

Entwurf und Implementierung paralleler Programme

Prof. Dr. Rolf Hennicker

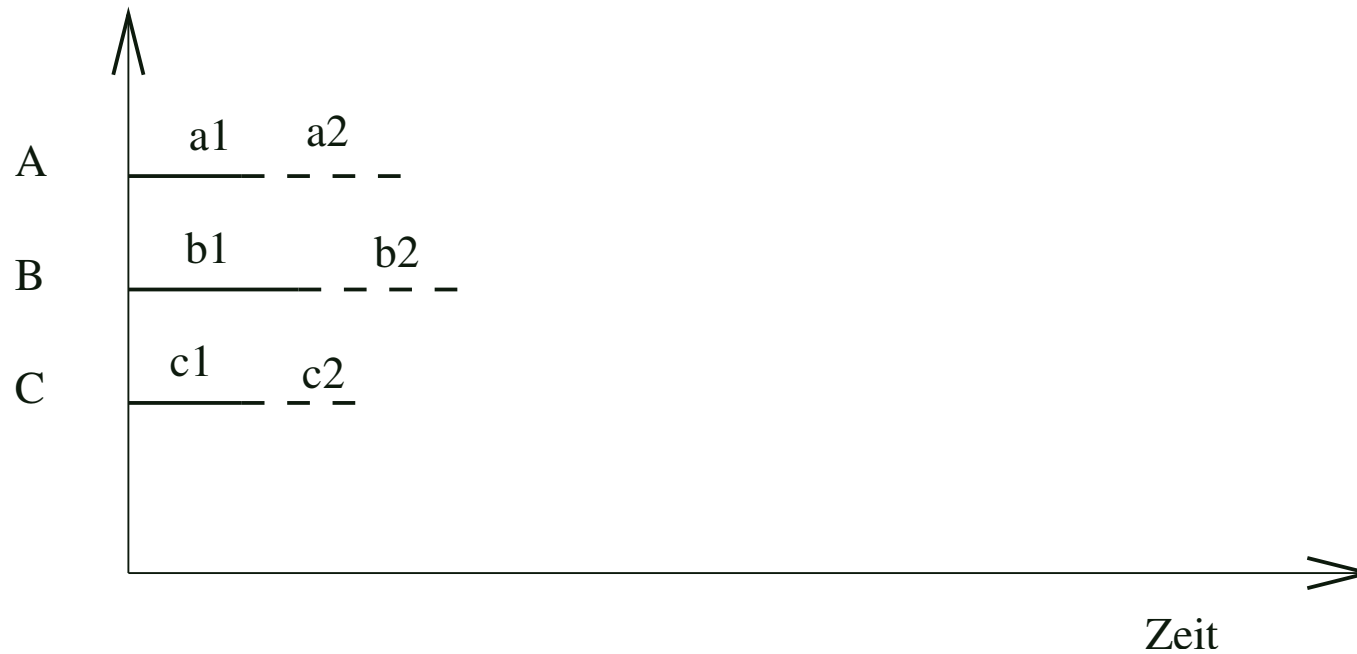
07.05.2009

Kapitel 3

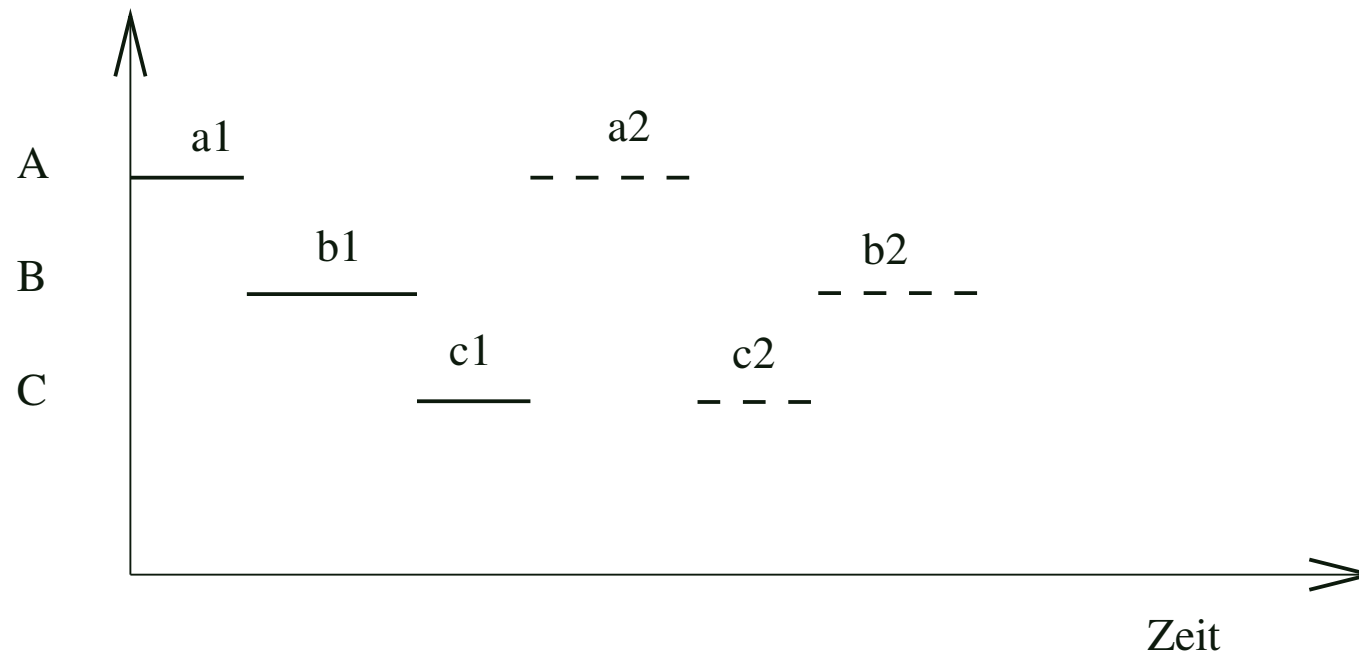
Parallele Prozesse

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

$$\parallel P = E.$$

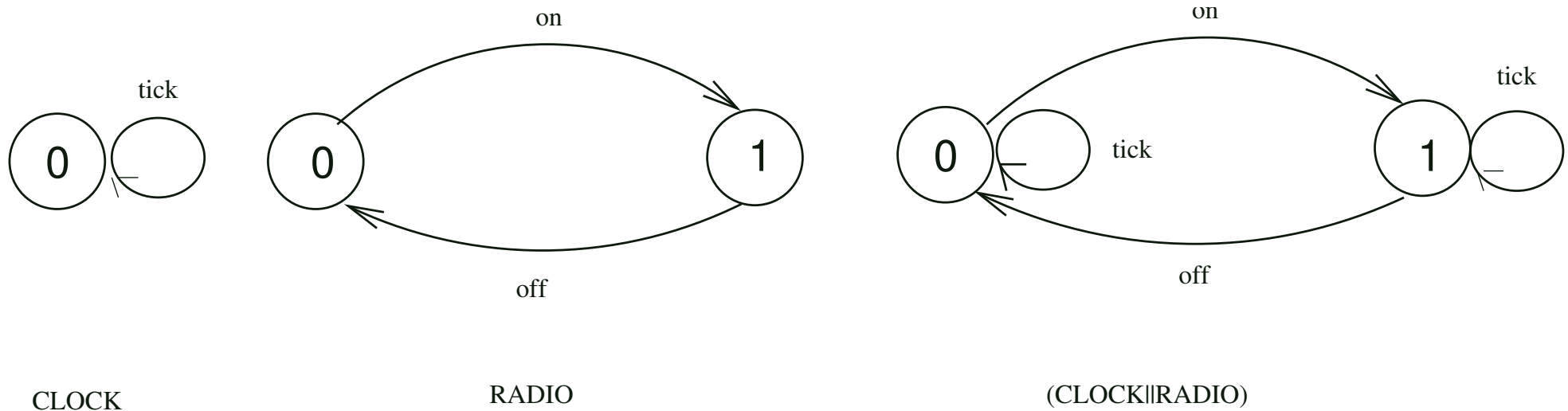
Beispiel:

CLOCK = (tick \rightarrow CLOCK).

RADIO = (on \rightarrow off \rightarrow RADIO).

||CLOCK_RADIO = (CLOCK||RADIO).

Zustandsmaschinen:



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. Diese Interaktion schränkt die möglichen Abläufe stark ein.

