Modul OMSI-2
im SoSe 2010

Objektorientierte Simulation mit ODEMx

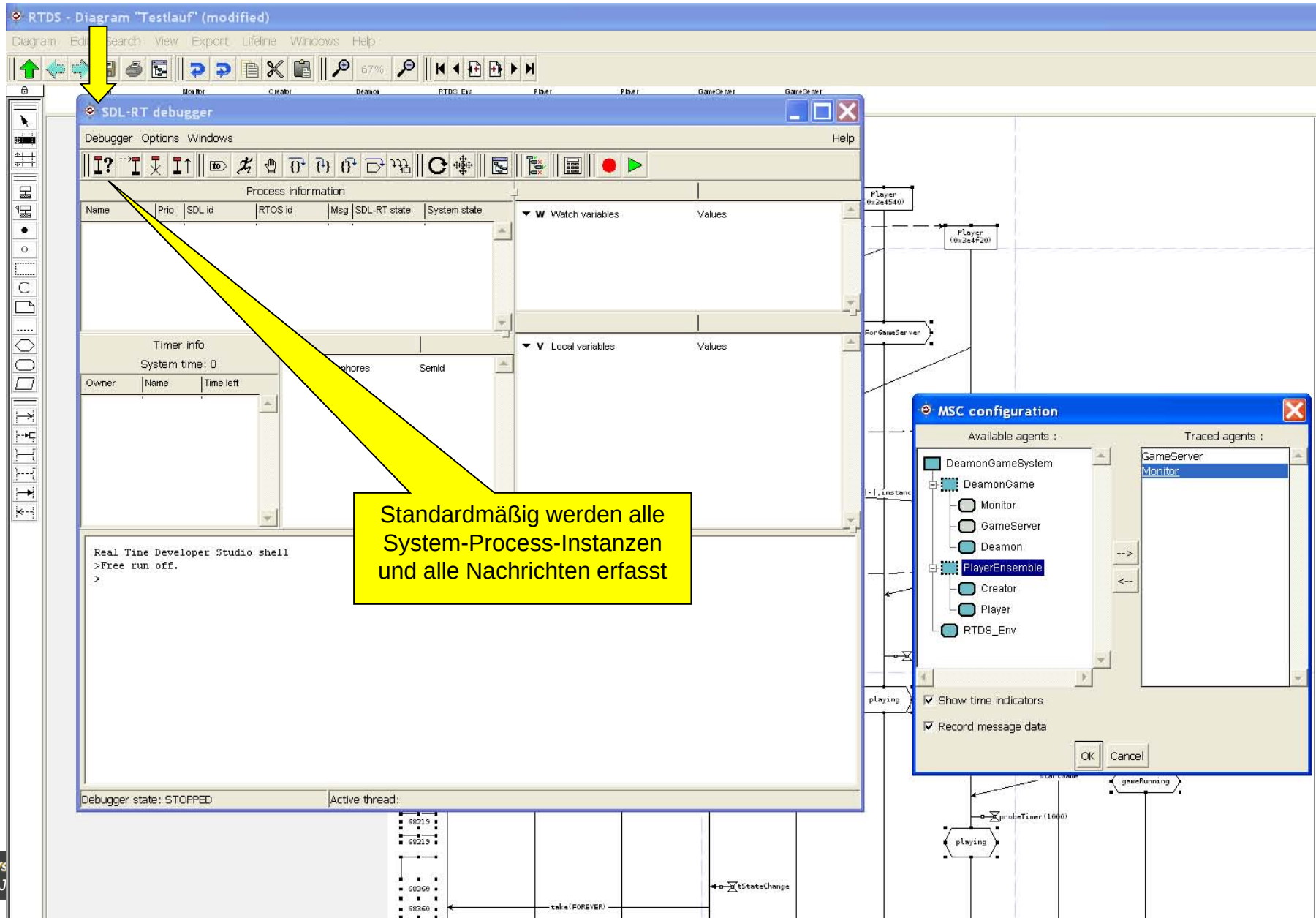
Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Andreas Blunk

fischer|ahrens|eveslage|blunk@informatik.hu-berlin.de

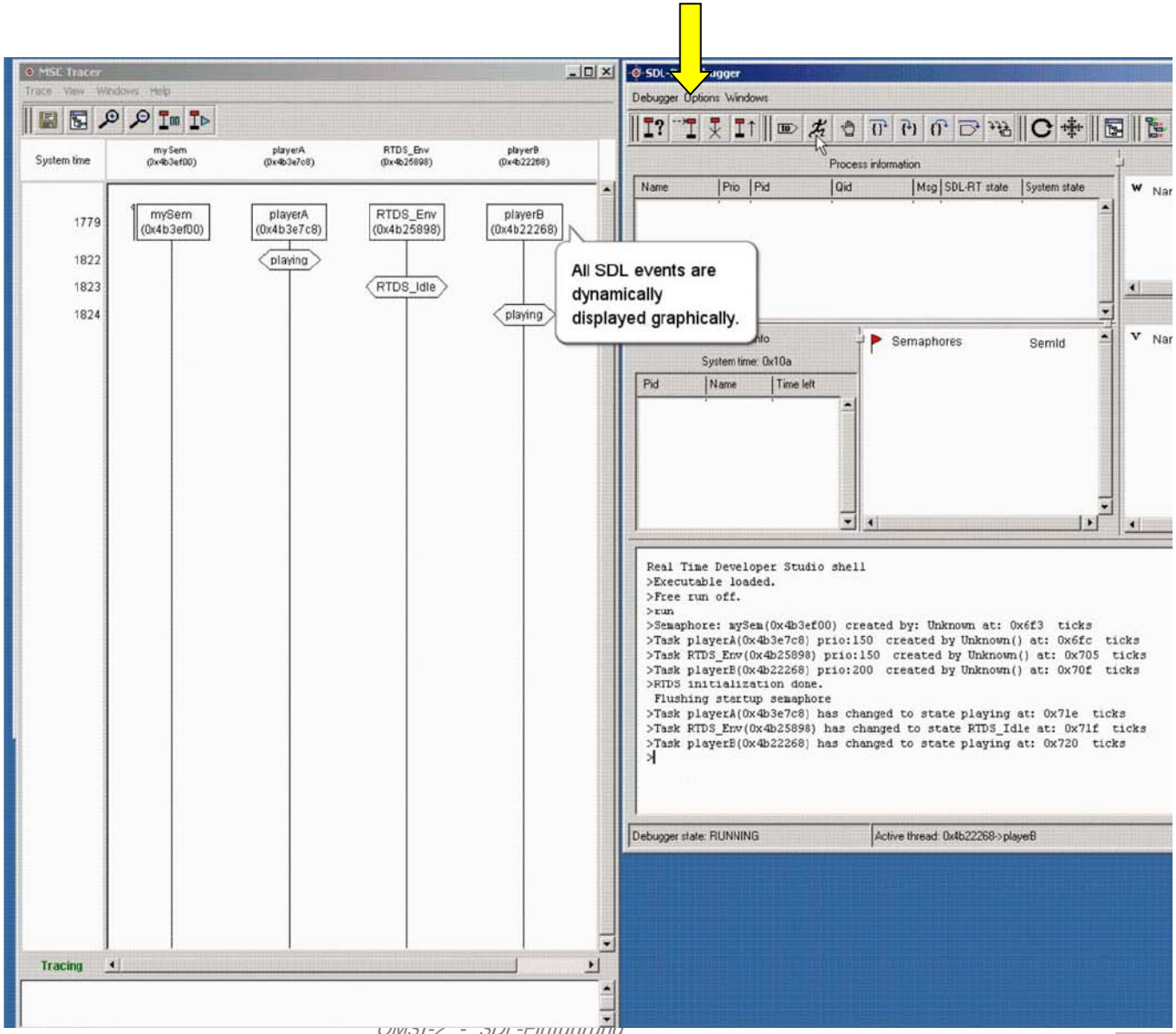
6. SDL-Einführung

1. Grundphilosophie
2. ITU-Standard Z.100
3. Werkzeuge
4. SDL-Grundkonzepte
5. Deamon-Game: Musterbeispiel (in UML-Strukturen)
6. DeamonGame:
   Struktur- und Verhaltensbeschreibung in SDL-RT
7. DeamonGame: Simulation
8. PragmaDev-Tutorial

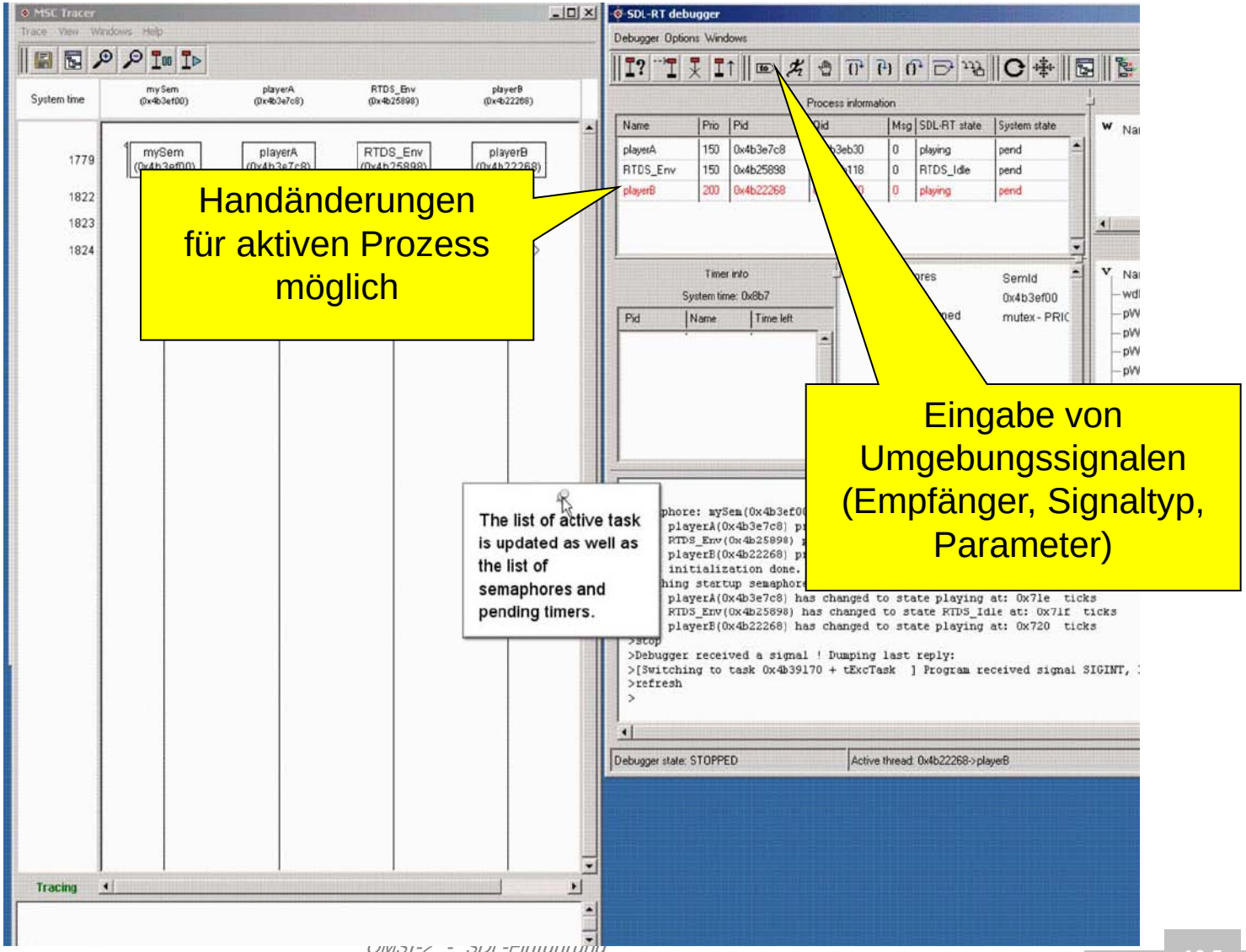
Konfiguration der SD-Ausgabe (MSC)



