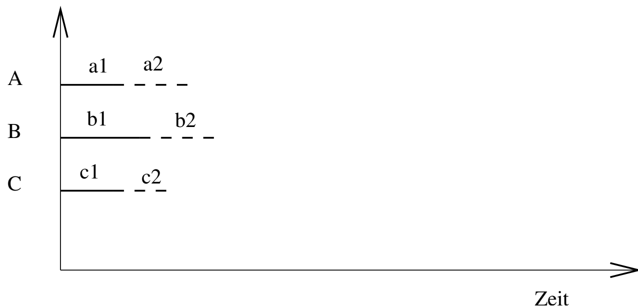
Kapitel 3

Parallele Prozesse

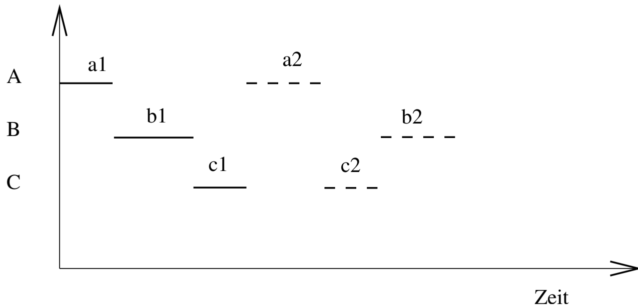
Prof. Dr. Rolf Hennicker

26.05.2011

Echte Parallelität



Quasi-(Pseudo-)Parallelität



Die Aktionen der einzelnen Prozesse werden bei Quasi-(Pseudo-)Parallelität miteinander "verzahnt" ausgeführt. Wir sprechen dann von **"Interleaving"**.

Beachte:

- Alle möglichen Verzahnungen müssen berücksichtigt werden.
- Die Reihenfolge der Aktionen eines Prozesses ist dieselbe wie bei echter Parallelität.

Die parallele Komposition von Prozessen wird im Folgenden durch Interleaving modelliert.

Parallele Komposition von Prozessen

Definition:

Sind $\mathsf{E}_1, \ldots, \mathsf{E}_n$ Prozessausdrücke, dann ist

$$(\mathsf{E}_1 \,||\, \mathsf{E}_2 \,||\, \ldots \,||\, \mathsf{E}_n)$$

ein Prozessausdruck (*parallele Komposition* von $\mathsf{E}_1, \ldots, \mathsf{E}_n$)
mit $\mathsf{FV}((\mathsf{E}_1 \,||\, \mathsf{E}_2 \,||\, \ldots \,||\, \mathsf{E}_n)) = \mathsf{FV}(\mathsf{E}_1) \cup \ldots \cup \mathsf{FV}(\mathsf{E}_n)$.

Wirkung:

Die (disjunkten) Aktionen von $\mathsf{E}_1, \ldots, \mathsf{E}_n$ werden verzahnt ausgeführt.

Deklaration paralleler Prozesse:

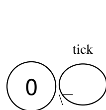
Seien E und F Prozessausdrücke und sei $\mathsf{P} \in \mathsf{PID}$ ein Prozessidentifikator mit $\mathsf{P} \notin \mathsf{FV}((\mathsf{E} \,||\, \mathsf{F}))$. Statt der Prozessdeklaration "$\mathsf{P} = (\mathsf{E} \,||\, \mathsf{F})$." schreiben wir dann

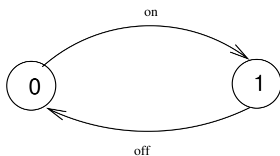
$$||\, \mathsf{P} = (\mathsf{E} \,||\, \mathsf{F}).$$

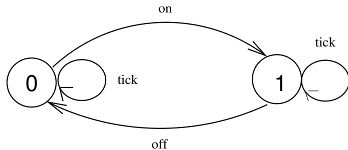
Beispiel:

$$\begin{aligned}&\text{CLOCK} = (\text{tick} \rightarrow \text{CLOCK}).\\&\text{RADIO} = (\text{on} \rightarrow \text{off} \rightarrow \text{RADIO}).\\&\|\text{CLOCK_RADIO} = (\text{CLOCK}\|\text{RADIO}).\end{aligned}$$

Zustandsmaschinen:







CLOCK

RADIO

(CLOCK||RADIO)

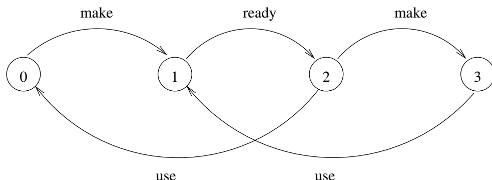
Prozessinteraktionen

- Prozessinteraktionen werden durch *gemeinsame* Aktionen ("shared actions") modelliert.
- Parallele Prozesse, die gemeinsame Aktionen haben, müssen diese gemeinsam ausführen, d.h. sie müssen sich synchronisieren.
- Die Synchronisation schränkt die möglichen Abläufe der parallelen Komposition ein.