$\text{MAKER2} = (\text{make} \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKER2}).$

$\text{USER2} = (\text{ready} \rightarrow \text{use} \rightarrow \text{used} \rightarrow \text{USER2}).$

$\parallel \text{MAKER_USER2} = (\text{MAKER2} \parallel \text{USER2}).$

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, \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:

$\text{CLIENT} = (\text{call} \rightarrow \text{wait} \rightarrow \text{continue} \rightarrow \text{CLIENT}).$

$\parallel \text{TWOCLIENTS} = (a:\text{CLIENT} \parallel b:\text{CLIENT}).$

Dabei ist $a:\text{CLIENT}$ eine Kurznotation für

$\text{CLIENT} \{a.\text{call}/\text{call}, a.\text{wait}/\text{wait}, a.\text{continue}/\text{continue}\}.$

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:

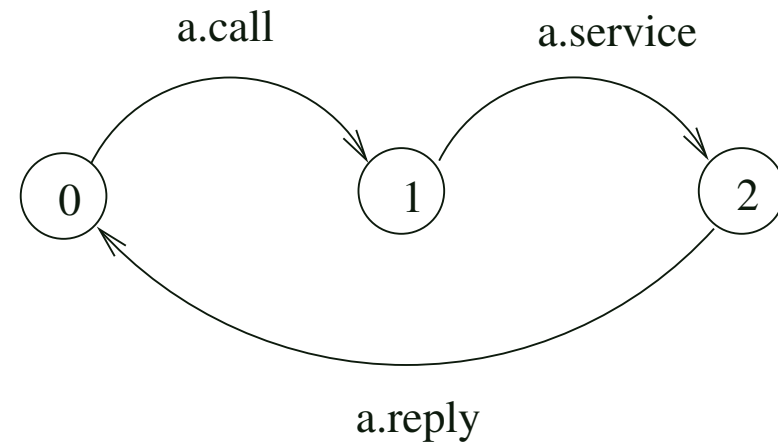
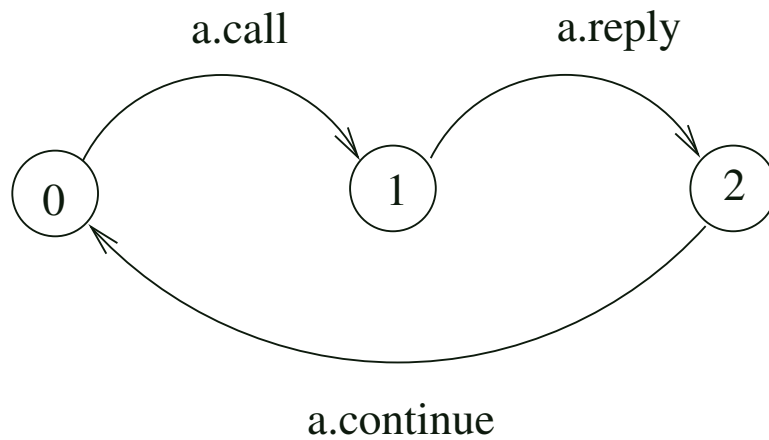
CLIENT = (call \rightarrow wait \rightarrow continue \rightarrow CLIENT).

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

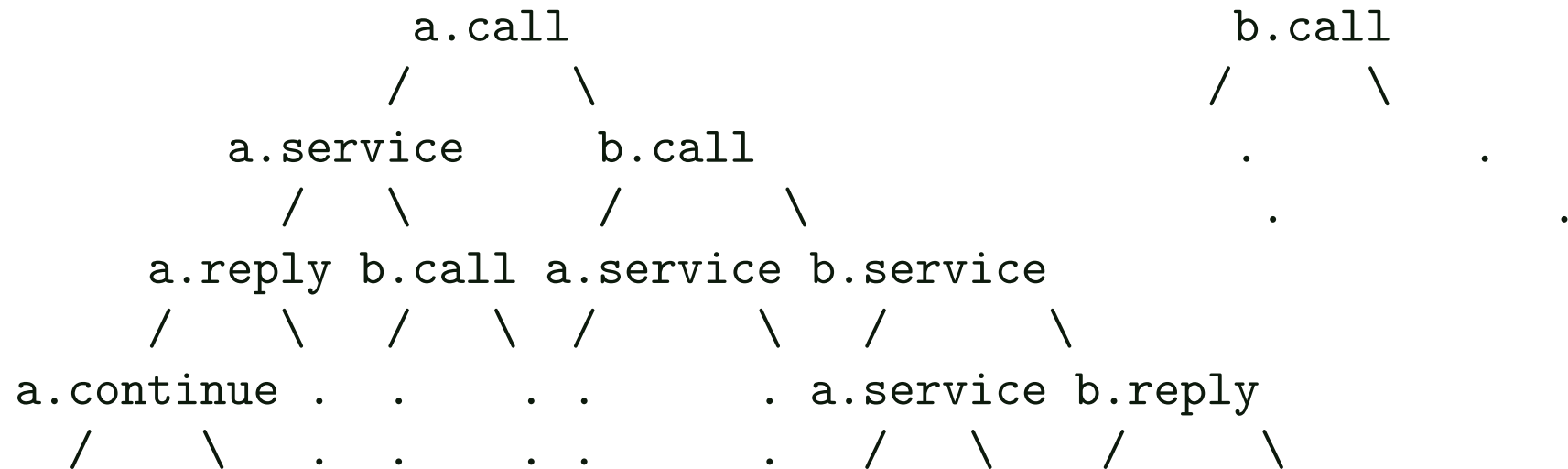
\parallel CLIENT_SERVER = (CLIENT \parallel SERVER) / {call/request, reply/wait}.

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:



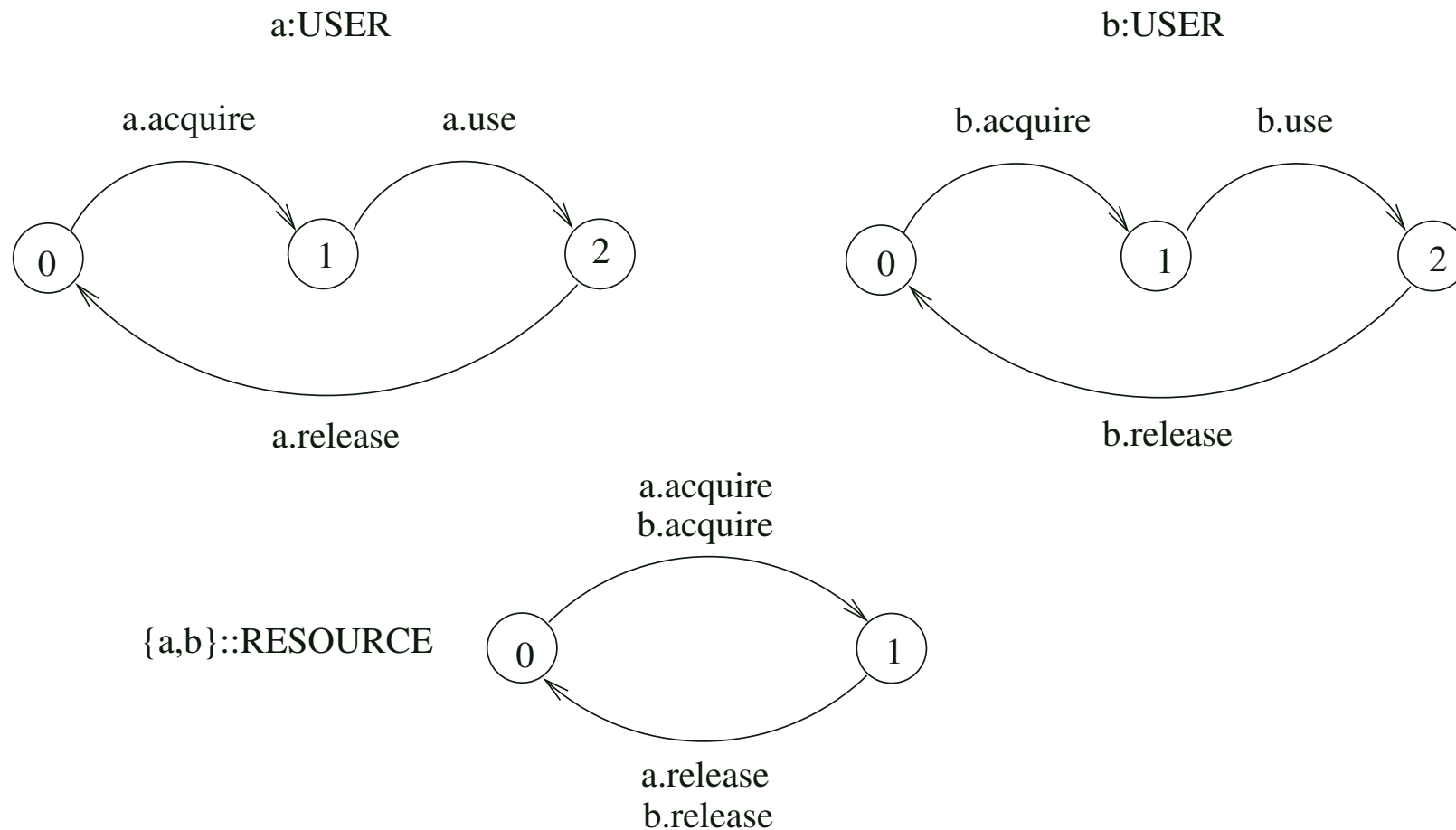
Das LTS ist zu komplex: Man will nur bestimmte Sichten sehen. Z.B. sind interne Abläufe für den Benutzer, der das System von außen betrachtet, uninteressant.

