersetzen Sie die feste Geburtenzahl durch eine entsprechende Rate und berücksichtigen Sie natürliche (nicht krankheitsbedingte) Tode.

c. Sorgen Sie dafür, dass alle Gruppengrößen ganzzahlig sind und führen Sie zufällige Schwankungen bei den Raten ein.

Aufgabe 3

- Implementieren Sie die stochastische Zustandsraum-Darstellung für `KlasseC` gemäß dem allgemeinen Muster in Simulink. Erstellen Sie daraus eine Variante des Modells `schule3` und vergleichen Sie die Ergebnisse mit der bisherigen Version.
- Erstellen Sie eine stochastische Zustandsraum-Darstellung für `KlasseD` und implementieren Sie diese gemäß dem allgemeinen Muster. Reproduziert die entsprechende Variante von `schule4` die bisherigen Ergebnisse?

Aufgabe 4

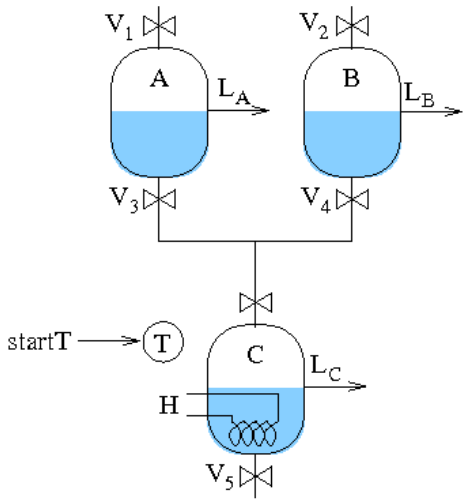
a. Erstellen Sie ein Modell `fehlerdetektor2` mit leicht geändertem Verhalten: Bei einer 1 im Zustand S3 gibt es 0 aus und wechselt nach S1, nicht nach S3. Die Idee dahinter: Nach drei Fehlern in Folge wird manuell aufgeräumt, man ist wieder bei 0. Der nächste Fehler ist also "der erste".

b. Untersuchen Sie mit den Modellen `fehlerdetektor1A` und `fehlerdetektor2` jeweils die Wahrscheinlichkeit p_3, drei Fehler hintereinander zu erhalten. Verwenden Sie dazu Einzelfehler-Wahrscheinlichkeiten $p = 0.3$ und $p = 0.03$.

c. Betrachten Sie die beiden Modellvarianten als Markov-Ketten. Berechnen Sie die Gleichgewichtsverteilungen und damit die exakte Antwort auf b.

Aufgabe 5

- In der Verfahrenstechnik werden häufig Batch-Prozesse eingesetzt, um chemische Prozesse kontrolliert ablaufen zu lassen:



- Im Beispielprozess werden zwei Stoffe in den Behältern A und B in einen dritten Behälter C gefüllt, wo sie bei Wärmezugabe reagieren und das Produkt abläuft. Dazu werden drei Sensoren L_A, L_B, L_C zur Messung der Füllstände eingesetzt und fünf Ventile V_1, ... V_5 sowie eine Heizung H angesteuert. Außerdem wird ein Timer T verwendet, der beim Einschalten der Heizung gestartet wird und mit T=1 anzeigt, dass der Prozess in C beendet ist.

- Alle Behälter sind zu Beginn leer. Der Prozess verläuft dann in folgenden Schritten:
  1. Behälter A und B werden gefüllt.
  2. A wird in C entleert, bis $L_C = 0.5$.
  3. B wird in C entleert, bis $L_C = 1$.
  4. C wird für eine feste Zeit beheizt, währenddessen werden A und B wieder aufgefüllt.
  5. C wird entleert und der Prozess wiederholt sich, beginnend mit 2.

- Der Einfachheit wird angenommen, dass Behälter A und B mindestens halb so groß sind wie Behälter C und dass der Prozess in C (Schritt 4) lange genug dauert, dass A und B währenddessen befüllt werden können.

a. Erstellen Sie einen Zustandautomaten zur Steuerung dieses Batch-Systems und implementieren Sie ihn zusammen mit interaktiven Eingabe-Elementen für die Füllstandsmesser (am einfachsten als Schieberegler) und den Timer.

b. Ersetzen Sie die Eingabe-Elemente durch ein einfaches Modell, das das Verhalten des Systems nachbildet.