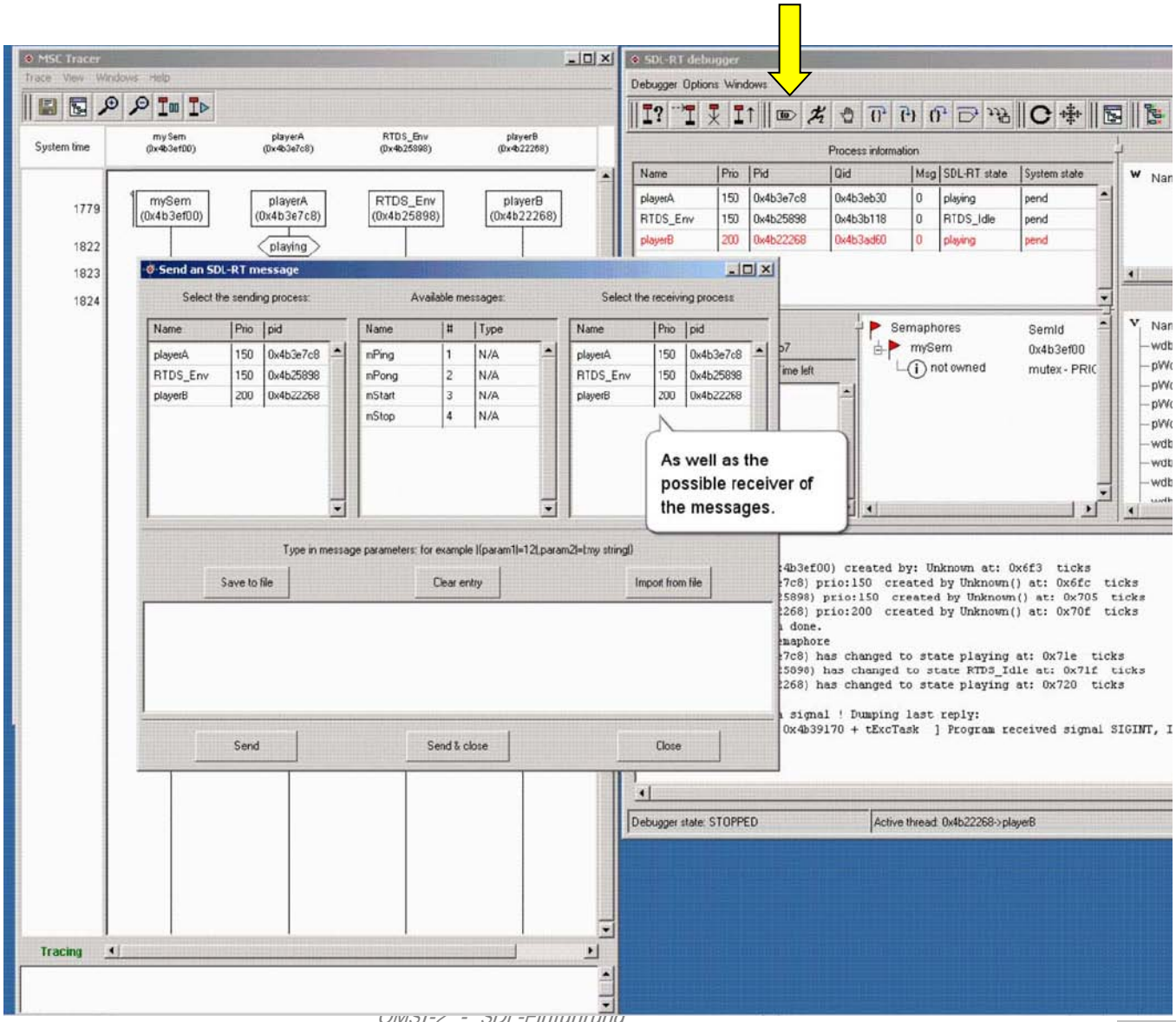
Einrichtung eines MSC-Trace-Diagramms



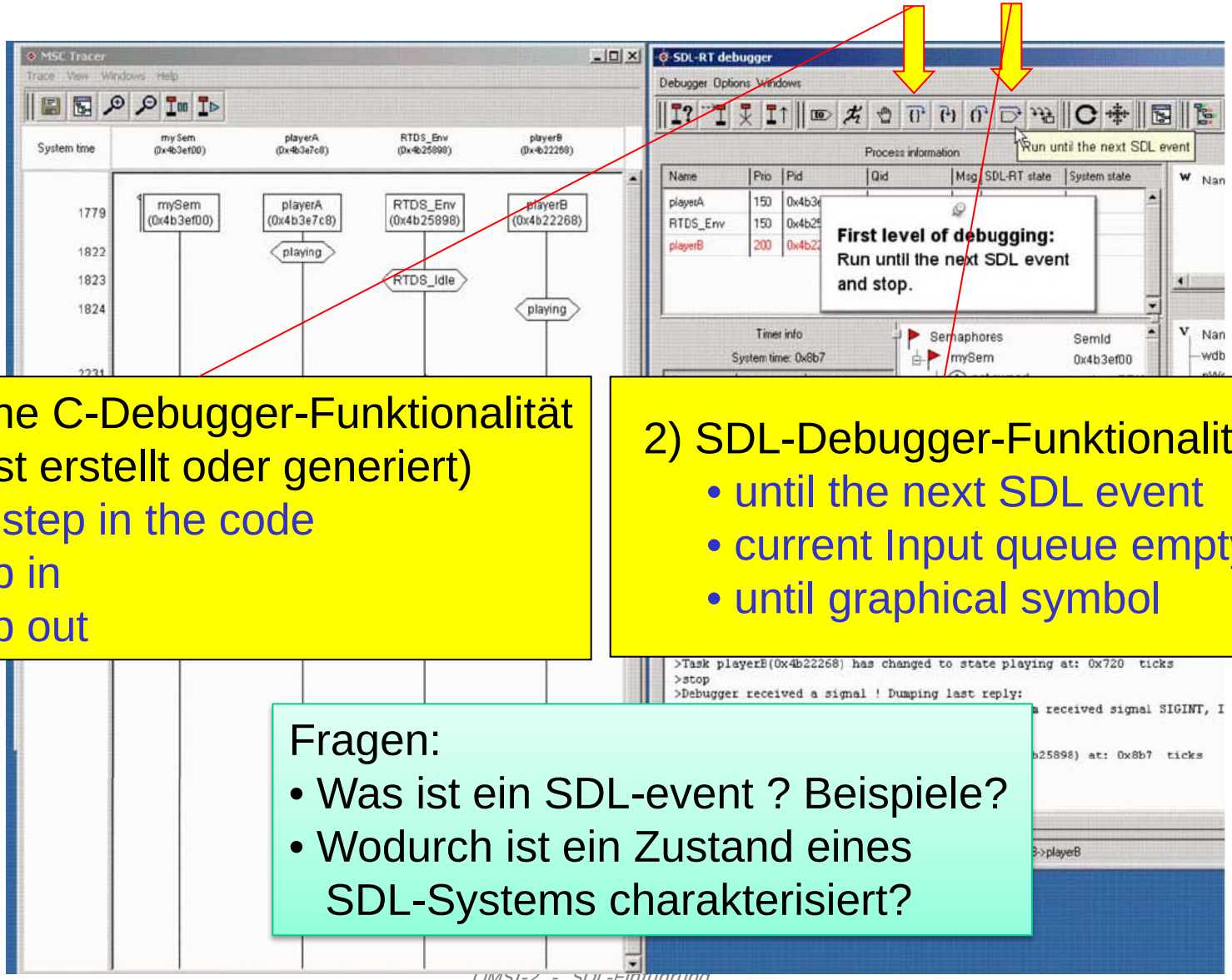
Eingabe von Umgebungsnachrichten



Start des Programms



Debugger-Steuerung



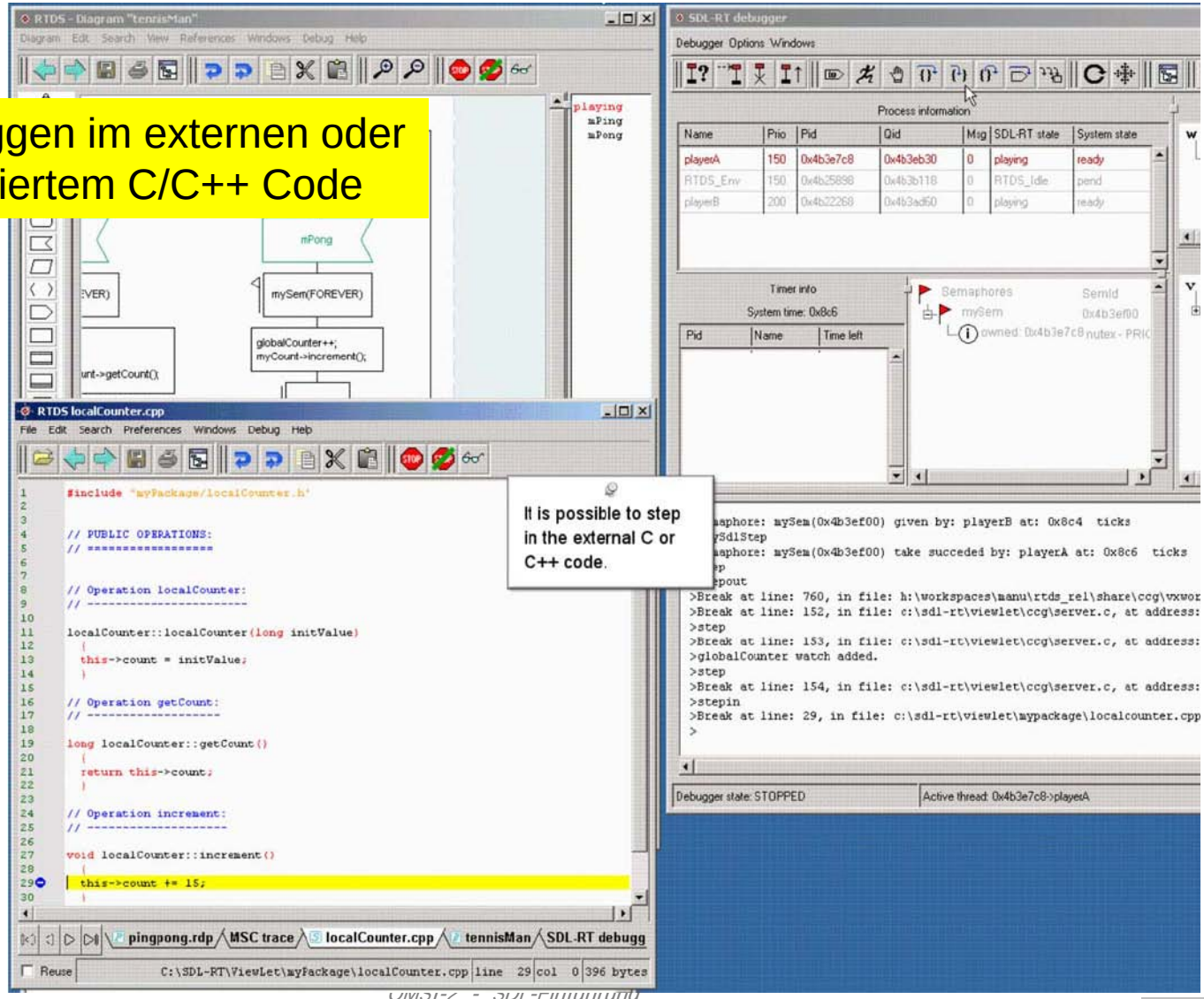
1) übliche C-Debugger-Funktionalität (selbst erstellt oder generiert)
- flat step in the code
- step in
- step out

2) SDL-Debugger-Funktionalität
- until the next SDL event
- current Input queue empty
- until graphical symbol

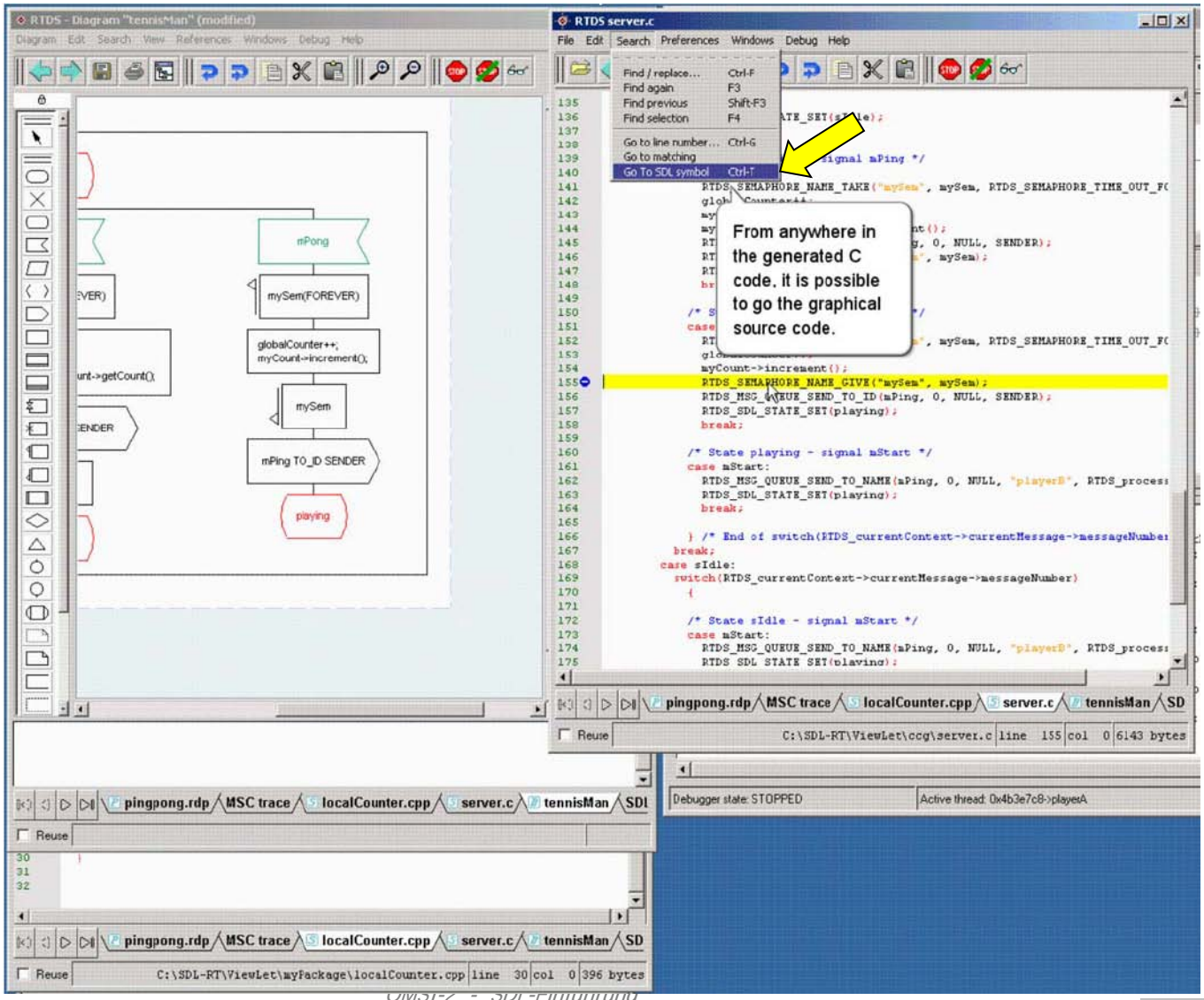
Fragen:
- Was ist ein SDL-event ? Beispiele?
- Wodurch ist ein Zustand eines SDL-Systems charakterisiert?

Wechsel:
SDL/GR-Sicht → C/C++ -Textsicht

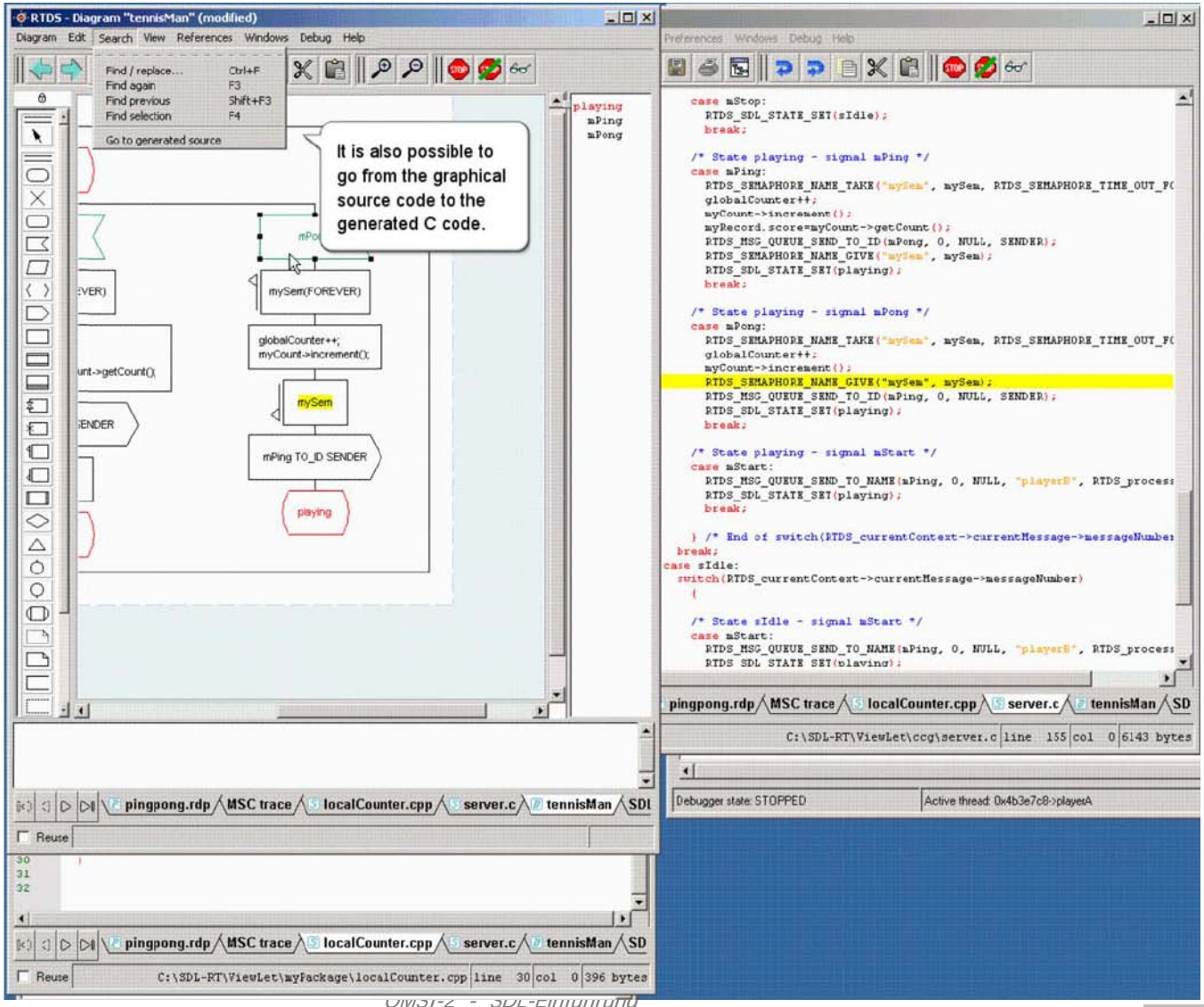
Debuggen im externen oder Generiertem C/C++ Code



