

Entwurf und Implementierung paralleler Programme

Prof. Dr. Rolf Hennicker

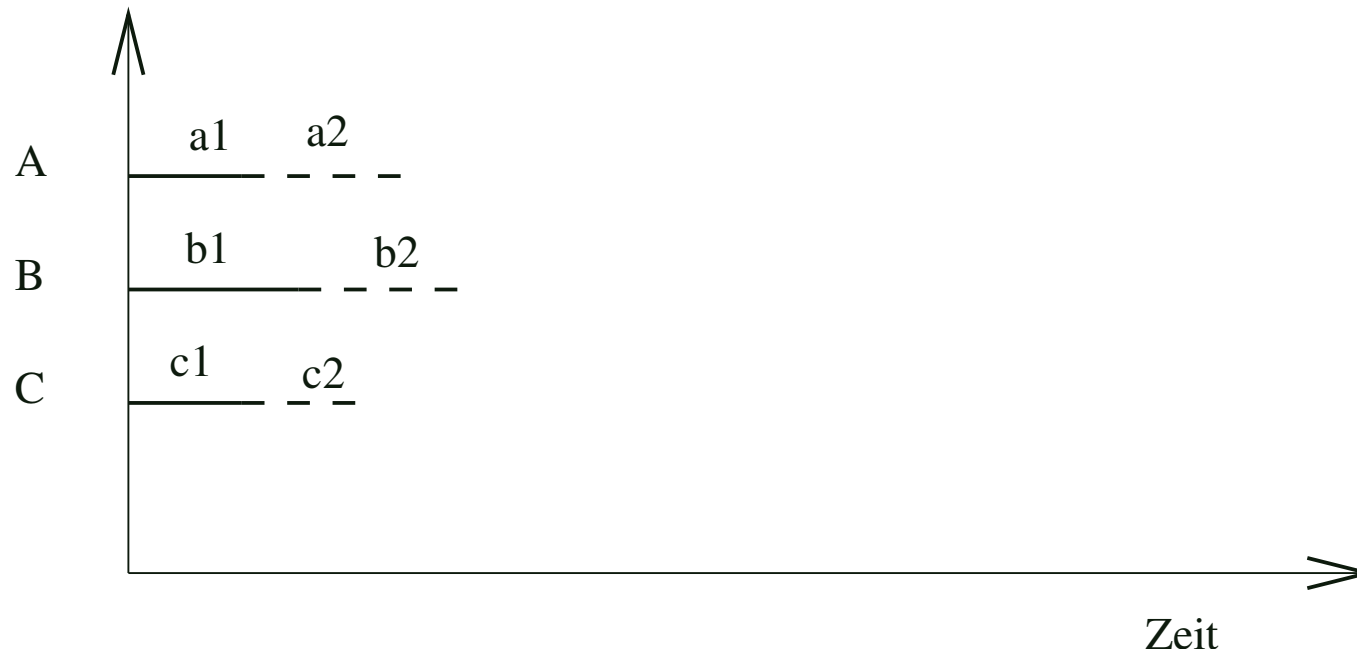
06.11.2006

Kapitel 3

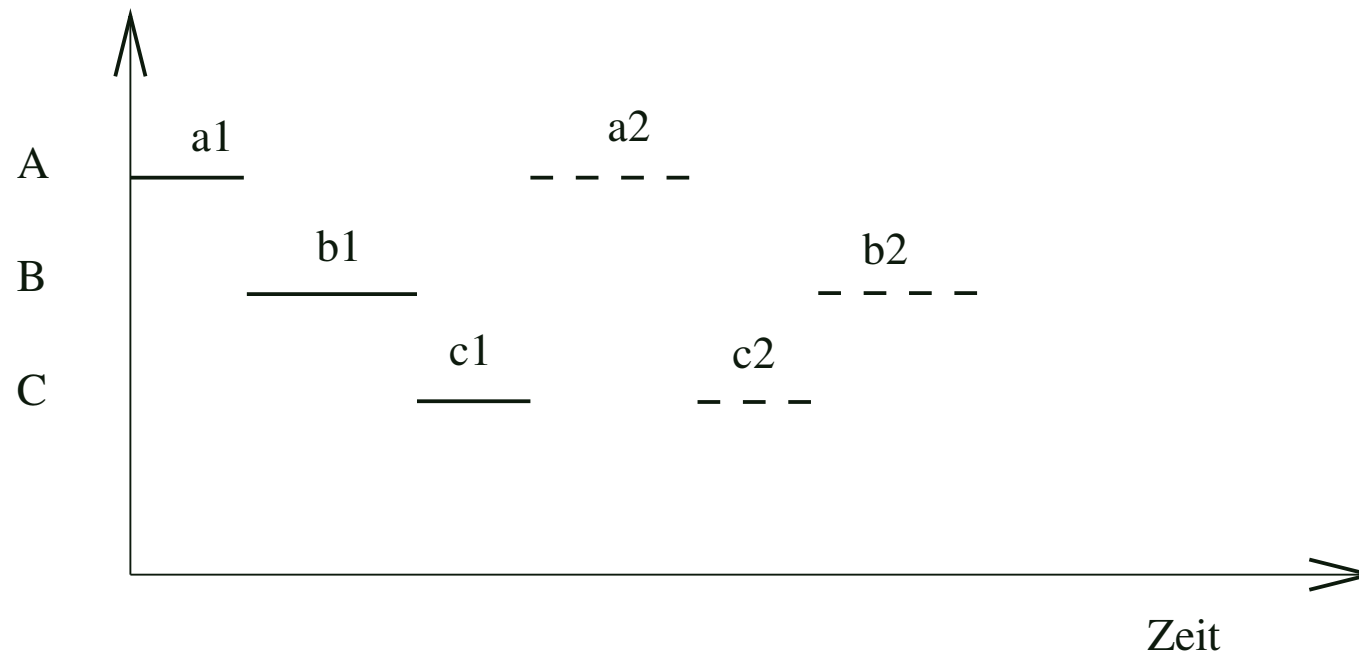
Parallele Prozesse

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.

Parallele Komposition von Prozessen

Definition:

Sind E_1, \dots, E_n Prozessausdrücke, dann ist

$$(E_1 \parallel E_2 \parallel \dots \parallel E_n)$$

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

Wirkung:

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

Deklaration paralleler Prozesse:

Sei E ein Prozessausdruck mit paralleler Komposition und sei $P \in \text{PID}$ ein Prozessidentifikator mit $P \notin FV(E)$. Statt der Prozessdeklaration " $P = E$." schreiben wir dann

$$\parallel P = E.$$

Beispiel:

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 Interaktion schränkt i.a. die möglichen Abläufe 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 (“Handshake“ Protokoll). Diese Interaktion schränkt die möglichen Abläufe stark ein.

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, \dots, a_k und n_1, \dots, n_k Aktionsnamen verschieden von τ . Dann ist

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

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

Wirkung:

Im LTS von E werden die Aktionsnamen a_1, \dots, a_k ersetzt durch n_1, \dots, 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}.