Beispiel:

MAKER = (make $\rightarrow$ ready $\rightarrow$ MAKER).
USER = (ready $\rightarrow$ use $\rightarrow$ USER).
$\|$MAKER_USER = (MAKER $\|$ USER).

Zugehöriges LTS:



Variante:

MAKER produziert erst dann weiter, wenn der USER die Benutzung bestätigt hat.

$\text{MAKER2} = (\text{make} \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKER2}).$
$\text{USER2} \quad = (\text{ready} \rightarrow \text{use} \rightarrow \text{used} \rightarrow \text{USER2}).$
$\|\text{MAKER_USER2} = (\text{MAKER2} \parallel \text{USER2}).$

Zugehöriges LTS:

Umbenennung von Aktionen

Die Umbenennung von Aktionen dient (vor allem)

- zur Erstellung verschiedener Kopien eines Prozesses,
- als Hilfsmittel zur Synchronisation paralleler Prozesse.

Allgemeine Voraussetzung: ACT = Labels $\cup \{\tau\}$

Definition:

Sei E ein (evt. paralleler) Prozessausdruck und seien $a_1, \ldots, a_k$ und $n_1, \ldots, n_k$ Aktionsnamen verschieden von τ. Dann ist

$$E\{n_1/a_1, \ldots, n_k/a_k\}$$

ein Prozessausdruck ("Relabelling") mit $FV(E\{n_1/a_1, \ldots, n_k/a_k\}) = FV(E)$.

Wirkung:

Im LTS von E werden die Aktionsnamen $a_1, \ldots, a_k$ ersetzt durch $n_1, \ldots, n_k$.

Beispiel für Umbenennung zur Erstellung von Prozess-Kopien:

CLIENT $=$ (call $\rightarrow$ wait $\rightarrow$ continue $\rightarrow$ CLIENT).
$\|$TWOCLIENTS $=$ (a:CLIENT $\|$ b:CLIENT).

Dabei ist a:CLIENT eine Kurznotation für
CLIENT $\{$a.call/call, a.wait/wait, a.continue/continue$\}$.

Abkürzende Schreibweisen für parallele Kompositionen von Prozesskopien

Sei `range ID = 1..N`

Die folgenden Ausdrücke

```
forall[i:1 ...N] a[i]:E
forall[i:ID] a[i]:E
a[i:1 ...N]:E
a[ID]:E
```

bezeichnen alle den Prozess

(`a[1]:E` ‖ ... ‖ `a[N]:E`)

Der Bezeichner a kann auch weggelassen werden.
Zum Beipiel bezeichnet `[ID]:E` den Prozess

(`[1]:E` ‖ ... ‖ `[N]:E`)

Synchronisation von Prozessen durch Umbenennung

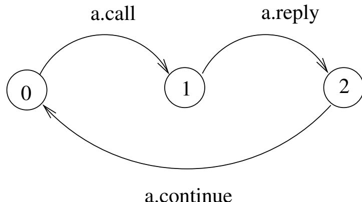
$$(E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n) \,/\, \{n_1/a_1, \ldots, n_k/a_k\} =_{def}$$
$$(E_1 \,\{n_1/a_1, \ldots, n_k/a_k\} \,||\, \ldots \,||\, E_n \,\{n_1/a_1, \ldots, n_k/a_k\})$$

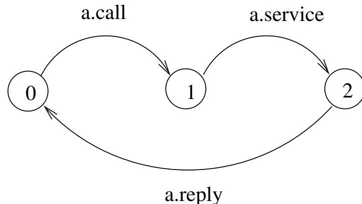
Beispiel für Umbenennung zur Synchronisation von Prozessen:

CLIENT = (call $\rightarrow$ wait $\rightarrow$ continue $\rightarrow$ CLIENT).
SERVER = (request $\rightarrow$ service $\rightarrow$ reply $\rightarrow$ SERVER).
$\|$CLIENT_SERVER = (CLIENT $\|$ SERVER) / {call/request, reply/wait}.

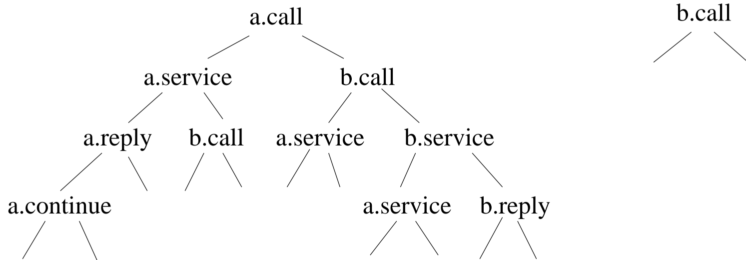
Beispiel für Synchronisation von Prozess-Kopien:

$\|$TWOCLIENTS_SERVER = (a:CLIENT $\|$ b:CLIENT $\|$ a:SERVER $\|$ b:SERVER) / {a.call/a.request, b.call/b.request, a.reply/a.wait, b.reply/b.wait}.





