Kapitel 3

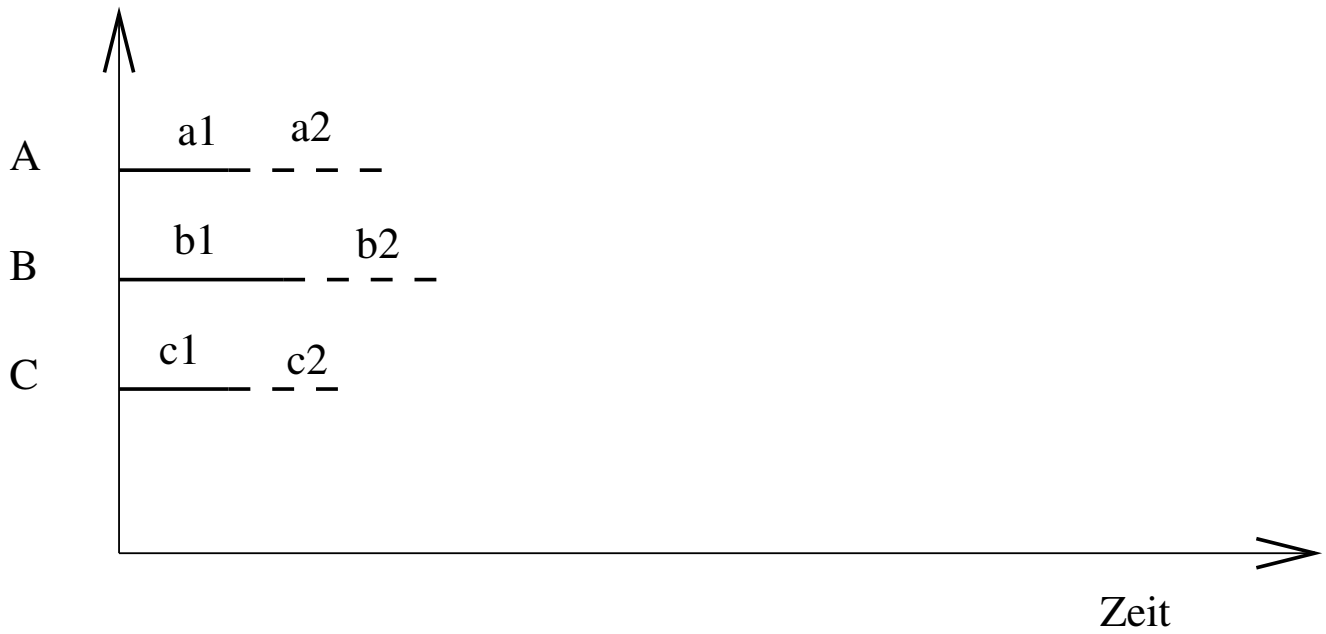
Parallele Prozesse

Prof. Dr. Rolf Hennicker

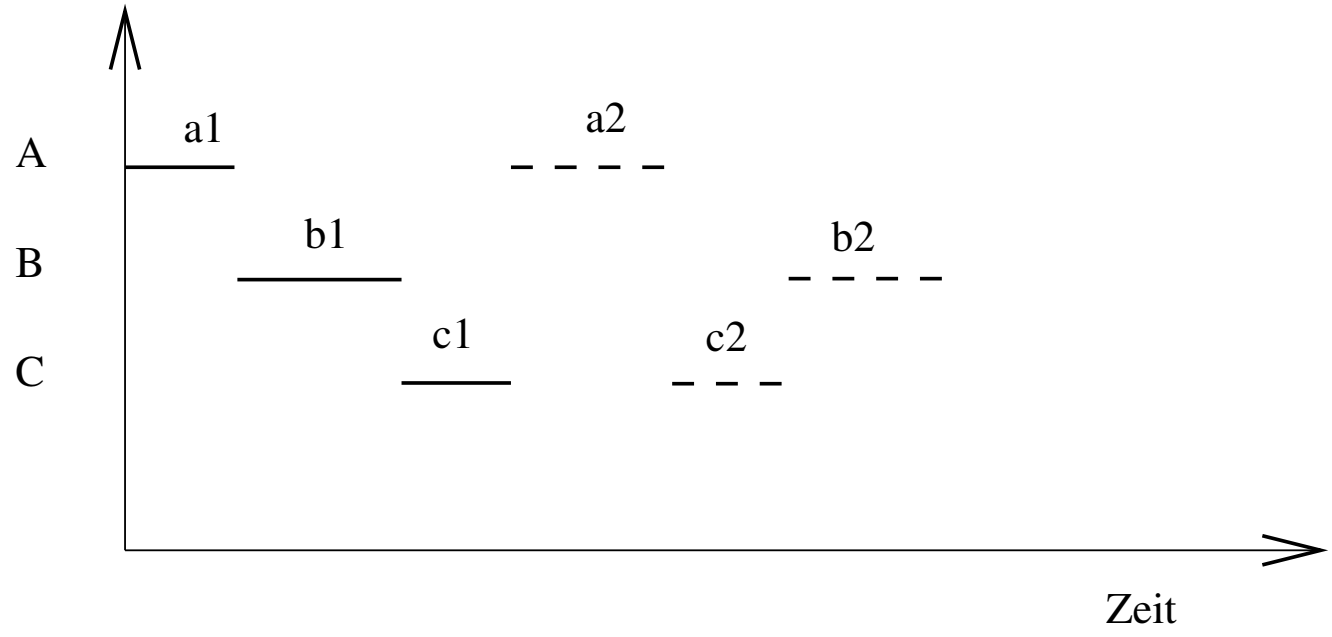
16.05.2013

3.1 Modellierung paralleler Prozesse

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.

6. Parallele Komposition von Prozessen

Bisher wurden 5 grundlegende Konstrukte für FSP Prozessausdrücke definiert.

Definition:

Sind $E_1, \ldots, E_n$ Prozessausdrücke, dann ist

$$(E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n)$$

ein Prozessausdruck (*parallele Komposition* von $E_1, \ldots, E_n$)
mit $FV((E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n)) = FV(E_1) \cup \ldots \cup FV(E_n)$.

Wirkung:

Die (disjunkten) Aktionen von $E_1, \ldots, E_n$ werden verzahnt ausgeführt.

Deklaration paralleler Prozesse:

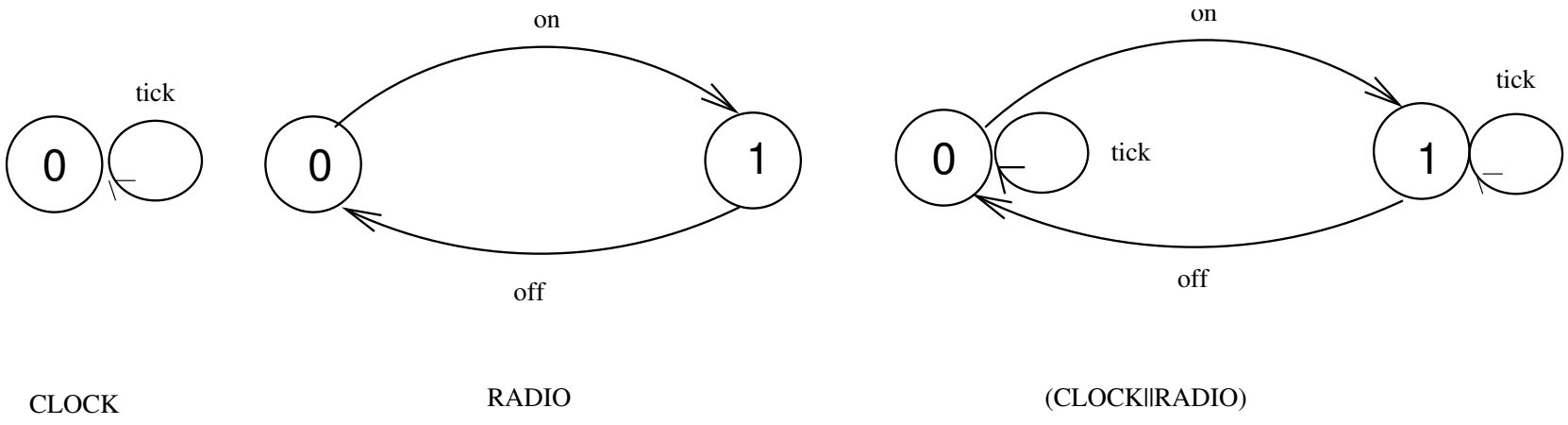
Seien E und F Prozessausdrücke und sei $P \in PID$ ein Prozessidentifikator mit $P \notin FV((E \,||\, F))$. Statt der Prozessdeklaration "$P = (E \,||\, F)$." schreiben wir dann

$$||\, P = (E \,||\, F).$$

Beispiel:

$\mathrm{CLOCK} = (\mathrm{tick} \rightarrow \mathrm{CLOCK}).$
$\mathrm{RADIO} = (\mathrm{on} \rightarrow \mathrm{off} \rightarrow \mathrm{RADIO}).$
$\|\mathrm{CLOCK\_RADIO} = (\mathrm{CLOCK}\|\mathrm{RADIO}).$

Zugehörige LTSe:

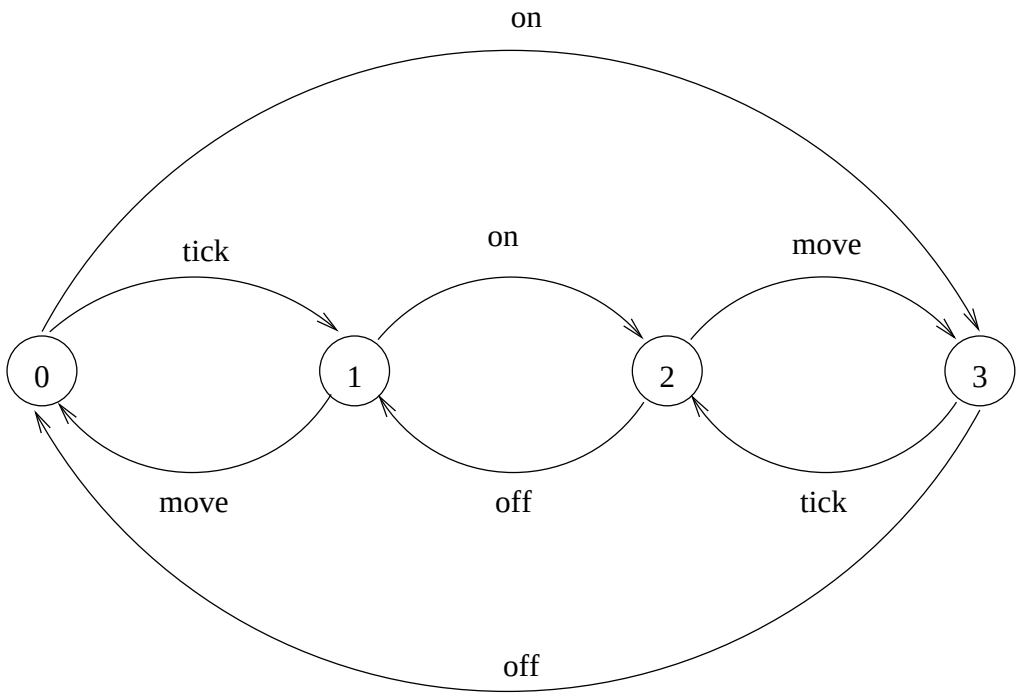


CLOCK RADIO (CLOCK||RADIO)

Beispiel:

CLOCK2 = (tick → move → CLOCK2).
RADIO = (on → off → RADIO).
||CLOCK2_RADIO = (CLOCK2||RADIO).

lts(CLOCK2):



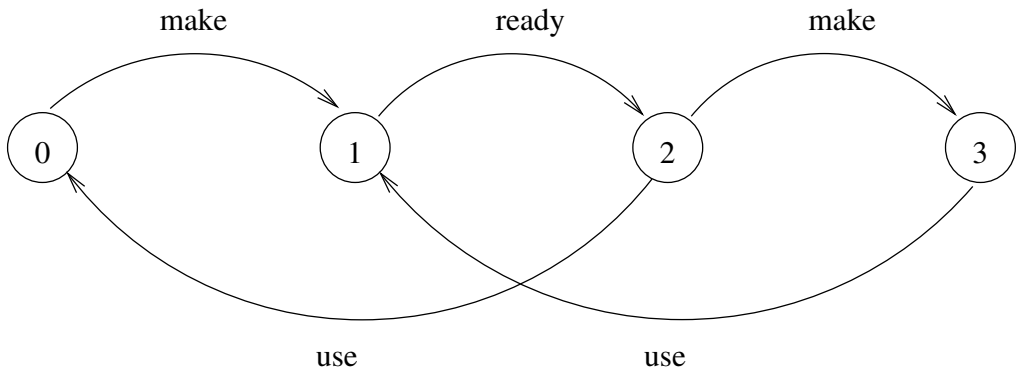
Prozessinteraktionen

- 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:

$\mathsf{MAKER} = (\mathsf{make} \rightarrow \mathsf{ready} \rightarrow \mathsf{MAKER}).$
$\mathsf{USER} \quad = (\mathsf{ready} \rightarrow \mathsf{use} \rightarrow \mathsf{USER}).$
$\|\mathsf{MAKER\_USER} = (\mathsf{MAKER} \parallel \mathsf{USER}).$

Zugehöriges LTS:



Variante:

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

$$\begin{aligned} &\text{MAKER2} = (\text{make} \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKER2}).\\ &\text{USER2} \quad = (\text{ready} \rightarrow \text{use} \rightarrow \text{used} \rightarrow \text{USER2}).\\ &\|\text{MAKER\_USER2} = (\text{MAKER2} \parallel \text{USER2}). \end{aligned}$$

Zugehöriges LTS:

7. Umbenennung von Aktionen

Die Umbenennung von Aktionen dient (vor allem)

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

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

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 $\mathrm{FV}(E\,\{n_1/a_1, \ldots, n_k/a_k\}) = \mathrm{FV}(E)$.

Wirkung:

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

Erstellung von Prozess-Kopien durch Umbenennung

Beispiel:

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

Dabei ist a:CLIENT eine abkürzende Schreibweise für
CLIENT{a.call/call, a.wait/wait, a.continue/continue}.

Man nennt dies **Process Labelling**.

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

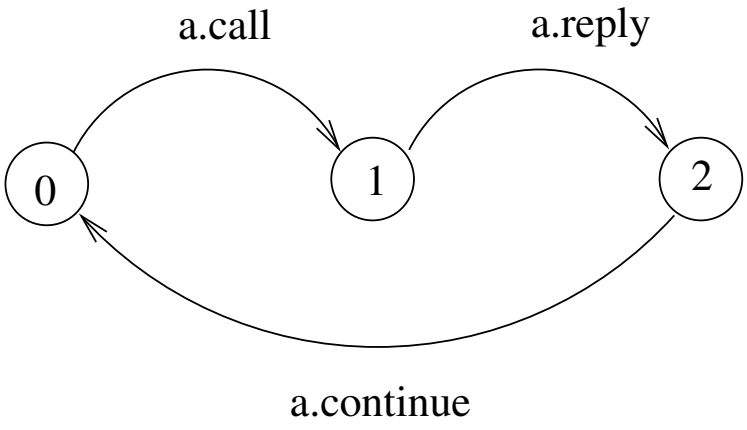
$$(E_1 \,||\, E_2 \,||\, \ldots \,||\, E_n) \,/\, \{n_1/a_1, \ldots, n_k/a_k\} =_{def}$$
$$(E_1 \,\{n_1/a_1, \ldots, n_k/a_k\} \,||\, \ldots \,||\, E_n \,\{n_1/a_1, \ldots, n_k/a_k\})$$

Beispiel:

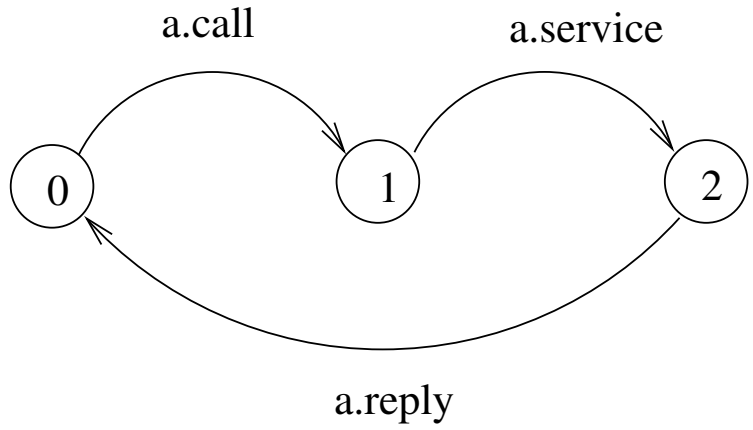
$\mathsf{CLIENT} = (\mathsf{call} \rightarrow \mathsf{wait} \rightarrow \mathsf{continue} \rightarrow \mathsf{CLIENT}).$
$\mathsf{SERVER} = (\mathsf{request} \rightarrow \mathsf{service} \rightarrow \mathsf{reply} \rightarrow \mathsf{SERVER}).$
$\|\mathsf{CLIENT\_SERVER} = (\mathsf{CLIENT} \parallel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

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}.