Synchronisation von Prozessen durch Umbenennung

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

Beispiel für Umbenennung zur Synchronisation von Prozessen:

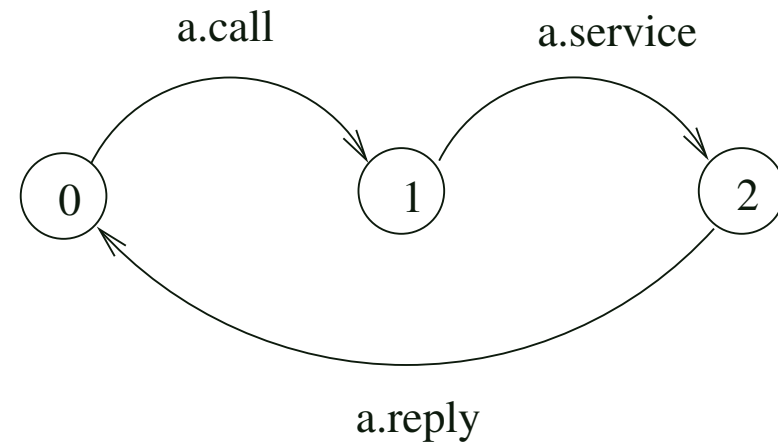
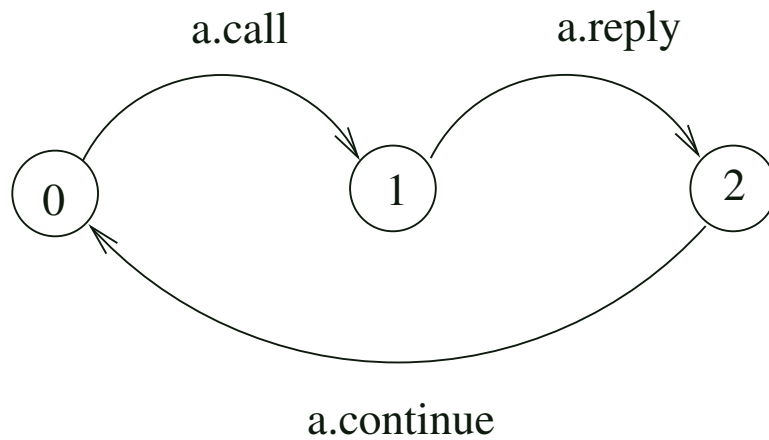
CLIENT = (call \rightarrow wait \rightarrow continue \rightarrow CLIENT).

SERVER = (request \rightarrow service \rightarrow reply \rightarrow SERVER).

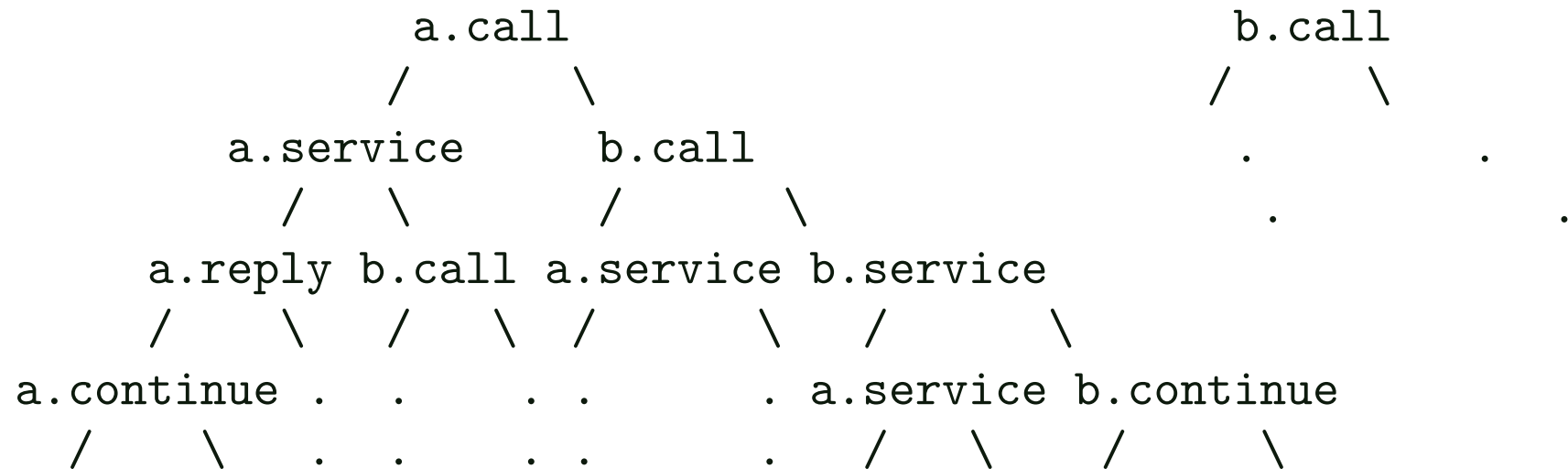
\parallel CLIENT_SERVER = (CLIENT \parallel SERVER) / {call/request, reply/wait}.

Beispiel für Synchronisation von Prozess-Kopien:

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



Traces:



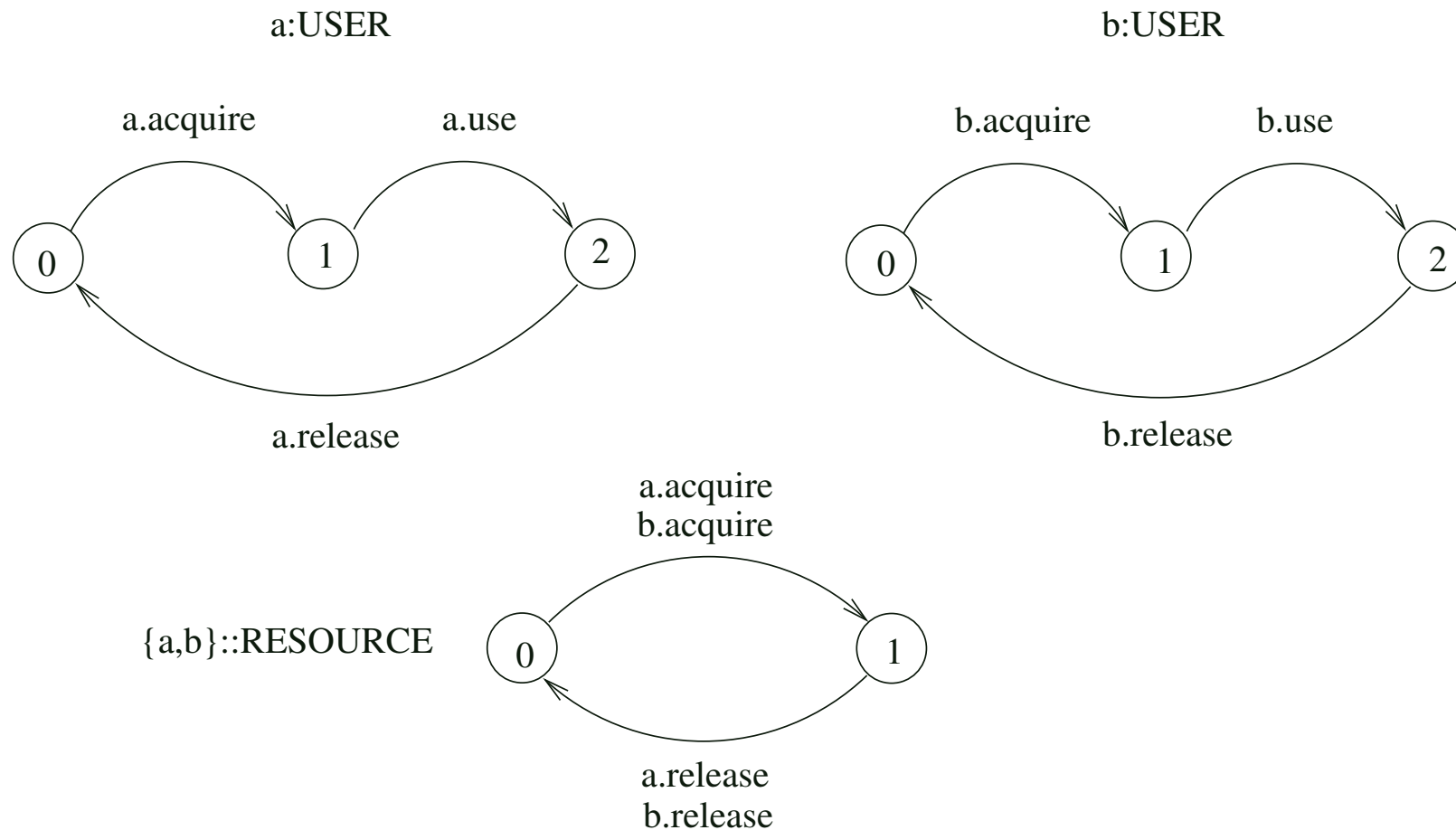
Das LTS ist zu komplex: Man will nur bestimmte Sichten sehen. Interne Abläufe sind für den Benutzer, der das System von außen betrachtet, uninteressant.