Beispiel (Resource-Sharing):

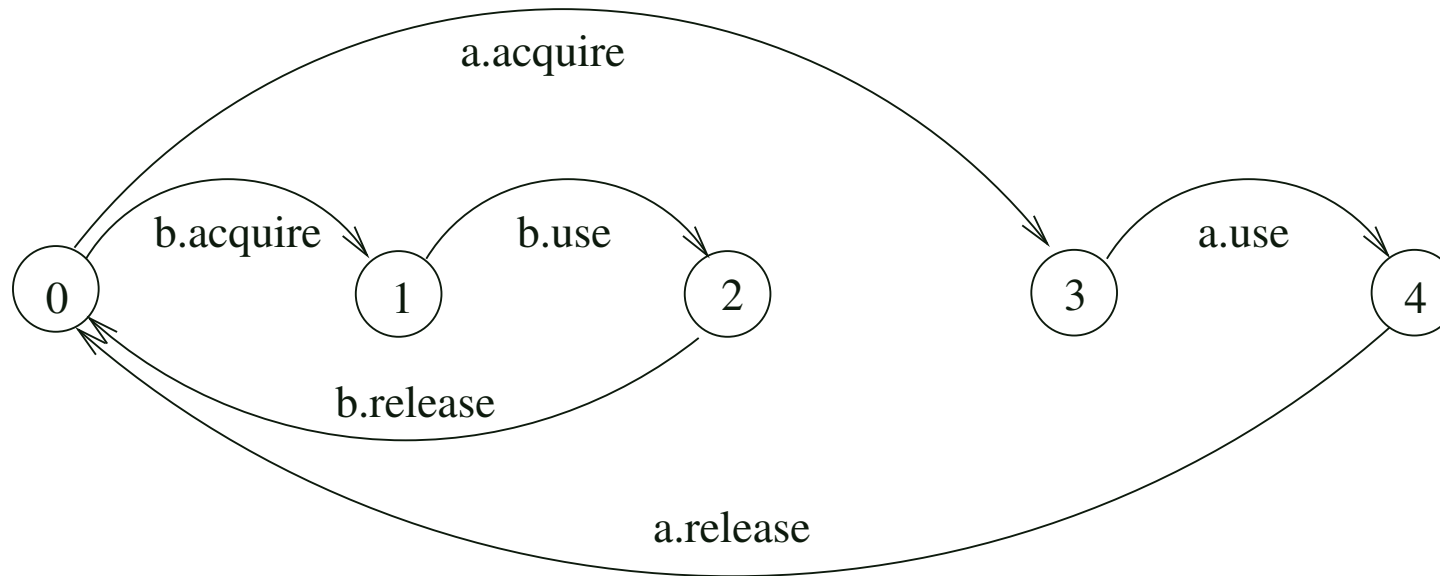
USER = (acquire \rightarrow use \rightarrow release \rightarrow USER).

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

\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

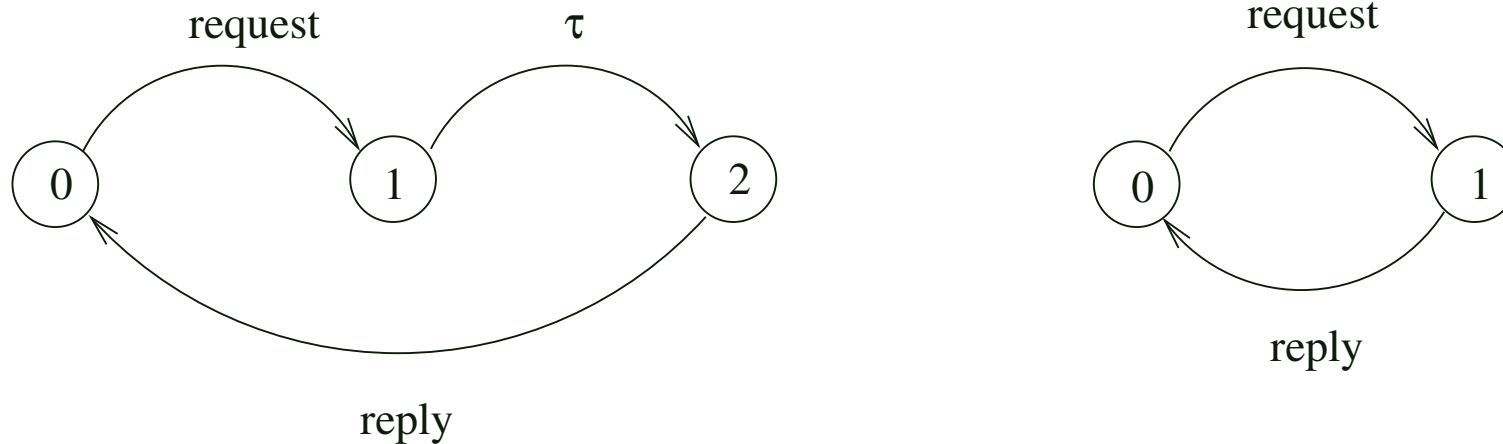
$$E \setminus H$$

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:

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


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

Bemerkung:

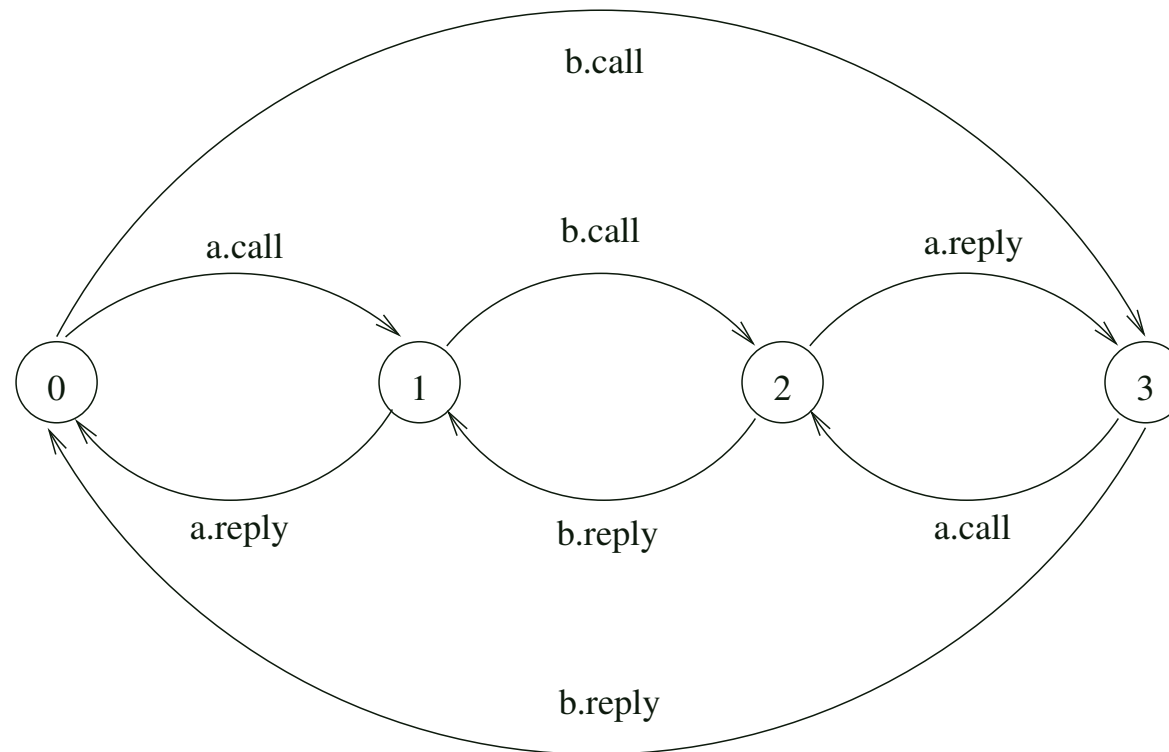
Zumeist wird “Hiding“ nach der parallelen Komposition angewandt, um von der Komplexität eines parallelen Systems zu abstrahieren.