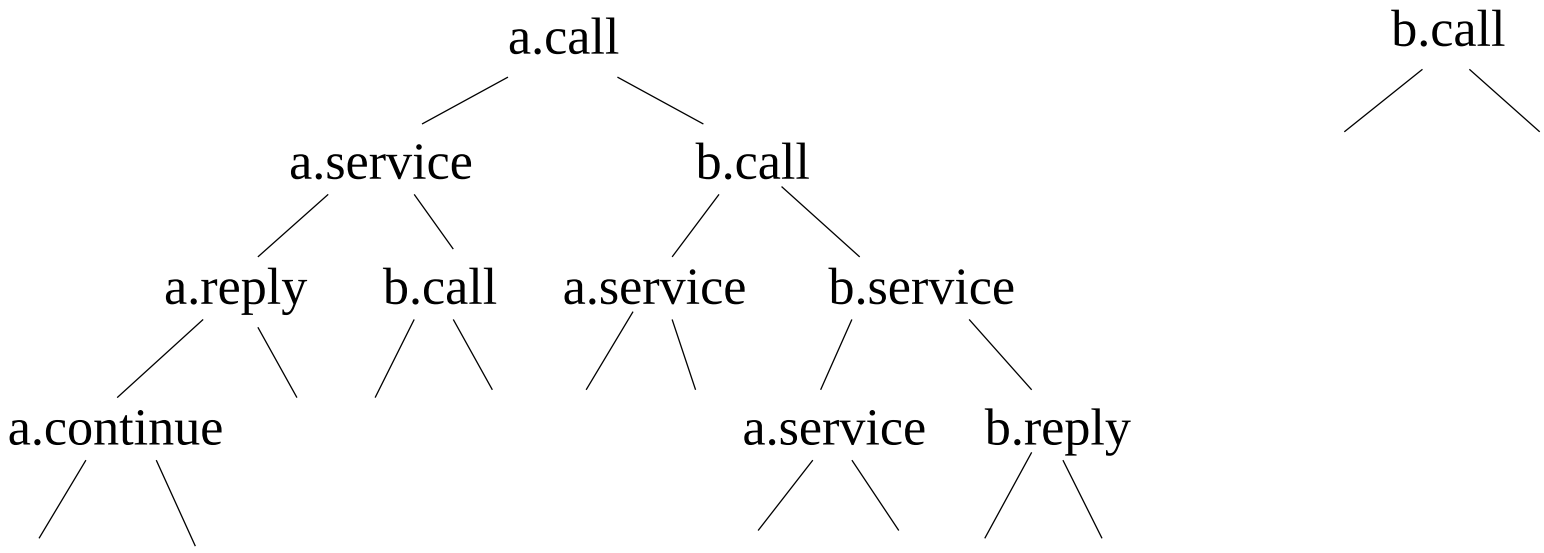


(a:CLIENT){... a.reply/a.wait ...}



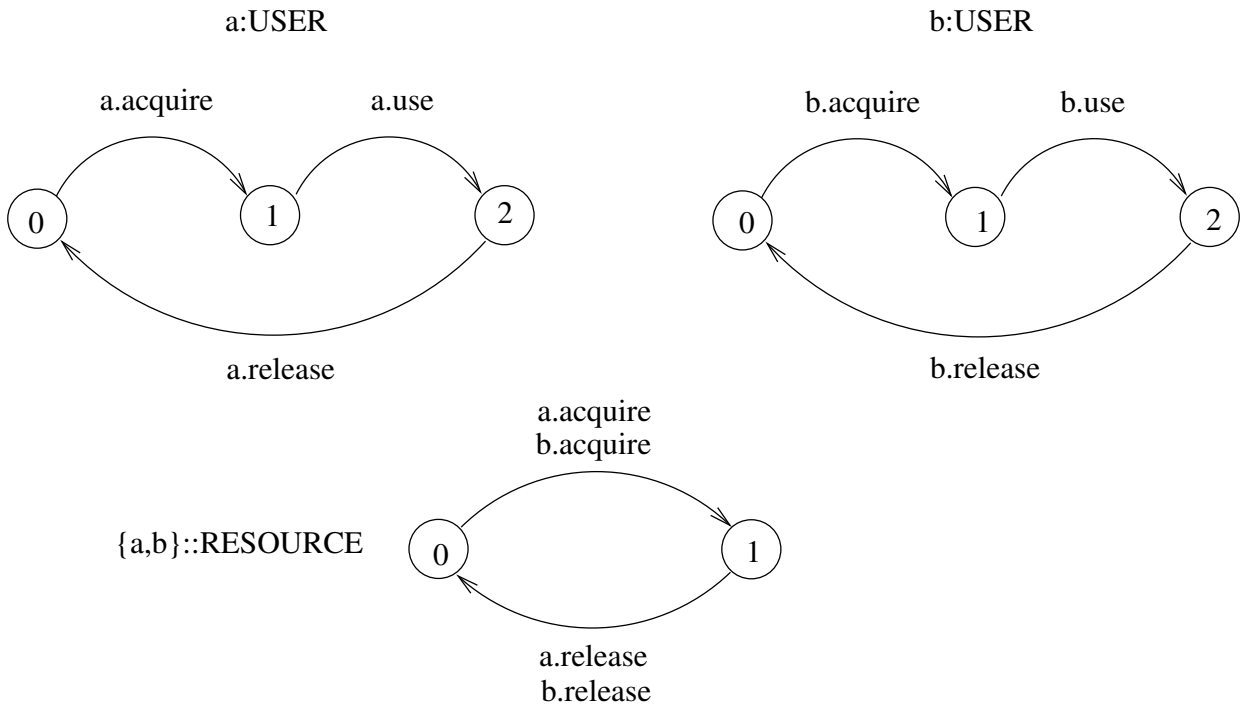
(a:SERVER){a.call/a.request ...}

Abläufe:

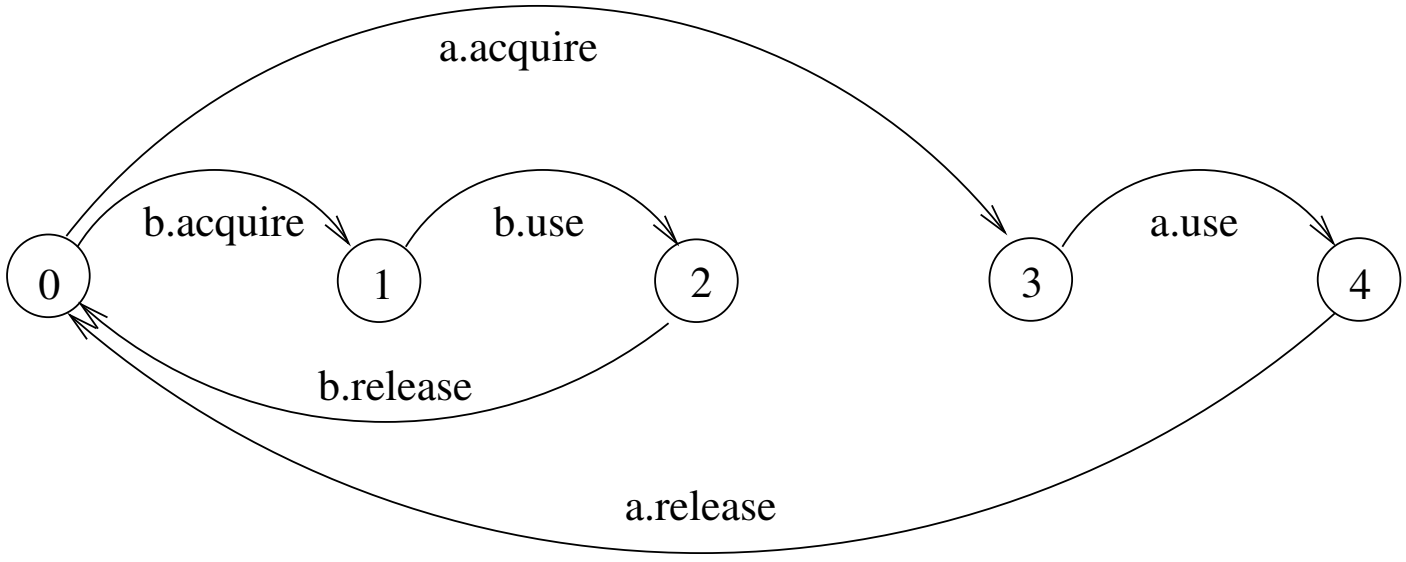

a.call
b.call
a.service
b.call
a.reply
b.call
a.service
b.service
a.continue
a.service
b.reply