Aufgabe 6

- Mit dem JK-Flipflop kann man einen einfachen Zähler aufbauen: Man verbindet einen Ausgang mit dem Takteingang des nächsten Flipflops und interpretiert die Q-Ausgänge als Binärziffern. Nachteil dieses **Asynchronzählers**: Die Schaltzeiten der Flipflops addieren sich. Daher baut man **Synchronzähler**, bei denen alle Flipflops denselben Takt bekommen und die Umschaltung durch geschickt gewählte Logikbausteine gesteuert wird.
- Erstellen Sie mit jeweils 4 JK-Flipflops
  - einen asynchronen Vorwärtszähler,
  - einen synchronen Vorwärtszähler,
  - einen synchronen Rückwärtszähler,
  - einen synchronen Vorwärts-/Rückwärtszähler, dessen Zählrichtung durch einen zusätzlichen Eingang bestimmt wird.

Aufgabe 7

- Entwickeln Sie eine Steuerung für einen Getränkeautomaten, der am Münzeinwurf Münzen von 10, 20, 50 Cent und 1 Euro annimmt. Man kann wählen zwischen
  - Cola (1.30 €)
  - Wasser (1.10 €)
  - Kaffee (2.20 €)
  - Kakao (2.00 €)
- Sobald die eingeworfene Summe den erforderlichen Betrag übersteigt, wird bei Cola oder Wasser eine entsprechende Flasche ausgeworfen und das Restgeld ausgegeben. Danach kann ein neues Getränk gewählt werden.
- Bei Kaffee oder Kakao wird ein Becher ausgeworfen und mit dem entsprechenden Pulver befüllt sowie das Restgeld ausgegeben. Gleichzeitig wird Wasser erhitzt. Ist das Wasser heiß genug, wird es in den Becher gefüllt und ein neues Getränk kann gewählt werden.
- Fügen Sie noch interaktive Komponenten zur Eingabe und Anzeige der Ausgaben hinzu und simulieren Sie das Ganze. Können Sie den Automaten verwirren?

Aufgabe 8

- Lösen Sie Aufgabe 5 mit Hilfe einer Statechart. Dabei sollen die dort vorausgesetzten Einschränkungen über Größe und Füllzeit der Behälter aufgehoben sein.

Aufgabe 9

- Zwar schickt das einfache Kreuzungsmodell einige Wagen auf Kreisbahnen, bis sie endlich herauskommen, aber es interessiert hier nur der Netto-Effekt: Wie viele Wagen fahren aus den einzelnen Richtungen ein bzw. kommen aus den einzelnen Richtungen heraus? Ziel der Aufgabe ist es, beim Kreuzungsmodell durch geschickte Parameterwahlen ein gewünschtes Verhalten zu erzielen.

  a. Betrachten Sie ein einzelnes von N einlaufendes Fahrzeug und berechnen Sie die Wahrscheinlichkeiten, dass es die Kreuzung in Richtung W, E oder N verlässt. Gehen Sie von einer ansonsten leeren Kreuzung aus und nehmen Sie an, dass bei den Wahlmöglichkeiten die Wahrscheinlichkeiten jeweils gleich groß sind. Berechnen Sie die entsprechenden Wahrscheinlichkeiten auch für die beiden anderen Einfahrmöglichkeiten.

  b. Nehmen Sie nun an, dass auf Platz X die Wahrscheinlichkeit für die Ausfahrt p_X beträgt (X = B, C, D). Wie groß sind nun die einzelnen Abbiege-Wahrscheinlichkeiten? Tipp: Die Formeln werden deutlich übersichtlicher, wenn man als Abkürzung definiert:

  $$q_X := 1 - p_X, \qquad X = B, C, D$$

  c. Aus W komme ein Verkehrsstrom Q_W = 10 Fahrzeuge/min, von denen 70% nach E fahren und 30% nach N abbiegen, aus E Q_E = 8 Fahrzeuge/min, von denen 80% nach W fahren und 20 % nach N abbiegen, sowie aus N Q_N = 6 Fahrzeuge/min, von denen 60% nach W und 40% nach E fahren, in Formeln

  $$\begin{aligned} Q_W &= Q_{WE} + Q_{WN} = 7.0 + 3.0 = 10.0 \\ Q_E &= Q_{EN} + Q_{EW} = 1.6 + 6.4 = 8.0 \\ Q_N &= Q_{NW} + Q_{NE} = 3.6 + 2.4 = 6.0 \end{aligned}$$

  mit entsprechenden Werten für alle vorkommenden Größen. Daraus ergeben sich die gesamten ausfahrenden Ströme A_X für die einzelnen Richtungen.