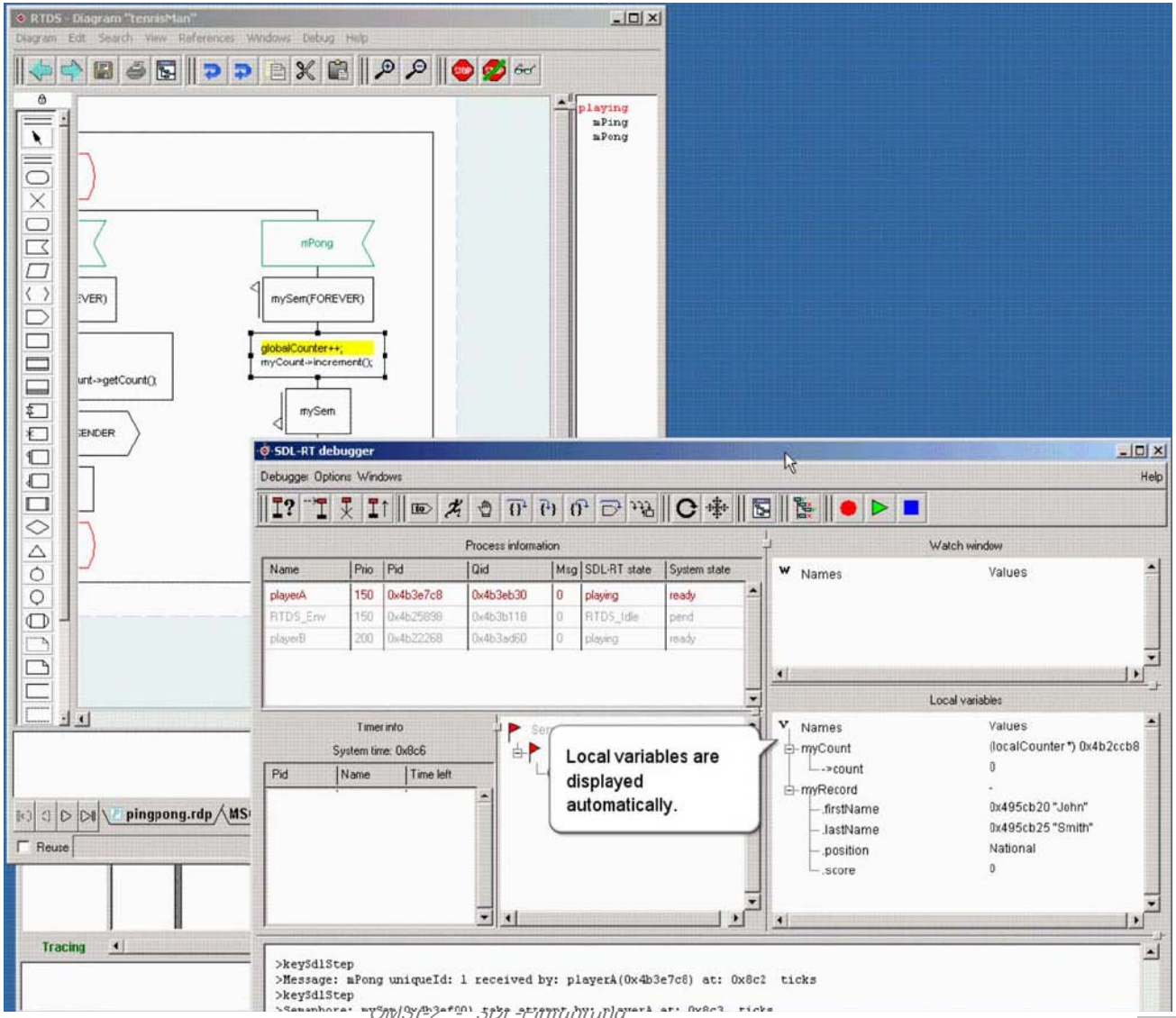
Rückkehr: Text → Grafik



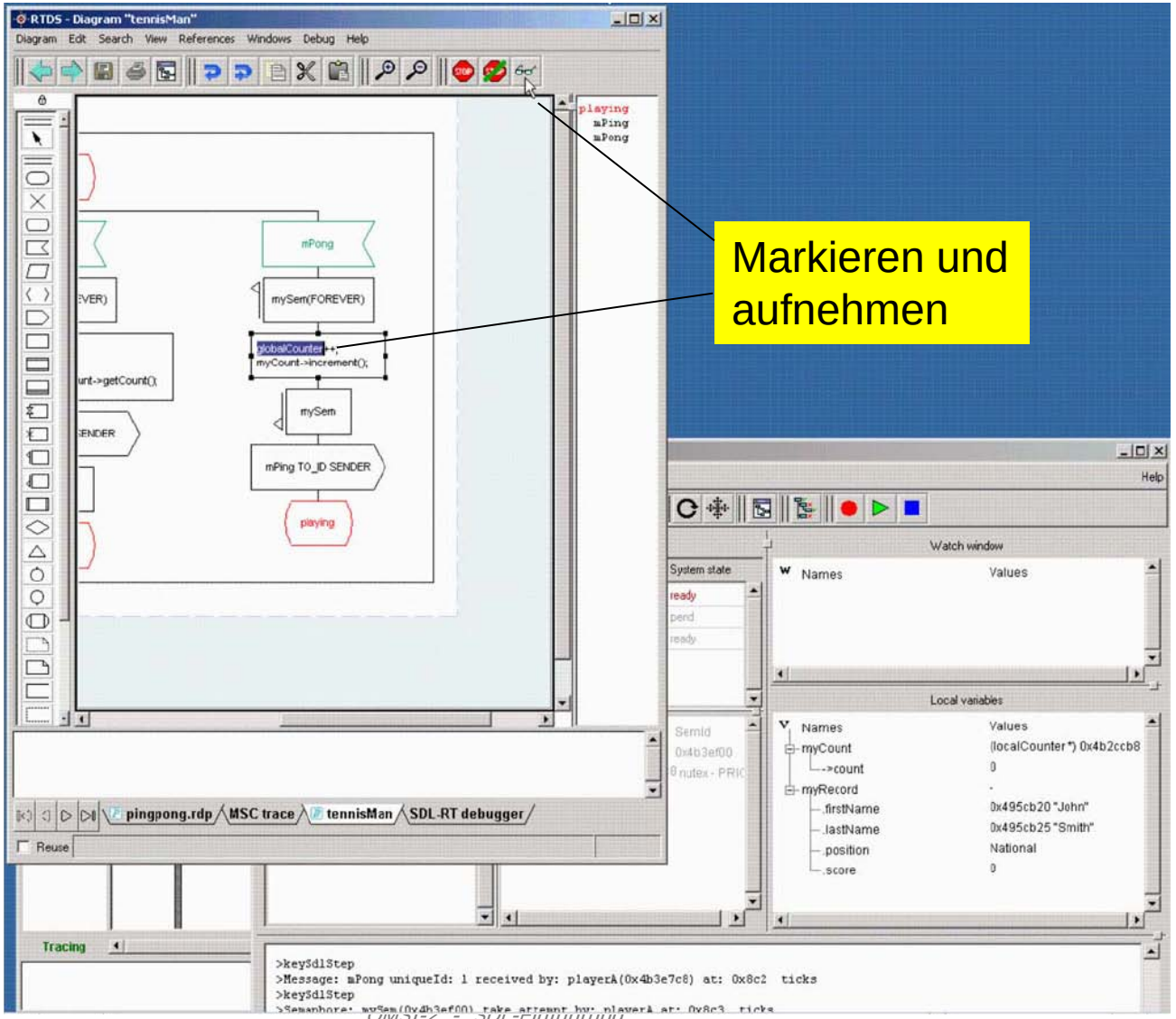
Grafik → Generierter Code



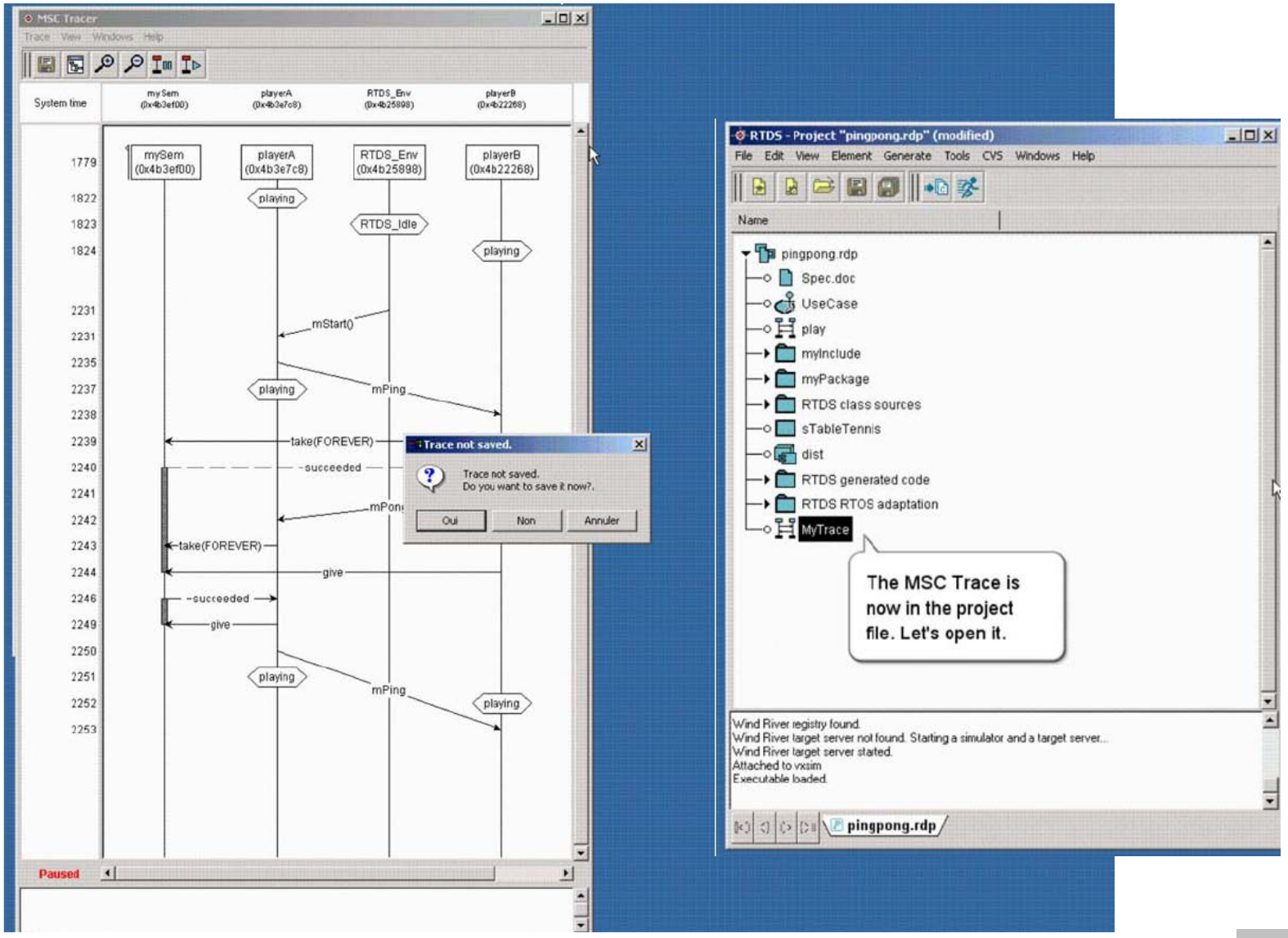
Anzeige von Variablenänderungen



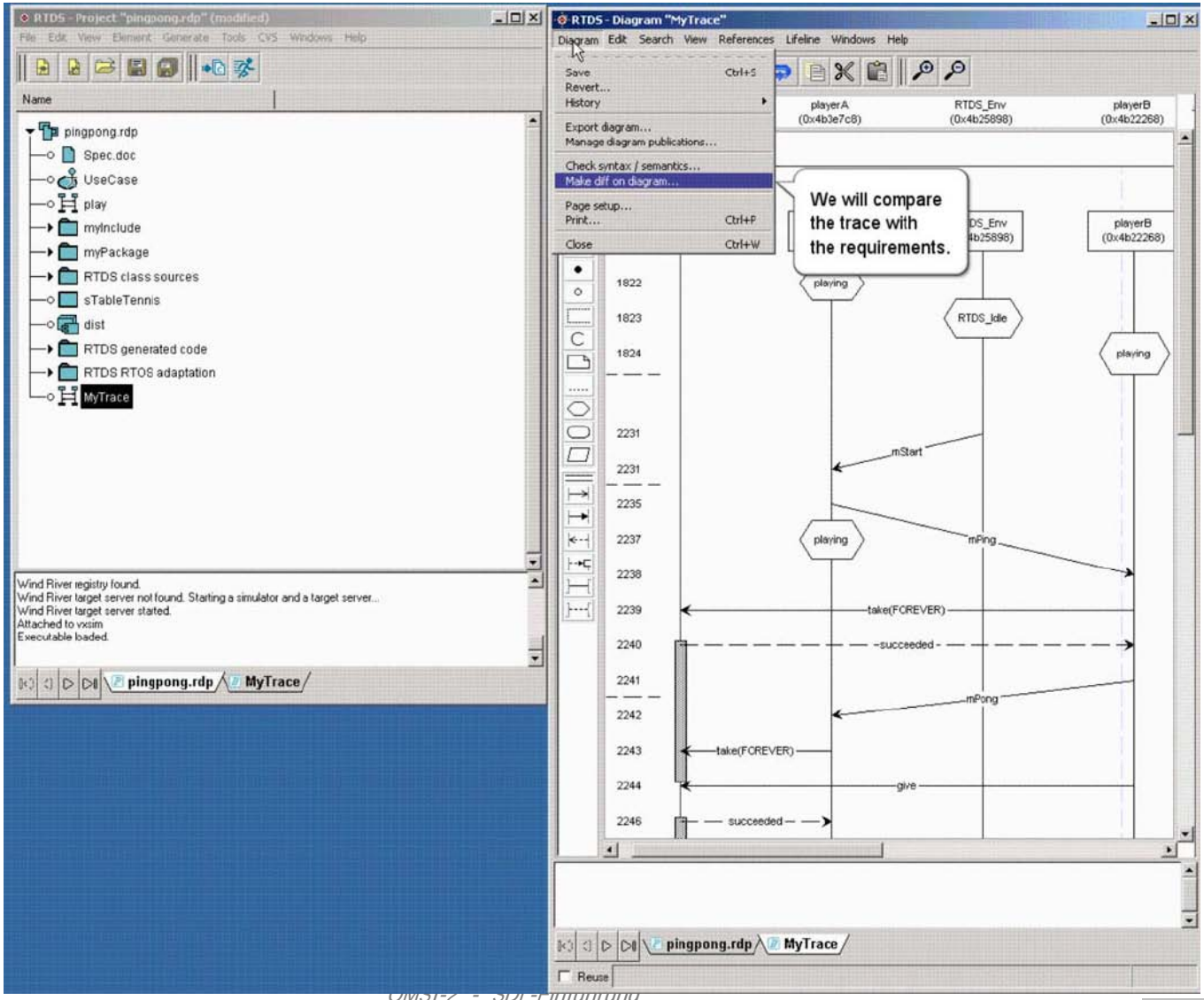
Setzen von Beobachtungspunkten



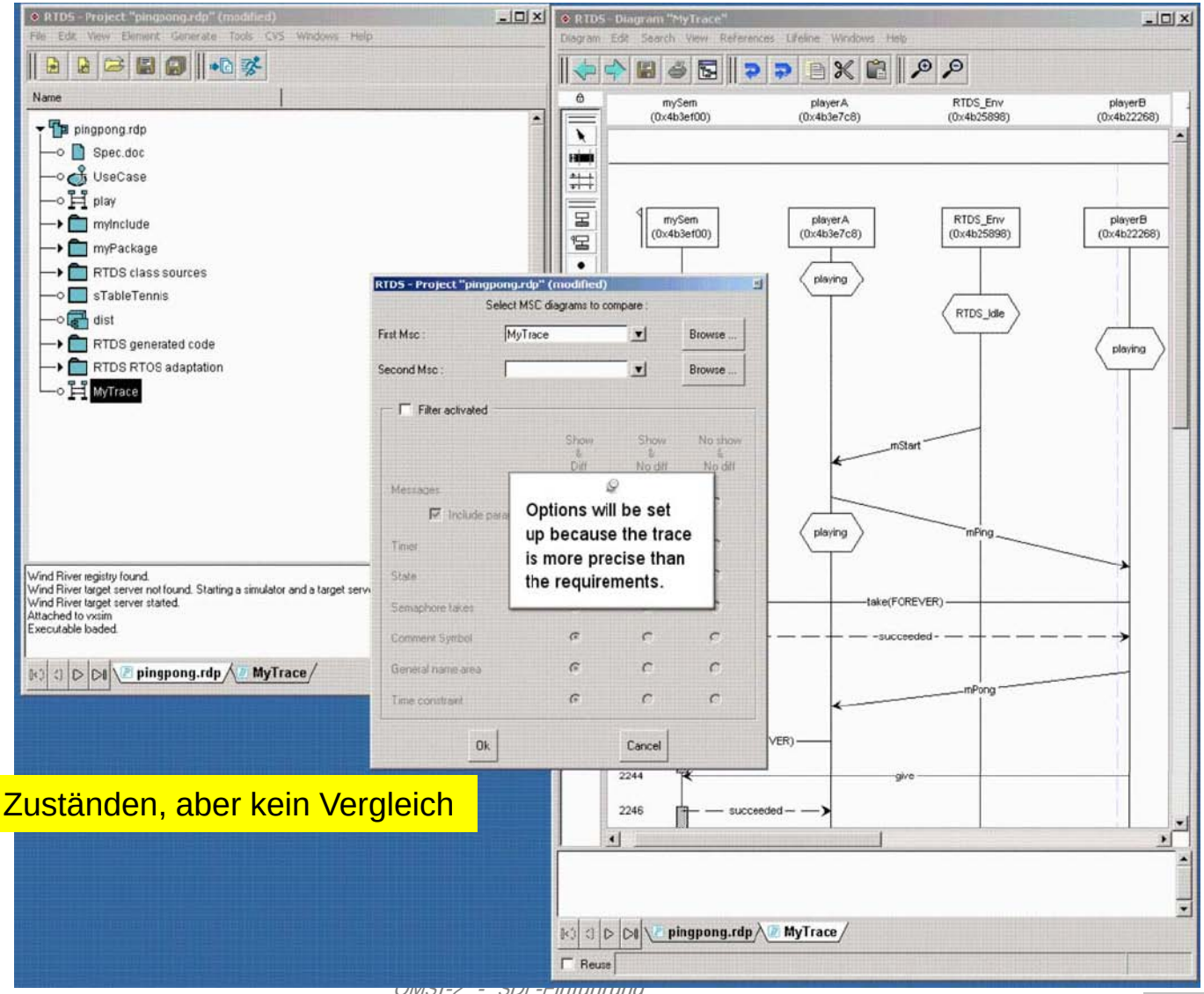
Speichern eines MSC-Diagramms



Vergleich: Anforderung - Simulationsresultat

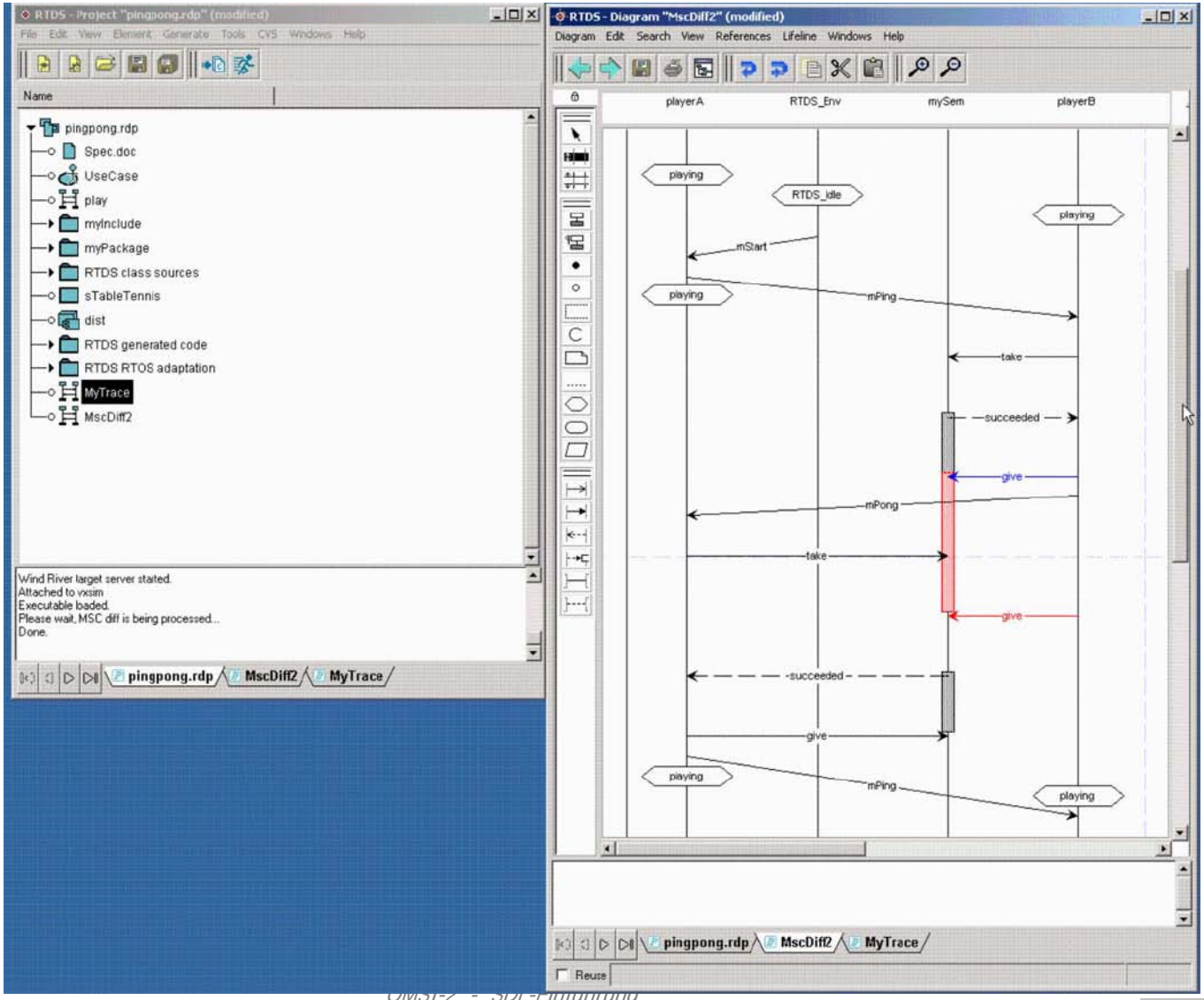


Benutzung von MSC-Filtern

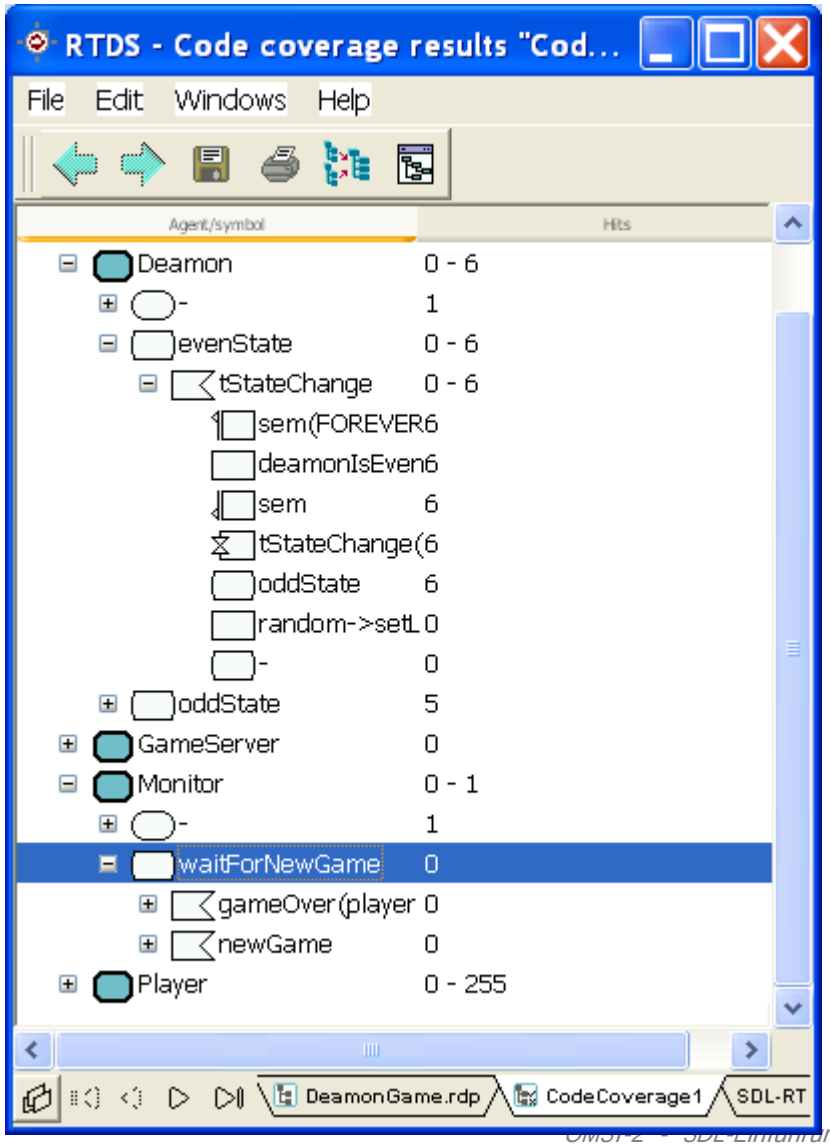


- Anzeige von Zuständen, aber kein Vergleich

Erzeugung einer Diff-Datei

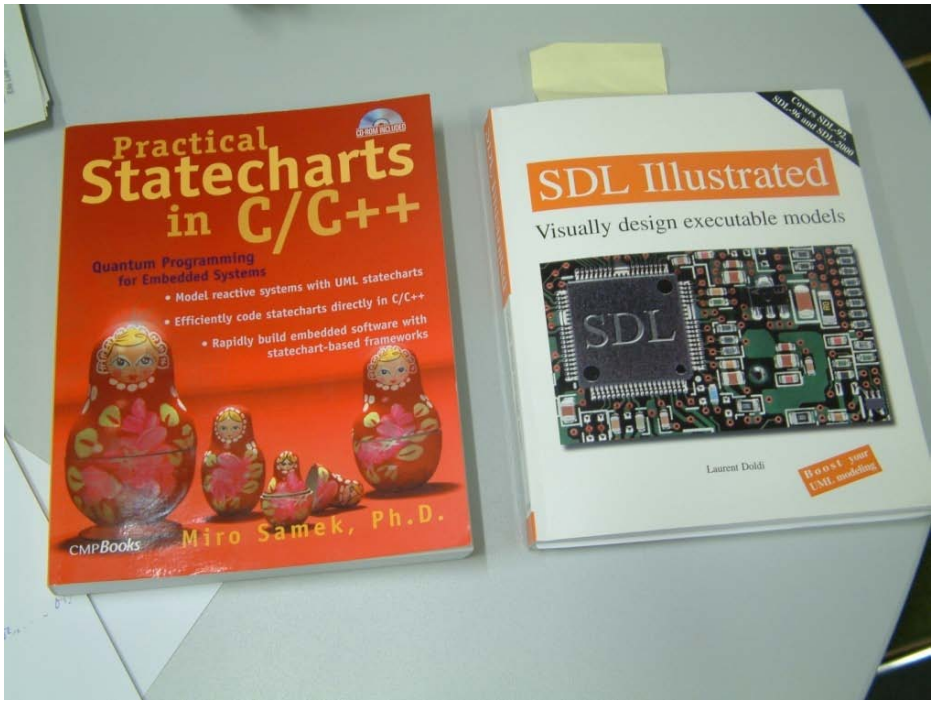


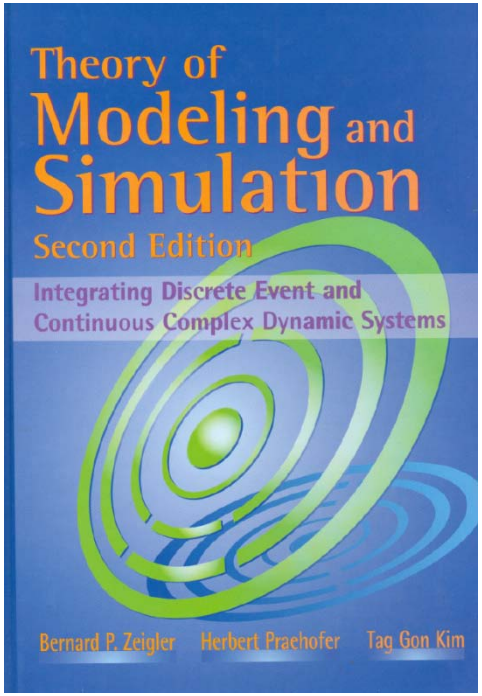
Überdeckungsgraph



verlangt unter *Options* des Projekt-Managers gewisse Einstellungen, da Erstellung ressourcenaufwändig ist

OMSI- Literaturhinweise





7. SDL-Konzepte *(Präzisierung, 1.Teil)*

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Nachrichtenadressierung
4. Timer

Vorschlag einer Namenskonvention

… für erstes Zeichen in Namen in Abhängigkeit des Modellelementetyps:

- b Block-Namen
- p Process-Namen
- t Timer-Namen
- s Semaphore-Namen
- g Gate-Namen

Struktur / Architetur

- System - Environment
  (Blocksymbol wird als System benutzt)

- Agent ist Element eines Systems
  - zwei Ausprägungen:
    - Block (ohne besondere Ausprägung im Zielcode),
    - Process (OS-Prozesse/Thread)
  - Mix von Prozessen und Blöcken auf einer Ebene ist erlaubt

- Prozess-Kardinalität: Default (1,)

Klassenbeschreibung

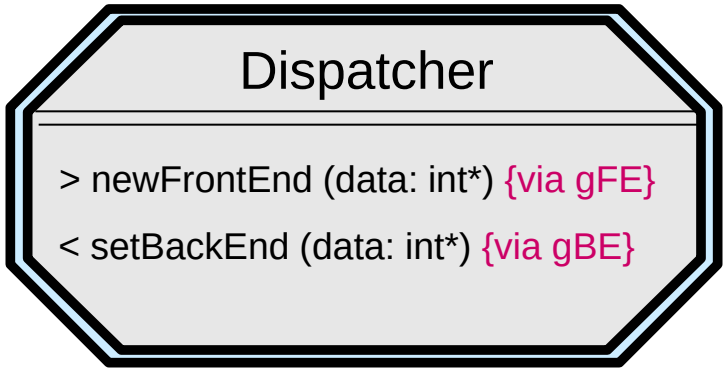
- 4 Formen
  - Interface-Klasse <<interface>>
  - System-Klasse <<system>>
  - Passive Klasse
  - Aktive Klasse : für Block-Typen, Process-Typen

In UML-Syntax

UserFactory
- lastUserId: int=0
+ <<create>> () - computeNext(): int + getUser(in userName: char*): User*

mehrere Konstruktoren (kein Return) möglich
nur ein Destruktor (<<delete>>)
In-, Out-, inOut-Parameter

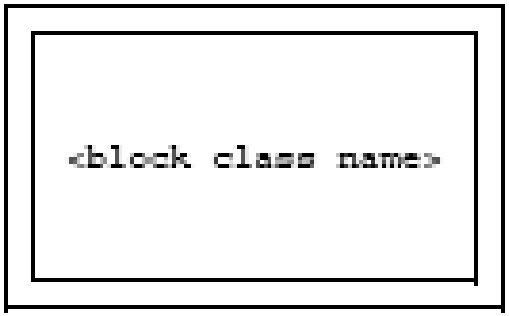
SDL-angepasste-Syntax