Beispiel (Resource-Sharing):

$$\begin{aligned}
\text{USER} &= (\text{acquire} \rightarrow \text{use} \rightarrow \text{release} \rightarrow \text{USER}).\\
\text{RESOURCE} &= (\text{acquire} \rightarrow \text{release} \rightarrow \text{RESOURCE}).\\
\|\text{RESOURCE_SHARE} &= (\text{a:USER} \parallel \text{b:USER} \parallel \{\text{a,b}\}\text{::RESOURCE}).
\end{aligned}$$



RESOURCE_SHARE



8. Verbergen von Aktionen

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

Definition:

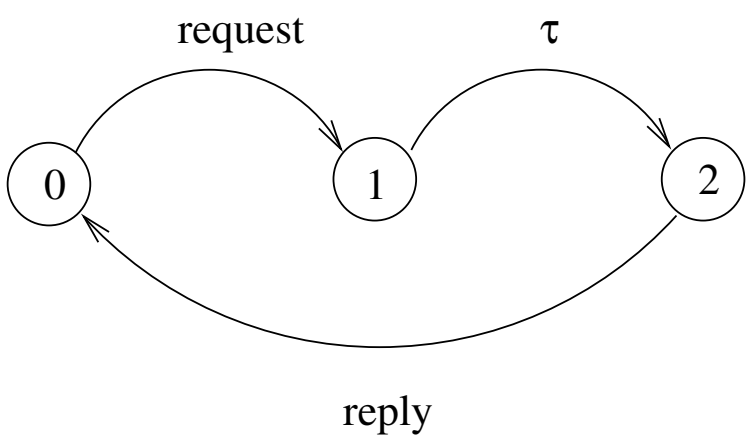
Sei E ein (event. paralleler) Prozessausdruck und sei H $\subseteq$ Labels eine Menge von Aktionsnamen ($\neq \tau$).
Dann ist E $\setminus$ H ein Prozessausdruck ("Hiding") mit FV(E $\setminus$ H) = FV(E).

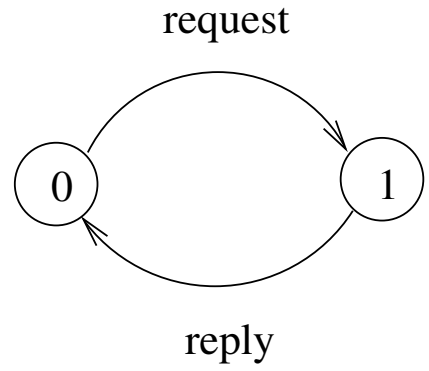
Wirkung:

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:

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





Neben dem LTS kann auch das *minimale, beobachtbar äquivalente* LTS berechnet werden.

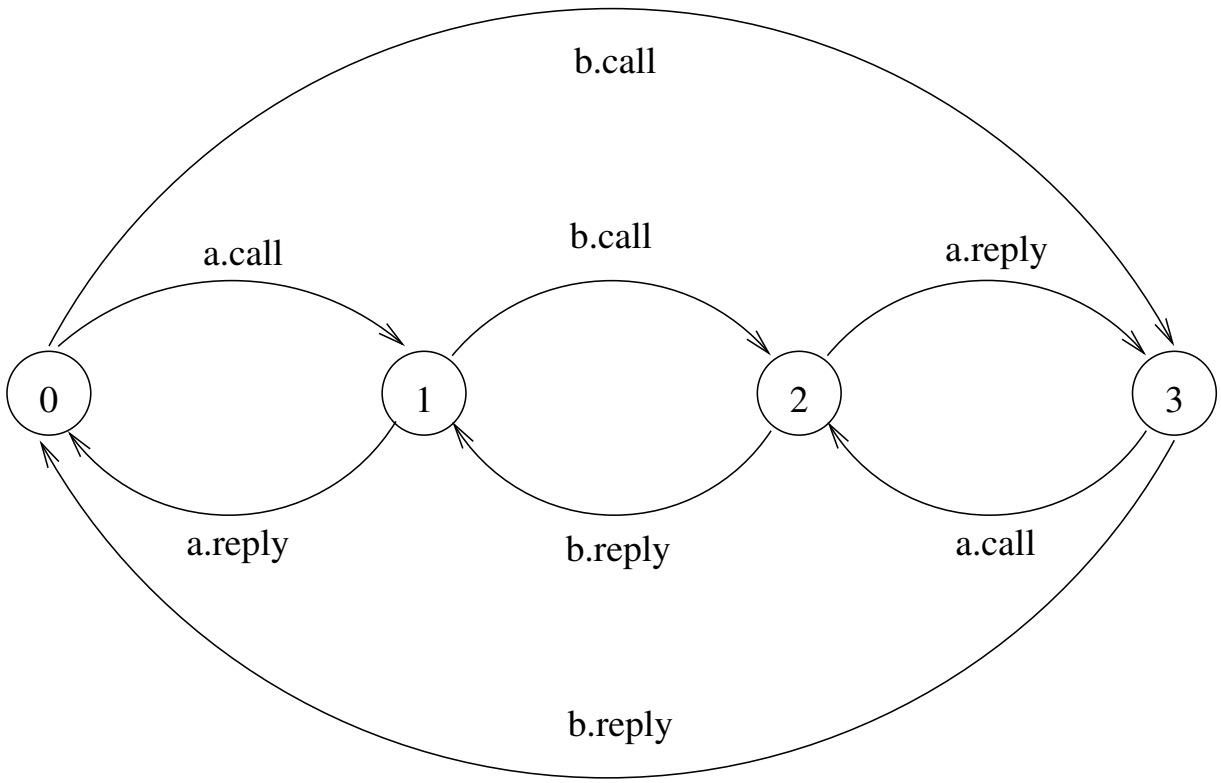
Bemerkung:

- 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

$$E \,@\, I$$

wobei E ein Prozessausdruck und $I \subseteq$ 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 verwendet, wobei gemeinsame Aktionen der Komponenten verborgen werden.
- Häufige Form von parallelen Prozessen mit Schnittstellen: $(P \| Q)/\{neu/alt\}@\{a_1,\ldots,a_k\}$

Beispiel:

(MAKER‖USER)@{make,use} entspricht (MAKER‖USER)\{ready}

9. Alphabeterweiterung

Definition:

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

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

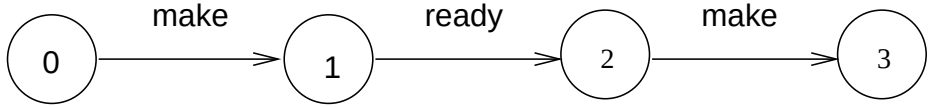
Beispiel: $\alpha((\text{MAKER} \parallel \text{USER}) @ \{\text{make, use}\}) = \{\text{make, use}\}$

Definition:

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

Beispiel:

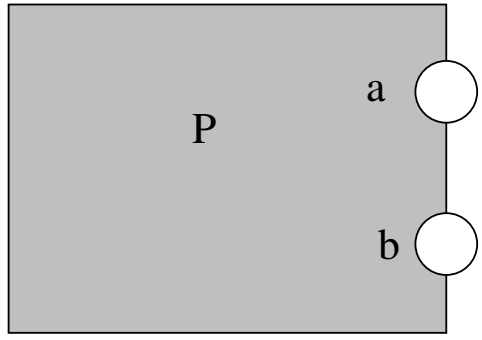
FMAKER = (make → ready → FMAKER) + {use}.
USER = (ready → use → USER).
‖FMAKER_USER = (FMAKER ‖ USER).



