parallel-prog-23-24 / blatt4-LTSA-visualisation /

picupup Add stuff from blatt 5 5e6e49b · last year History

Name	Name	Last commit date
..		
png	Add self solved final versi...	last year
4-1.lts	Add self solved final versi...	last year
4-1.txt	Add self solved final versi...	last year

parallel-prog-23-24 / blatt4-LTSA-visualisation / Top

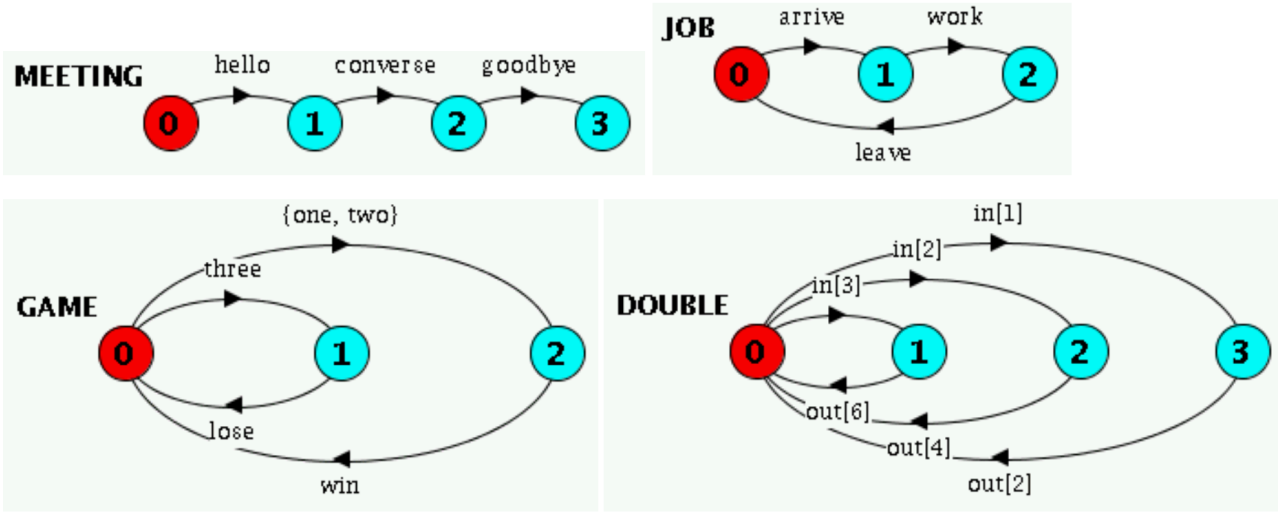
4-2-watter-sensor....	Add self solved final versi...	last year
4-3-Brause-Autom...	Add self solved final versi...	last year
4-3-Brause-Autom...	Add self solved final versi...	last year
4-4-b-Fertigungs-...	Add self solved final versi...	last year
4-5-Speicher-schi...	Add self solved final versi...	last year
4-6-a-serverClient...	Add self solved final versi...	last year
4-6-b-ClienServer...	Add self solved final versi...	last year
4-7-Museum.lts	Add stuff from blatt 5	last year
README.md	Add self solved final versi...	last year
README.pdf	Add self solved final versi...	last year

README.md

4.1

Geben Sie fur jeden der sieben im folgenden als Zustandsgraphen dargestellten Prozesse die zugehörige algebraische FSP-Beschreibung an. Uberprüfen Sie Ihre Lösung mit Hilfe des Analysetools LTSA und geben Sie einige Traces aus.