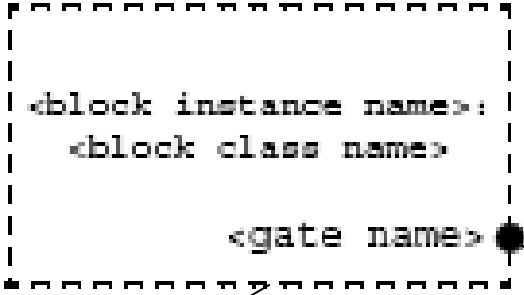


keine Attribute
Operations → PseudoNachrichten (nurEinParameter)
Property für Gate-Angabe

SDL-RT: Block-Typen (Block-Klassen)

Einschränkungen gegenüber Z.100





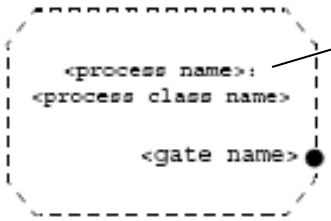
keine Vererbung
keine Virtualität
keine Kontextparameter

Instanzmengen sind nicht zulässig

Gates ohne Atleast-Typ-Angabe

SDL-RT: Process-Typen (Process-Klassen)

Einschränkungen gegenüber Z.100



Instanzmengenbezeichener
:
Process-Typbezeichner

Vererbung eingeschränkt gegenüber Z.100
- Spezialisierung: Zustände und Transitionen wie Z.100
- aber keine Kontextparameter

Redefinition von
- Transitionen (alle sind implizit virtuell)
- Gates (Virtualität ist anzugeben)
- Datentypen

Abstrakte Process-Typen, falls
- abstrakte Trigger
- abstrakte Gates

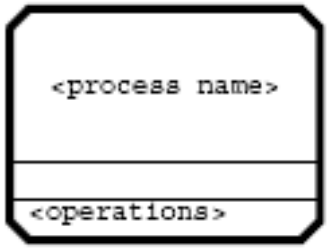
Präfix: ABSTRACT

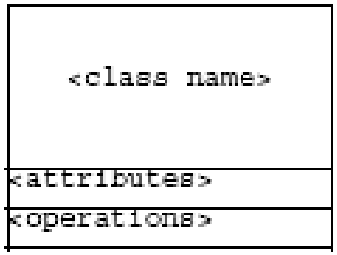
Präfix: VIRTUAL

UML-Klassendiagramme

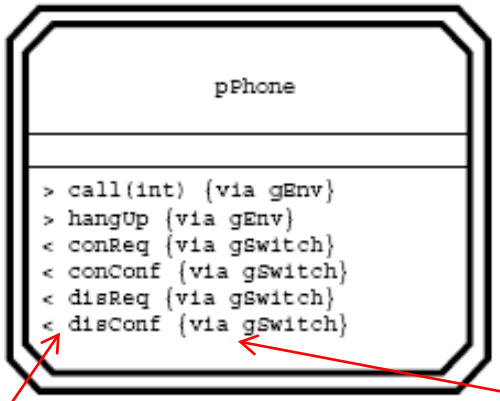
- Aktive Klassen, passive Klassen

vordefinierte Stereotypen
System, Block, Blockclass,
Process, Processclass,





aktive Klassen haben keine Attribut-Kompartments (Attribute werden woanders definiert)
Operationen sind ein- und ausgehende Nachrichten (asynchron) mit Gate-Angabe
Leider: hier keine Angaben von Nachrichtenlisten erlaubt !!! (Grund unklar)

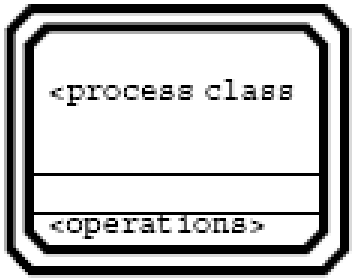


SDL-RT-Doko fehlerhaft:

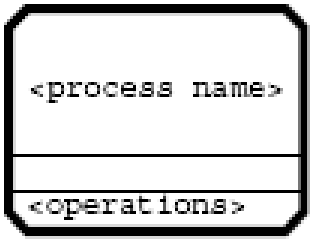
kein Leerzeichen

Doppelpunkt statt Leerzeichen

Vordefinierte graphische Symbole von stereotypisierter Klassen



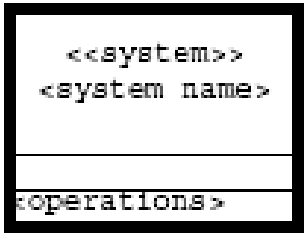
Class stereotyped as
a class of process



Class stereotyped as
a process

Class stereotyped as
a block

Class stereotyped as
a class of block



Class stereotyped as
a system

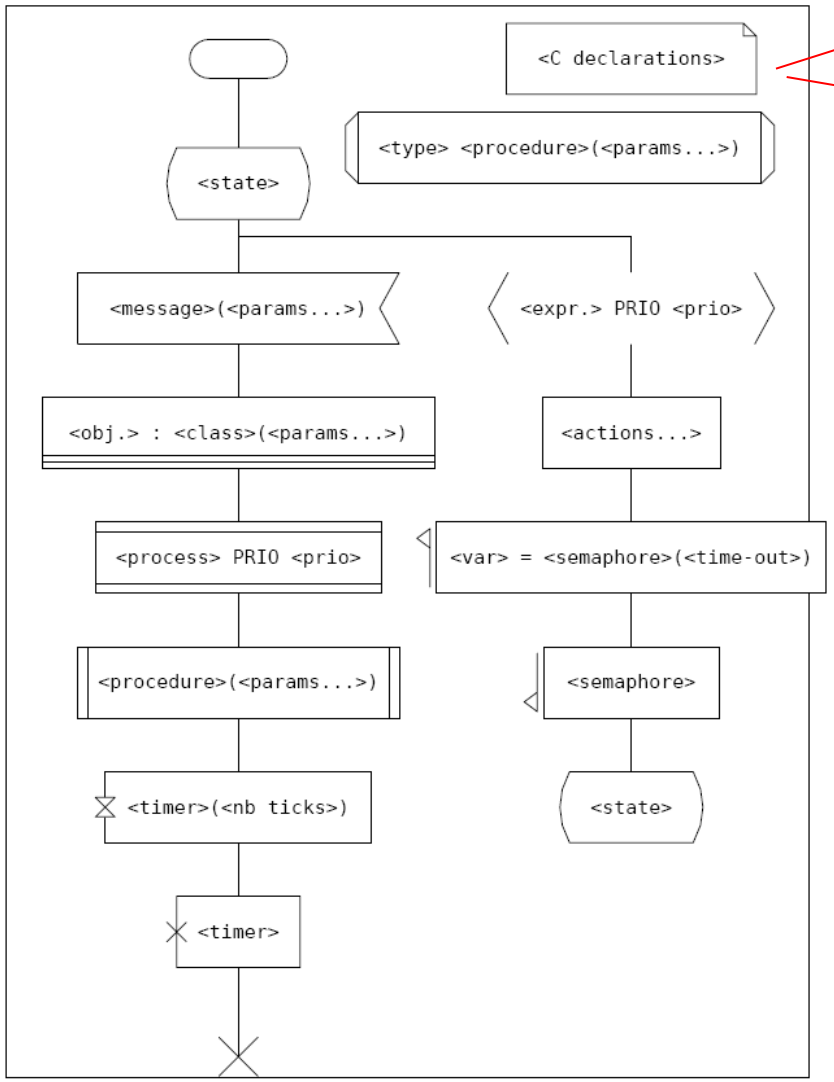
Datentypen

- zwei Möglichkeiten für Nutzung von C-Datentypen

  – externes Header-File (Projekt-Manager)
  #include dann auf Block- oder Process-Level möglich