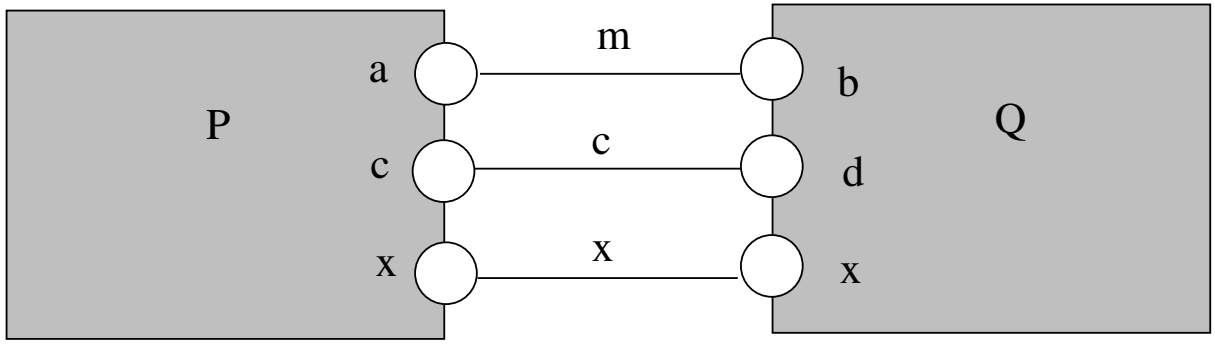
Strukturdiagramme

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

Strukturdiagramm eines Prozesses mit Alphabet {a,b}

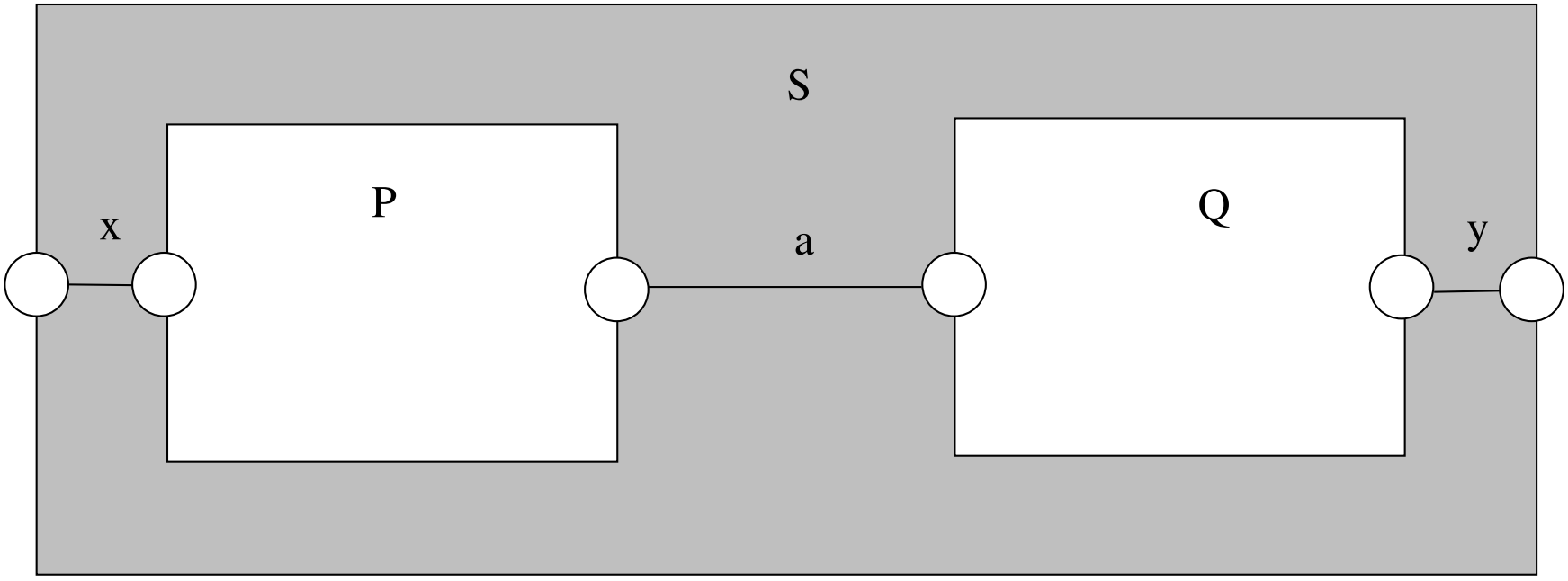


Strukturdiagramm von zwei parallelen Prozessen



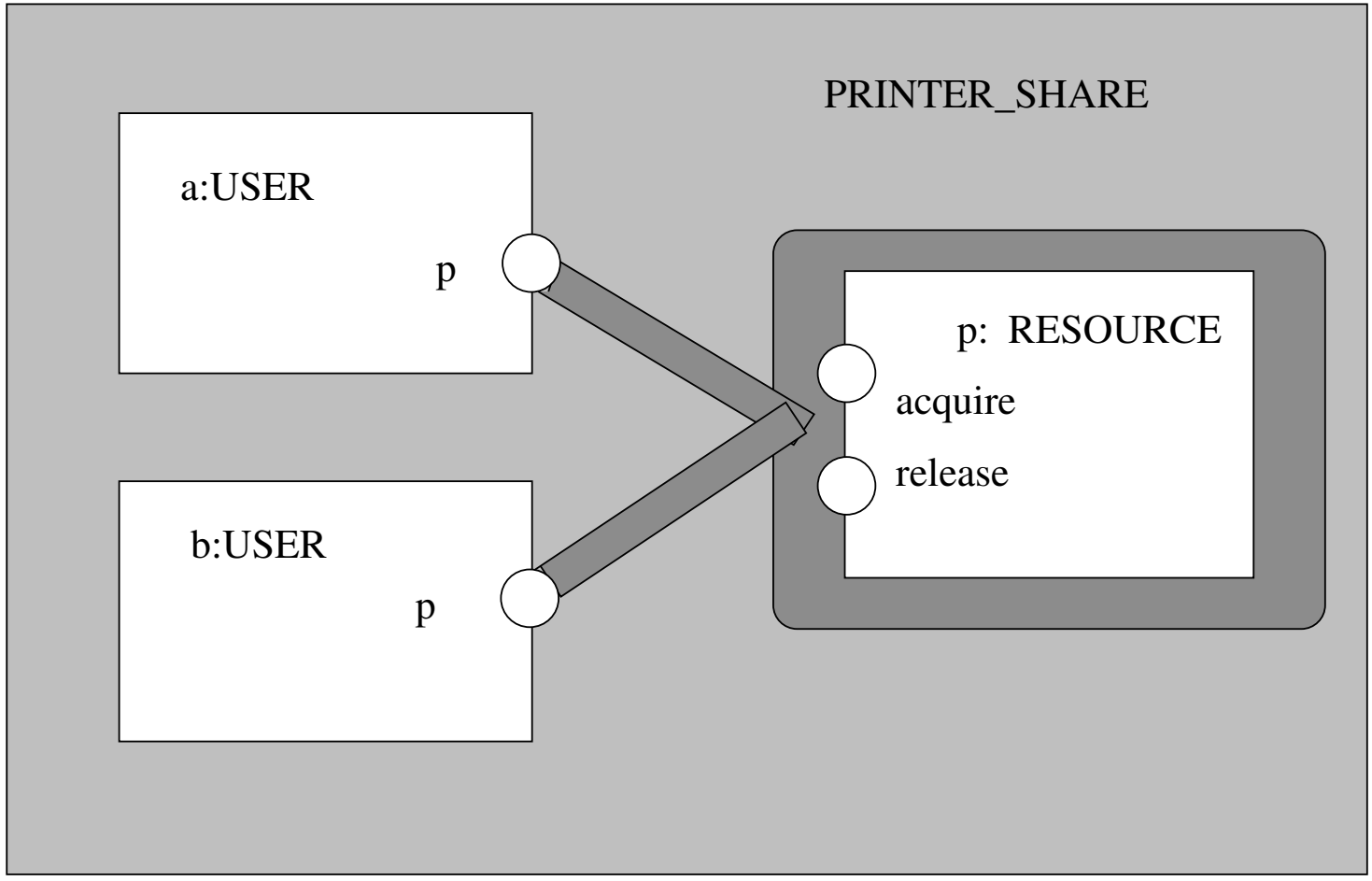
(P||Q) / {m/a, m/b, c/d}

Strukturdiagramm von zwei parallelen Prozessen mit Schnittstelle



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

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

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

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

von Transitionen in Δ gibt, wobei „$\xrightarrow{\tau^*}_\Delta$“ 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 $q \xrightarrow{\tau^*}_\Delta p$.

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

(2) Falls q' $\overset{a}{\Longrightarrow}_{\Delta'}$ p', dann existiert p $\in$ S mit q $\overset{a}{\Longrightarrow}_{\Delta}$ 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, E, F, G Prozessausdrücke, R eine Umbenennung und H eine Menge von Labels.