  Geben Sie mit Hilfe der Ergebnisse aus b.) Formeln an, wie man bei gegebenen Ausfahr-Wahrscheinlichkeiten p_X die Ausgangsströme aus den Eingangsströmen erhält (unter der Annahme, dass sich die Ströme nicht beeinflussen). Bei bekannten Strömen können daraus die Werte der p_X numerisch bestimmt werden. Welche Werte ergeben sich hier?

  Achtung: Der Newton-Solver hat Probleme, da die Jacobi-Matrix singulär ist (prüfen!). Was bedeutet das und wie kann man das Problem lösen?

  d. Überprüfen Sie die gefundene Lösung explizit mit Hilfe des Modells `Abzweigung1`. Schicken Sie dabei alle drei Eingangsströme getrennt los, so dass sie sich nicht beeinflussen.

  e. Wie ändert sich das Ergebnis, wenn man die Ströme gleichzeitig losschickt, aber die Abzweigung mit Ampelsteuerung verwendet?

Aufgabe 10

- Erweitern Sie das Kreuzungsmodell um eine Linksabbiegerspur für die von W ankommenden Fahrzeuge incl. eigener Ampel. Nutzen Sie die Ampelsignale, um Kreisfahrer innerhalb der Kreuzung zu verhindern und testen Sie, ob es funktioniert. Stellen Sie dann die Parameter so ein, dass sich die Verkehrsströme aus Aufgabe 9c ergeben.

Aufgabe 11

a. Herr Dr. Pünktlich behandelt Patienten nur nach Terminvereinbarung, Termine sind im Abstand von 10 Minuten von 8 Uhr bis 10:50 Uhr. Es werde angenommen, dass alle Patienten pünktlich kommen. Die Behandlungszeit beträgt im Schnitt ebenfalls 10 min, ist aber exponentiell verteilt.

   Untersuchen Sie die durchschnittliche Wartezeit eines Patienten, die maximale Wartezimmerbelegung, die Auslastung des Arztes und den Überhang (Arbeitszeit nach 11 Uhr). Führen Sie dazu 100 Läufe für je einen Tag durch und bestimmen die Mittelwerte.

b. In der Praxis seiner Kollegin Frau Dr. Locker kommen Patienten unangemeldet, im Schnitt alle 10 min mit exponentiell verteilten Zeitabständen. Einlass ist ebenfalls von 8:00 Uhr bis 10:50 Uhr. Berechnen Sie die gleichen Kennwerte wie in a. und vergleichen Sie die Ergebnisse. Geben Sie auch die Anzahl behandelter Patienten an.

Aufgabe 12

- Der Durchsatz an einer Werkzeugmaschine soll verdoppelt werden. Dazu kann entweder eine zweite identische Maschine angeschafft werden oder die vorhandene durch eine neue, doppelt so schnelle ersetzt werden. Die Gesamtkapazität der Warteschlange für ankommende Werkstücke sei K, die bessere Lösung sei die mit geringerer Zahl von Blockaden.
- Konkret sei der (neue) Ankunftsprozess exponentiell mit Rate $\lambda = 2/\text{min}$, die Bearbeitungszeit der alten Maschine sei normalverteilt mit $m = 0.8$ min und $\sigma = 0.16$ min, die der schnelleren Maschine $m = 0.4$ min und $\sigma = 0.08$ min. Die Kapazität betrage $K = 14$.
- Vergleichen Sie bzgl. der Zahl der Blockaden folgende Prozesse:
  a. eine Queue mit Kapazität K und eine schnelle Maschine,
  b. eine Queue mit Kapazität K und zwei langsame Maschinen,
  c. zwei Queues je mit Kapazität K/2, die abwechselnd bedient werden, und zwei langsame Maschinen.

Aufgabe 13*

- Schreiben Sie ein Matlab-Programm, das das DES-Modell und die Eventroutinen des D/D/1-Beispiels implementiert und reproduzieren Sie damit die obigen Ergebnisse.