  – SDL-Text-Box
    - Block-Level → daraus wird separate C-Header-Datei
      (globale Typen, globale Variablen)
    - Process-Level → für Process-lokale Variablen
    - Procedure-Level → für Prozedur-lokal Variablen

Process-lokale Deklarationen



gestrichelt: SDL-Deklaration

voll: C-Deklaration

SDL-Signale und Signallisten

- Message statt Signal

- MESSAGE msg1, msg2(int),
  msg3(char*, struct MyType*, double)
  - Parameter: gültige C-Typen (auch Zeiger)

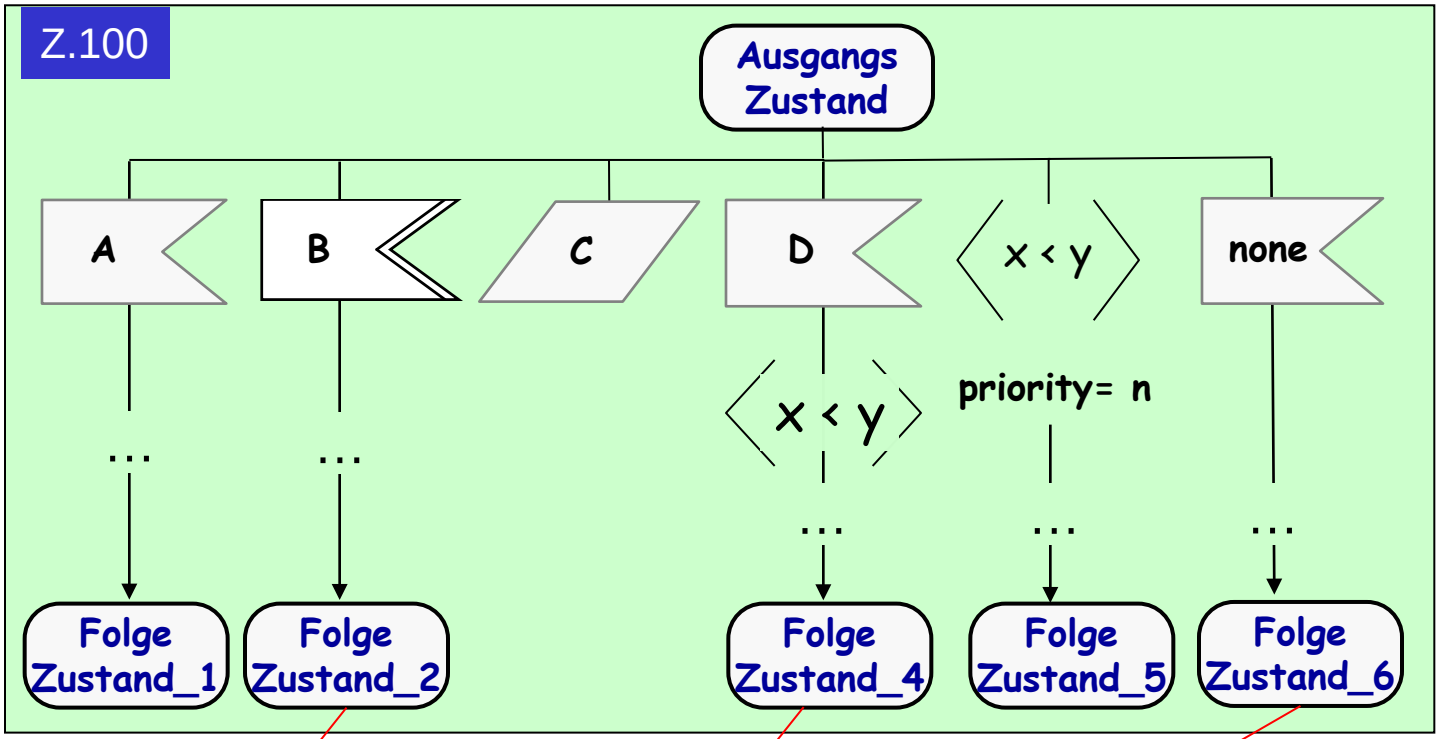
- MESSAGE_LIST myList= ((subList), msg1, msg3);

- implizite Dekalaration von
  - Timern und
  - Timeout-Nachrichten

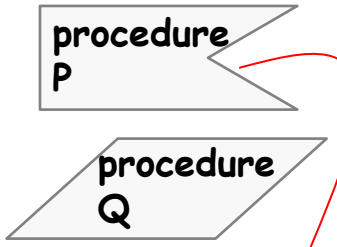
7. SDL-Konzepte *(Präzisierung, 1.Teil)*

1. Modellstruktur
2. Einfacher Zustandsautomat: Trigger-Arten
3. Nachrichtenadressierung
4. Timer

Die Triggervarianten von SDL *im Überblick*



weitere Trigger



nicht in SDL-RT kann aber modelliert werden

nicht in SDL-RT kann aber modelliert werden

nicht in SDL-RT kann aber modelliert werden

nicht in SDL-RT kann aber modelliert werden

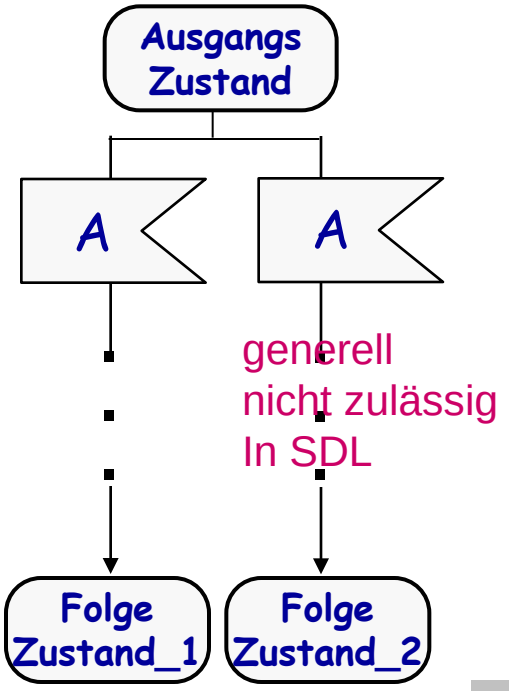
Normale Signaleingabe: *Input ohne Parameter*

state Ausgangszustand;

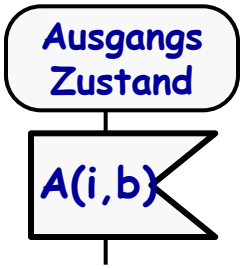
input A;

Achtung:
ein Signal darf für je ein Ausgangszustand nicht mehr als einmal als Trigger vorgesehen werden

- der Übergang aus *Ausgangszustand* (in dem die Prozessinstanz verharrt), wird ausgeführt, sobald *A* zum ersten Signal im Puffer wird
- das Signal *A* wird konsumiert
  (d.h. Parameter werden evtl. übernommen,
  SENDER wird gesetzt,
  Signal wird im Puffer gestrichen)
- hat ein Prozess ein Signal unverändert weiterzugeben, sind zunächst alle Parameter zu kopieren



Normale Signaleingabe: *mit Parameterübernahme*

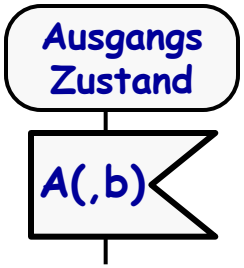


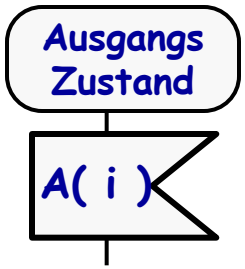

```
state Ausgangszustand;
input A (i, b);
```

```
MESSAGE A (int, bool);
/* beliebige Signatur */
```

```
int i,
bool b;
```

- beide Parameter werden übernommen (d.h. in die Variablen *i* und *b* kopiert)
- Vorraussetzung dafür ist, dass *i* und *b* lokale Variablen des Empfängerprozesses sind und zuweisungs-kompatible Typen repräsentieren
- ***sender***-Variable wird gesetzt

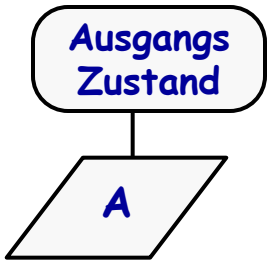




Achtung:
eine partielle Übernahme der Parameter ist möglich

- hat ein Prozess ein Signal unverändert weiterzugeben, sind zunächst alle Parameter zu kopieren

Signalzurückstellung: Save




```
state Ausgangszustand;
save A;
```

```
MESSAGE A (int, bool);
/* beliebige Signatur */
```

- das aktuelle Signal *A* wird zurückgestellt, das nachfolgende Signal wird damit zum neuen aktuellen Signal
- ist *A* das einzige Signal, verharrt der Prozess im Ausgangszustand bis ein weiteres Signal eintrifft
- sobald ein neu eingetroffenes Signal zu einem Zustandsübergang geführt haben sollte, ist die Zurückstellung von *A* wieder aufgehoben
- weder die Parameter von A werden kopiert, noch wird **sender** aktualisiert

Save- Semantik

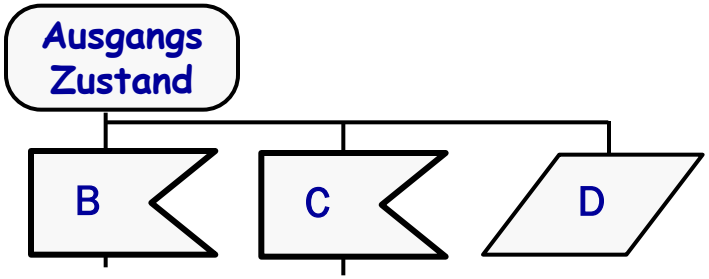
Z.100

The saved signals are retained in the input port in the order of their arrival.

The effect of the save is valid only for the state to which the save is attached.

In the following state, signal instances that have been "saved" are treated as normal signals

Signalverwerfung: implizites Discard

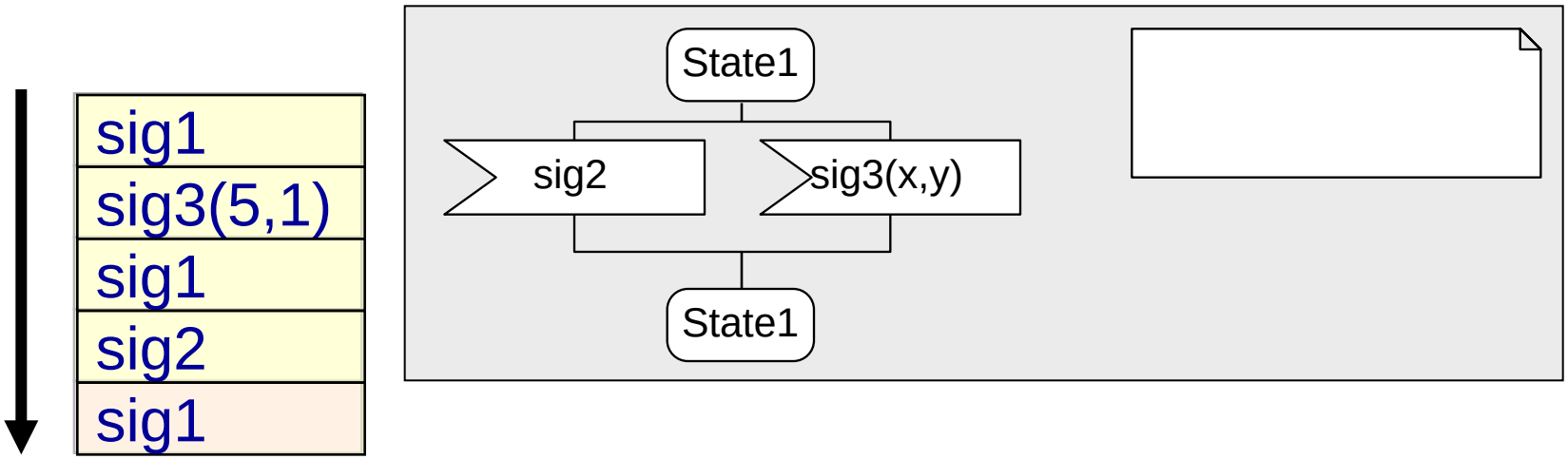



```
MESSAGE A (int, bool);
/* beliebige Signatur */
```

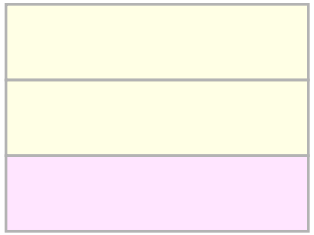
- Sollte für einen Zustand, indem sich der Prozess befindet, **kein Trigger** für das aktuelle Signal vorgesehen sein,
  wird es (ohne Kopie seiner expliziten und impliziten Parameter) **verworfen**

- o.B.d.A.: sei *A* dieses Signal für *Ausgangszustand*, wird dieses (und nur dieses) *A* verworfen

Trigger-Beispiel-1



nachher (im Zustand *State1*)

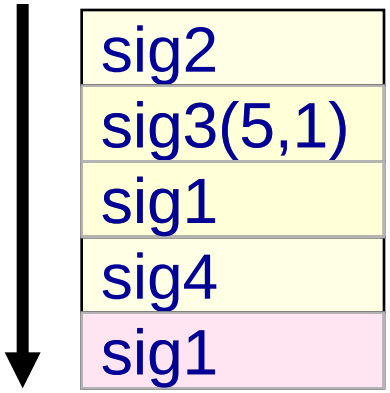


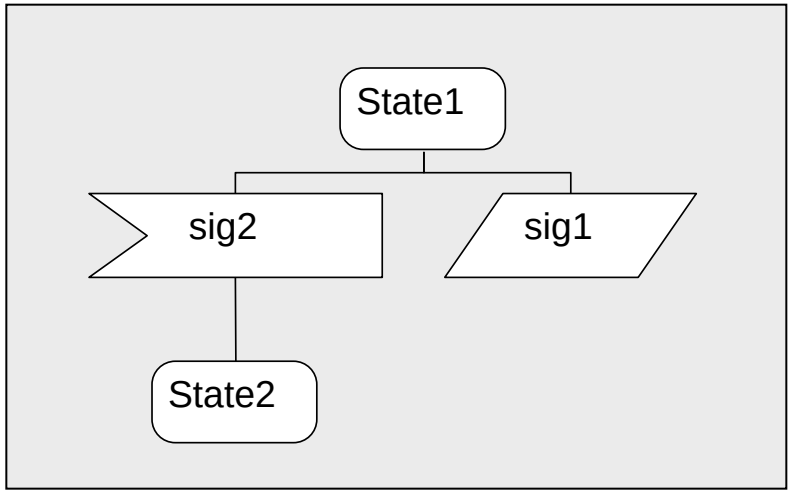
1. sig1 → discard
2. sig2 → input
3. sig1 → discard
4. sig3 → input mit Parameterübergabe
5. sig1 → discard

sender= PId-Wert des Senders von **sig3**

Trigger-Beispiel-2

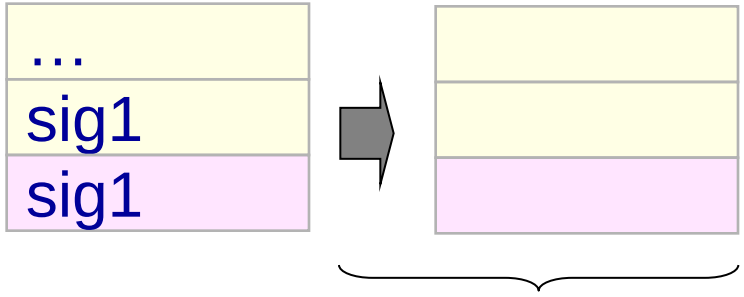
vorher





1. sig1 → save
2. sig4 → discard
3. sig1 → save
4. sig3 → discard
5. sig2 → input