Beispiel (Resource-Sharing):

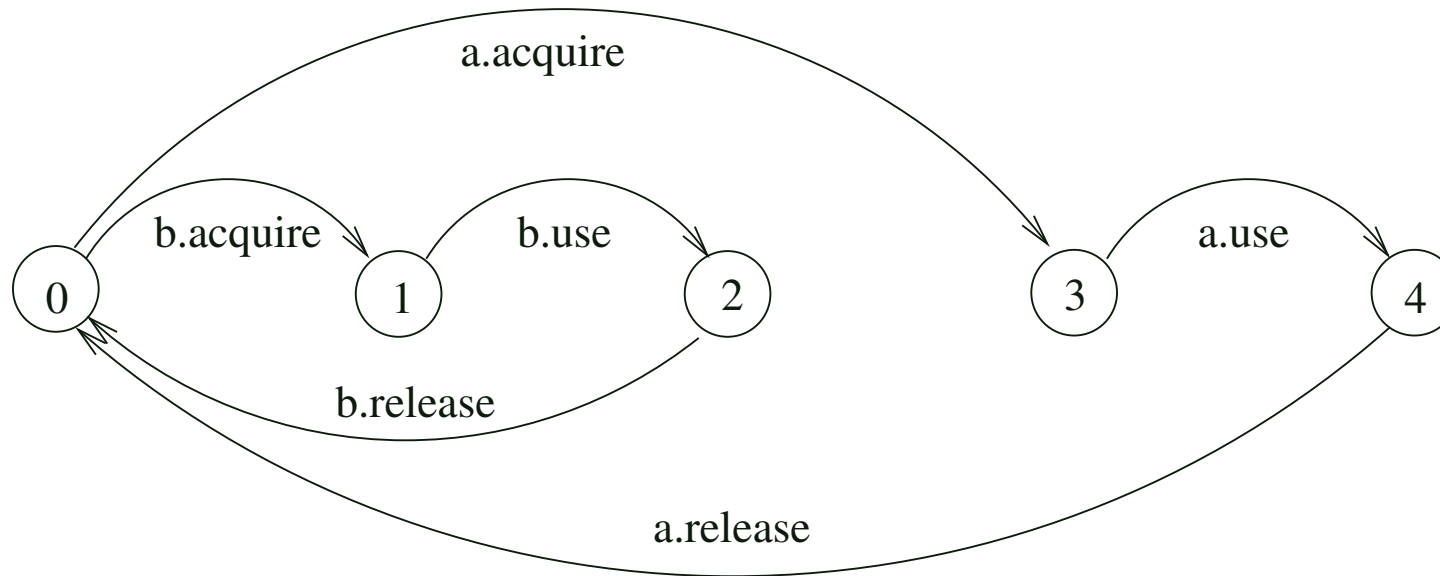
USER = (acquire \rightarrow use \rightarrow release \rightarrow USER).

RESOURCE = (acquire \rightarrow release \rightarrow RESOURCE).

\parallel RESOURCE_SHARE = (a:USER \parallel b:USER) \parallel {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 $H \subseteq \text{Labels}$ eine Menge von Aktionsnamen. Dann ist

$$E \setminus H$$

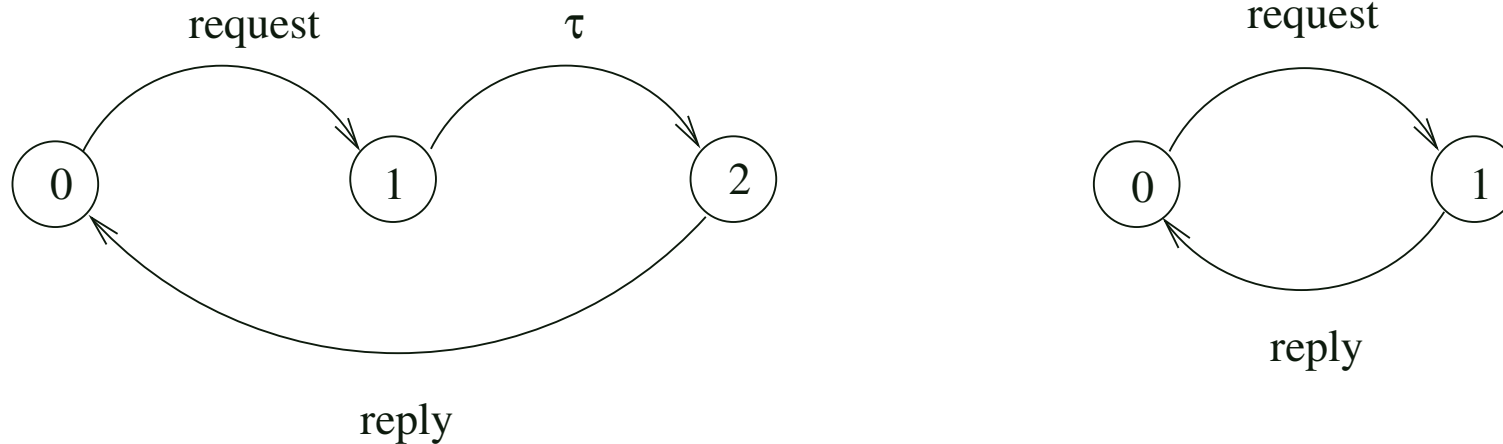
ein Prozessausdruck (“Hiding”) mit $FV(E \setminus H) = FV(E)$.

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:

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



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 τ nicht als gemeinsame Aktion aufgefasst werden.

Beispiel:

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

Minimales, beobachtbar äquivalentes LTS:

Schnittstellen-Operator

$$E @ I$$

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

Wirkung:

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

Bemerkung:

- 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:
 $(E \parallel F) / \{\text{neu/alt}\} @ \{a_1, \dots, a_k\}$

Beispiel:

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

Alphabeterweiterung

Definition:

(1) Sei $T = (S, A, \Delta, q)$ ein LTS.

Dann heißt die Menge $\alpha T =_{def} A \setminus \{\tau\}$ das *Alphabet* von T .

(2) Sei E ein Prozessausdruck mit $\text{Its}(E) = T$.

Dann heißt die Menge $\alpha E =_{def} \alpha T$ das *Alphabet* von E .

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

Definition:

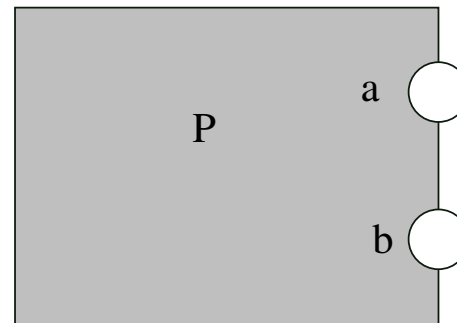
Sei E ein Prozessausdruck und $B \subseteq \text{Labels}$ eine Menge von Aktionen. Dann ist die *Alphabeterweiterung* $E + B$ ein Prozessausdruck mit $\text{FV}(E + B) = \text{FV}(E)$.

Beispiel:

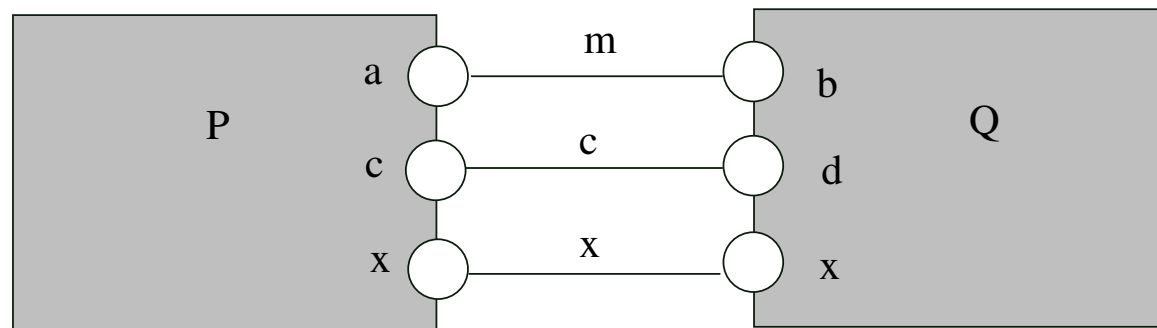
Strukturdiagramme

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

Strukturdiagramm eines Prozesses mit Alphabet $\{a,b\}$

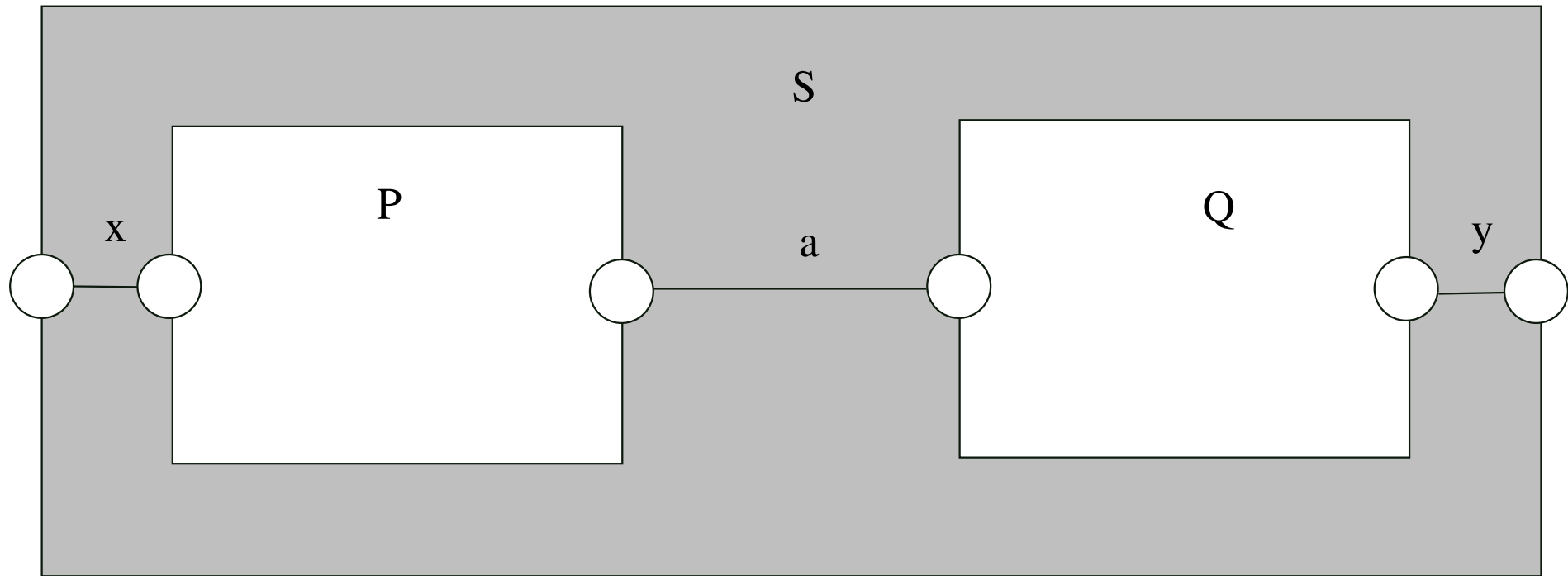


Strukturdiagramm von zwei interaktiven Prozessen



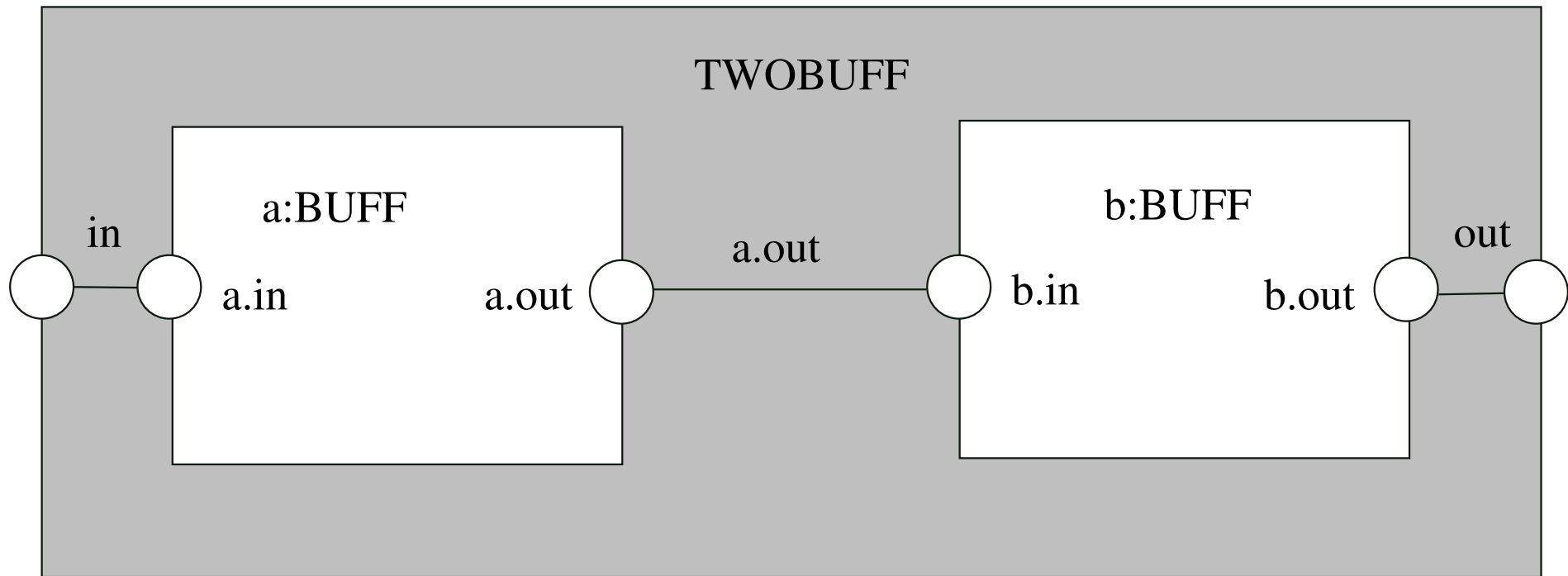
$(P \parallel Q) / \{m/a, m/b, c/d\}$

Strukturdiagramm von interaktiven Prozessen mit Schnittstellen

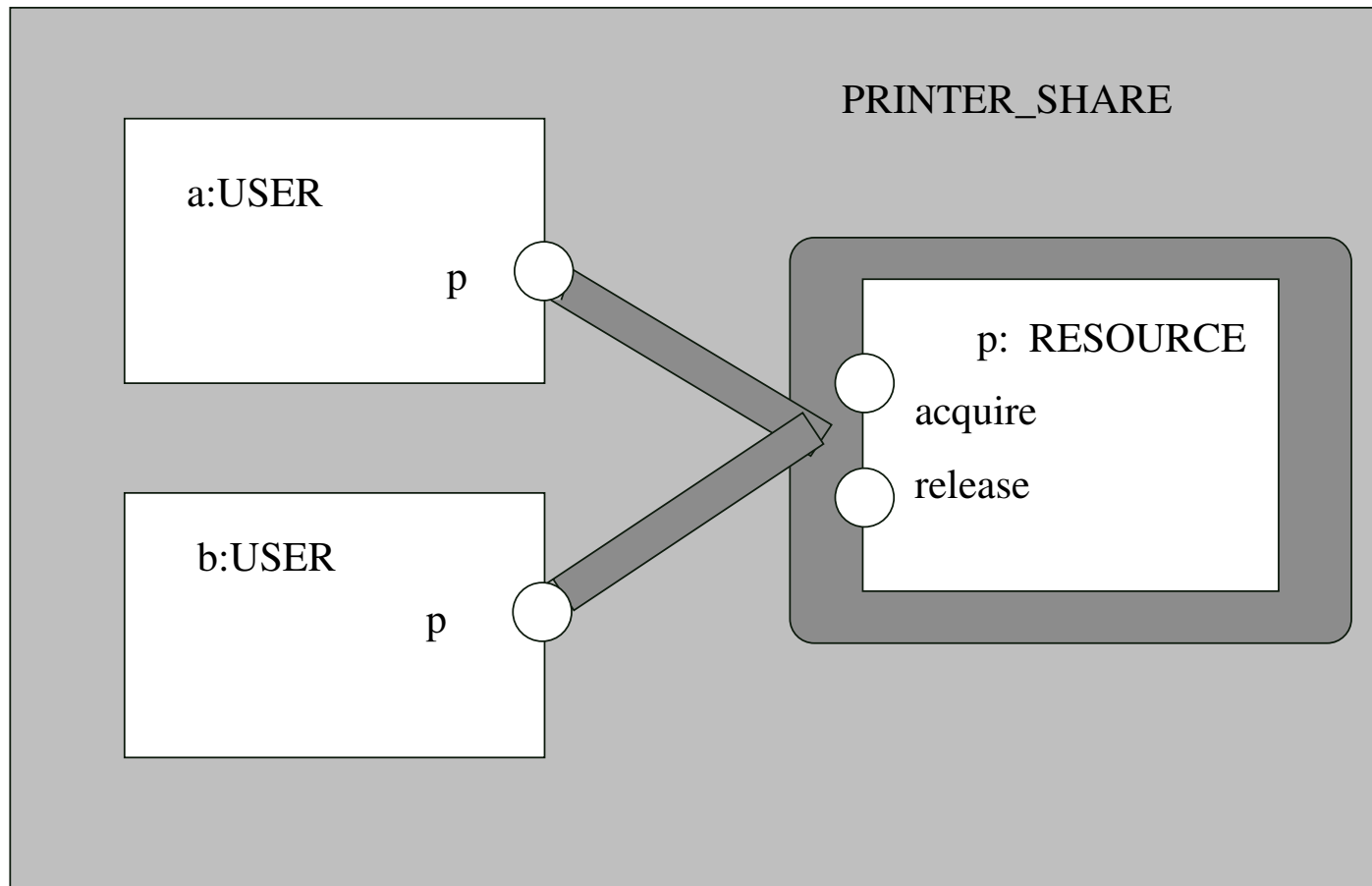


$$\parallel S = (P \parallel Q) @ \{x, y\}$$

Beispiel (Komposition zweier Puffer mit Schnittstelle):



Strukturdiagramm von Prozessen mit Ressource-Sharing



$\text{RESOURCE} = (\text{acquire} \rightarrow \text{release} \rightarrow \text{RESOURCE}).$

$\text{USER} = (\text{p.acquire} \rightarrow \text{use} \rightarrow \text{p.release} \rightarrow \text{USER}).$

$\text{||PRINTER_SHARE} = (\text{a:USER} \parallel \text{b:USER} \parallel \{ \text{a,b} \} :: \text{p:RESOURCE}).$

3.2 Semantik von parallelen Prozessen

Die induktive Definition der Funktion $\text{Its}: \mathcal{E} \longrightarrow \mathcal{T}$ wird folgendermaßen erweitert auf:

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

Beobachtbare Äquivalenz

Zwei Prozesse sind beobachtbar äquivalent, wenn sie für einen (externen) Beobachter, der keine (internen) τ -Aktionen sehen kann, dasselbe Verhalten haben.

Definition:

Seien $T, T' \in \mathcal{T}$ zwei LTSs und sei $a \in \text{Labels} \cup \{\epsilon\}$.

T geht mit a modulo τ über in T' , geschrieben $T \xrightarrow{a} T'$, wenn es $U, V \in \mathcal{T}$ gibt mit

$$T \xrightarrow{\tau^*} U \xrightarrow{a} V \xrightarrow{\tau^*} T'$$