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

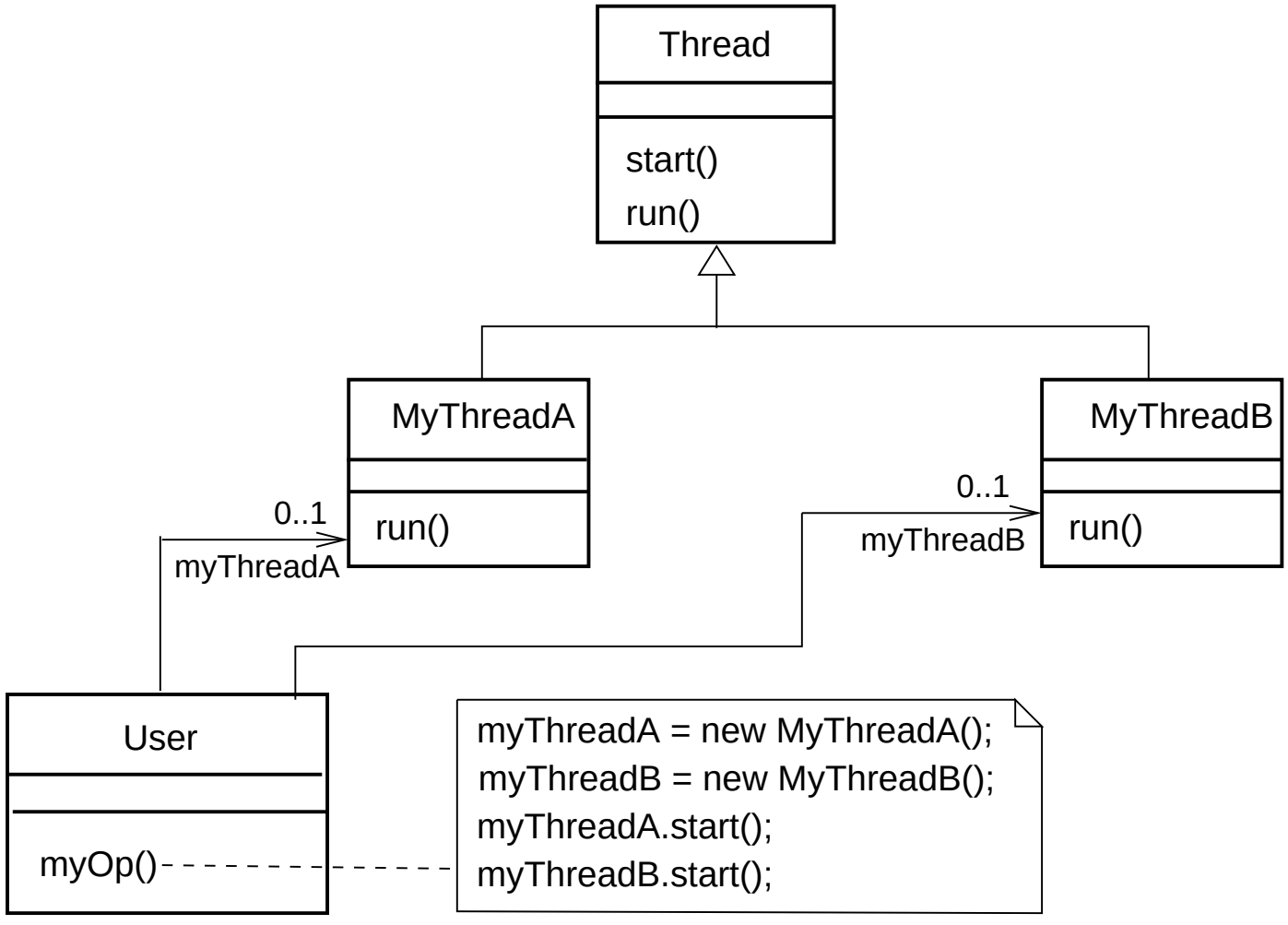
Kongruenzeigenschaften:

- $E \approx F \implies (a \to E) \approx (a \to F)$
- $E \approx F \implies (E \| G) \approx (F \| G)$
- $E \approx F \implies E\{R\} \approx F\{R\}$
- $E \approx F \implies E \setminus H \approx F \setminus H$

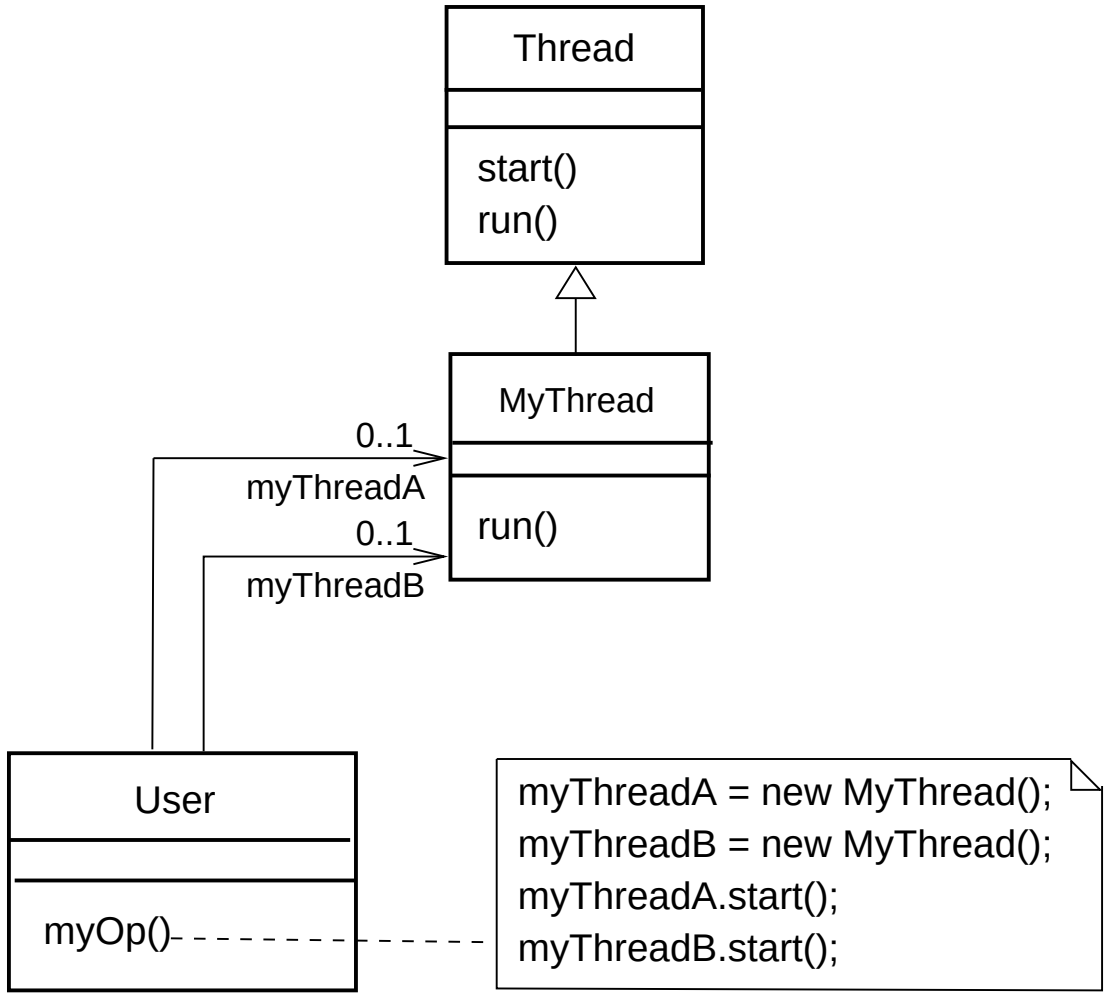
Beachte: $\approx$ ist keine Kongruenz bzgl. der Auswahl.