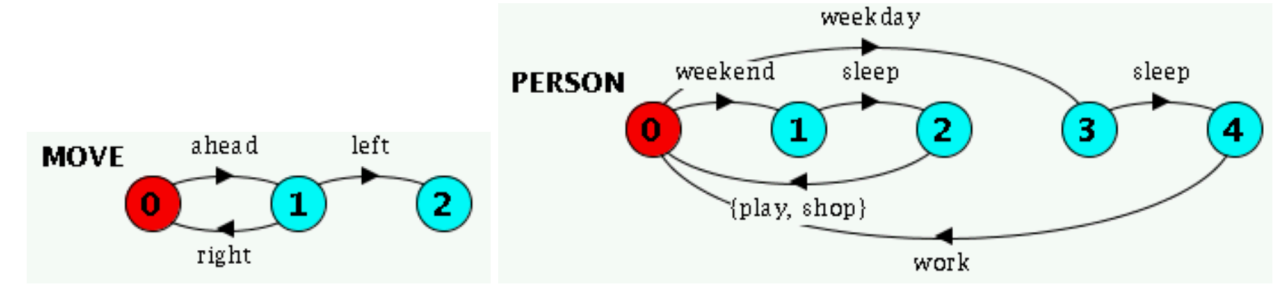
a

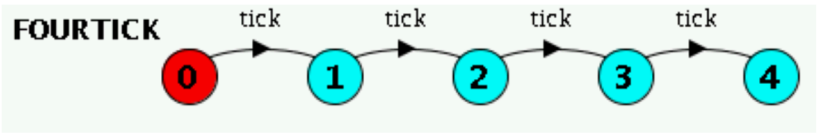


Antwort

```
MEETING = (hello -> converse -> goodby -> STOP).
JOB = (arrive -> work -> leave -> JOB).
GAME = (three -> lose -> GAME
| one -> WIN
| two -> WIN),
WIN = (win -> GAME).
DOUBLE = (in[1] -> out[2] -> DOUBLE
| in[2] -> out[4] -> DOUBLE
| in[3] -> out[6] -> DOUBLE).
```

b





Antwort

```
MOVE = (ahead -> MIDDLE),
MIDDLE = (right -> MOVE | left -> STOP).
PERSON = (weekend -> sleep -> DOSOMETHING
| weekday -> sleep -> work -> PERSON),
DOSOMETHING = (play -> PERSON | shop -> PERSON).
FOURTICK = (tick -> tick -> tick -> tick -> STOP).
```

4.2

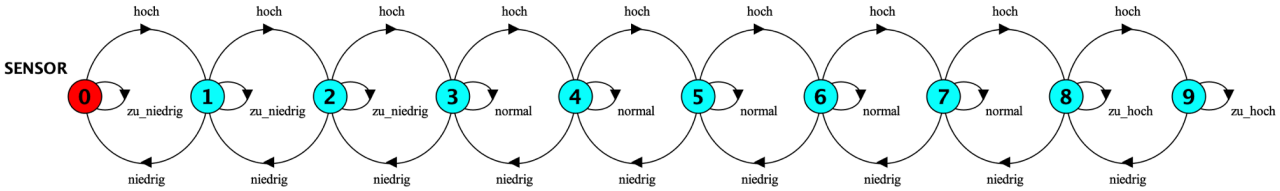
Ein Sensor misst den Wasserstand in einem Tank, wobei der Pegel die Werte 0,...,9 annehmen kann (5 ist der Anfangspegel). In Abhangigkeit des Wasserstandes gibt der Sensor die Meldungen "zu niedrig" (bei Pegel kleiner als 3), "zu hoch" (bei Pegel größer als 7) oder "normal" (sonst) aus. Modellieren Sie den Sensor als FSP SENSOR.

Hinweis: Das Alphabet von SENSOR ist {pegel[0..9], niedrig, hoch, normal}.