Aufgabe 14

- Betrachten Sie statt des D/D/1-Beispiels ein M/M/1-System mit Mittelwerten `tG = 1` und `tS = 0.83`. Der Generator erzeuge nur 10 Entitäten, das Programm stoppt, wenn alle 10 Entitäten das System verlassen haben.
  a. Formulieren Sie entsprechende Eventroutinen und spielen Sie das Verhalten mit einer Event-Tabelle manuell nach.
  b. * Passen Sie das Matlab-Programm aus Aufgabe 13 für dieses Modell an.

Aufgabe 15

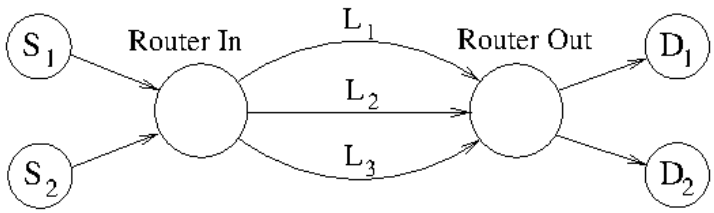
- Überprüfen Sie die Ergebnisse von `testOvenA` und `testOven` bis t = 13 mit Hilfe einer Entitäten-Tabelle, analog zur Prüfung von `testConveyor`.
- Spielt die Reihenfolge gleichzeitiger Ereignisse eine Rolle?

Aufgabe 16

- Beide Maschinen der Fertigungsstraße sollen von einer einzigen Person betreut werden. Dazu wird Maschine 2 so umgebaut, dass sie ebenfalls vier Teile in zwei Zeiteinheiten bearbeitet. Teile erreichen eine Maschine nur, wenn der Betreuer gerade da ist. Er wechselt abhängig vom Zustand der Maschinen zwischen ihnen hin und her.
- Erweitern Sie das Modell um den Betreuer und vergleichen Sie das Ergebnis. Kann man den theoretisch möglichen Durchsatz erreichen?

Aufgabe 17

- Erweitern Sie das Modell `sendmessage2D` um einen weiteren Generator S_2, der seine Message über das gleiche Netzwerk an einen Empfänger D_2 schickt



- Fügen Sie auch Displays für die Laufzeiten der beiden Messages hinzu.
- Vergleichen Sie die Ergebnisse für Startzeiten t = 15 und t = 6 der 2. Message (alle anderen Werte wie bisher).

Aufgabe 18

- Verifizieren Sie das Modell `supplychain1`, indem Sie manuell eine Tabelle ähnlich wie bei `testDistributor2` erstellen und sie mit dem Ergebnis der Simulation vergleichen. Wählen Sie dazu geschickte Parameterwerte, so dass die wichtigsten Ereignisse frühzeitig auftreten, insbesondere eine verzögerte Bestellung in einer Fabrik und eine nicht ausgeführte Bestellung beim Distributor.
- Setzen Sie die `Initial waiting time` der Distributoren auf 24.1/24.2/24.3, um eine zu große Zahl gleichzeitiger Events zu verhindern. Dadurch wird der hohe Grad an Zufall (welche Order wird bei welcher Fabrik zuerst ausgeführt) verringert und eine manuelle Vorausberechnung erst möglich.
- Hinweis: Durch Spielen mit der globalen `seed` kann man das Auftreten bestimmter Ereignisse etwas früher erreichen und erspart sich damit übermäßig lange Tabellen.

Aufgabe 19

- Die Lagerhaltung in den `Distributor`- und `Factory`-Komponenten des Versorgungsketten-Modells wird bei zunehmender Zahl an Produkt-Typen sehr umständlich und unübersichtlich. Vereinfachen Sie dies, indem Sie in den entsprechenden Komponenten jeweils nur einen `Entity Store` vorsehen, der die Produkte aller Typen enthält.

Aufgabe 20

- Die Datei `dataEx20.dat` enthält Ausfallzeiten einer Maschine. Finden Sie eine Verteilungsfunktion, die die Verteilung dieser Werte näherungsweise beschreibt.