wobei „ $\xrightarrow{\tau^*}$ “ für eine beliebige (endliche) Anzahl von τ -Übergängen steht und falls $a = \epsilon$, $U \xrightarrow{a} V$ für $U = V$ steht.

Beispiele:

Definition (Beobachtbare Äquivalenz von LTSen):

Die *beobachtbare Äquivalenz* (*schwache Äquivalenz*) ist die größte Relation $\approx \in \mathcal{T} \times \mathcal{T}$, so dass für alle $T, T' \in \mathcal{T}$ mit $T \approx T'$ gilt:

$$(0) \quad \alpha T = \alpha T'.$$

$$(1) \quad T \xrightarrow{a} R \quad (a \in \text{Labels} \cup \{\epsilon\}, R \in \mathcal{T}) \implies \\ \exists R' \in \mathcal{T} \text{ mit } T' \xrightarrow{a} R' \text{ und } R \approx R'$$

$$(2) \quad T' \xrightarrow{a} R' \quad (a \in \text{Labels} \cup \{\epsilon\}, R' \in \mathcal{T}) \implies \\ \exists R \in \mathcal{T} \text{ mit } T \xrightarrow{a} R \text{ und } R \approx R'$$

Bemerkungen:

- Eine Relation $\text{Rel} \subseteq \mathcal{T} \times \mathcal{T}$, die die oben genannten Eigenschaften (0), (1) und (2) erfüllt (mit Rel statt \approx), nennt man *schwache Bisimulation*.
- \approx ist die Vereinigung aller schwachen Bisimulationen.
(Beweis: Die Vereinigung von schwachen Bisimulationen ergibt wieder eine schwache Bisimulation).
- \approx ist eine Äquivalenzrelation.
(Beweis analog zur starken Äquivalenz.)

Beispiele:

Definition (Beobachtbare Äquivalenz von Prozessen):

Zwei Prozesse $E, F \in \mathcal{E}$ sind *beobachtbar äquivalent* (*schwach äquivalent*), geschrieben $E \approx F$, wenn gilt: $\text{Its}(E) \approx \text{Its}(F)$.

Beispiele:

Gesetze für beobachtbare Äquivalenz

$$(a \rightarrow E \mid b \rightarrow F) \approx (b \rightarrow F \mid a \rightarrow E)$$