4.2 Antwort

```
const N = 9 // Max
const START_LEVEL = 5 // Start point

SENSOR = PEGEL[0],
PEGEL[i:0..N] = (when (i < N) hoch -> PEGEL[i+1]
| when (i > 0) niedrig -> PEGEL[i-1]
| when (i < 3) zu_niedrig -> PEGEL[i]
| when (i > 7) zu_hoch -> PEGEL[i]
| when (i < 8 && i > 2) normal -> PEGEL[i]).
```

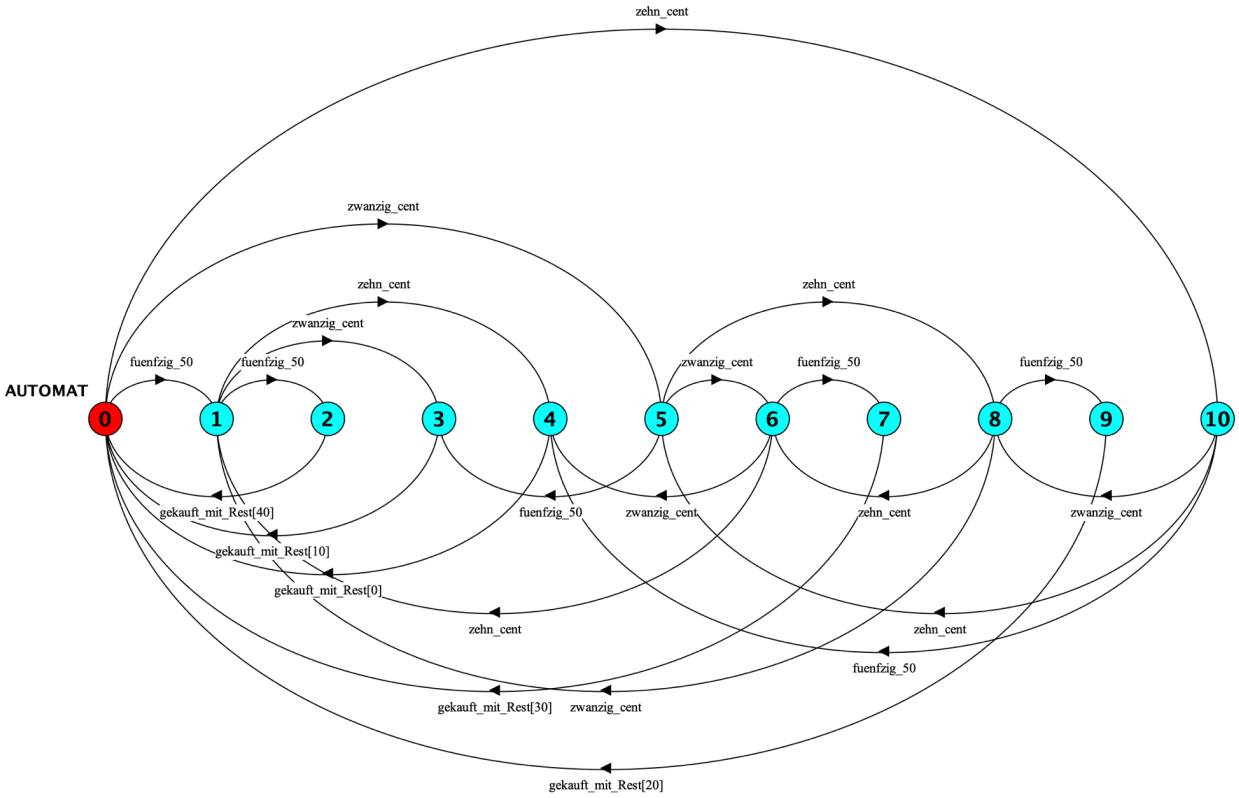


4.3

In einem Getrankeautomaten kostet ein Becher süße Brause 60 Cent. Der Automat nimmt 10-, 20- und 50-Cent-Munzen an und gibt gegebenenfalls Wechselgeld zurück. Beschreiben Sie den Getränke- automaten algebraisch durch FSP und geben Sie den entsprechenden Zustandsgraphen an.