Aufgabe 21

- Das Modell `fertigungskette2` ist eine stochastische Version von `fertigungsketteB` mit folgenden Änderungen:
  - Der Generator hat exponentiell verteilte Ankunftsintervalle, mean = 0.7,
  - Maschine 1/2 haben normalverteilte Servicezeiten mit $t_S = 2$, $\sigma_S = 0.3$ für Maschine 1 und $t_S = 1$, $\sigma_S = 0.2$ für Maschine 2.

  a. Bestimmen Sie 95%-Konfidenzintervalle für die mittlere Belegung von Eingangs- und Zwischenlager mit dem Replication/Deletion-Verfahren.

  b. Wie groß müssen die Lager sein, damit die Wahrscheinlichkeit, dass ein Lager innerhalb von 1000 Schritten nicht ausreicht, kleiner ist als 2%?

  c. Überprüfen Sie Ihr Ergebnis mit Hilfe einer Modellvariante, die die in b. ermittelten Lagergrößen verwendet und zählt, wie oft ein eingehendes Werkstück am Eingangs- bzw. Zwischenlager abgewiesen werden musste.

Aufgabe 22

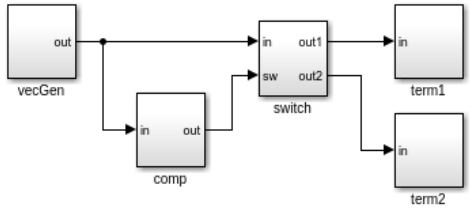
a. Erstellen Sie ein AtomicDEVS-Modell für ein negativ getriggertes JK-Flipflop (vgl. kap3-2).

b. Erstellen Sie einen Block, der beliebige vorgegebene binäre Signale erzeugt. Als Parameter habe er den Startzustand `z0` und einen Vektor `tVec` von Zeiten, zu denen der Ausgangswert umspringt. Verwenden Sie ihn, um das JK-Flipflop damit zu testen.

c. Erstellen Sie ein Schieberegister aus vier JK-Flipflops als CoupledDEVS-Modell und testen Sie es.

Aufgabe 23

- Untersuchen Sie folgendes System:



- Dabei sind die einzelnen Komponenten wie folgt definiert:
  - Vectorgenerator `vectorgen`
    - 1 Ausgang, keine Eingänge
    - zwei (gleich lange) Parametervektoren `tVec`, `yVec`
    - gibt zu den Zeiten `tVec(i)` jeweils den Wert `yVec(i)` aus
  - Vergleichsbaustein `comparator`
    - 1 Eingang, 1 Ausgang
    - gibt 1 am Ausgang aus, wenn Eingang ≥ 0, sonst 0
  - `Umschalter switch`
    - 2 Eingänge (`in`, `sw`), 2 Ausgänge (`out1`, `out2`)
    - gibt Eingangswert von `in` nach `out1` bzw. `out2` aus, wenn `sw = 0` bzw. `sw = 1`
  - Terminator `terminator1`
    - wie bisher, aber mit zwei Zustandsgrößen `val`, `ni`
    - `val` speichert den einlaufenden Wert
    - `ni` speichert Anzahl der eingelaufenen Werte
    - hilfreich, um Ausgangswerte anzeigen zu können

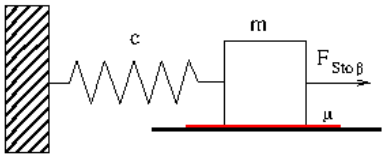
a. Erstellen Sie alle AtomicDEVS-Komponenten und testen Sie sie in möglichst einfachen Testprogrammen.

b. Erstellen Sie das Gesamtmodell als CoupledDEVS-Modell und testen Sie es. Funktioniert es? Falls nein: Warum nicht?

c. * Erstellen Sie eine neue Switch-Komponente, um das Problem aus b. zu umgehen und ein funktionierendes Gesamtmodell zu erhalten (vgl. [14]).

Aufgabe 24

- Ein wichtiges Beispiel für ein System, das zwischen zwei verschiedenen Verhaltensweisen umschaltet, ist eine Masse mit Gleit- und Haftreibung. Konkret sei eine Masse an einer Feder gegeben, die auf einer Unterlage gleitet bzw. haftet. Zusätzlich kann noch ein kurzer Kraftstoß $F_{Stoß}(t)$ aufgebracht werden, um die Masse nach dem Haften wieder in Bewegung zu bringen:



- Die Bewegungsgleichung lautet dann

$$m\ddot{x} = -cx + F_R + F_{Stoß}(t)$$

mit der Reibungskraft F_R.