3.3 Java-Programme mit mehreren Threads

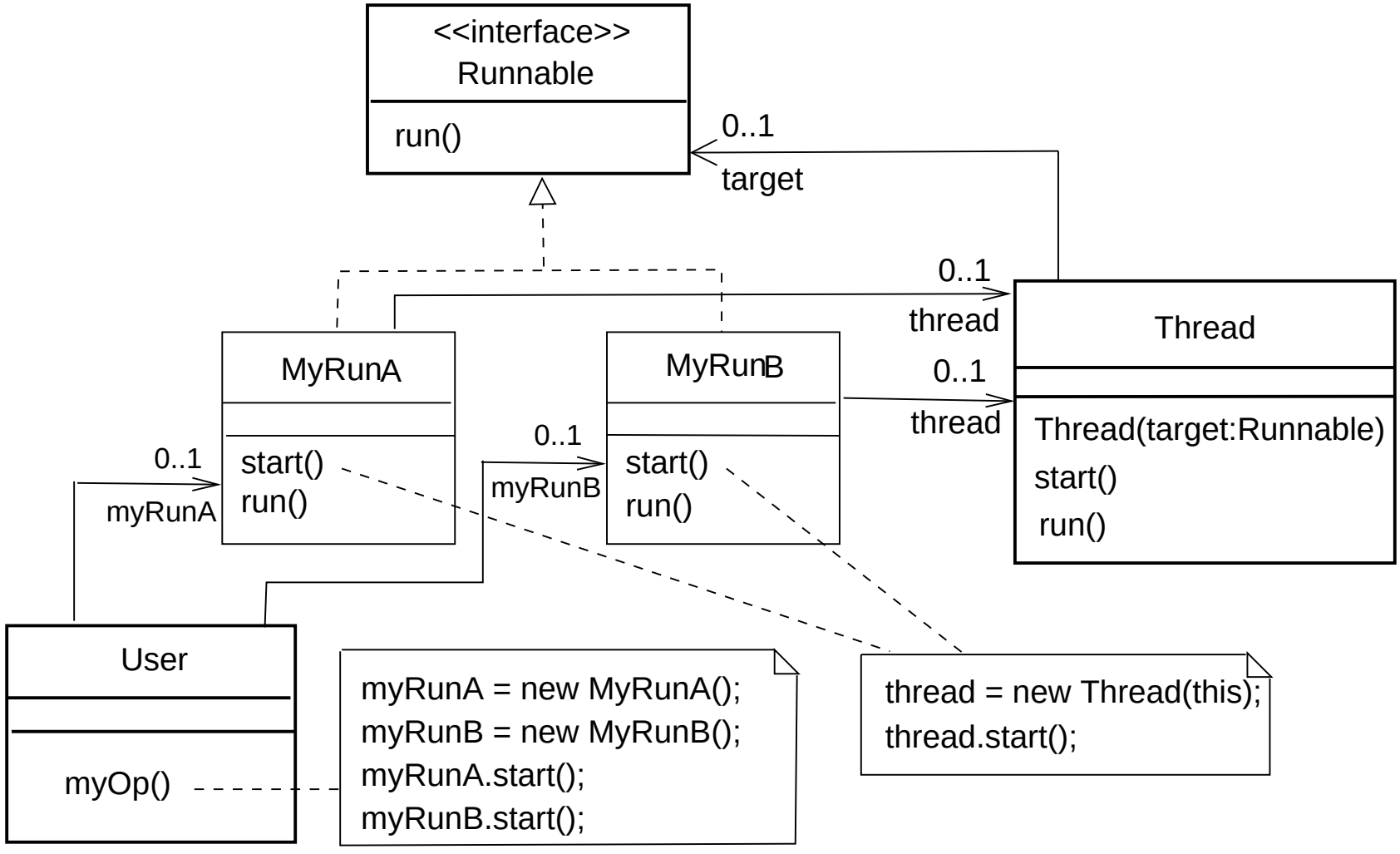
Realisierung mittels Vererbung



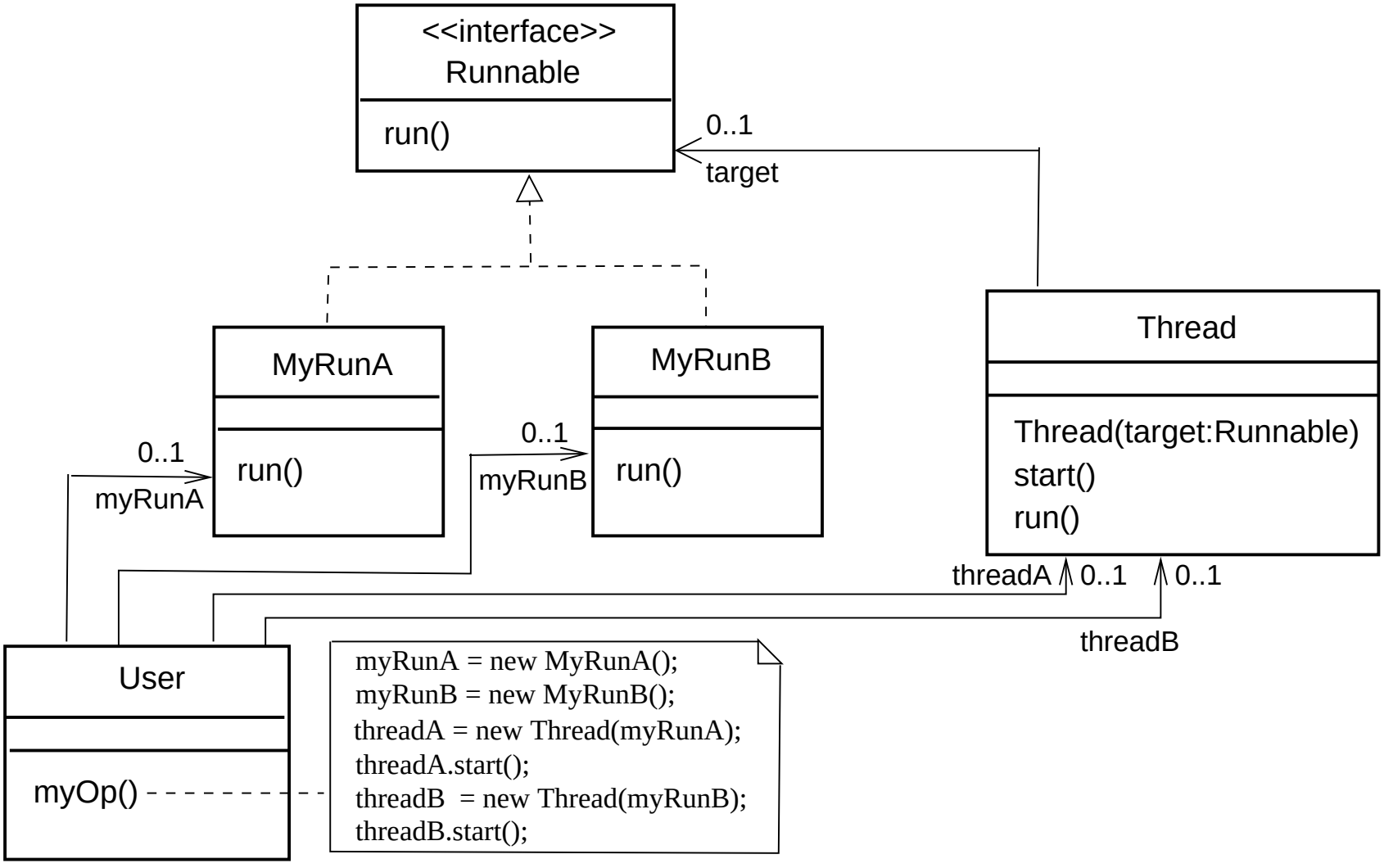
Realisierung mittels Vererbung (Variante)



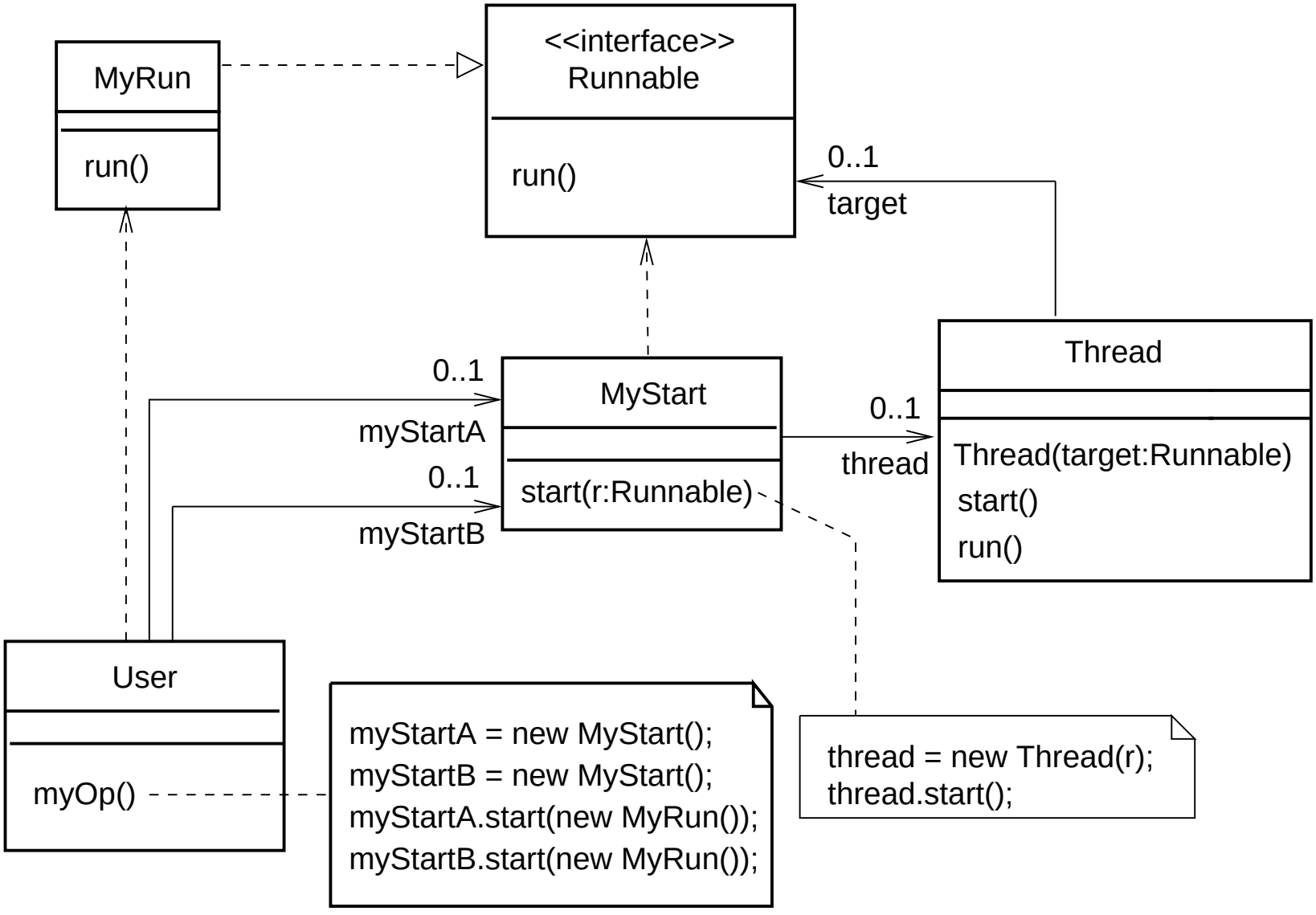
Realisierung durch Verwendung des Interfaces “Runnable“