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:
 $(P \parallel Q) / \{\text{neu/alt}\} @ \{a_1, \dots, a_k\}$

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:

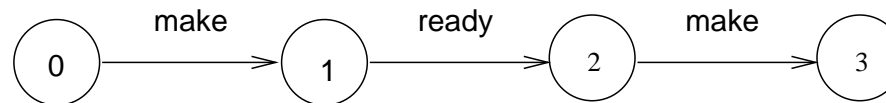
Sei E ein Prozessausdruck und $B \subseteq \text{Labels}$ eine Menge von Aktionen. Dann ist die *Alphabeterweiterung* $E + B$ ein Prozessausdruck mit $\text{FV}(E + B) = \text{FV}(E)$.

Beispiel:

$\text{FMAKER} = (\text{make} \rightarrow \text{ready} \rightarrow \text{FMAKER}) + \{\text{use}\}$.

$\text{USER} = (\text{ready} \rightarrow \text{use} \rightarrow \text{USER})$.

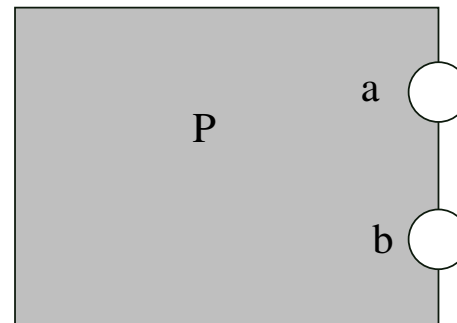
$\parallel \text{FMAKER_USER} = (\text{FMAKER} \parallel \text{USER})$.