4.3 Antwort

```
const BRAUSE_COST = 60 /
const RangMax = 2*BRAUSE_COST
AUTOMAT = INPUT[0],
INPUT[i:0..RangMax] = (when (i < BRAUSE_COST) zehn_cent ->
INPUT[i + 10]
| when (i < BRAUSE_COST) zwanzig_cent -> INPUT[i + 20]
| when (i < BRAUSE_COST) fuenfzig_cent -> INPUT[i + 50]
| when (i >= BRAUSE_COST) gekauft_mit_Rest[i - 60] -> INPUT[0]).
```



4.4

a

Zeigen Sie, dass die im Folgenden definierten Prozesse `S1` und `S2` das gleiche Verhalten zeigen:

```
P = (a -> b -> P).
Q = (c -> b -> Q).
||S1 = (P || Q).

S2 = ( a -> c -> b -> S2
     | c -> a -> b -> S2).
```

Antwort:

Die Prozesse `S1` und `S2` teilen beide die aktion b. Beide prozesse teile aber nicht die Aktion a oder c. Das heißt in der paralleisierungs Prozess können a und c nach ein ander durchgeführt werden, before die die Aktion b abgearbeitet wird.

b

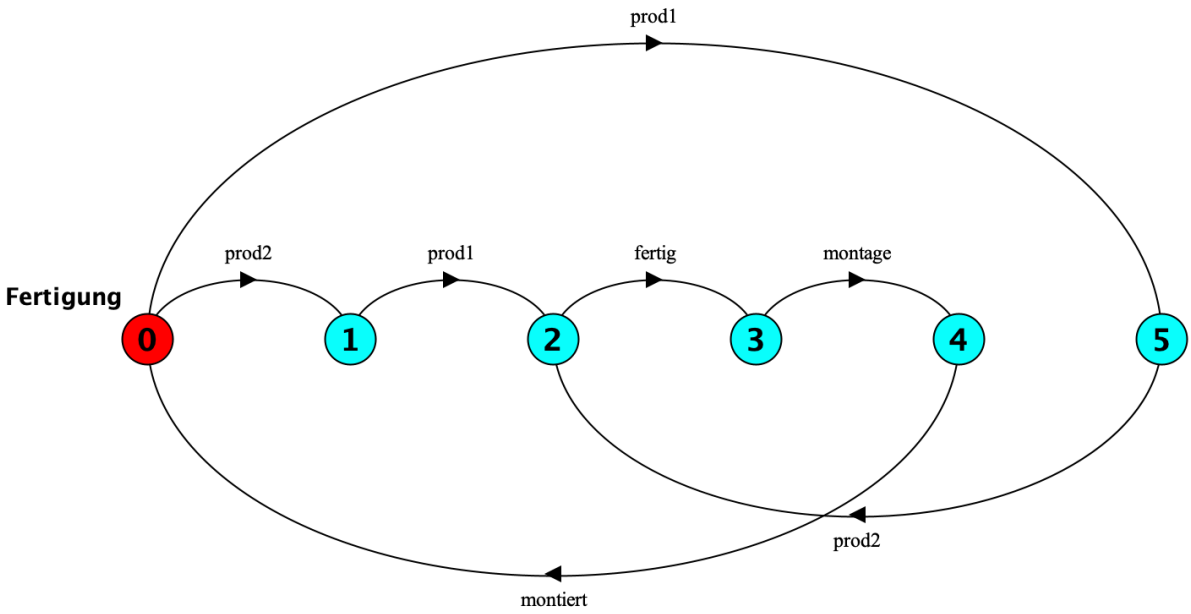
Finden Sie einen Prozess, der ohne die Verwendung der parallelen Komposition `||` definiert ist, aber das gleiche Verhalten wie der folgende Prozess `Fertigung` zeigt.

```
Prod1 = (prod1 -> fertig -> montiert -> Prod1).
Prod2 = (prod2 -> fertig -> montiert -> Prod2).
||Produktion = (Prod1 || Prod2).

Montage = (fertig -> montage -> montiert -> Montage).
||Fertigung = (Produktion || Montage).
```

4.4.b Antwort

```
Fertigung = (prod1 -> prod2 -> Middle
                                | prod2 -> prod1 -> Middle),
                                Middle = (fertig -> montage ->
montiert -> Fertigung).
```



4.5

Der Prozess `Speicher = (rein -> raus -> Speicher)` . modelliert das Verhalten einer Speicherzelle, indem er zunachst eine rein- und dann eine raus-Aktionen ausfuhrt.