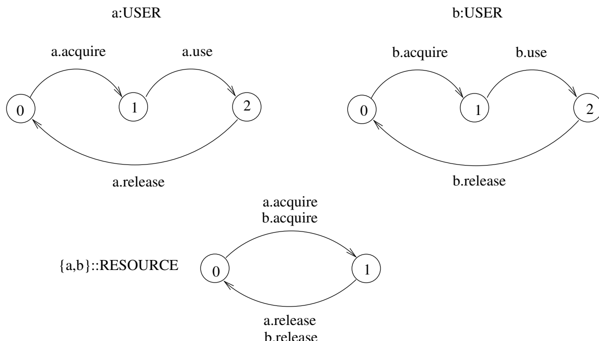
Traces:



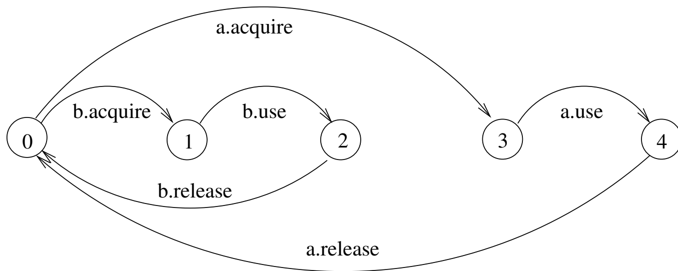
Das LTS ist zu komplex: Man will nur bestimmte Sichten sehen.
Zum Beispiel sind interne Abläufe für den Benutzer, der das System von außen betrachtet, uninteressant.

Beispiel (Resource-Sharing):

USER $=$ (acquire $\rightarrow$ use $\rightarrow$ release $\rightarrow$ USER).
RESOURCE $=$ (acquire $\rightarrow$ release $\rightarrow$ RESOURCE).
$\|$RESOURCE_SHARE $=$ (a:USER $\|$ b:USER) $\|$ $\{$a,b$\}$::RESOURCE).



RESOURCE_SHARE



Verbergen von Aktionen

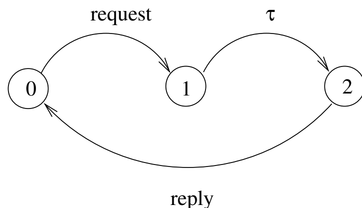
Das Verbergen von Aktionen (“Hiding“) dient zur Abstraktion von Aktionen, die unter einem bestimmten Gesichtspunkt “nicht relevant“ sind.

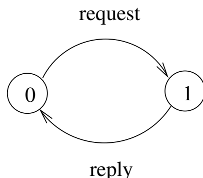
Definition:
Sei E ein (event. paralleler) Prozessausdruck und sei $H \subseteq$ Labels eine Menge von Aktionsnamen.
Dann ist $E \setminus H$ ein Prozessausdruck (“Hiding“) mit $FV(E \setminus H) = FV(E)$.

Wirkung:
Die Aktionen aus H werden verborgen und im LTS von E in eine spezielle Aktion τ (tau) umbenannt. τ heißt “unsichtbare“ (unbeobachtbare, stille, interne) Aktion.

Beispiel:

$\|$SERVER2 = SERVER $\setminus$ {service}.





Bei einem Prozess mit Hiding kann neben dem LTS auch das *minimale, beobachtbar äquivalente* LTS berechnet werden.

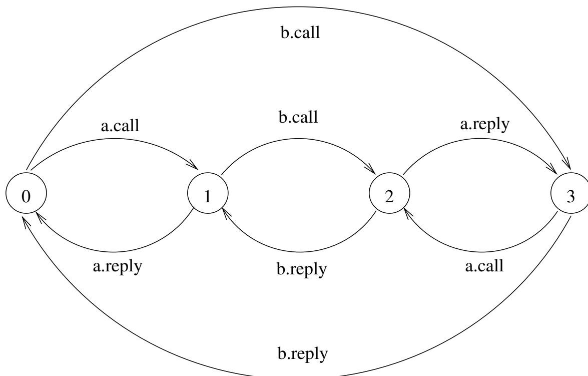
Bemerkung:

- Zumeist wird “Hiding“ nach der parallelen Komposition angewandt, um von der Komplexität eines parallelen Systems zu abstrahieren.
- Wird es vorher angewandt, dann darf bei der parallelen Komposition nicht bzgl. τ synchronisiert werden; τ ist intern und daher keine gemeinsame Aktion parallel laufender Prozesse.