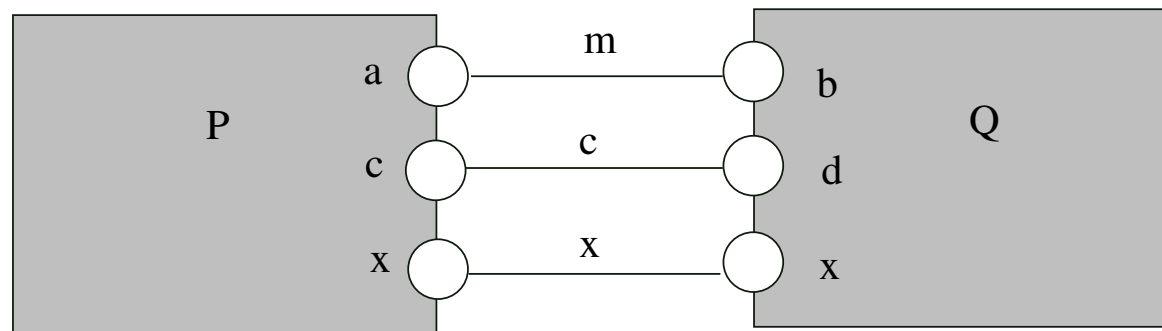
Strukturdiagramme

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

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

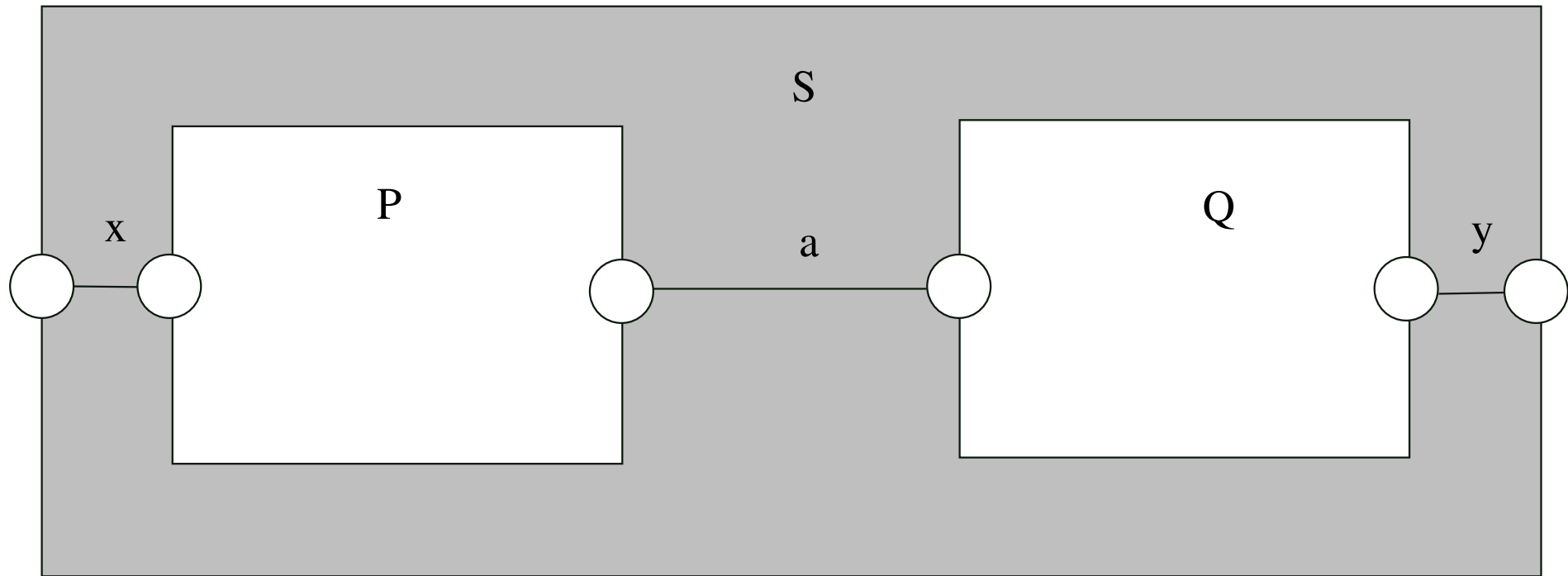


Strukturdiagramm von zwei interaktiven Prozessen



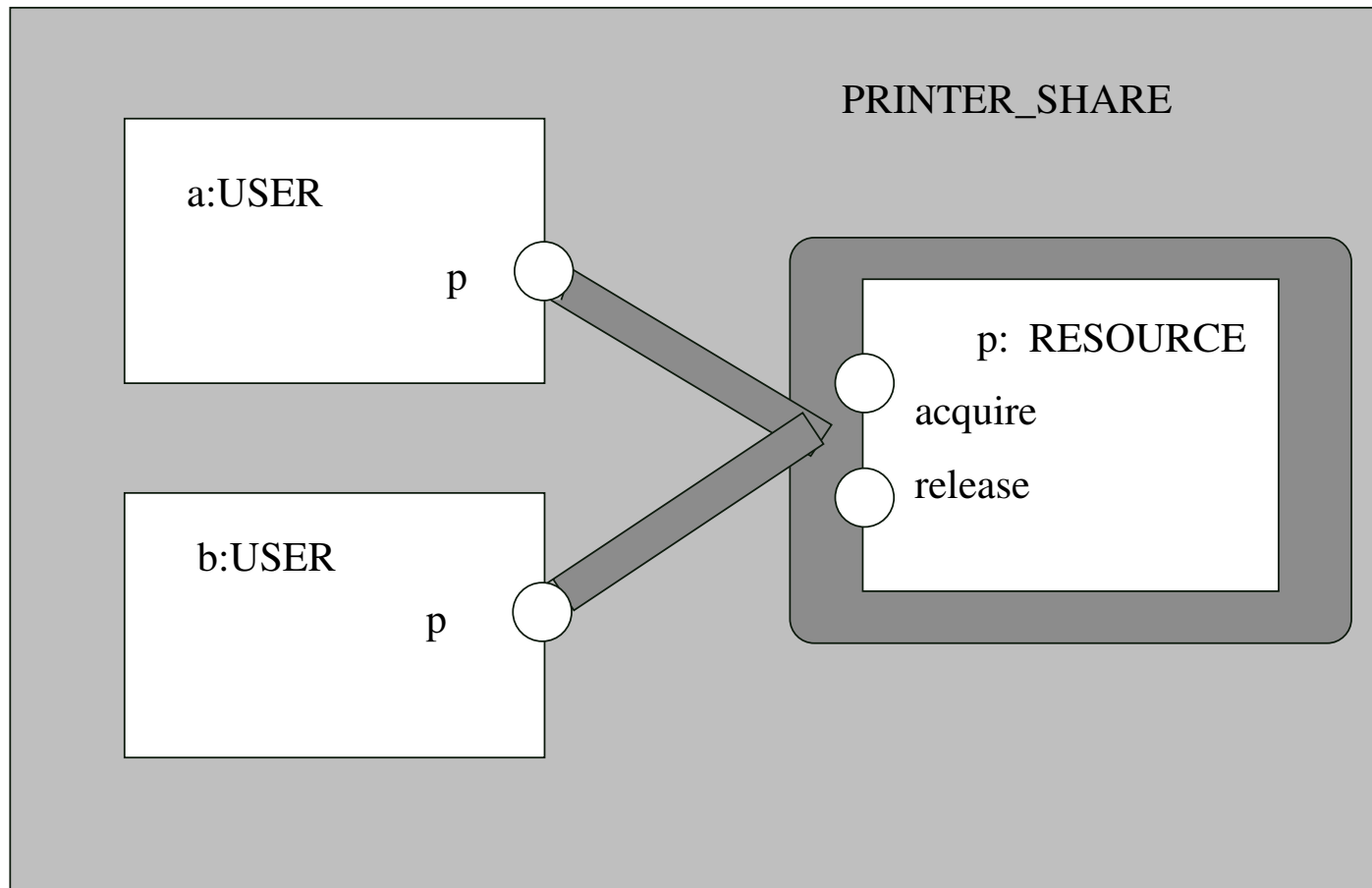
$(P||Q) / \{m/a, m/b, c/d\}$

Strukturdiagramm von interaktiven Prozessen mit Schnittstellen



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

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
- Alphabeterweiterung

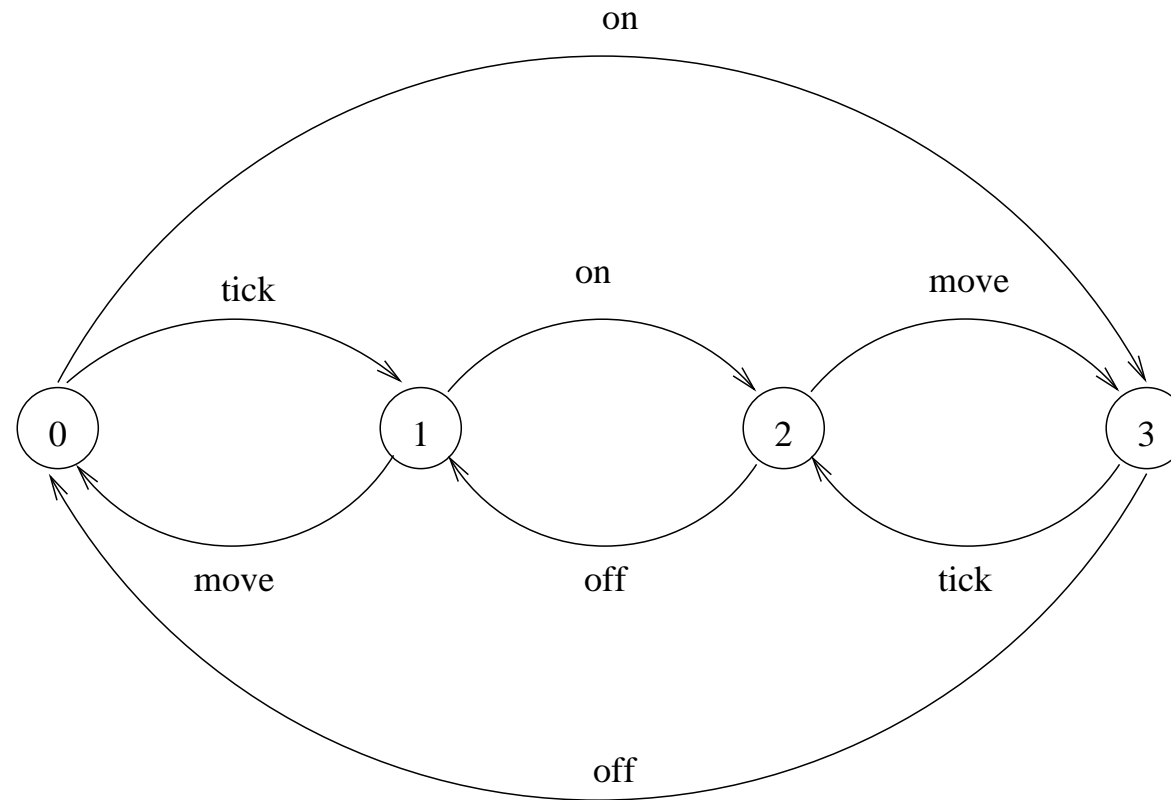
Beispiel:

CLOCK2 = (tick \rightarrow move \rightarrow CLOCK2).

RADIO = (on \rightarrow off \rightarrow RADIO).

\parallel CLOCK2_RADIO = (CLOCK2 \parallel RADIO).

Its(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 $a \in \alpha T \cup \{\epsilon\}$.

q geht mit a modulo τ über in p (bzgl. Δ), geschrieben $q \xRightarrow{a}_{\Delta} p$, 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 und $u \xrightarrow{\epsilon} v$ für $u = v$ steht.

Definition (Schwache Bisimulation):

Seien $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$ mit $\alpha T = \alpha T'$.

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

(2) Falls $q' \xRightarrow{a}_{\Delta'} p'$, dann existiert $p \in S$ mit $q \xRightarrow{a}_{\Delta} p$ 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 $T, T' \in \mathcal{T}$, $T = (S, A, \Delta, q_0)$, $T' = (S', A', \Delta', q_0')$.

T und T' sind *beobachtbar äquivalent*, geschrieben $T \approx T'$, wenn gilt:

- (a) $\alpha T = \alpha T'$.
- (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 LTSen 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 bisimilar*), geschrieben $E \approx F$, wenn gilt: $\text{Its}(E) \approx \text{Its}(F)$.

Beispiele:

Gesetze für beobachtbare Äquivalenz

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

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

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

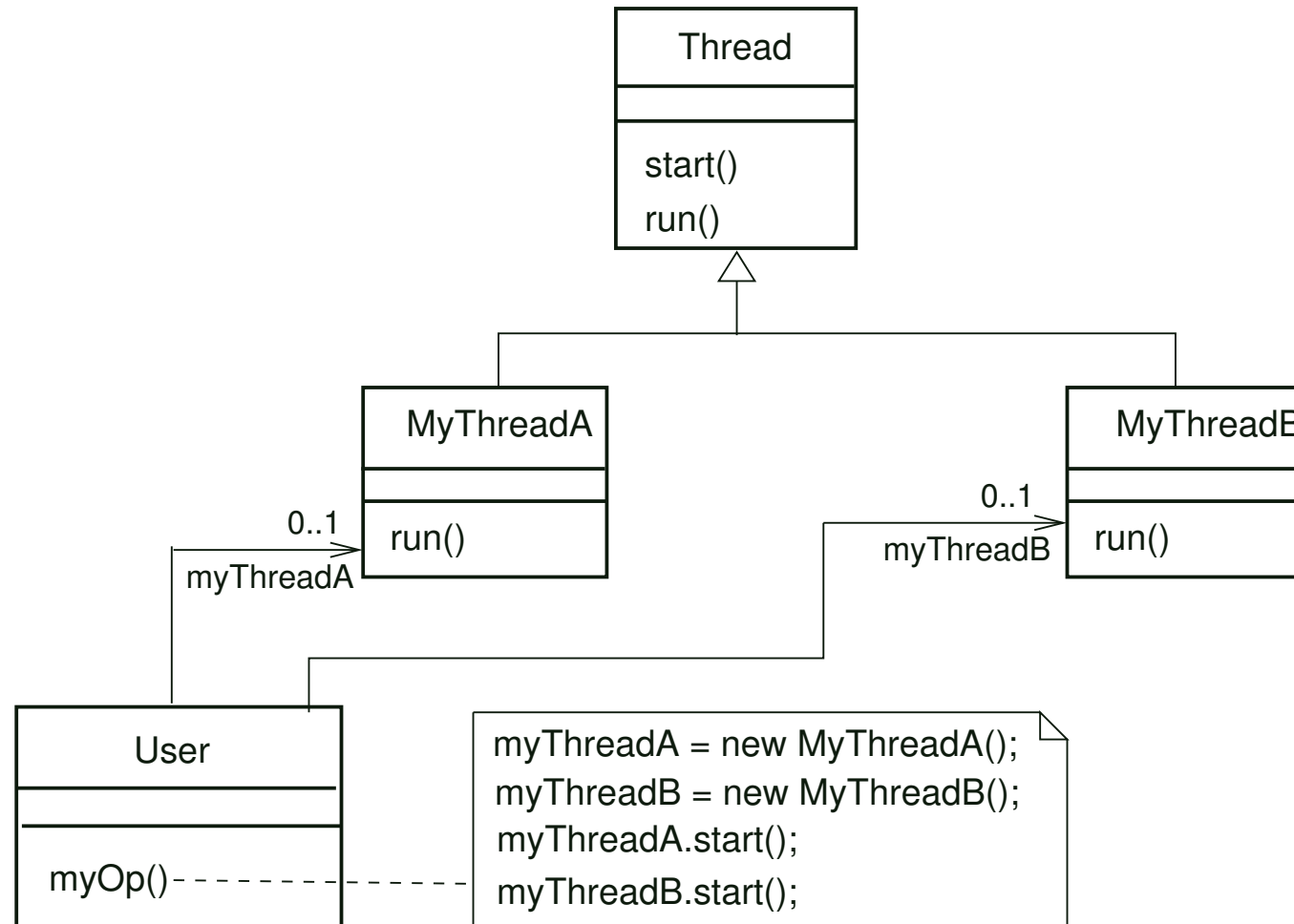
$$((E \parallel F) \parallel G) \approx (E \parallel (F \parallel G))$$

$$(E \parallel \text{STOP}) \approx E$$

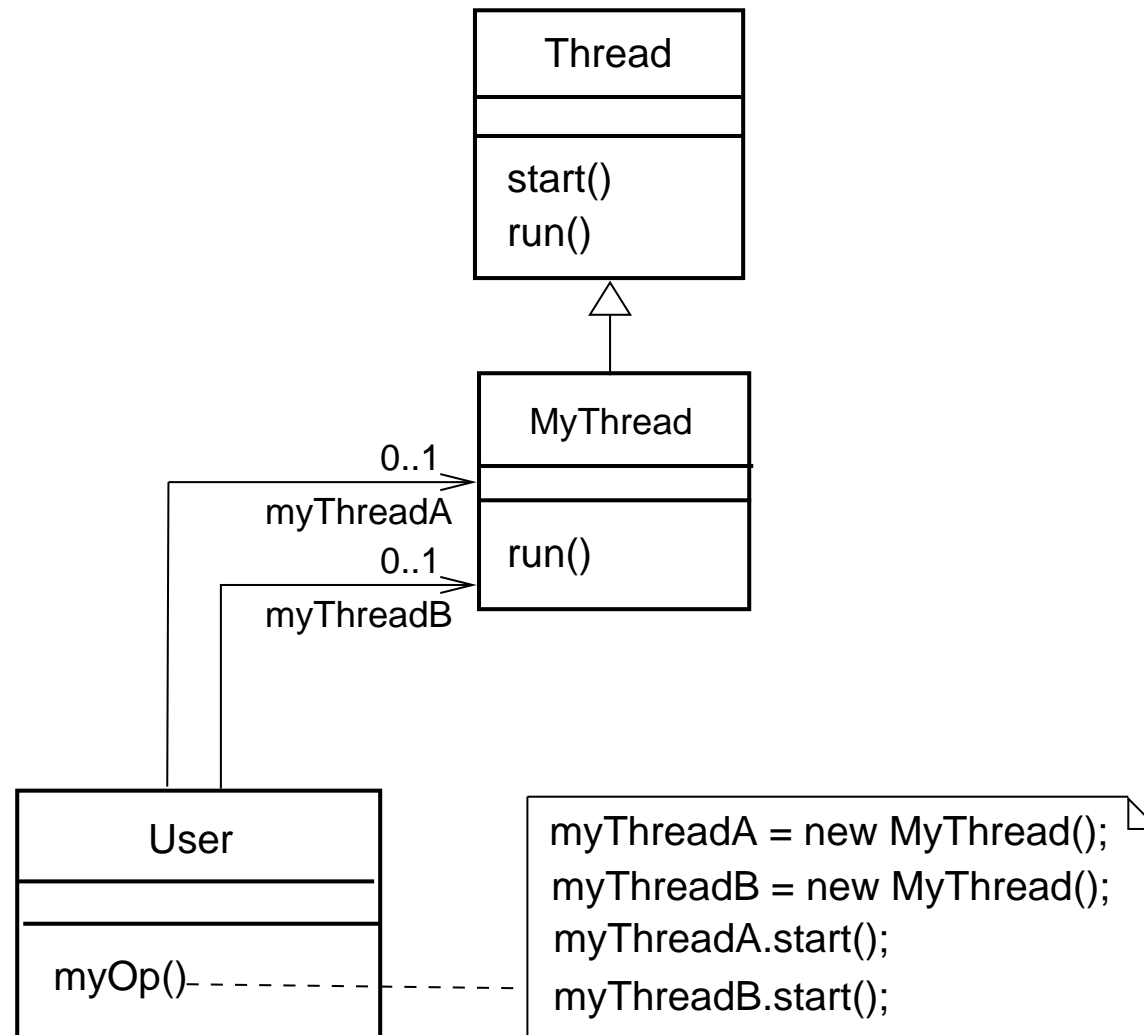
$$(a \rightarrow E) \setminus \{a\} \approx E \quad \text{falls } a \notin \alpha E.$$