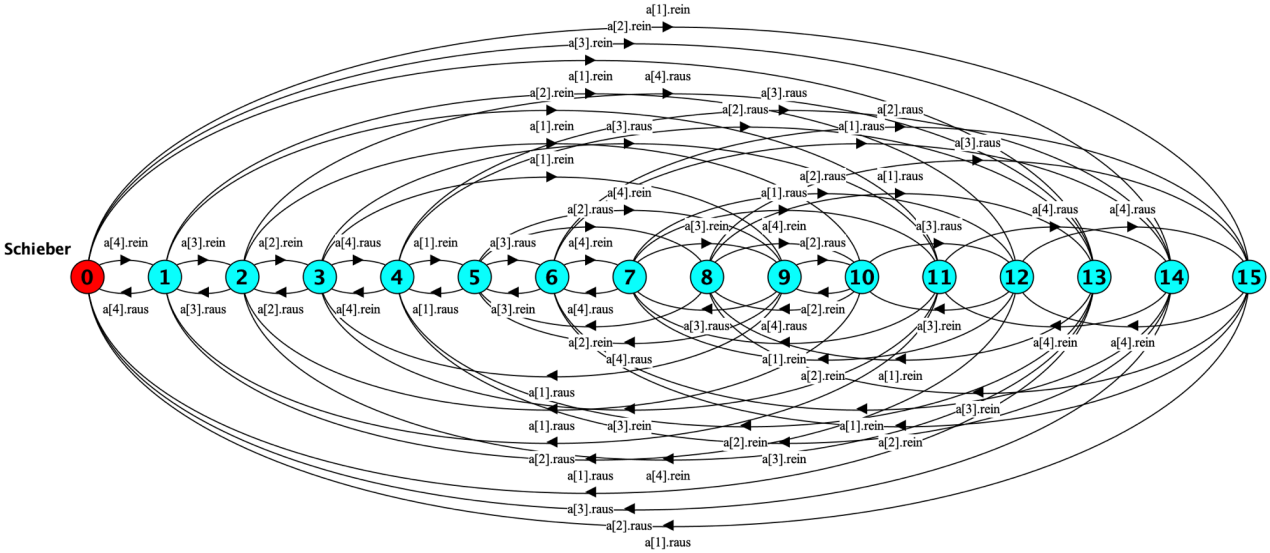
Definieren Sie unter Verwendung von `Speicher` und mit Hilfe der parallelen Komposition einen Prozess `Schieber` , der das Verhalten eines Schieberegisters mit N = 4 internen Speicherzellen modelliert – d. h. das Schieberegister besitzt nach außen nur die Aktionen `rein` bzw. `raus` ; im Innern des Schieberegisters werden die Daten von einer Zelle zur nachsten weitergeschoben.

Zeichnen Sie zuerst ein Strukturdiagramm und testen Sie dann Ihre Variante für N ≤ 4 im Analysetool LTSA.

Hinweis: In FSP kann der Befehl `forall` sowohl in der parallelen Komposition als auch beim Relabelling verwendet werde.