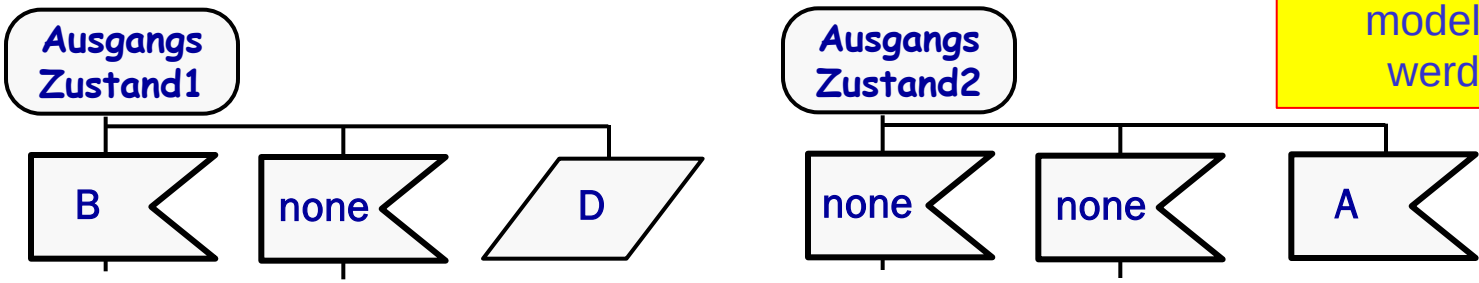
nachher (im Zustand *State2*)



falls keine weiteren Übergänge in *State2*

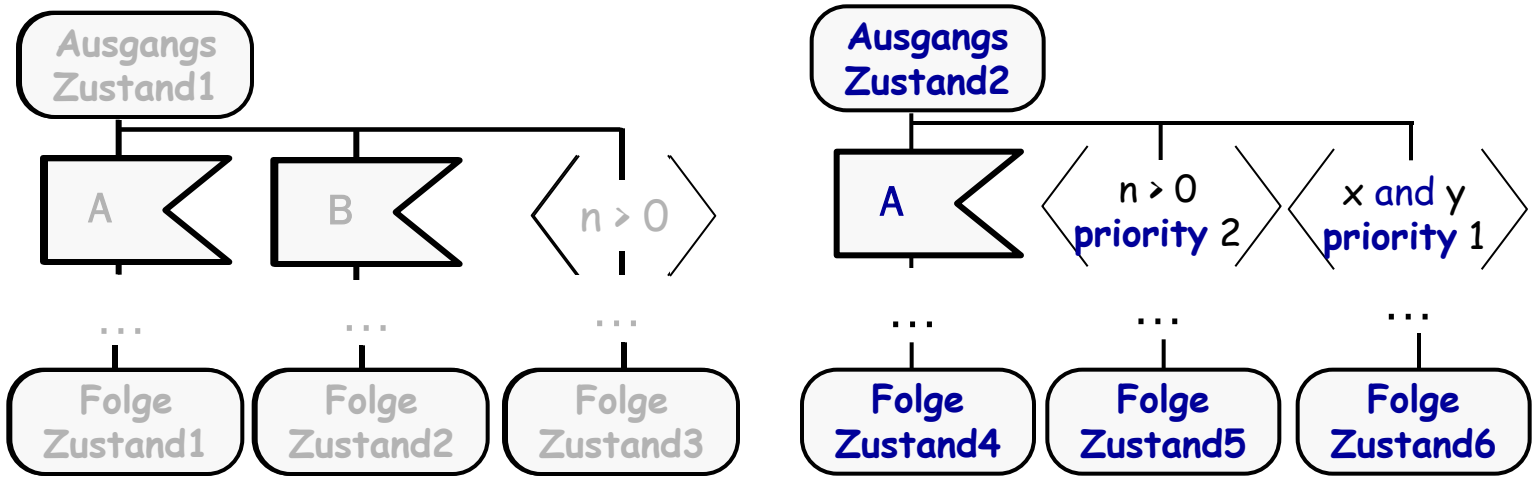
Spontaner Zustandsübergang

nicht in SDL-RT kann aber modelliert werden



- ein spontaner Übergang benötigt **keine** bestimmte Belegung des Signalpuffers – er kann auch erfolgen, wenn der Puffer leer ist
- ob und wann der spontane Übergang vollzogen wird, ist **nichtdeterministisch** bestimmt. Mit Cinderella (Werkzeug) lässt sich eine Wahrscheinlichkeit einstellen.
- der **sender**-Wert nach einem spontanen Übergang ist **null**
- der spontane Übergang wird benötigt, um Störeinflüsse auf das Verhalten eines Systems nachbilden zu können
- mehrere spontane Übergänge für einen Zustand sind zulässig, **maximal** kann aber nur **einer** bei einem Zustandsübergang zur Realisierung kommen

Trigger: Continuous-Signal



der Übergang aus *Ausgangszustand1* in *FolgeZustand3* wird ausgeführt

- falls $n > 0$ ist (*n* als lokale Variable des Prozesses) und
- sich weder *A* noch *B* im Puffer befindet

➔ normale Input-Trigger sind höher priorisiert als Continuous-Trigger

der Übergang aus *Ausgangszustand2* in *FolgeZustand6* wird ausgeführt

- wenn sich *A* **nicht** im Puffer befindet und (*x* and *y*) den Wert true liefert
- der Übergang wird **unabhängig** vom *n*-Wert ausgeführt

➔ niedrigster Prioritätswert bedeutet höchste Priorität

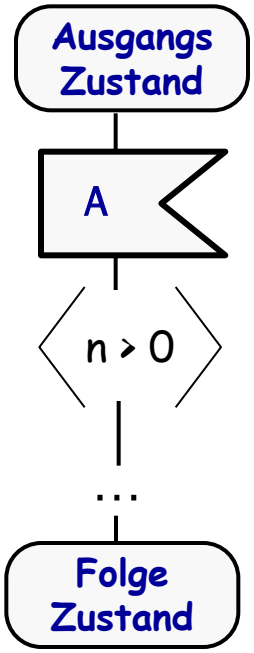
Trigger: Input mit Bedingung

der normale Input-Trigger kann gesteuert werden

- der Übergang kann nur vollzogen werden, wenn A das aktuelle Signal im Puffer **und** die zusätzliche Bedingung erfüllt ist
- da die Bedingung nur von lokalen Variablen abhängig ist, besteht eine reale **Deadlock**-Gefahr für den Prozess

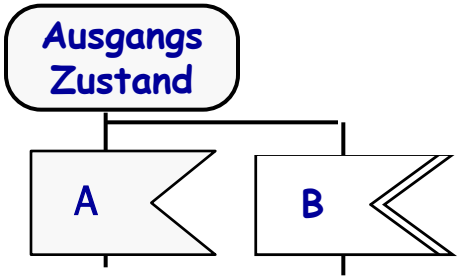
➔ **Lösung**: alternative Transitionen über

- normale Inputs
- priorisierte Input



Achtung: Eine Bedingung kann nicht über explizite oder implizite Signalparameter gebildet werden – wenn doch, dann über die Werte des zuletzt ausgeführten Zustandsübergangs

Priorisierter Zustandsübergang



- der Übergang, ausgelöst durch ein Signal B im Ausgangszustand, ist gegenüber allen anderen **priorisiert**, unabhängig von der Position von B im Puffer
  A behält seine Position im Puffer bei
- der normale Input bei Konsumtion von A kommt im Ausgangszustand nur dann zum Ausführung, wenn sich **keine** Signale B im Puffer befinden

Achtung: SDL kennt keine Signalpriorisierung,
dafür aber Trigger-Priorisierung

Rangfolge:

(1) Priorisierter Input

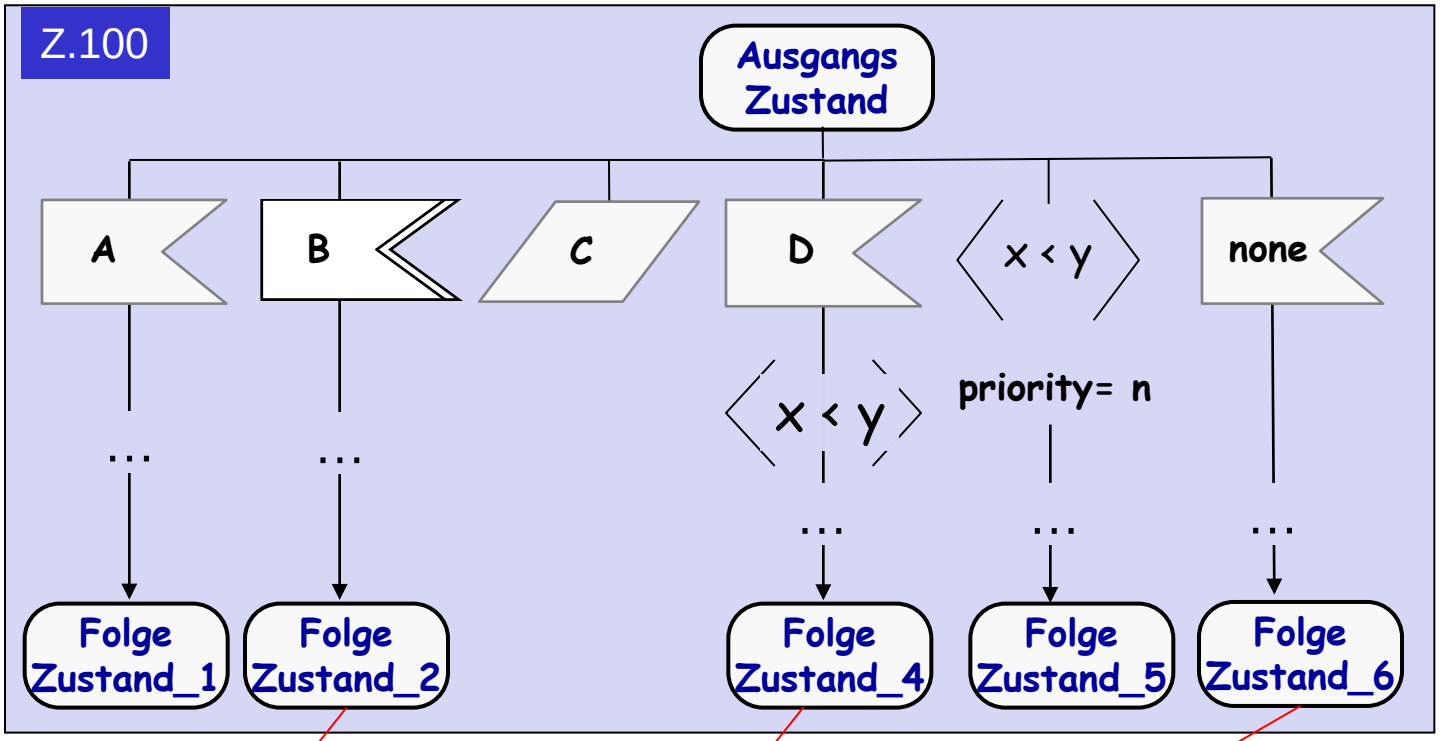
(2) Normaler Input

(3) Input mit Bedingung

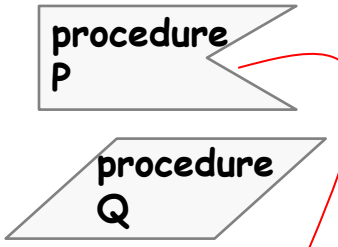
(4) Continuous-Signal (mit Prioritätsklassen)
bei Gleichheit: erfolgt eine nichtdeterminierte Auswahl

} überlagert durch spontane Trigger

Die Triggervarianten von SDL *im Überblick*



weitere Trigger



nicht in SDL-RT kann aber modelliert werden

nicht in SDL-RT kann aber modelliert werden

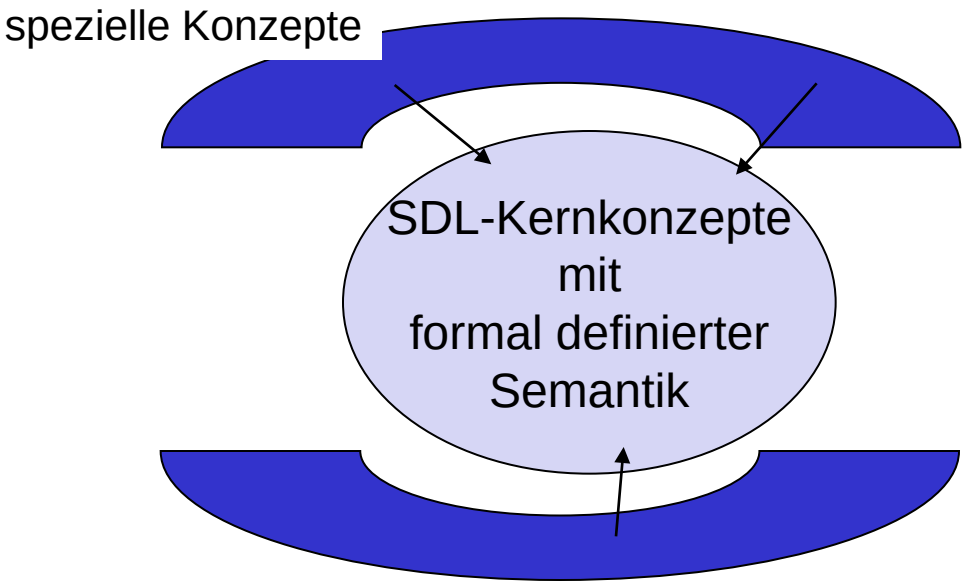
nicht in SDL-RT kann aber modelliert werden

nicht in SDL-RT kann aber modelliert werden

Semantik-Definition der Trigger

Beherrschung der semantischen Komplexität

- Mehrzahl neuer Konzepte in **SDL-92/96** wird durch Transformation auf Kernkonzepte definiert



Beispiel:
- Priority Input,
- Continuous Signal Input,
- Input mit Vorbedingung
- (und RemoteProcedure-Input/Save)

werden auf

- normalen Input,
- Save,
- Discard
- (und gewöhnliche Prozeduren)

zurückgeführt

7. SDL-Konzepte *(Präzisierung, 1.Teil)*

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Nachrichtenadressierung
4. Timer

Zwei Adressierungsarten

(1) OUTPUT *msg-name (param-List)* **TO_X** *receiver*

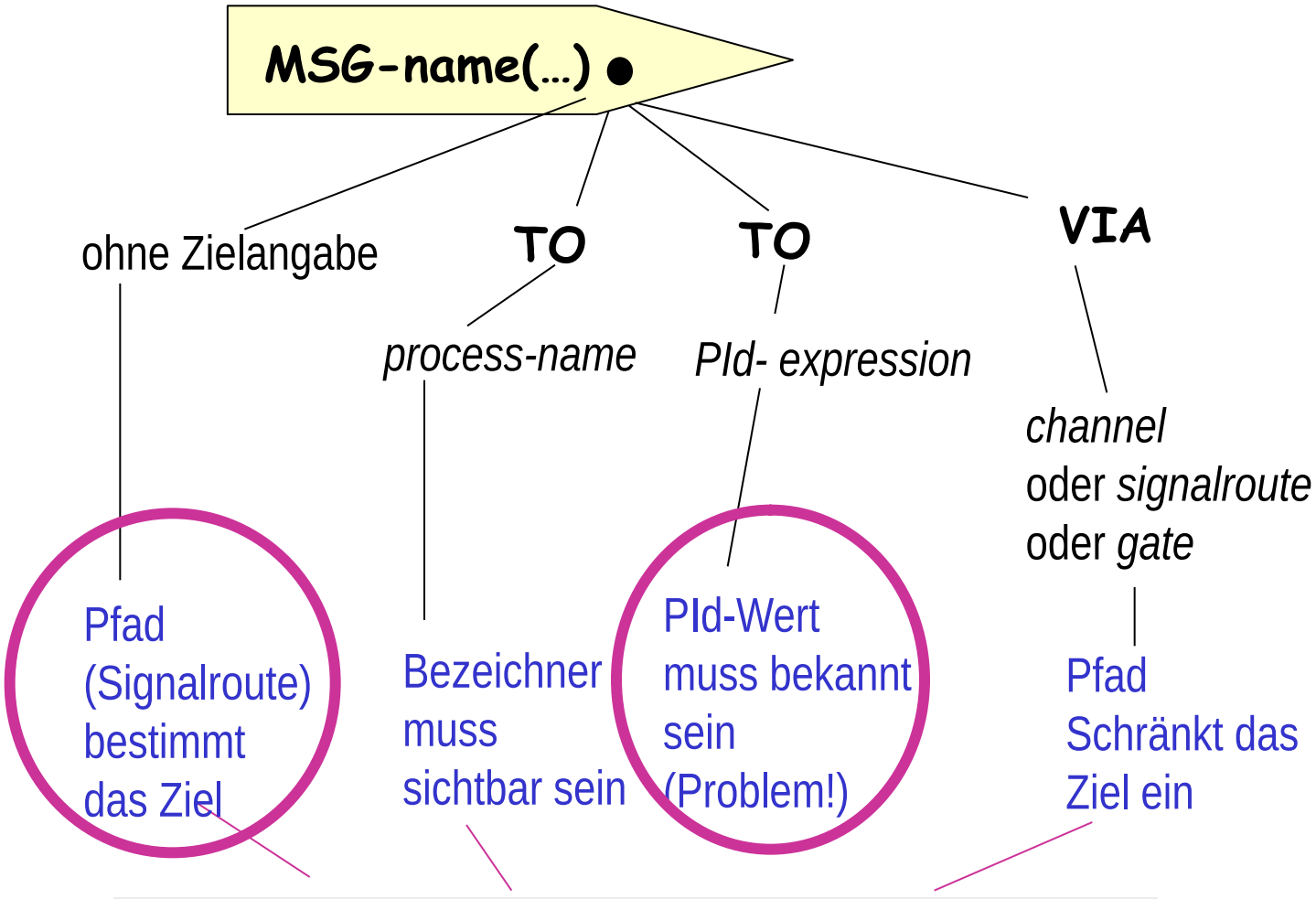
(2) OUTPUT *msg-name (param-List)* **VIA** *medium*

- Parameter: Werte, Referenzen
  bei Referenzen ist wichtig: Welcher Prozess legt Lebenslauf fest.
  häufig:
  - Sender besitzt Datum nicht mehr, nachdem gesendet worden ist,
  - Empfänger gibt Speicherplatz frei (daraus entstehen Probleme)

- **TO_X**:
  - **ID**: PARENT, SELF, OFFSPRING, SENDER, oder RTDS_QueueId
  - **NAME**: erster generierter Prozess einer Menge
    (nicht benutzen, wenn Name nicht 1-deutig)
  - **ENV**