Beispiel:

$\|$TCLIENTS_SERVER =
    TWOCLIENTS_SERVER \ {{a,b}.continue, {a,b}.service}.

Bezüglich beobachtbarer Äquivalenz minimalisiertes LTS:



Schnittstellen-Operator

$$\text{E}\,@\,\text{I}$$

wobei E ein Prozessausdruck und $\text{I} \subseteq$ Labels eine Menge von Aktionen ($\neq \tau$) ist.

Wirkung:

Alle Aktionen von E, die nicht in I vorkommen, werden verborgen.

Bemerkungen:

- I heißt *Schnittstelle* ("Interface") des Prozesses.
- Schnittstellen werden meist zur Beschreibung der von einem komplexen (parallelen) System angebotenen Dienste und zur Verbergung interner gemeinsamer Aktionen der Komponenten verwendet.
- Häufige Form von parallelen Prozessen mit Schnittstellen: $(\text{P}\|\text{Q})/\{\text{neu}/\text{alt}\}@\{\text{a}_1,\ldots,\text{a}_k\}$

Beispiel:

$(\text{MAKER}\|\text{USER})@\{\text{make},\text{use}\}$

Alphabeterweiterung

Definition:

(1) Sei $\mathsf{T} = (\mathsf{S}, \mathsf{A}, \Delta, \mathsf{q})$ ein LTS.
Dann heißt die Menge $\alpha\mathsf{T} =_{def} \mathsf{A} \setminus \{\tau\}$ das *Alphabet* von T.

(2) Sei E ein Prozessausdruck mit $\mathsf{lts}(\mathsf{E}) = \mathsf{T}$.
Dann heißt die Menge $\alpha\mathsf{E} =_{def} \alpha\mathsf{T}$ das *Alphabet* von E.

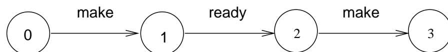
Beispiel: $\alpha((\mathsf{MAKER} \,\|\, \mathsf{USER}) \,@\, \{\mathsf{make}, \mathsf{use}\}) = \{\mathsf{make}, \mathsf{use}\}$

Definition:

Sei E ein Prozessausdruck und $\mathsf{B} \subseteq$ Labels eine Menge von Aktionen. Dann ist die *Alphabeterweiterung* $\mathsf{E} + \mathsf{B}$ ein Prozessausdruck mit $\mathsf{FV}(\mathsf{E} + \mathsf{B}) = \mathsf{FV}(\mathsf{E})$.

Beispiel:

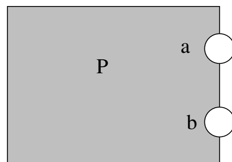
$\mathsf{FMAKER} = (\mathsf{make} \to \mathsf{ready} \to \mathsf{FMAKER}) + \{\mathsf{use}\}.$
$\mathsf{USER} = (\mathsf{ready} \to \mathsf{use} \to \mathsf{USER}).$
$\|\mathsf{FMAKER_USER} = (\mathsf{FMAKER} \,\|\, \mathsf{USER}).$



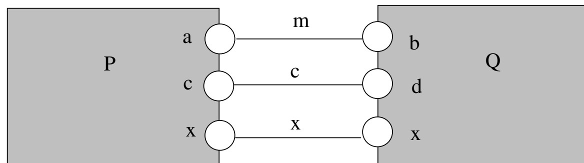
Strukturdiagramme

Strukturdiagramme zeigen den *strukturellen Aufbau* komplexer Systeme (Prozesse) mit Schnittstellen und (internen) Verbindungen zwischen Komponenten.