$$(a \rightarrow E \mid a \rightarrow E) \approx (a \rightarrow E)$$

$$(E \parallel F) \approx (F \parallel E)$$

$$((E \parallel F) \parallel G) \approx (E \parallel (F \parallel G))$$

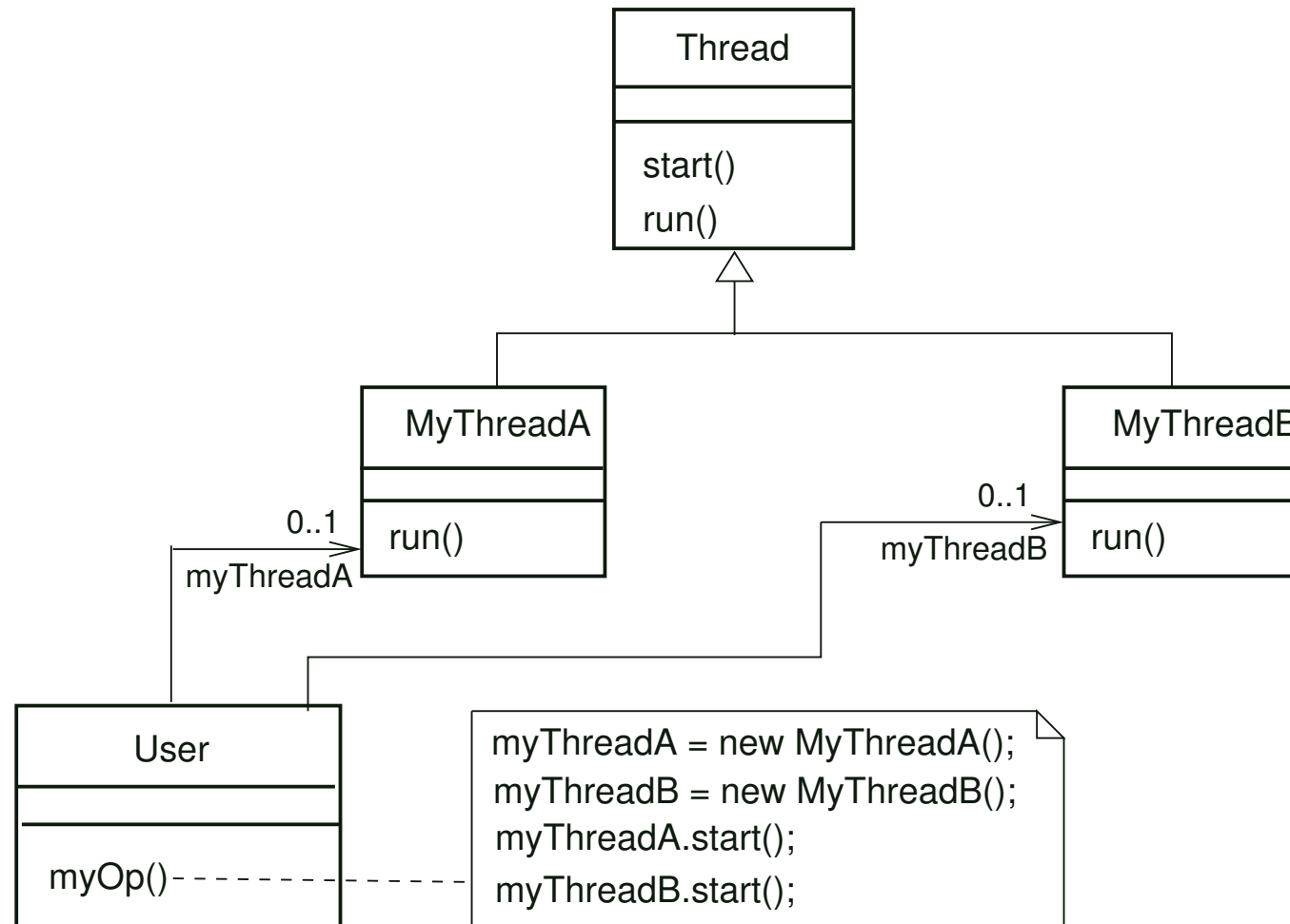
$$(E \parallel E) \approx E$$

$$(E \parallel STOP) \approx E$$

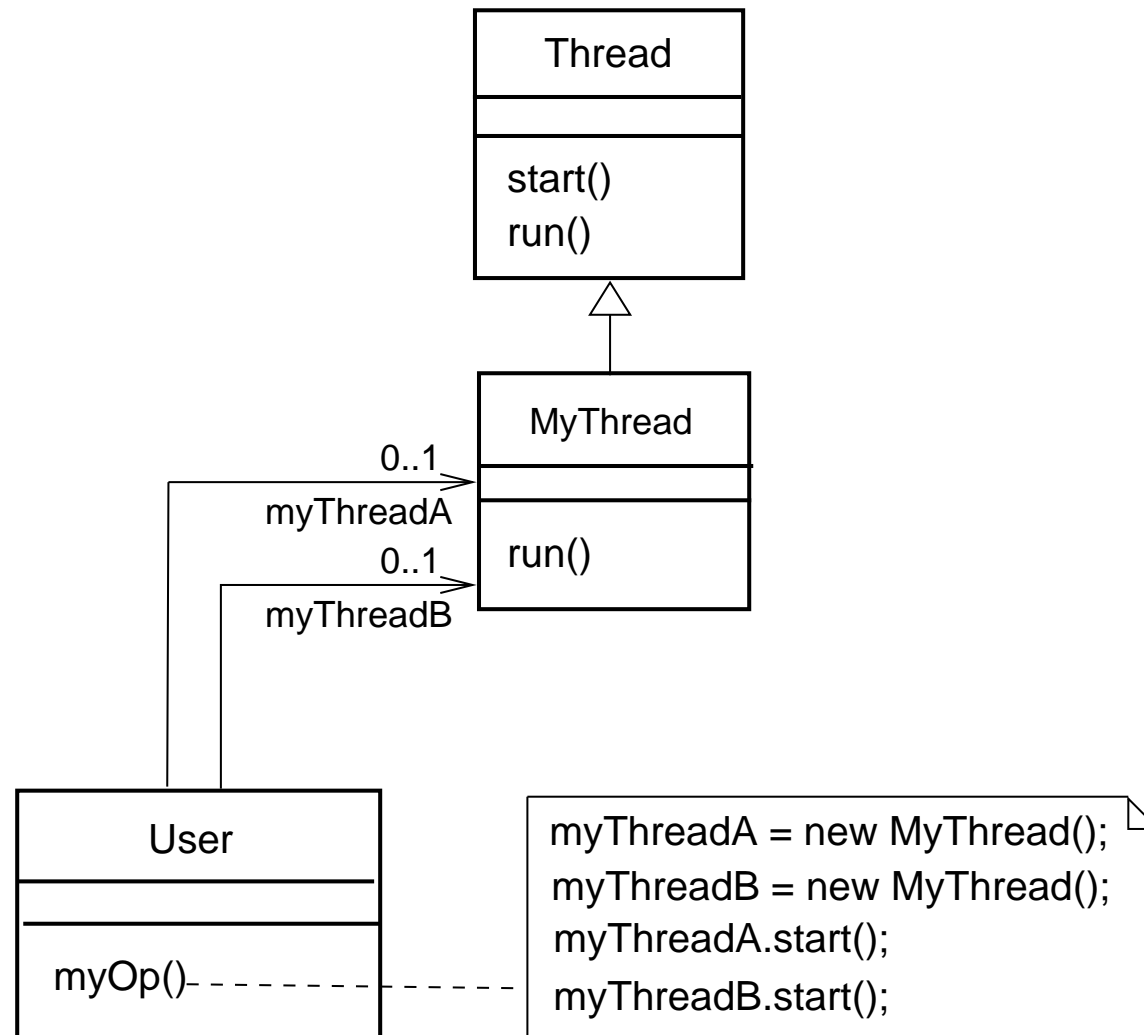
$$(a \rightarrow E) \setminus \{a\} \approx E$$