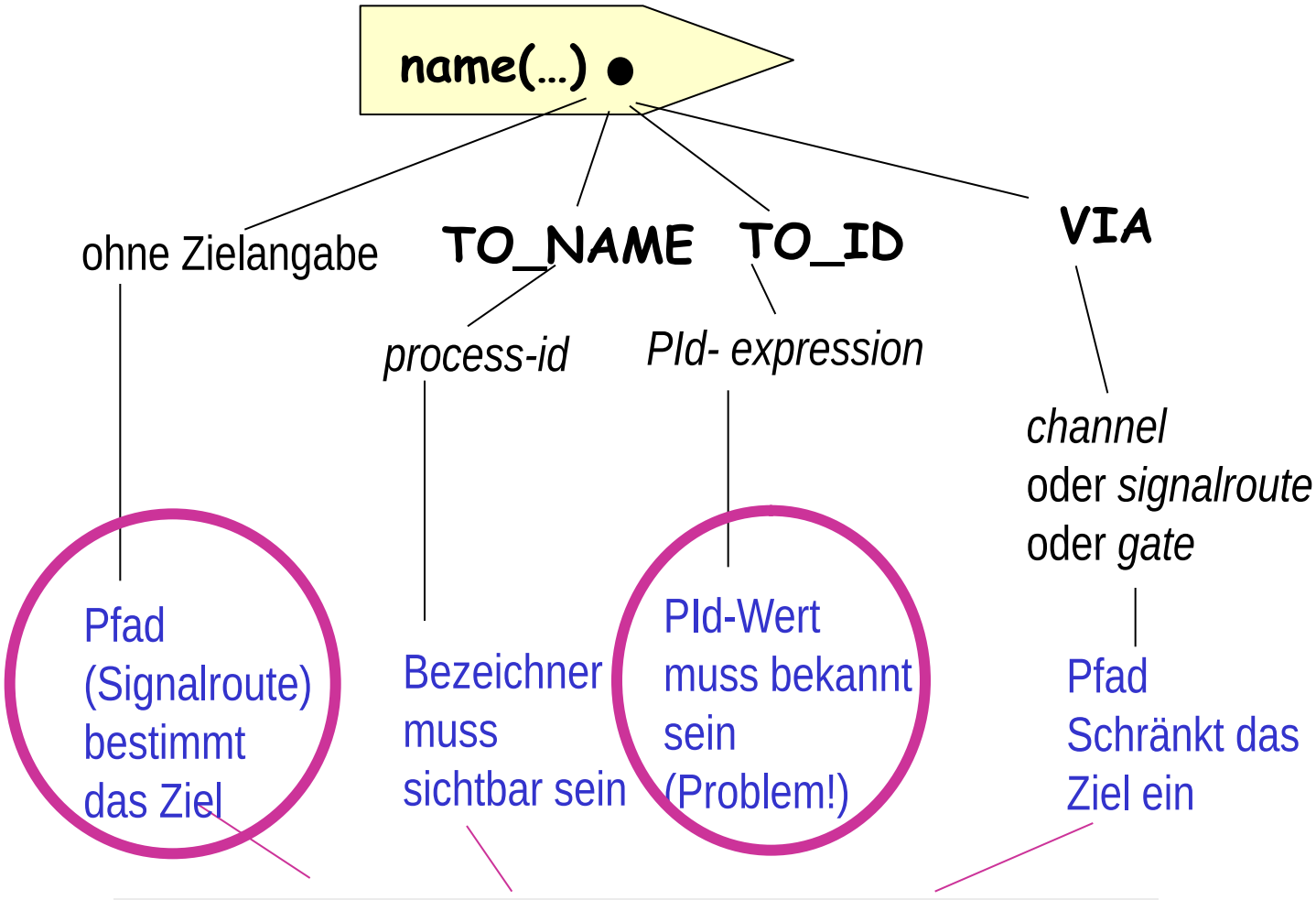
- medium:
  - Name von Channel oder Gate (verbunden mit aktuellem Prozess)
    keine Mehrdeutigkeit zulässig!

SDL-Standard: Nachrichten-Adressierungsarten



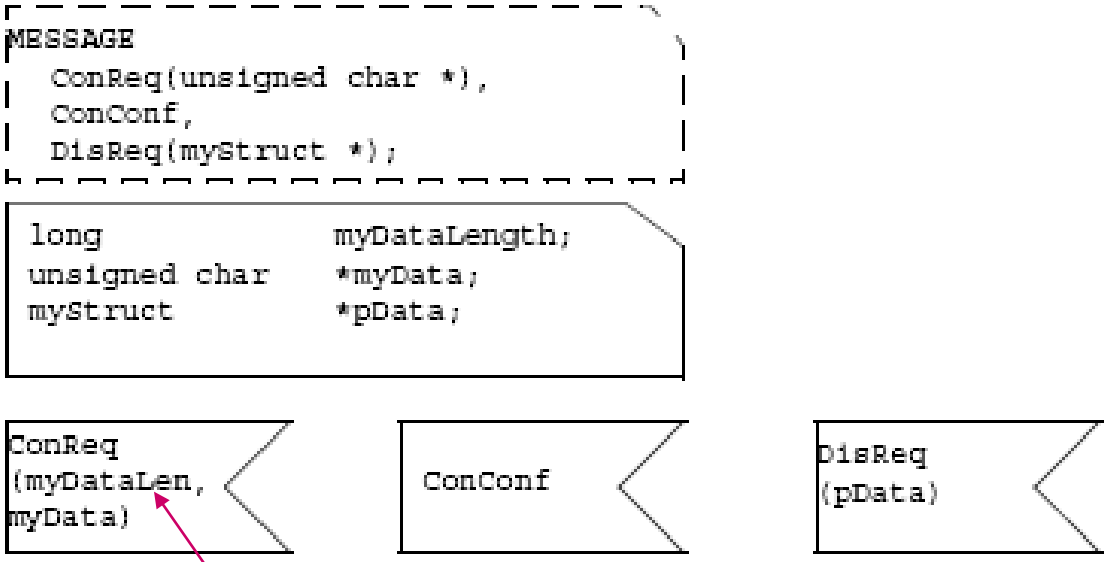
bei Mehrdeutigkeiten der Empfänger-Instanz erfolgt eine nichtdeterminierte Auswahl

SDL-RT: Nachrichten-Adressierungsarten



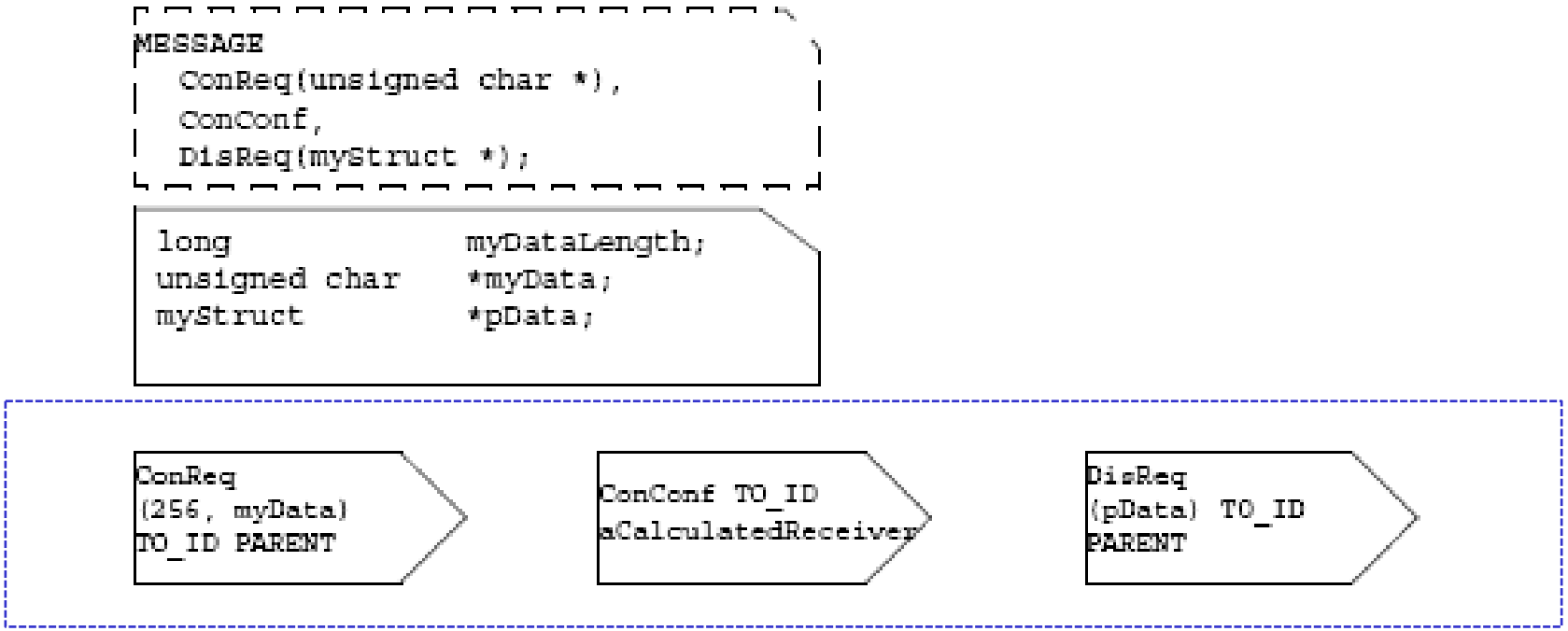
bei Mehrdeutigkeiten der Empfänger-Instanz erfolgt eine ~~nicht~~determinierte Auswahl

SDL-RT: Zeiger als Parameter



unsigned char* man kann optional auf die Länge zugreifen (ist als erster Parameter zu übergeben)

SDL/RT- Adressierungsarten



Adressierungsart TO_NAME: wird rudimentär unterstützt

Spezifische Adressierungsklauseln:
-TO_ID (PID-Ausdruck)
- TO_NAME
- TO_ENV

7. SDL-Konzepte *(Präzisierung, 1.Teil)*

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Nachrichtenadressierung
4. Timer

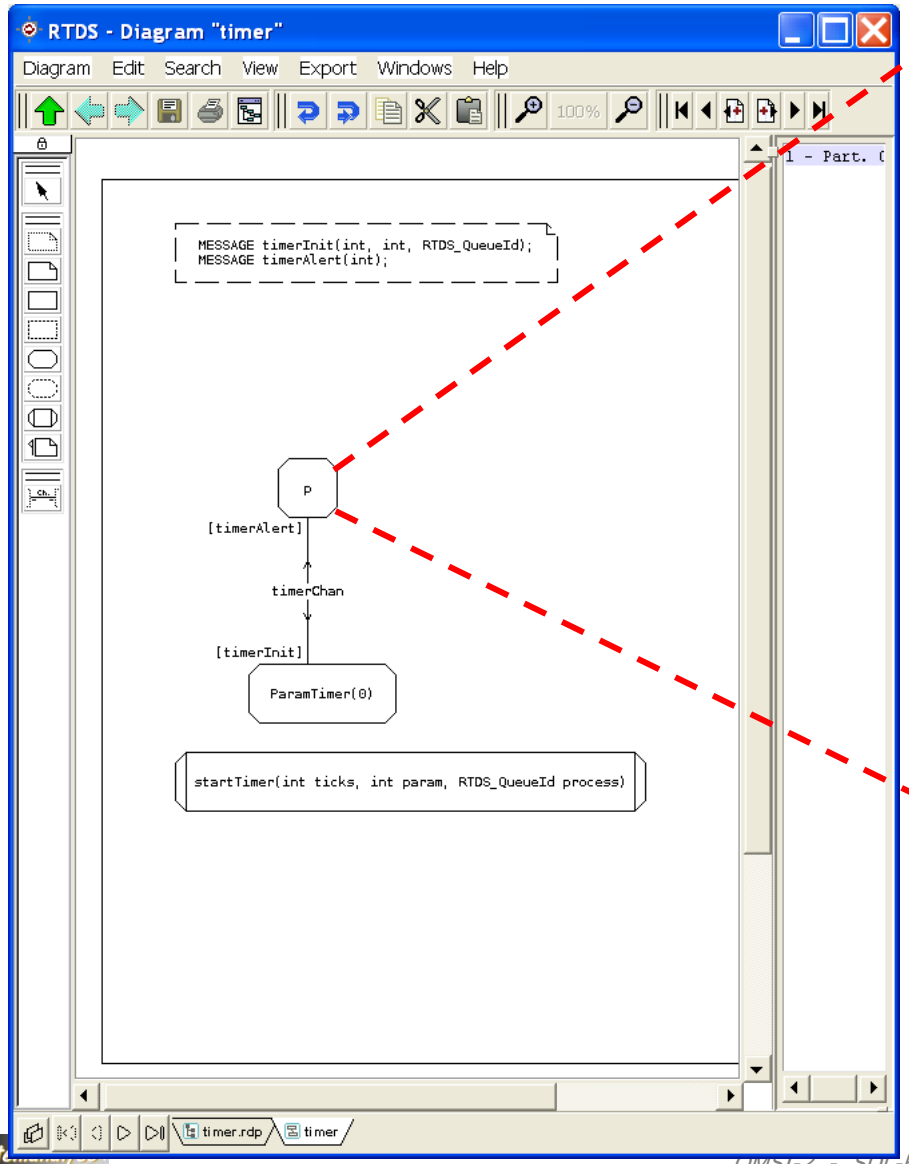
Timer-Konzept

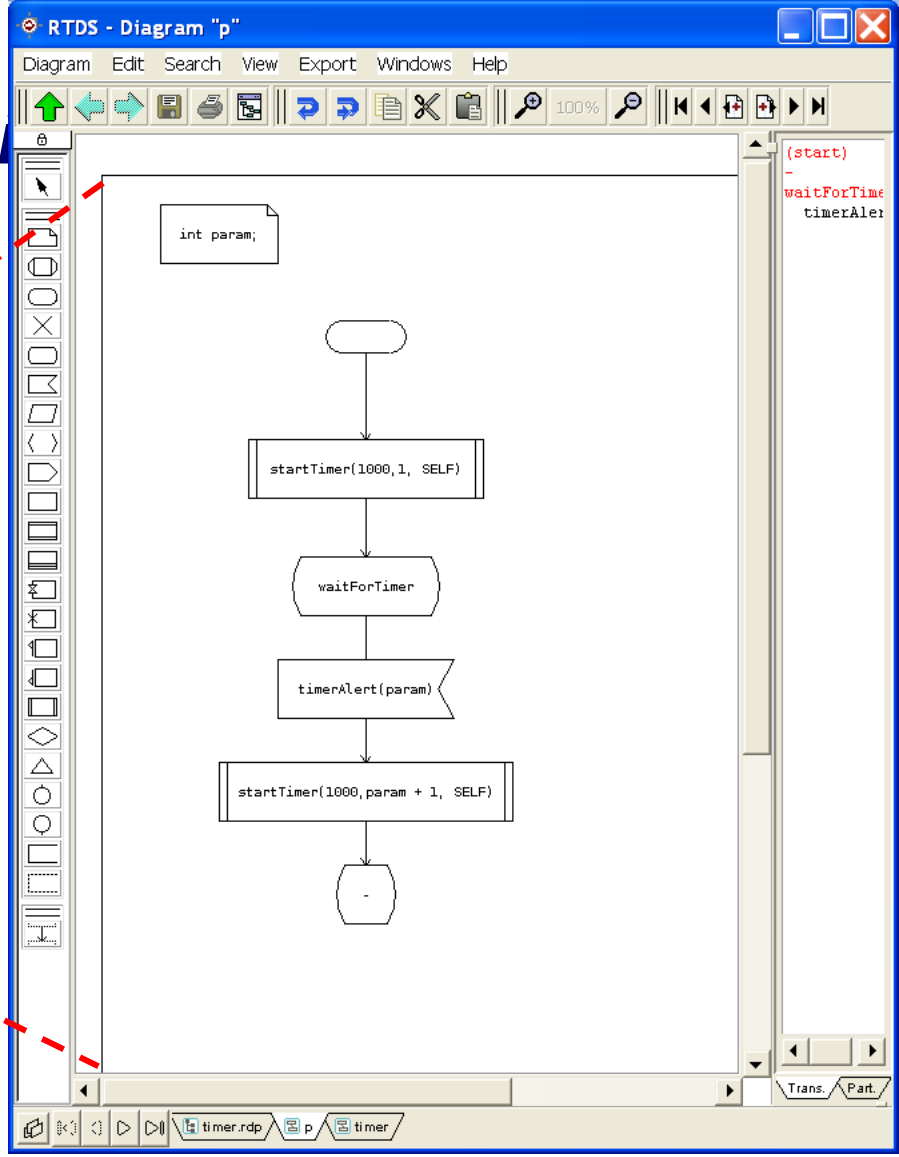
- Grundkonzept zur Verhinderung von Deadlocks und Lifelocks in realen verteilten Systemen
  - Starten: start/restart bei Angabe der Timeoutzeit
  - Stoppen: stop
  - Timeout: Timout-Nachrichten unterscheidbar durch Namen des auslösenden Timers
  - Jeder Prozess kann überbeliebig viele Timer verfügen

- Z.100-SDL erlaubt Parametrisierung/Indizierung von Timern zur Identifikation (SDL-RT nicht)