Strukturdiagramm eines Prozesses mit Alphabet {a,b}

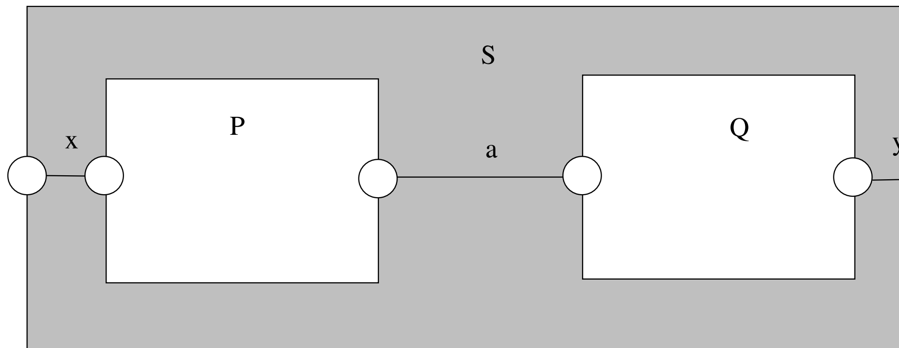


Strukturdiagramm von zwei interaktiven Prozessen



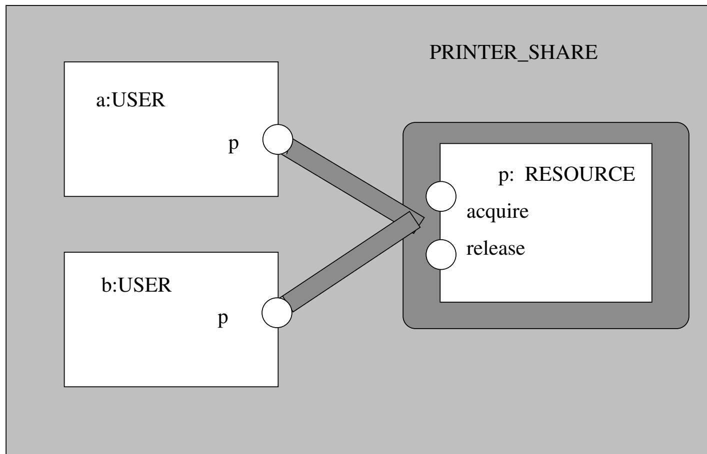
(P||Q) / {m/a, m/b, c/d}

Strukturdiagramm von interaktiven Prozessen mit Schnittstellen



$\| \; S = (P \| Q)@\{x,y\}$

Strukturdiagramm von Prozessen mit Ressource-Sharing



RESOURCE = (acquire –> release –> RESOURCE).
USER = (p.acquire –> use –> p.release –> USER).
||PRINTER_SHARE = (a:USER||b:USER||{a,b}::p:RESOURCE).

3.2 Semantik von parallelen Prozessen

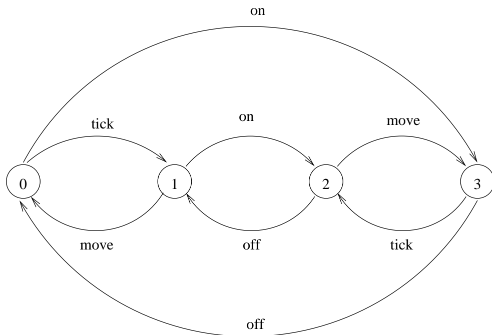
Die induktive Definition der Funktion lts: $\mathcal{E} \longrightarrow \mathcal{T}$ wird folgendermaßen (vgl. Vorlesungsmitschrift) erweitert auf:

- Parallele Komposition von Prozessen
- Umbenennung
- Hiding (Verbergen von Aktionen) und
- Alphabeterweiterung

Beispiel:

CLOCK2 $=$ (tick $\rightarrow$ move $\rightarrow$ CLOCK2).
RADIO $=$ (on $\rightarrow$ off $\rightarrow$ RADIO).
$\|$CLOCK2_RADIO $=$ (CLOCK2$\|$RADIO).

lts(CLOCK2):



Beobachtbare Äquivalenz

Zwei Prozesse sind beobachtbar äquivalent, wenn sie für einen (externen) Beobachter, der keine τ-Aktionen sehen kann, nicht unterschieden werden können.

Definition:

Sei $T = (S, A, \Delta, q_0)$ ein LTS, seien $q, p \in S$ zwei Zustände und sei $a \in \alpha T$.
q *geht mit* a *modulo* τ *über in* p, geschrieben $q \overset{a}{\Longrightarrow}_\Delta p$, wenn es eine Folge

$$q \xrightarrow{\tau^*} u \xrightarrow{a} v \xrightarrow{\tau^*} p$$

von Transitionen in Δ gibt, wobei „$\xrightarrow{\tau^*}$“ für eine beliebige (endliche) Anzahl von τ-Übergängen in Δ steht (eventuell auch keinen, d.h. $q = u$ oder $v = p$).

Wir schreiben $q \overset{\epsilon}{\Longrightarrow}_\Delta p$ für $q \xrightarrow{\tau^*} p$.

Definition (Schwache Bisimulation):

Seien T, T' $\in \mathcal{T}$, T = (S, A, Δ, q_0), T' = (S', A', Δ', q_0') mit αT = αT'.
Eine **schwache Bisimulation** zwischen T und T' ist eine Relation R $\subseteq$ S $\times$ S',
so dass für alle (q, q') $\in$ R und für alle a $\in \alpha T \cup \{\epsilon\}$ gilt:

(1) Falls q $\stackrel{a}{\Longrightarrow}_{\Delta}$ p, dann existiert p' $\in$ S' mit q' $\stackrel{a}{\Longrightarrow}_{\Delta'}$ p' und (p, p') $\in$ R.