- Solange die Geschwindigkeit $v \neq 0$, ist die Reibung durch die Gleitreibungskraft

$$F_g = -\mu_g mg\,\mathrm{sign}(v)$$

gegeben, bei $v = 0$ wird stattdessen die (größere) Haftreibungskraft F_h auftreten. Sie ist der Summe F_{ext} der äußeren Kräfte (hier: Feder- und Stoßkraft) entgegengesetzt, solange diese die maximale Haftreibungskraft

$$F_{h,max} = \mu_h mg$$

nicht überschreitet, aber nie betragsmäßig größer als $F_{h,max}$. Dies kann man z. B. so in Formeln fassen:

$$F_h = -\mathrm{sign}(F_{ext})\,\min(F_{h,max}, |F_{ext}|)$$

Insgesamt hat man dann die Reibungskraft

$$F_R = \begin{cases} F_g & | \quad v \neq 0 \\ F_h & | \quad |F_{ext}| < \mu_h mg \end{cases}$$

- Erstellen Sie ein Simulink-Model für die Bewegung der Masse bei gegebener Anfangsauslenkung x_0 und -geschwindigkeit v_0. Wählen Sie $F_{Stoß}$ so, dass die Masse nach einer Haftphase wieder in Bewegung kommt und verwenden Sie folgende Parameterwerte:
  - m = 1 kg
  - g = 9.81 m/s^2
  - c = 100 N/m
  - μ_g = 0.8
  - μ_h = 1.6
  - x_0 = 0.8 m
  - v_0 = 3 m/s

Aufgabe 25

- Erstellen Sie eine Variante `ovenhyb2` des Ofenmodells, bei dem die Prozesskontroll-Schicht nicht mit Stateflow, sondern mit SimEvents implementiert wird. Auf eine Cooldown-Phase wird verzichtet, ansonsten soll sich das Modell wie vorher verhalten.