4.5 Antwort

```
Speicher = (rein -> raus -> Speicher).
||Schieber(N=4) = (a[i:1..N]:Speicher).
```



4.6

a

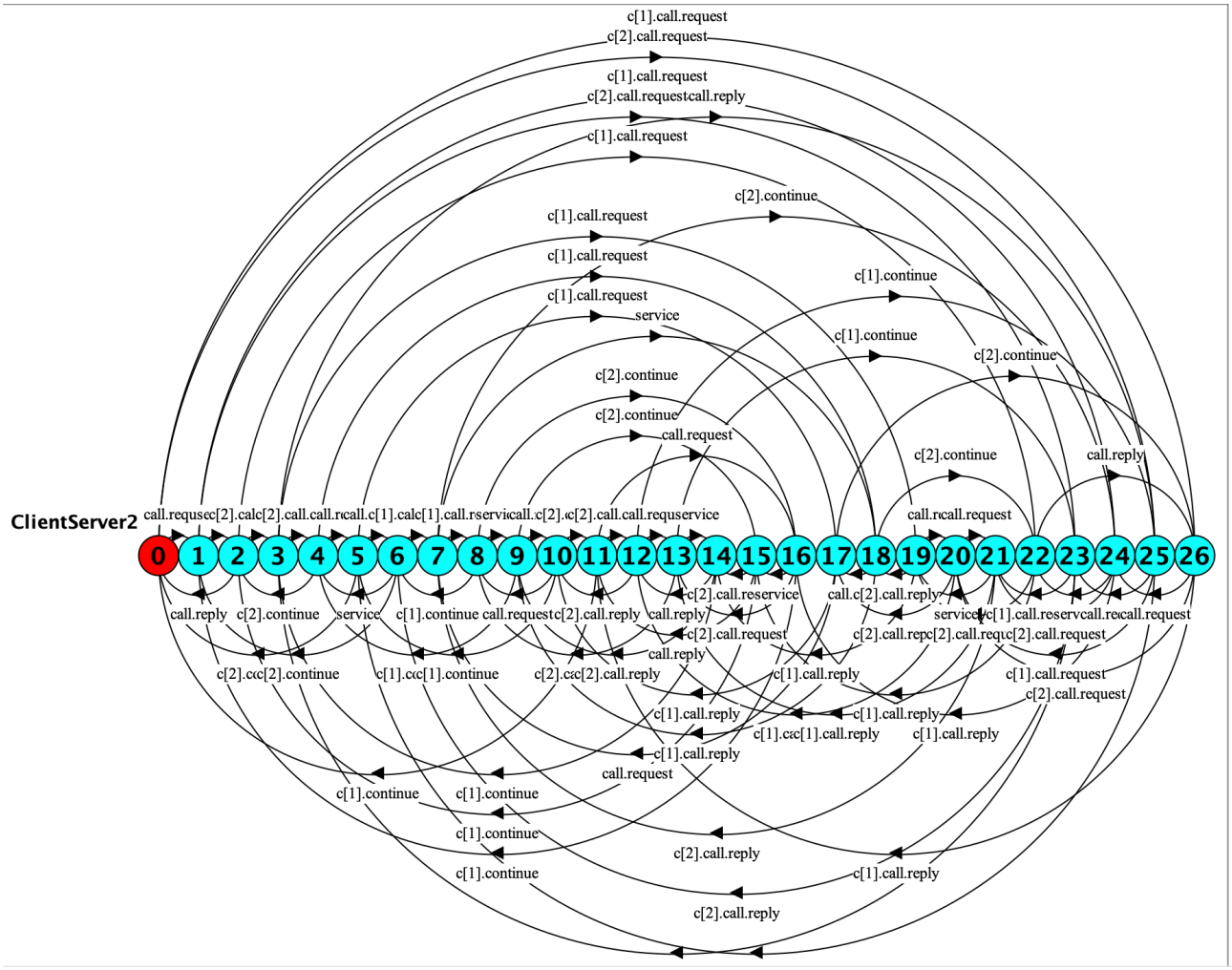
Modifizieren Sie den Client-/Server-Prozess aus der Vorlesung so, dass mehr als ein Client den Dienst des Servers abrufen kann.