3.3 Java-Programme mit mehreren Threads

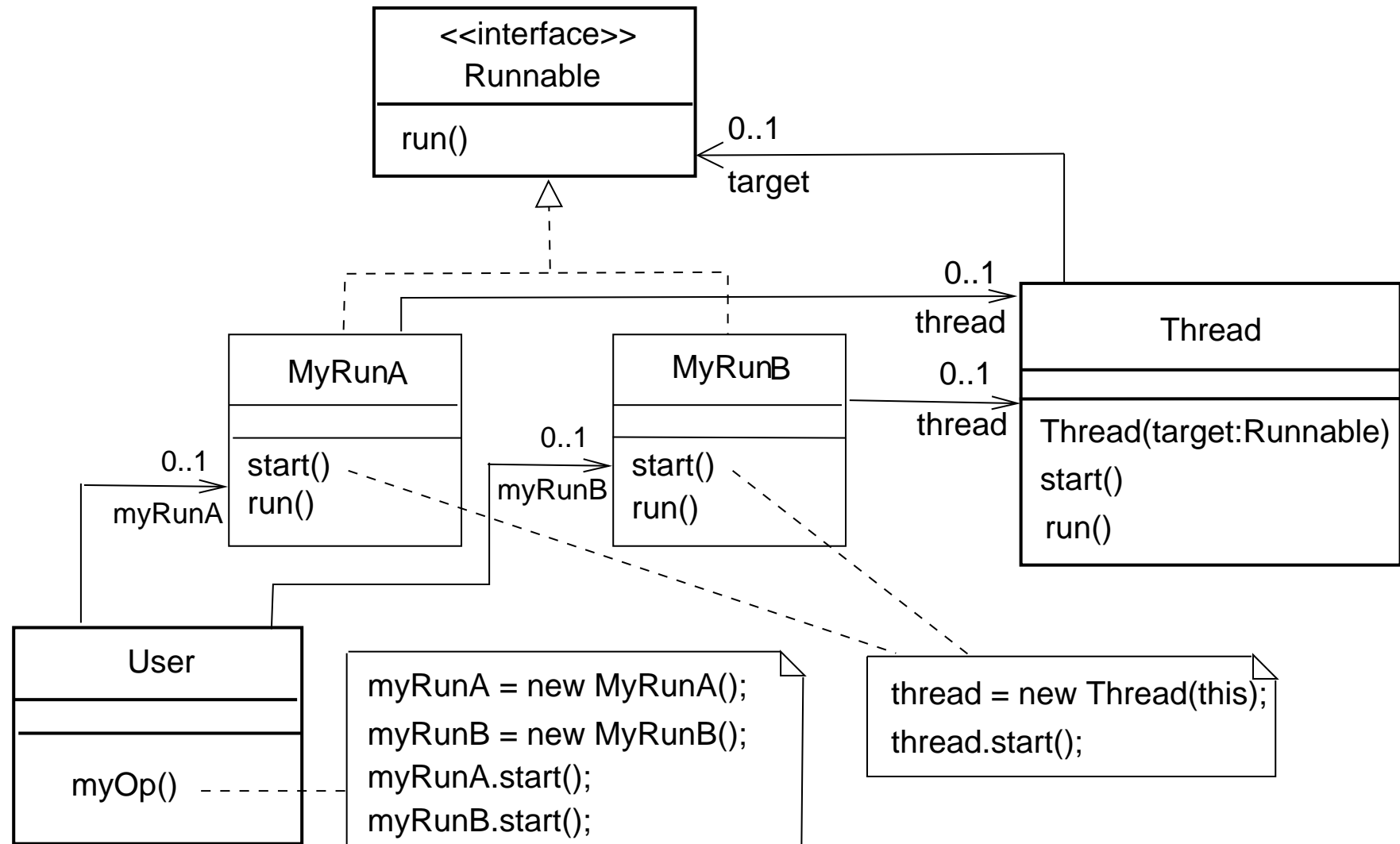
Realisierung mittels Vererbung



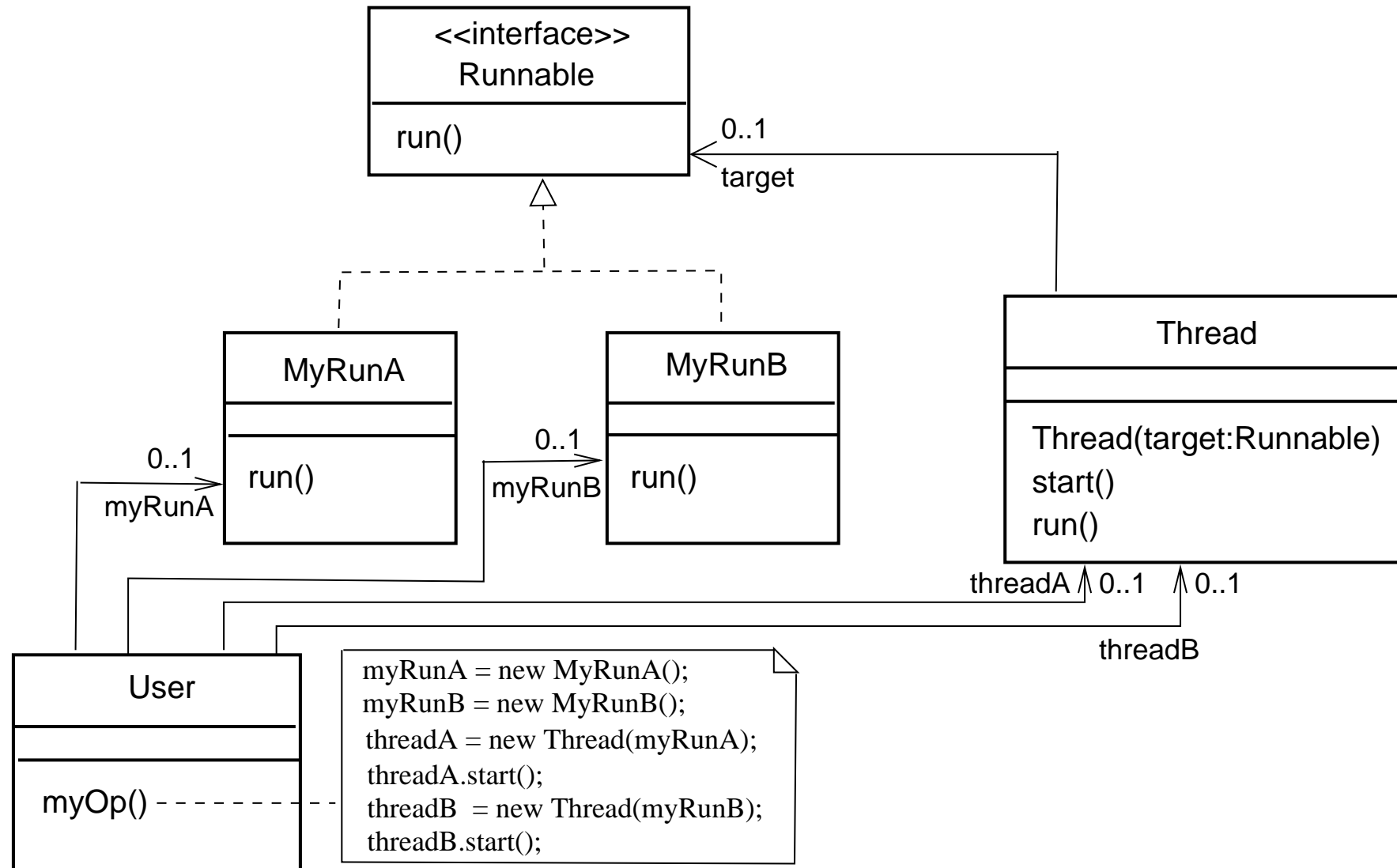
Realisierung mittels Vererbung (Variante)



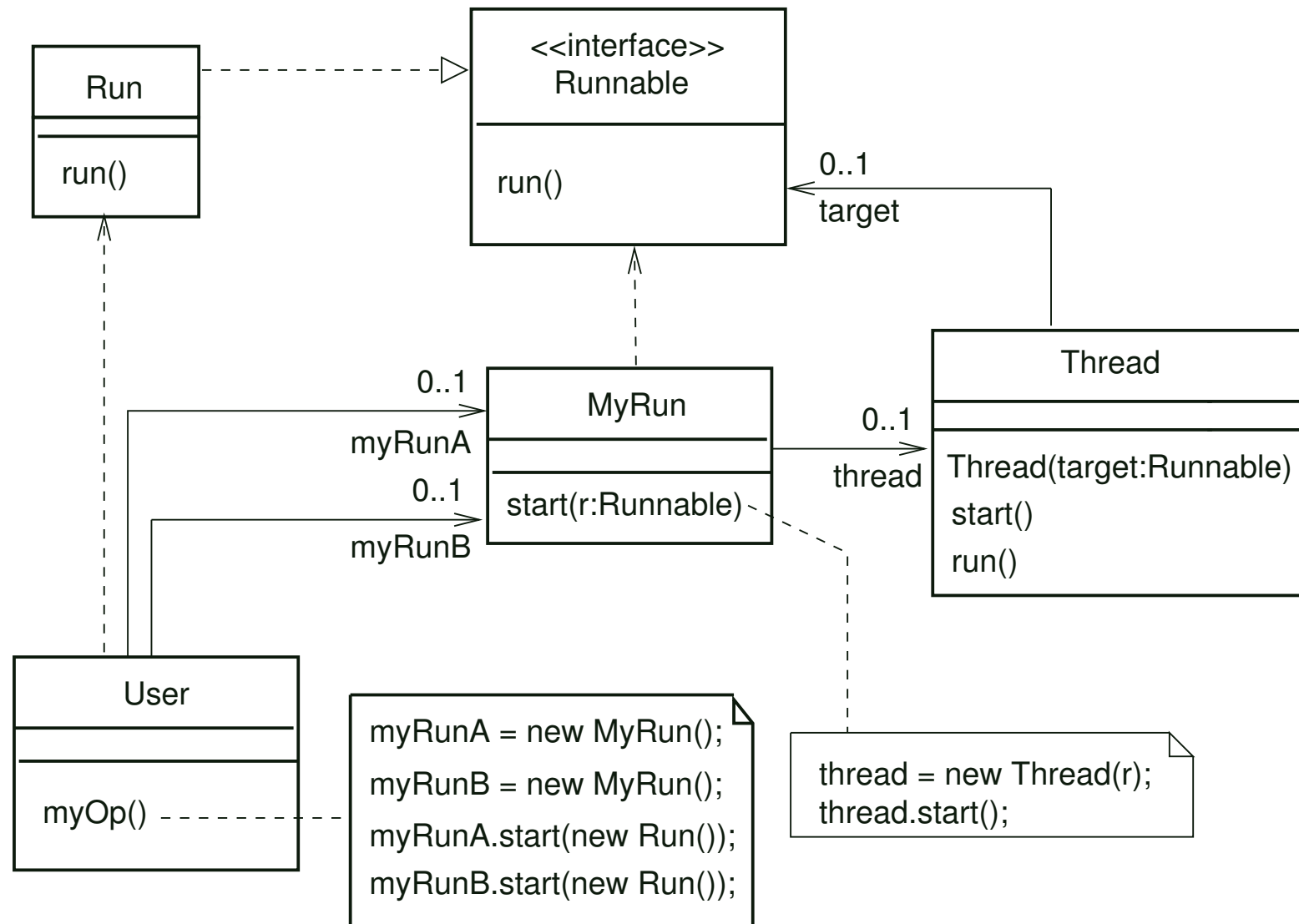
Realisierung durch Verwendung des Interfaces "Runnable"