Anhang

Literatur

1. U. Hedtstück: Simulation diskreter Prozesse: Methoden und Anwendungen
   Springer Vieweg 2013

2. G. Hübner: Stochastik
   Vieweg+Teubner, 5. Aufl. 2009

3. C. G. Cassandras, S. Lafortune: Introduction to Discrete Event Systems
   Springer, 3. Aufl. 2009

4. J. Lunze: Ereignisdiskrete Systeme
   De Gruyter Oldenbourg, 2. Aufl. 2012

5. A. M. Law: Simulation Modeling and Analysis
   Mcgraw-Hill Publ.Comp., 5. Aufl. 2014

6. W. Reisig: Understanding Petri Nets
   Springer 2013

7. W. D. Kelton, R. P. Sadowski, N. B. Zupick: Simulation with Arena
   McGraw-Hill, 6. Aufl. 2015

8. B. P. Zeigler, A. Muzy, E. Kofman: Theory of Modeling and Simulation
   Elsevier Academic Press, 3. Aufl. 2019

9. K. Gutenschwager, M. Rabe, S. Spieckermann, S. Wenzel: Simulation in Produktion und Logistik
   Springer, 2017

Nachweise

Text

1. L. M. S. Dias, A. A. C. Vieira, G. A. B. Pereira, J. A. Oliveira: Discrete simulation software ranking - a top list of the worldwide most popular and used tools. In: Proc. 2016 Winter Sim. Conf., p. 1060-1071, 2016 (online).
2. Junglas, P.: Probabilistic state space models – A theoretical framework with practical relevance, SNE Simulation Notes Europe **29**(1), p. 33-38 (2019) (online).
3. D. Harel: Statecharts: A visual formalism for complex systems. Science of Computer Programming, **8**, p. 231 - 274, 1987 (online).
4. G. Hamon: A Denotational Semantics for Stateflow. In: EMSOFT 2005: Proc. of the Fifth ACM Workshop on Embedded Software, pp. 164-172. Association for Computing Machinery, Jersey City, NJ (2005) (online).
5. G. Hamon, J. Rushby: Statecharts: An Operational Semantics for Stateflow. International Journal on Software Tools for Technology Transfer (STTT) **9** (5-6); Special section FASE'04/05, p. 447–456 (2007) (online).
6. A. Di Febbraro, D. Giglio, N. Sacco: Modular representation of urban traffic systems based on hybrid petri nets. In: Proc. Intelligent Transportation Systems, p. 866-871, 2001 (online).
7. A. Di Febbraro, D. Giglio, N. Sacco: On applying petri nets to determine optimal offsets for coordinated traffic light timings. In: Proc. of IEEE 5th International Conference on Intelligent Transportation Systems, p. 773-778, 2002 (online).
8. C. R. Vázquez et al: Hybrid Petri net model of a traffic intersection in a urban network. In: IEEE International Conference on Control Applications, p. 658-664, 2010 (online).
9. S. M. Tauböck: C14 Supply Chain Management - Definition. Simulation News Europe **11** (32/33), p. 42-43 (2001) (online).
10. Austermann, L., Junglas, P., Schmidt, J., Tiekmann, C.: Conceptional problems of transaction-based modeling and its implementation in SimEvents 4.4, SNE Simulation Notes Europe **27**(3), p. 137-142 (2017) (online).
11. H. Bossel: Systeme, Dynamik, Simulation: Modellbildung, Analyse und Simulation komplexer Systeme, Books on Demand, 2004
12. P. D. Welch: The statistical analysis of simulation results. The computer performance modeling handbook **22** (1983), 268-328.
13. A. C. H. Chow: Parallel DEVS: A parallel, hierarchical, modular modeling formalism and its distributed simulator. Transact. Soc. Comp. Simulation, **13**(2), 55-68, (1996).
14. F. Preyser, B. Heinzl, P. Raich, W. Kastner: Towards Extending the Parallel-DEVS Formalism to Improve Component Modularity. In: ASIM 2016, Workshop der ASIM/GI-Fachgruppen STS/GMMS, Lippstadt, p. 83-89 (2016) (online).
15. F. Preyser, B. Heinzl, W. Kastner: RPDEVS: Revising the Parallel Discrete Event System Specification. In: Proc. 9th Vienna Int. Conf. on Mathematical Modelling, Wien, p. 269-274, (2018) (online).
16. T. Schwatinski. T. Pawletta: An Advanced Simulation Approach for Parallel DEVS with Ports. In: Proc. 2010 Spring Simulation Multiconference, p. 147-154 (2010) (online).
17. Homepage of MatlabDEVS.
18. T. Pawletta, C. Deatcu, O. Hagendorf, S. Pawletta, G. Colquhoun: DEVS-Based Modeling and Simulation in Scientific and Technical Computing Environments. In: Proc. of DEVS Integrative M&S Symposium (DEVS'06) - Part of the 2006 Spring Simulation Multiconference (SpringSim'06), p. 151-158 (2006) (online).
19. C. Deatcu, B. Freymann, T. Pawletta: PDEVS-based hybrid system simulation toolbox for MATLAB. In Proc. Symp. Theory of Modeling & Simulation. Society for Computer Simulation International, p. 989-1000 (2017) (online).

20. A. Körner, F. Breitenecker: State Events and Structural-dynamic Systems: Definition of ARGESIM Benchmark C21. SNE Simulation Notes Europe **26**(2), p. 117-122 (2016) (online).
21. P. A. Fritzson: Principles of Object-Oriented Modeling and Simulation with Modelica 3.3. Wiley & Sons, New York, 2015.
22. J.-P. Disselkamp, P. Junglas, A. Niehüser, P. Schönfelder: A Solution to ARGESIM Benchmark C21 'State Events and Structural-dynamic Systems' based on Modelica Components. SNE Simulation Notes Europe **28**(2), p. 39-48 (2018) (online).
23. J.-P. Disselkamp, P. Junglas, A. Niehüser, P. Schönfelder: Implementing the Argesim C21 benchmark with Modelica components. In: ASIM 2018, Workshop der ASIM/GI-Fachgruppen STS/GMMS, Heilbronn, p. 197-202 (2018) (online).
24. Th. Pawletta, A. Schmidt, P. Junglas: A Multimodeling Approach for the Simulation of Energy Consumption in Manufacturing. SNE Simulation Notes Europe **27**(2), p. 115-124 (2017) (online).
25. A. Schmidt: Variantenmanagement in der Modellbildung und Simulation unter Verwendung des SES/MB Frameworks. Diss. U. Rostock (2019) (online).