4-6-a Antwort (Nicht gelöst)

Bei diese Aufgabe war ich nicht sicher wie man es machen solle.

```
Client2 = (call.request -> call.reply -> continue -> Client2).
Server2 = (accept.request -> service -> accept.reply ->
Server2).

||ClientServer2(N=2) = (c[i:1..N]:Client2 || Server2)
/ {call/accept}.
```



b

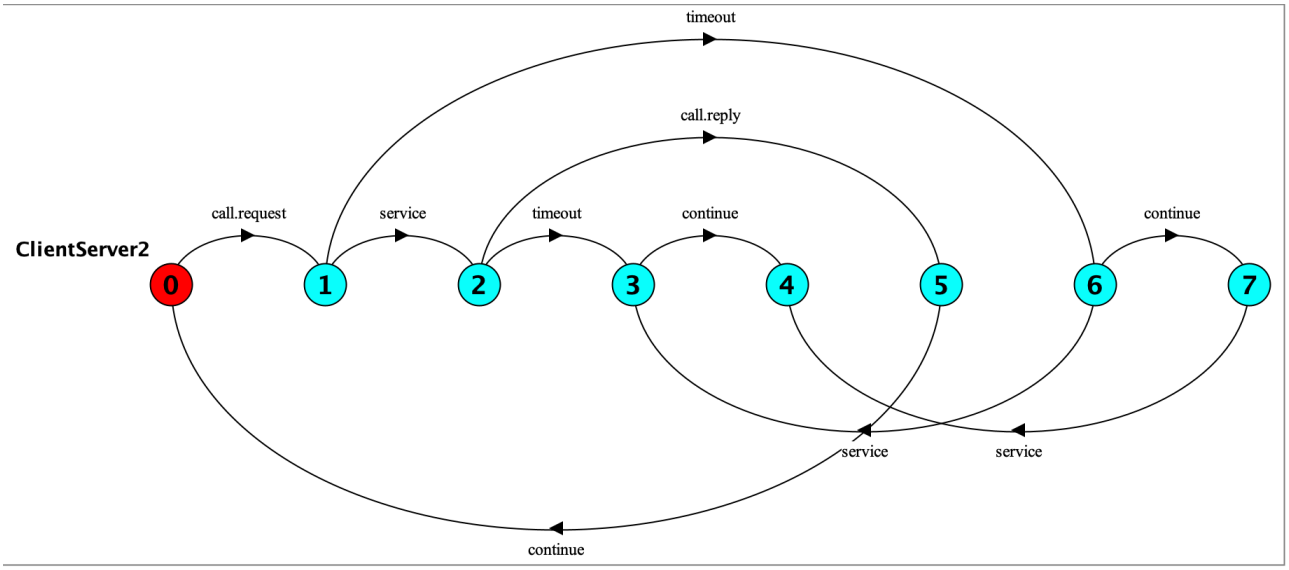
Ändern Sie Ihren Prozess aus dem ersten Aufgabenteil so, dass die `call` -Aktion der Clients statt mit einer Antwort des Servers auch mit einem Timeout beendet werden kann.

4-6-b Antwort

Gelöst ohne die anwendung von dem N fachen Clien Kommunikation.

```
Client2 = (call.request -> (call.reply -> CONTINUE | timeout ->
CONTINUE)),
CONTINUE = (continue -> Client2 ).
Server2 = (accept.request -> service -> accept.reply ->
Server2).

||ClientServer2 = (Client2 || Server2)
/ {call/accept}.
```



4.7

Ein Museum besitzt einen Eingang und einen Ausgang, die jeweils durch ein Drehkreuz gesichert und mit einer zentralen Steuerung verbunden sind. Zur Offnungszeit gibt der Museumswächter an der zentralen Steuerung den Eingang ins Museum frei, woraufhin die beiden Drehkreuze entsperrt werden.