3.3 Java-Programme mit mehreren Threads

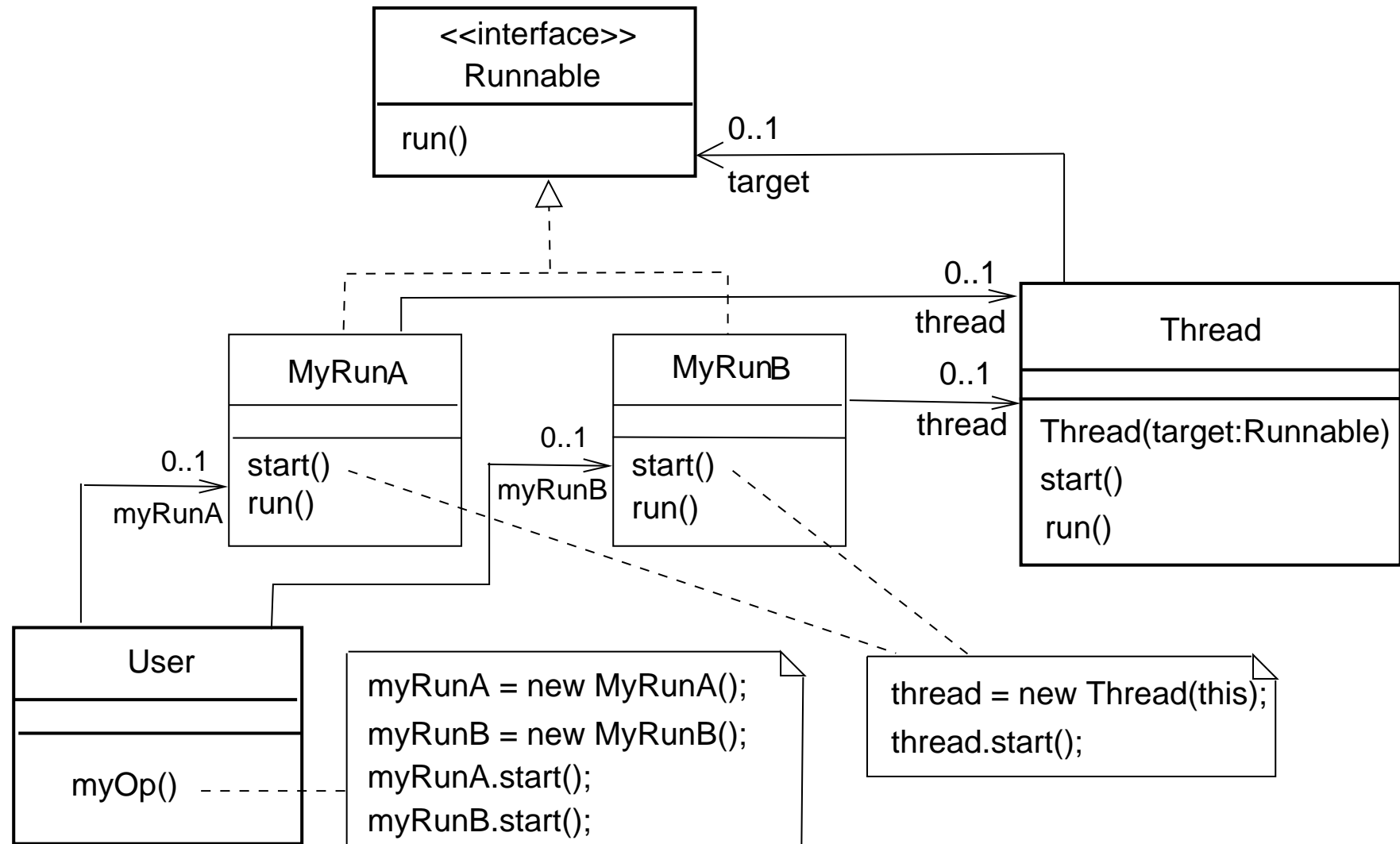
Realisierung mittels Vererbung



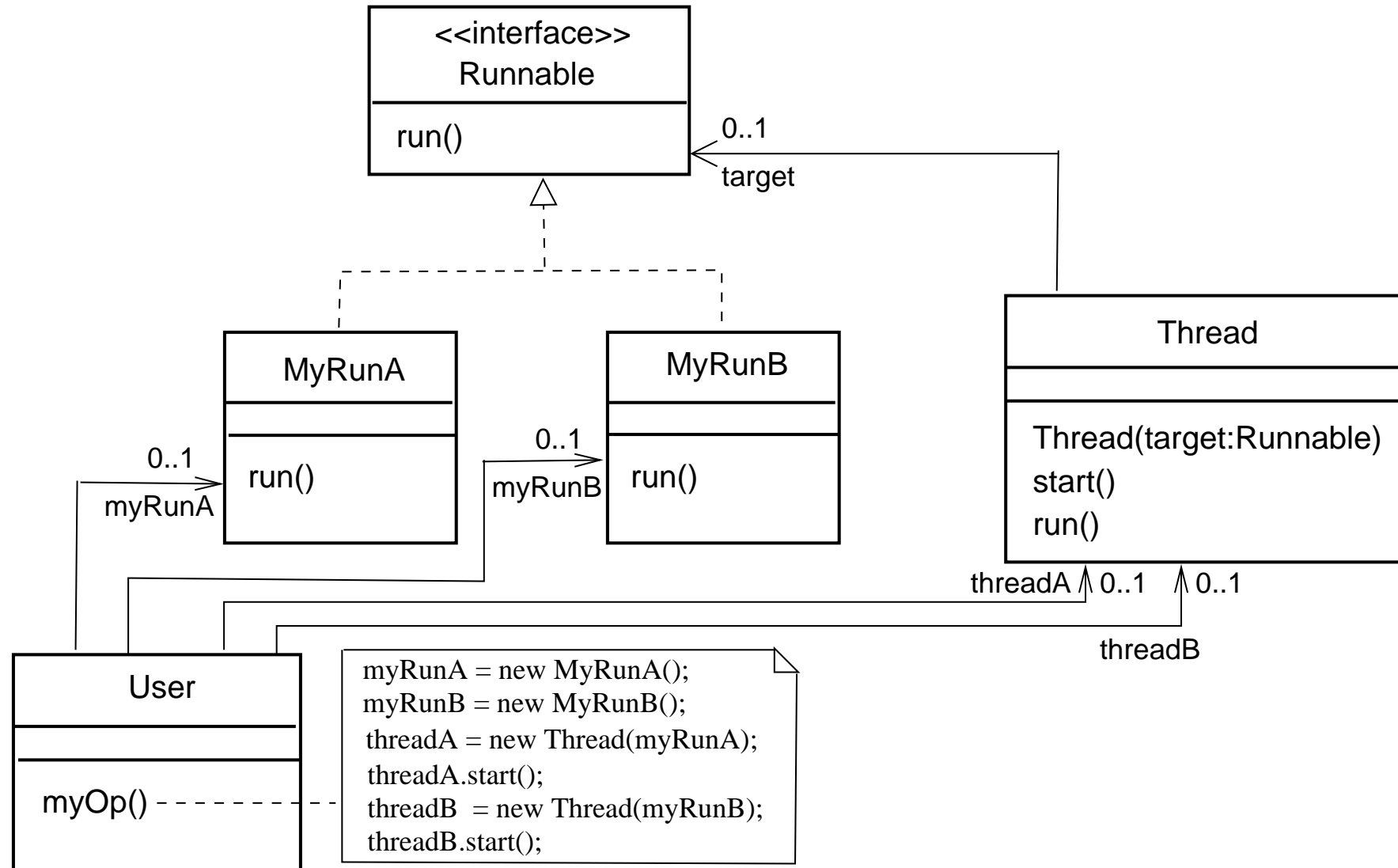
Realisierung mittels Vererbung (Variante)



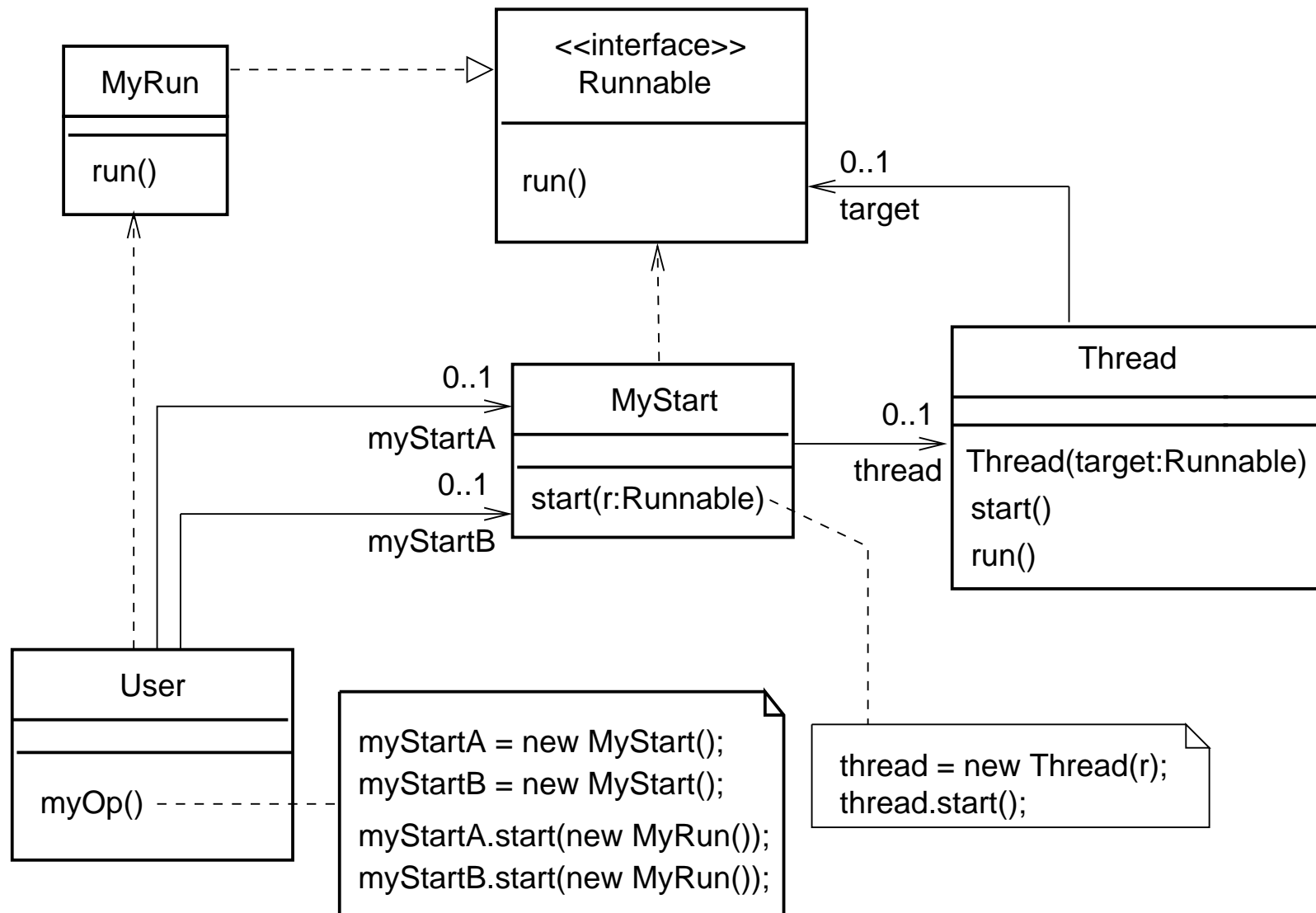
Realisierung durch Verwendung des Interfaces "Runnable"