(2) Falls q' $\stackrel{a}{\Longrightarrow}_{\Delta'}$ p', dann existiert p $\in$ S mit q $\stackrel{a}{\Longrightarrow}_{\Delta}$ p und (p, p') $\in$ R.

Bemerkung:

Jede starke Bisimulation zwischen zwei LTSen T und T' ist auch eine schwache Bisimulation zwischen T und T' Die Umkehrung gilt jedoch nicht!

Definition (Beobachtbare Äquivalenz von LTSen):

Seien T, T' $\in \mathcal{T}$, T = (S, A, Δ, q_0), T' = (S', A', Δ', q_0').
T und T' sind **beobachtbar äquivalent (schwach bisimular)**, geschrieben T $\approx$ T', wenn gilt:

(a) T und T' haben dasselbe Alphabet, d.h. αT = αT'.

(b) Es gibt eine schwache Bisimulation R $\subseteq$ S $\times$ S' zwischen T und T', so dass (q_0, q_0') $\in$ R.

Bemerkung:

Stark äquivalente LTSe sind auch beobachtbar äquivalent.
Die Umkehrung gilt jedoch nicht!

Lemma:

$\approx$ ist eine Äquivalenzrelation auf $\mathcal{T}$.
(Beweis analog zur starken Äquivalenz.)

Beispiele:

Definition (Beobachtbare Äquivalenz von Prozessen):
Zwei Prozesse E, F $\in \mathcal{E}$ sind **beobachtbar äquivalent (schwach bisimular)**, geschrieben E $\approx$ F, wenn gilt: lts(E) $\approx$ lts(F).

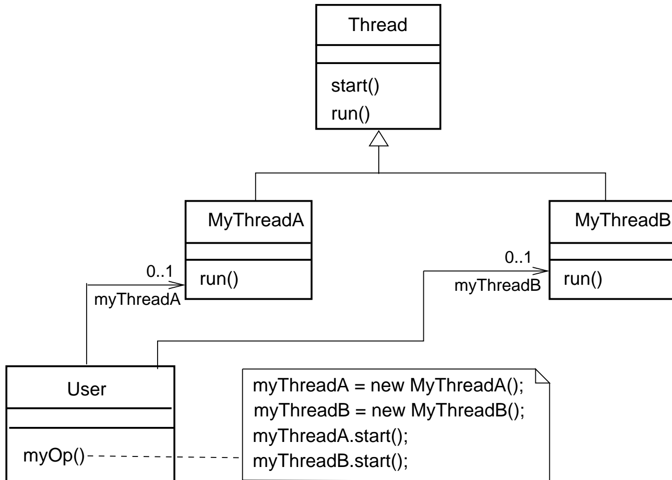
Beispiele:

Algebraische Gesetze für beobachtbare Äquivalenz

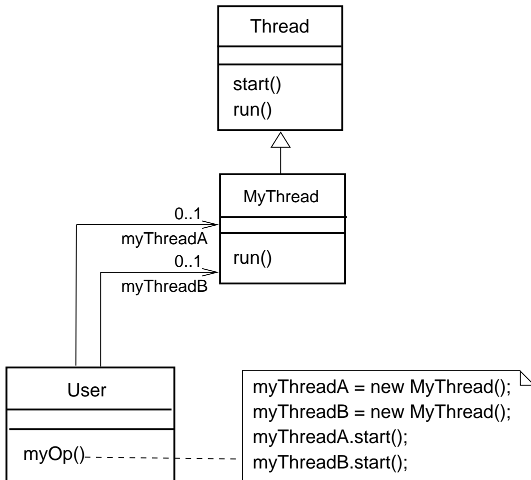
Seien a, b Aktionen und E, F, G Prozessausdrücke.

- $(\mathsf{a} \to \mathsf{E} \mid \mathsf{b} \to \mathsf{F}) \approx (\mathsf{b} \to \mathsf{F} \mid \mathsf{a} \to \mathsf{E})$
- $(\mathsf{a} \to \mathsf{E} \mid \mathsf{a} \to \mathsf{E}) \approx (\mathsf{a} \to \mathsf{E})$
- $(\mathsf{E} \| \mathsf{F}) \approx (\mathsf{F} \| \mathsf{E})$
- $((\mathsf{E} \| \mathsf{F}) \| \mathsf{G}) \approx (\mathsf{E} \| (\mathsf{F} \| \mathsf{G}))$
- $(\mathsf{E} \| \mathsf{STOP}) \approx \mathsf{E}$
- $(\tau \to \mathsf{E}) \approx \mathsf{E}$
- $(\mathsf{a} \to \mathsf{E}) \setminus \{\mathsf{a}\} \approx \mathsf{E}$ falls $\mathsf{a} \notin \alpha\mathsf{E}$.