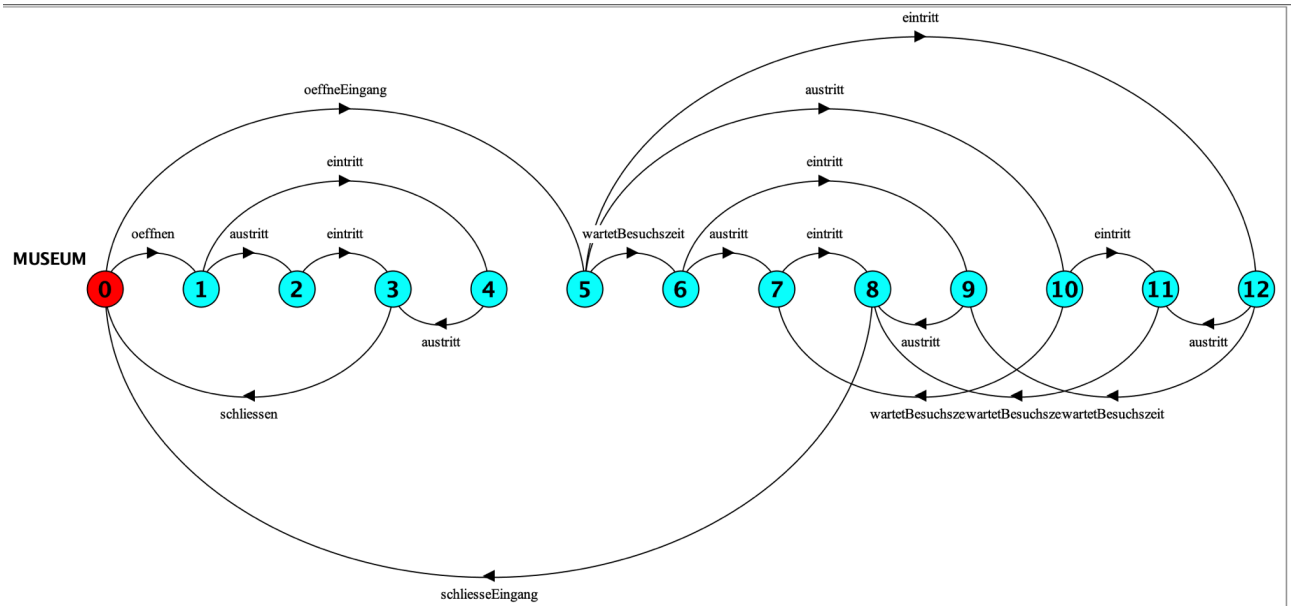
Nach Ablauf der Besuchszeit sperrt er an der zentralen Steuerung den Eintritt weiterer Besucher. Nach der Schließung kann nur noch das Drehkreuz am Ausgang passiert werden (solange sich noch Besucher in den Raumen befinden). Da es sich um ein sehr kleines Museum handelt, erlaubt die Steuerung auch nur N Besuchern gleichzeitig, sich in den Raumen aufzuhalten.

Modellieren Sie das Museum durch die parallele Komposition der Prozesse `Eingang`, `Ausgang`, `Steuerung` und `Waerter`.

4.7 Antwort:

```
WAERTER = (oeffneEingang -> wartetBesuchszeit ->
schliesseEingang -> WAERTER).
STEUERUNG = (oeffnen -> schliessen -> STEUERUNG).
EINGANG = (entsperren -> eintritt -> sperren -> EINGANG).
AUSGANG = (entsperren -> austritt -> sperren -> AUSGANG).

||MUSEUM = (EINGANG || AUSGANG || STEUERUNG || WAERTER)
/{oeffneEingang/entsperren, oeffnen/entsperren,
schliesseEingang/sperren, schliessen/sperren }.
```


MUSEUM
0
1
2
3
4
5
6
7
8
9
10
11
12
oeffneEingang
oeffnen
austritt
eintritt
eintritt
austritt
schliessen
schliesseEingang
eintritt
austritt
wartetBesuchszeit
austritt
eintritt
eintritt
austritt
eintritt
austritt
wartetBesuchszewartetBesuchszewartetBesuchszeit