Realisierung mit “Runnable“ (Variante 1)



Realisierung mit “Runnable“ (Variante 2)



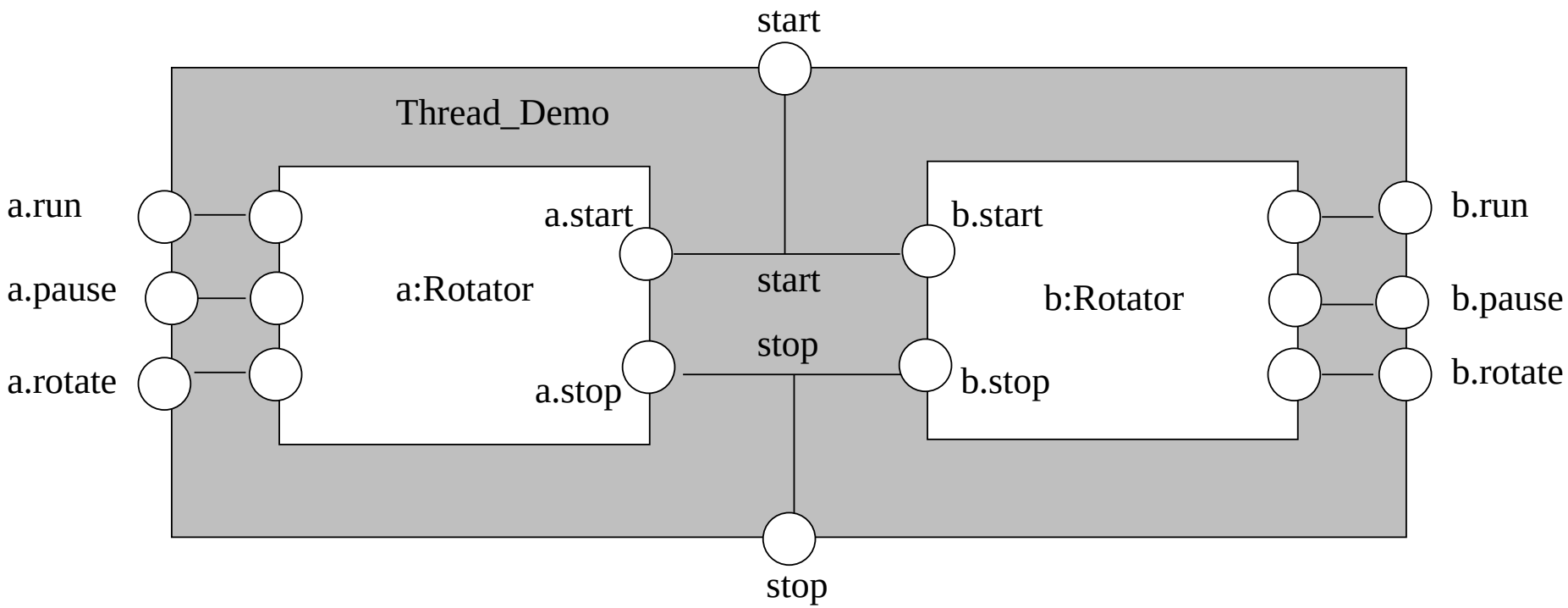
Beispiel (Rotierende Segmente):
vgl. [Magee, Kramer]

Zwei voneinander unabhägige Threads rotieren ein Kreissegment.

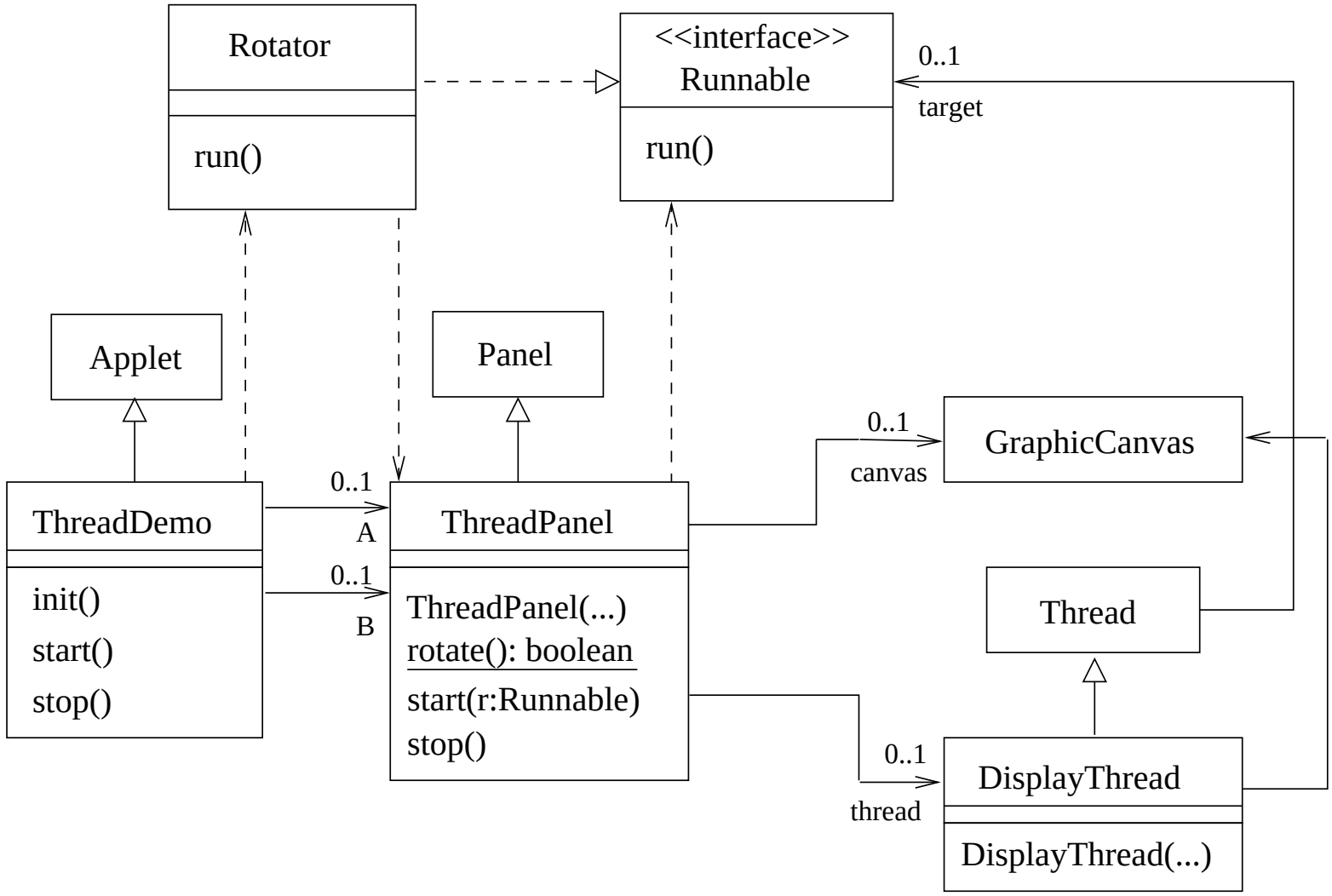
Modellierung:

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

$\|\text{THREAD\_DEMO} =$
(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) {}
   }
}
```