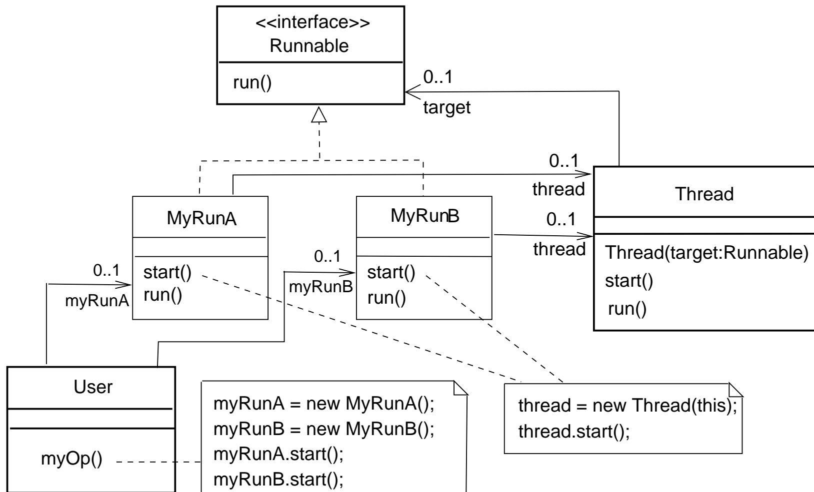
Realisierung mittels Vererbung



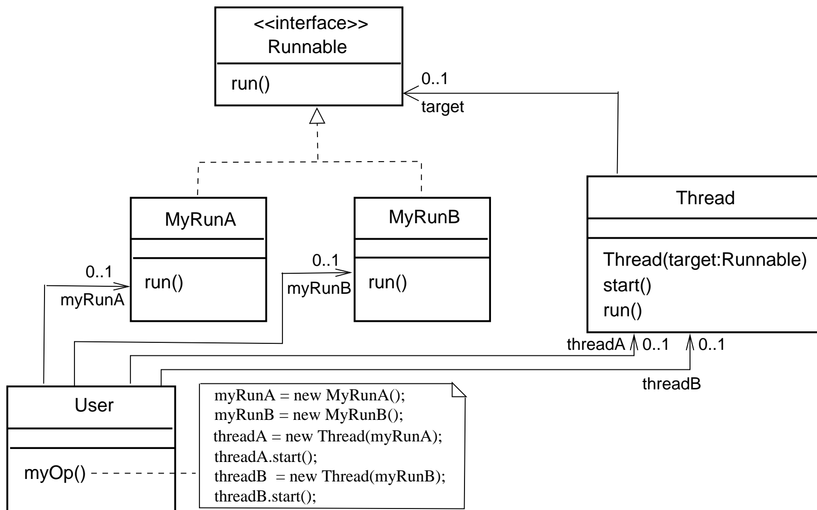
Realisierung mittels Vererbung (Variante)



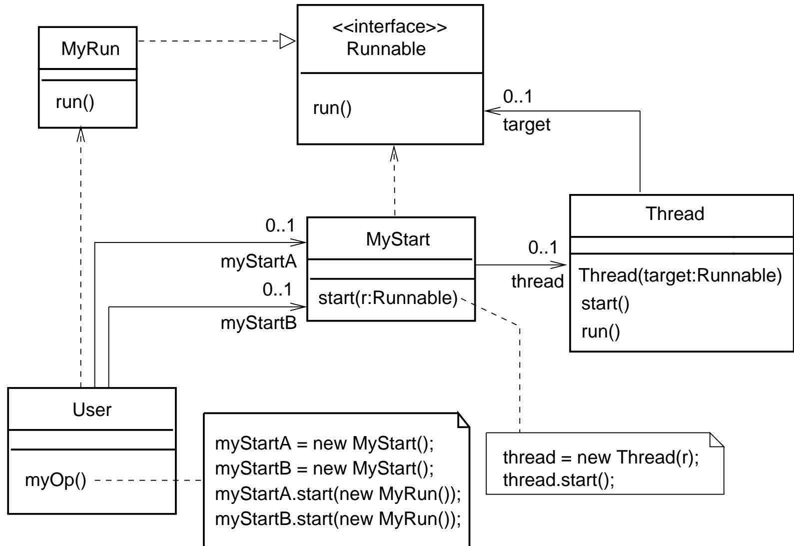
Realisierung durch Verwendung des Interfaces “Runnable“



Realisierung mit "Runnable" (Variante 1)



Realisierung mit "Runnable" (Variante 2)



Beispiel (Rotierende Segmente):
vgl. [Magee, Kramer]