Beispielmodelle

- schule1.slx
- discSimLib.slx + Icons-Verzeichnis
- schule2.slx
- binomial.m
- startBinomial.m
- schule3.slx
- schule4.slx
- testKlasseD.slx
- fehlerdetektor1A.slx
- fehlerdetektor1B.slx
- rsflipflopFS.slx
- jkflipflopFS.slx
- schieberegisterA.slx
- schieberegisterB.slx
- schieberegisterC.slx
- fahrstuhlFS1.slx
- robcontrol1A.slx
- robcontrol1B.slx
- robcontrol2A.slx
- robcontrol2B.slx
- robcontrol3.slx
- DiscSimLib.mo
- BeschwerdeA.mo
- BeschwerdeB.mo
- BeschwerdeC.mo
- Abzweigung1A.mo
- Abzweigung1B.mo
- Abzweigung2A.mo
- TestStrasse.mo
- Strassennetz.mo
- singleserver1A.slx
- singleserver1B.slx
- singleserver1C.slx
- singleserver2A.slx
- singleserver2B.slx
- singleserver3A.slx
- singleserver3B.slx

- singleserver3C.slx
- queuecapacity1A.slx
- queuecapacity1B.slx
- queuecapacity1C.slx
- computeQueueCapacity.m
- reworking1A.slx
- reworking1B.slx
- reworking2A.slx
- reworking2B.slx
- reworking2C.slx
- reworking2D.slx
- testConveyor.slx
- testOvenA.slx
- testOvenB.slx
- fertigungsketteA.slx
- fertigungsketteB.slx
- sendmessage1A.slx
- sendmessage1B.slx
- sendmessage1C.slx
- sendmessage2A.slx
- sendmessage2B.slx
- sendmessage2C.slx
- sendmessage2D.slx
- sendmessage2E.slx
- testWholesalers.slx
- testFactory1.slx
- testFactory2.slx
- testFactory3.slx
- testFactory4.slx
- testDistributor1A.slx
- testDistributor1B.slx
- testDistributor1C.slx
- testDistributor2.slx
- supplychain1.slx
- singleserver4A.slx
- singleserver4B.slx
- runSingleserver4B.m
- kap8_3a.m

- kap8_3b.m
- kap8_3c.m
- kap8_3d.m
- fifo3A.slx
- fifi3B.slx
- am_discrete_template.m
- cm_discrete_template.m
- am_generator.m
- am_terminator.m
- am_serverA.m
- am_queueA.m
- model_singleserverA.m
- run_singleserver.m
- analyse_singleserver.m
- am_server.m
- am_queue.m
- model_singleserver.m
- huepfballA.slx
- huepfballB.slx
- huepfballC.slx
- huepfballD.slx
- FreeFallPendulum.mo
- ovenhyb1.slx
- dataEx20.dat

Prüfungsleistung

- Die Aufgaben im Skript haben unterschiedliche Punktezahlen gemäß folgender Tabelle:

Nr.	1*	2	3*	4*	5	6	7	8	9*	10	11	12	13*	14*	15*	16	17	18*	19	20*	21*	22	23*	24	25
Punkte	2	2	2	2	3	2	3	2	3	2	2	1	3	2+2	1	3	2	3	3	2	3	2	3+1	2	2

  - Die mit * markierten Aufgaben haben theoretische Anteile.
  - Bei Aufgabe 14 kann 14a (mit 2 Punkten) oder 14 a+b (mit 4 Punkten) gewählt werden.
  - Bei Aufgabe 23 kann 23a+b (mit 3 Punkten) oder 23 a-c (mit 4 Punkten) gewählt werden.
- Wählen Sie Aufgaben mit einer Gesamtpunktzahl von 10 (Zweiergruppe) bzw. 15 (Dreiergruppe). Dabei gelten folgende Einschränkungen:
  - Mindestens eine Aufgabe hat theoretische Anteile,
  - aus jeder der drei Gruppen (1-8), (11-19), (9,10,20-25) ist mindestens eine Aufgabe enthalten,
  - jede Aufgabe kann höchstens von einer Gruppe gewählt werden.
- Abzugeben sind eine schriftliche Ausarbeitung sowie ein Datenträger mit den lauffähigen Modellen und der Ausarbeitung als PDF-Datei.
- Beachten Sie auch die Erläuterungen zu Formalitäten bei Hausarbeiten.
- Hinweis: Die Beschreibung eines Submodells beginnt in der Regel mit der Außenansicht (dem Interface), d. h. seiner Aufgabe, den Ein- und Ausgängen sowie den Parametern, erst dann (falls nötig) mit der Innenansicht (der Implementierung).