Realisierung mit "Runnable" (Variante 1)



Realisierung mit "Runnable" (Variante 2)



Beispiel (Rotierende Segmente):

vgl. [Magee, Kramer]

Zwei voneinander unabhängige Threads rotieren ein Kreissegment.

Modellierung:

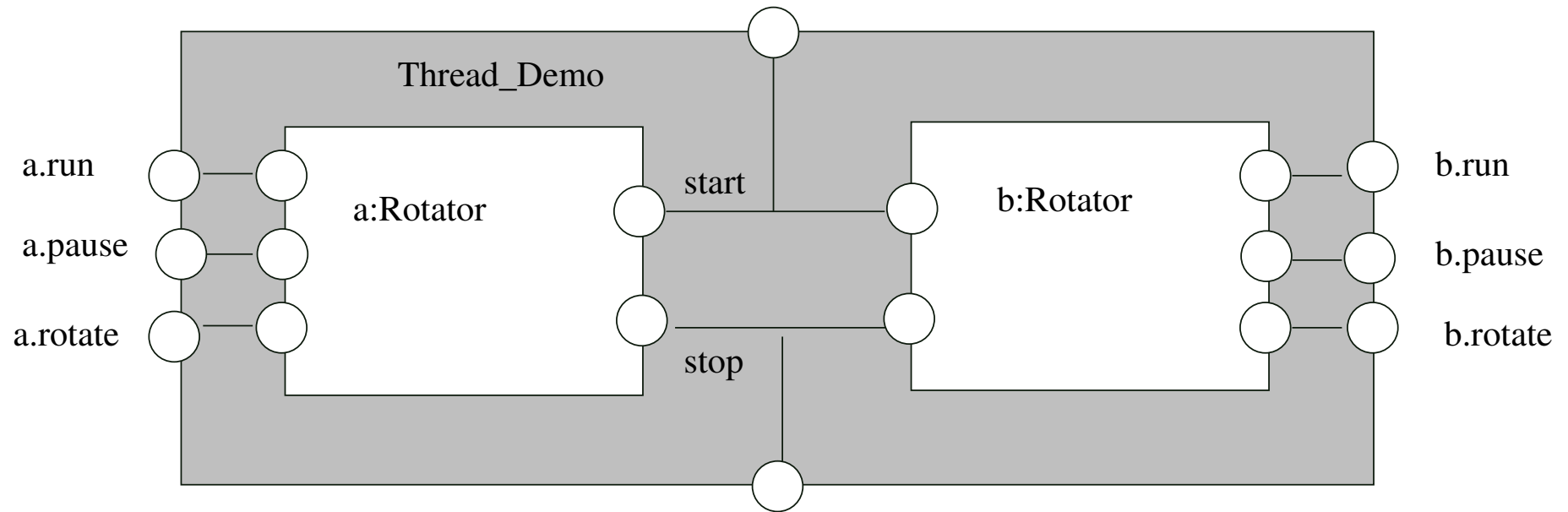
```

ROTATOR  = (start → RUN),
RUN      = ({rotate, run} → RUN
           | pause → PAUSED
           | stop  → STOP),
PAUSED   = (run → RUN
           | pause → PAUSED
           | stop  → STOP).

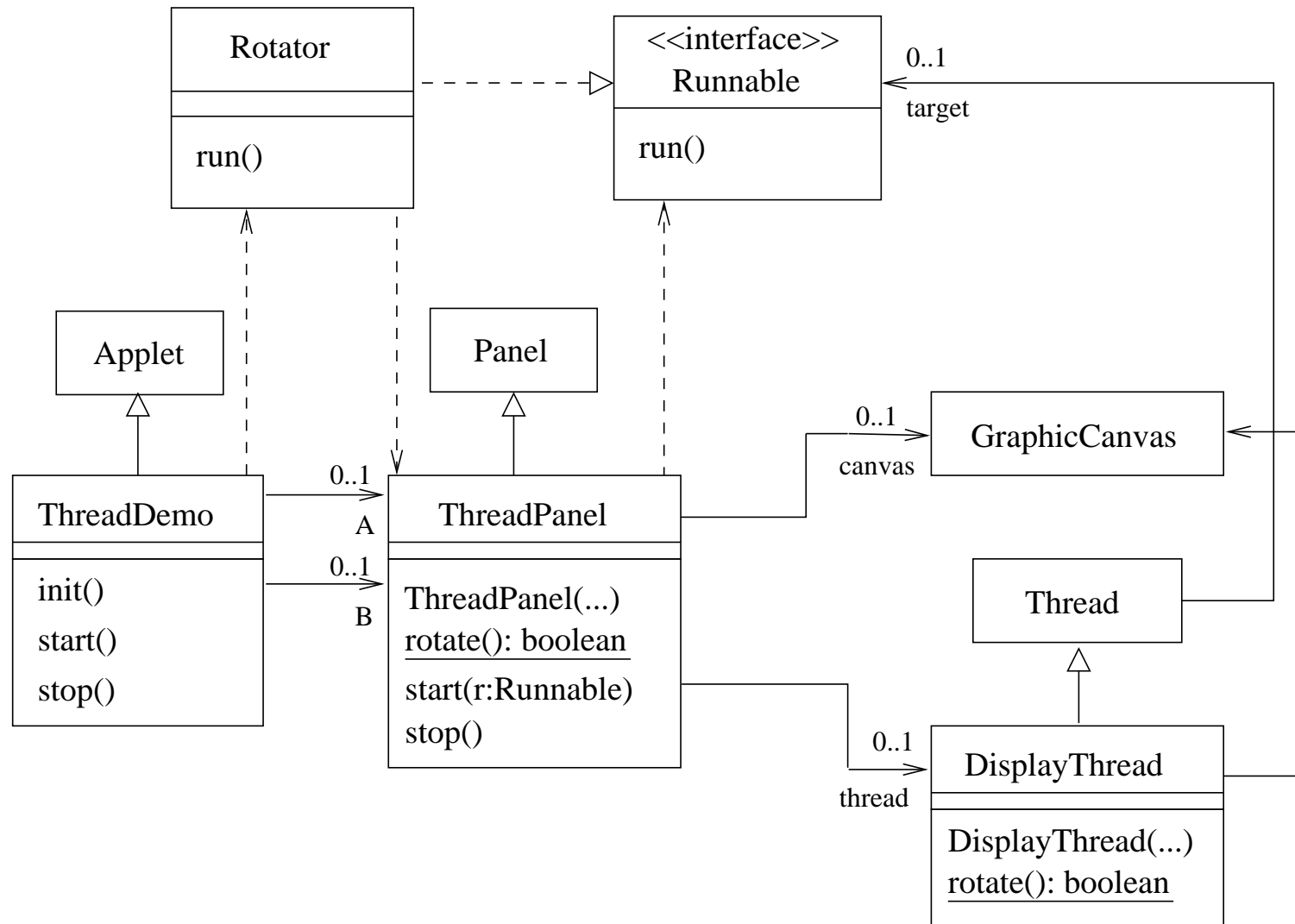
```

||THREAD_DEMO =

(a:ROTATOR||b:ROTATOR)/{start/{a,b}.start,stop/{a,b}.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(display, 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) {}
    }
}
```