Realisierung mit "Runnable" (Variante 1)



Realisierung mit "Runnable" (Variante 2)



Beispiel (Rotierende Segmente):

vgl. [Magee, Kramer]

Zwei voneinander unabhängige Threads rotieren ein Kreissegment.

Modellierung:

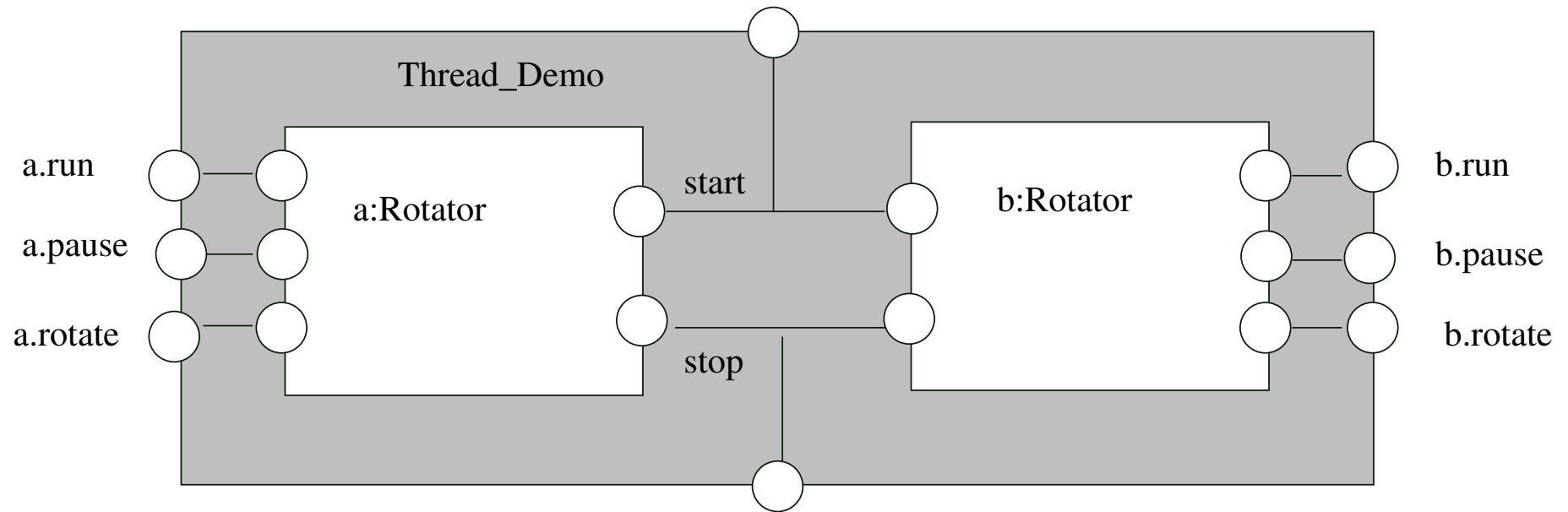
ROTATOR = (start → PAUSED),

PAUSED = (run → RUN
 | pause → PAUSED
 | stop → STOP),

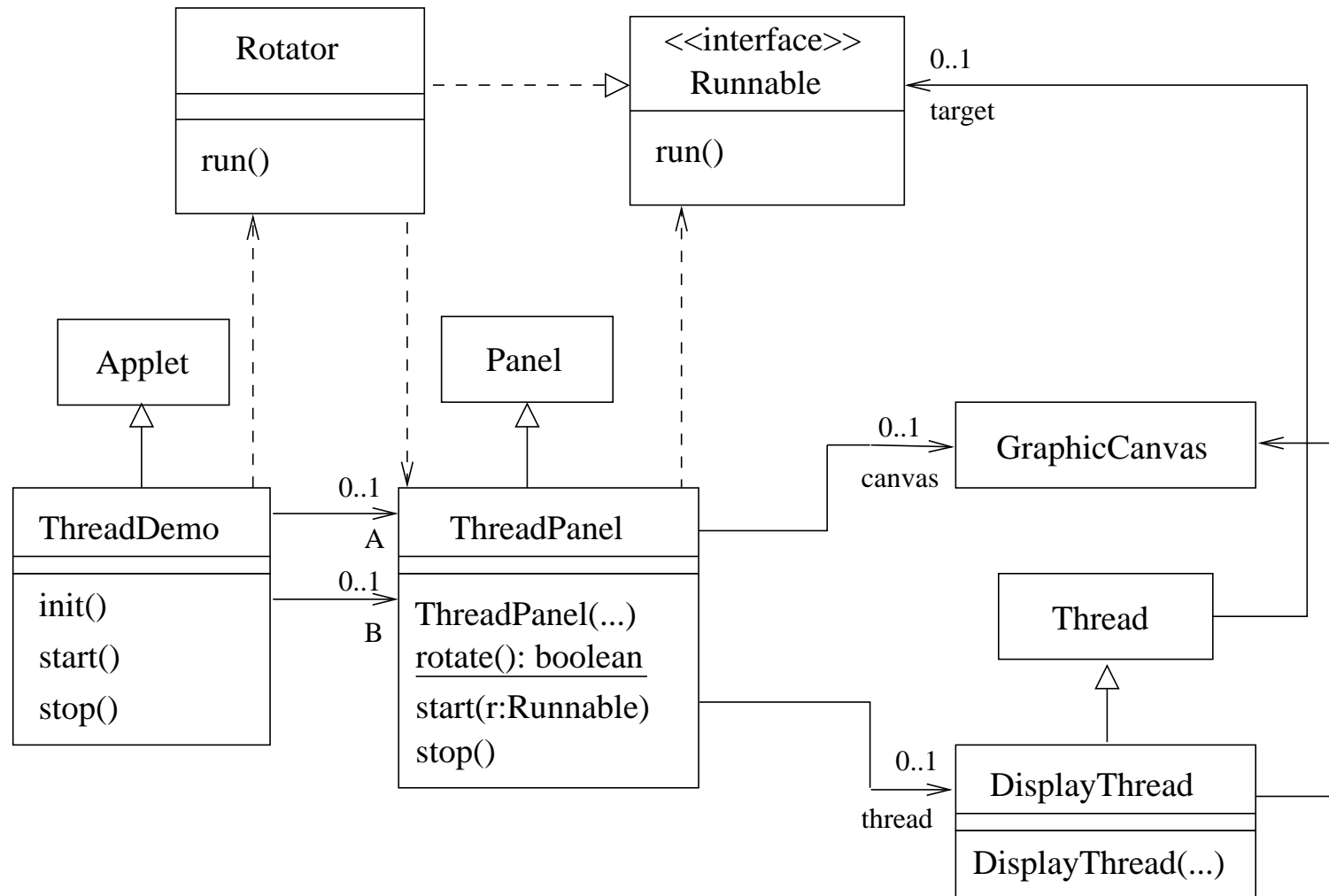
RUN = ({rotate, run} → RUN
 | pause → PAUSED
 | stop → STOP).

||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(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) {}  
    }  
}
```