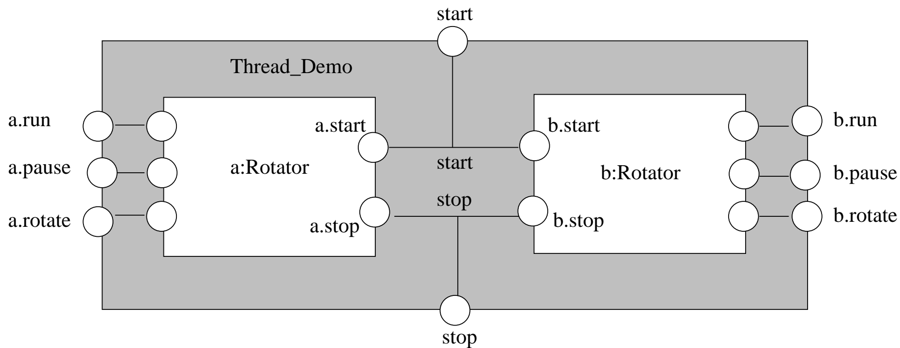
Zwei voneinander unabhägige Threads rotieren ein Kreissegment.

Modellierung:

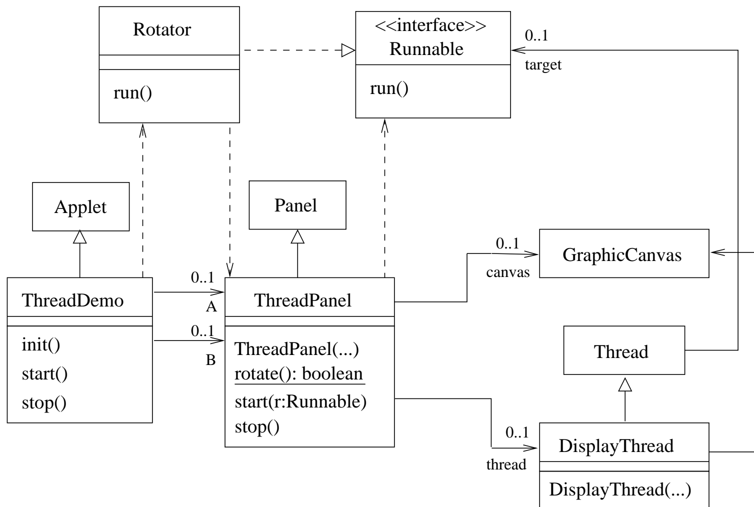
$$
\begin{array}{lcl}
\mathsf{ROTATOR} & = & (\mathsf{start} \rightarrow \mathsf{PAUSED}), \\
\mathsf{PAUSED} & = & (\mathsf{run} \rightarrow \mathsf{RUN} \\
 & & \mid \mathsf{pause} \rightarrow \mathsf{PAUSED} \\
 & & \mid \mathsf{stop} \rightarrow \mathsf{STOP}), \\
\mathsf{RUN} & = & (\{\mathsf{rotate}, \mathsf{run}\} \rightarrow \mathsf{RUN} \\
 & & \mid \mathsf{pause} \rightarrow \mathsf{PAUSED} \\
 & & \mid \mathsf{stop} \rightarrow \mathsf{STOP}).
\end{array}
$$

$\|$THREAD_DEMO $=$

(a:ROTATOR$\|$b:ROTATOR)/$\{$start/$\{$a,b$\}$.start,stop/$\{$a,b$\}$.stop$\}$.


start
Thread_Demo
a.run
a.pause
a.rotate
a.start
a:Rotator
a.stop
start
stop
b.start
b:Rotator
b.stop
b.run
b.pause
b.rotate
stop

Implementierung des Modells:



Java-Code:

```
public class ThreadDemo extends Applet {
  ThreadPanel A,B;

  public void init() {
    A = new ThreadPanel("Thread A", Color.blue);
    B = new ThreadPanel("Thread B", Color.blue);
    add(A); add(B);
  }
  public void start() {  // synchronisation
    A.start(new Rotator());
    B.start(new Rotator());
  }
  public void stop() {
    A.stop();
    B.stop();
  }
}
```

```
public class ThreadPanel extends Panel {
   DisplayThread thread;
   GraphicCanvas canvas;
   // construct display with title and segment color c
   public ThreadPanel(String title, Color c) {...}
   // rotate display of currently running thread 6 degrees
   // return value not used in this example
   public static boolean rotate() throws InterruptedException {...}

   // create a new thread with target r and start it running
   public void start(Runnable r) {
      thread = new  DisplayThread(canvas, r,...);
      thread.start();
   }
   // stop the thread using interrupt()
   public void stop() {thread.interrupt(); }
}
public class Rotator implements Runnable {
   public void run() {
      try {
         while(true)  ThreadPanel.rotate();
      } catch(InterruptedException e) {}
   }
}
```