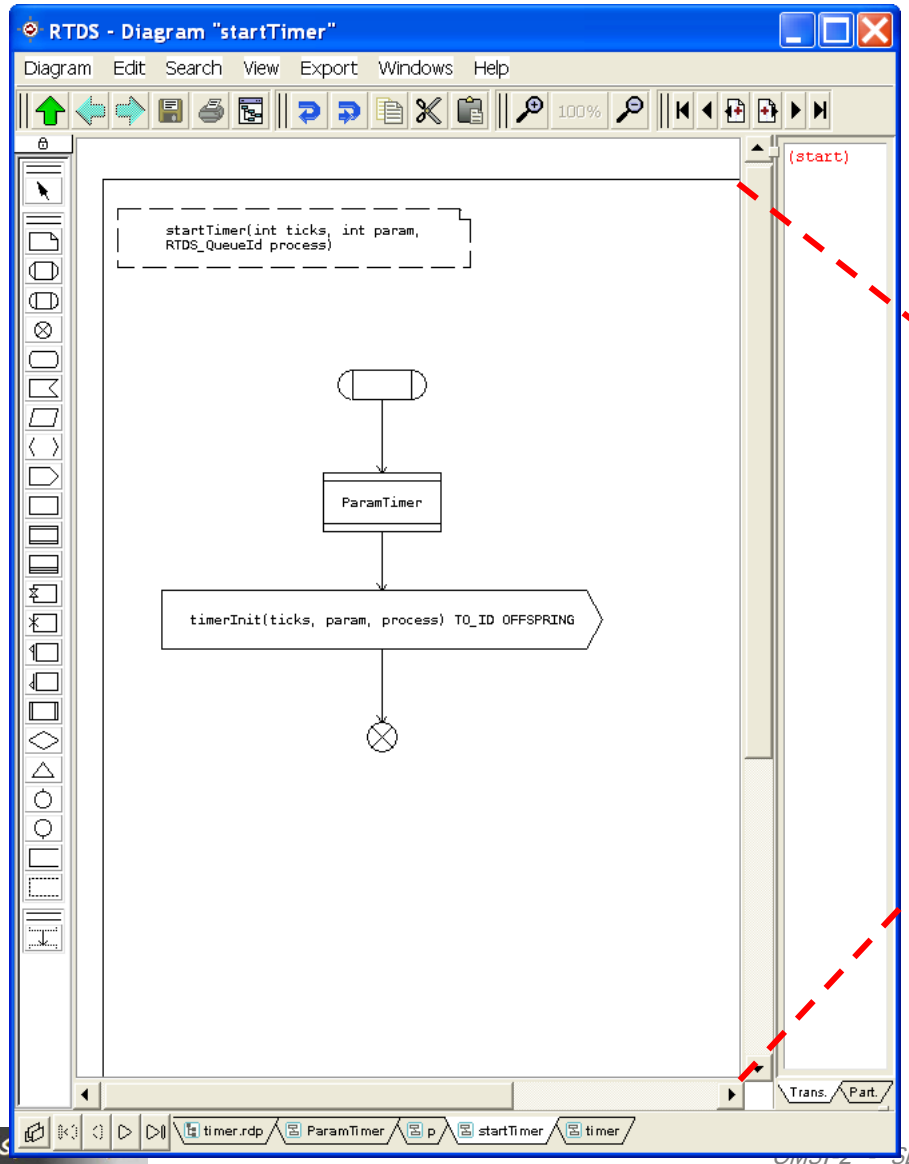
- In SDL/RT bleibt nur ein nutzereigenes Ersetzungsmodell

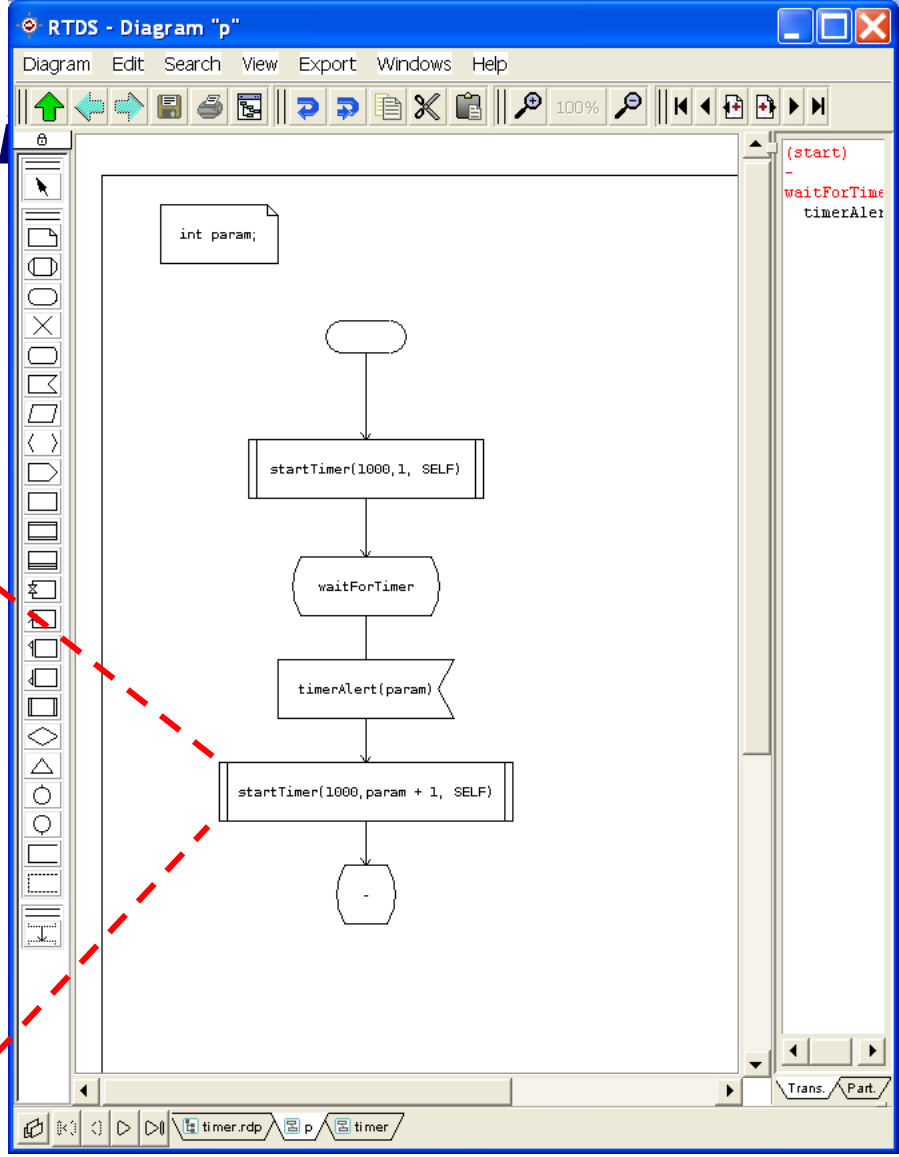
Timer als eigenständi



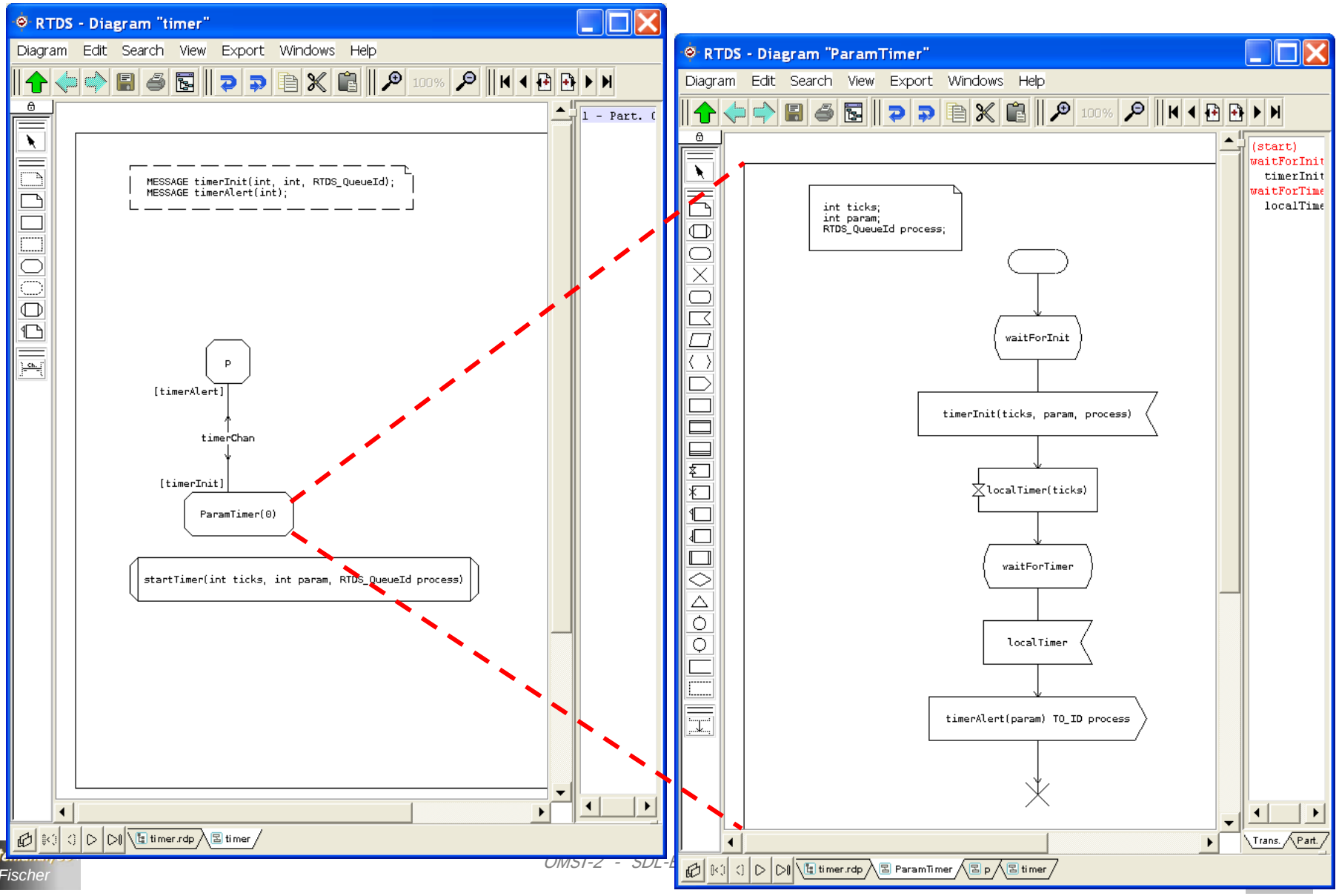


Timer als eigenständi





Timer als eigenständiger Prozess



8. Protokollentwicklung in SDL

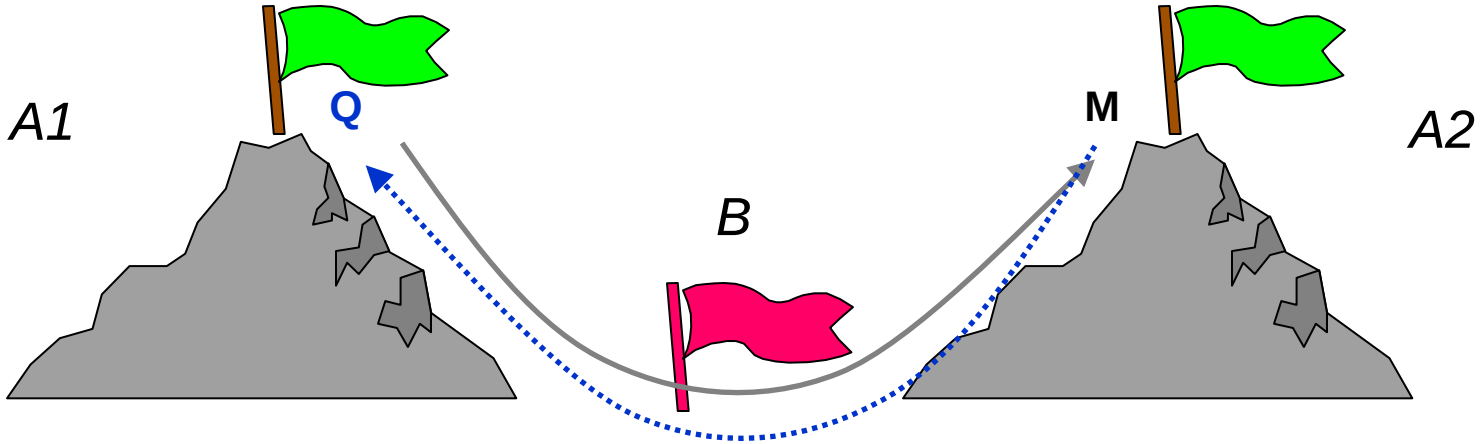
- OSI-Schichtenmodell (Konzept)
- InRes-Protokoll (Pseudo-Rechnernetzprotokoll)
- Umsetzung in SDL/RT
- weiterer Ausbau

Protokolle im Altertum

Übertragung von Nachrichten über unsichere „Kanäle“
bei Quittieren dieser Nachrichten

Ziel

Absprache eines gemeinsamen Angriffstermins der verteilten grünen Armee



Fall: *A1* erhält tatsächlich eine Bestätigung Q für Nachricht M von *A2*
Wie sicher können sich A1 und A2 sein, dass Termin gilt ?

Protokoll (Protocol)

- Regeln zur Kommunikation zwischen Partnerinstanzen
  innerhalb einer Kommunikationsebene
  - Wer muss wann aus welchem Grund eine Information an seinen Partner schicken
  - Vereinbarungen von Struktur und Codierungen der auszutauschenden Informationen

- Schichten-Struktur zur Beherrschung komplexer Übertragungsdienste für verbundene Rechner (als Knoten in einem Rechnernetz)
  - in realen Systemen gibt es nur eine einzige physisch vorhandene horizontale Verbindung, die anderen sind abstrakt / virtuell

- ISO-Standard zum Aufbau von Kommunikationsprotokollen:
  Open System Interconnection
  - Konzepte unabhängig von ihrer Implementierung)

Dienst (Service)

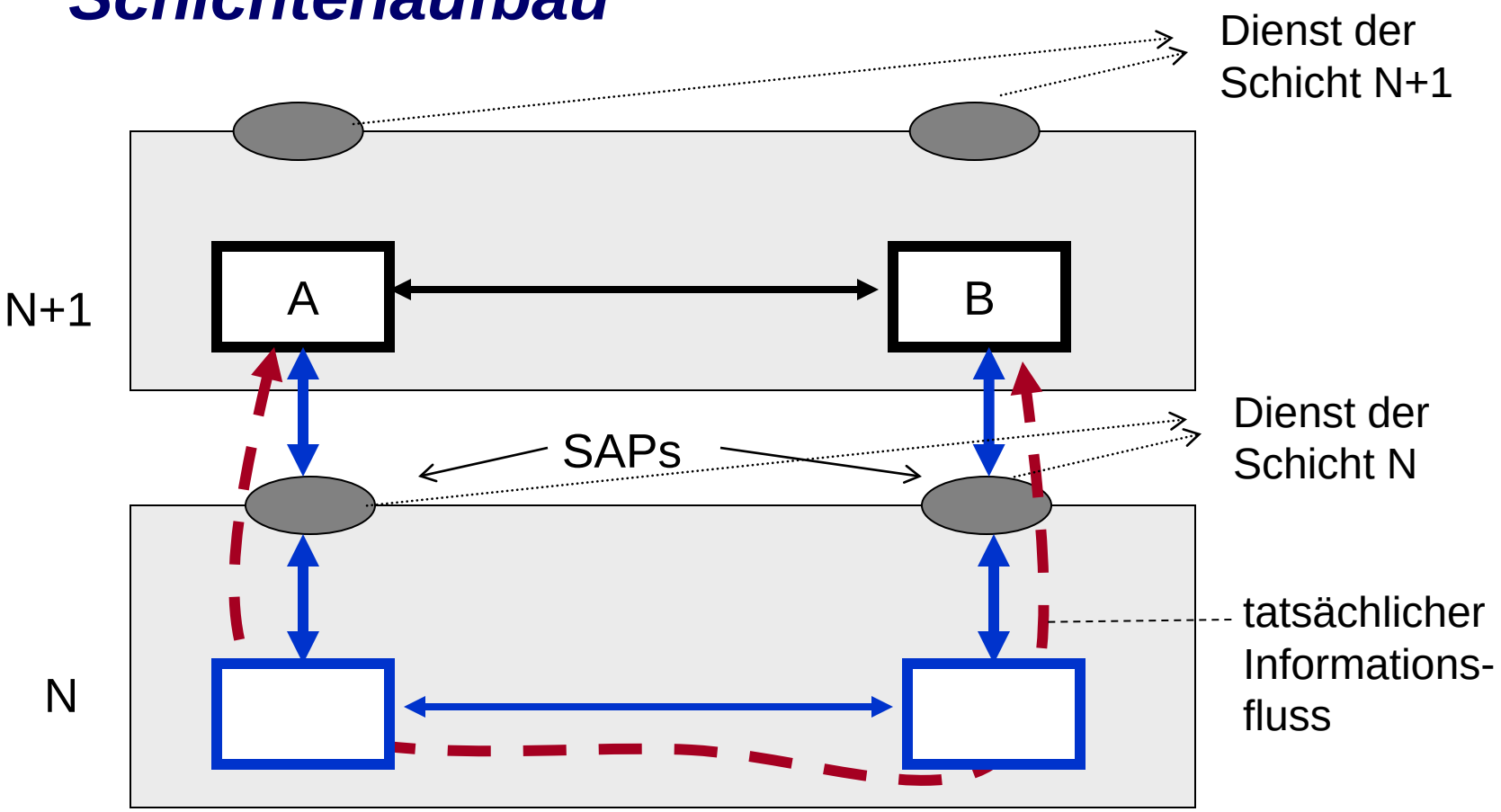
- Eine Partnerinstanz einer Schicht N (*layer N*) bietet der darüber liegenden Schicht (*layer N+1*) eine Dienstleistung (*Service*)

- Kommandos zur Dienstinanspruchnahme heißen Dienstelemente/Dienstprimitive (*Service Primitives*, SP)

- Schnittstelle einer Schicht bzgl. eines Dienstes heißt Dienstzugangspunkt (*Service Access Point*, SAP)

- es besteht eine vertikale Instanzbeziehung zwischen den Dienst-Beteiligten unterschiedlicher Schichten (Diensterbinger-Dienstnutzer)

Schichtenaufbau

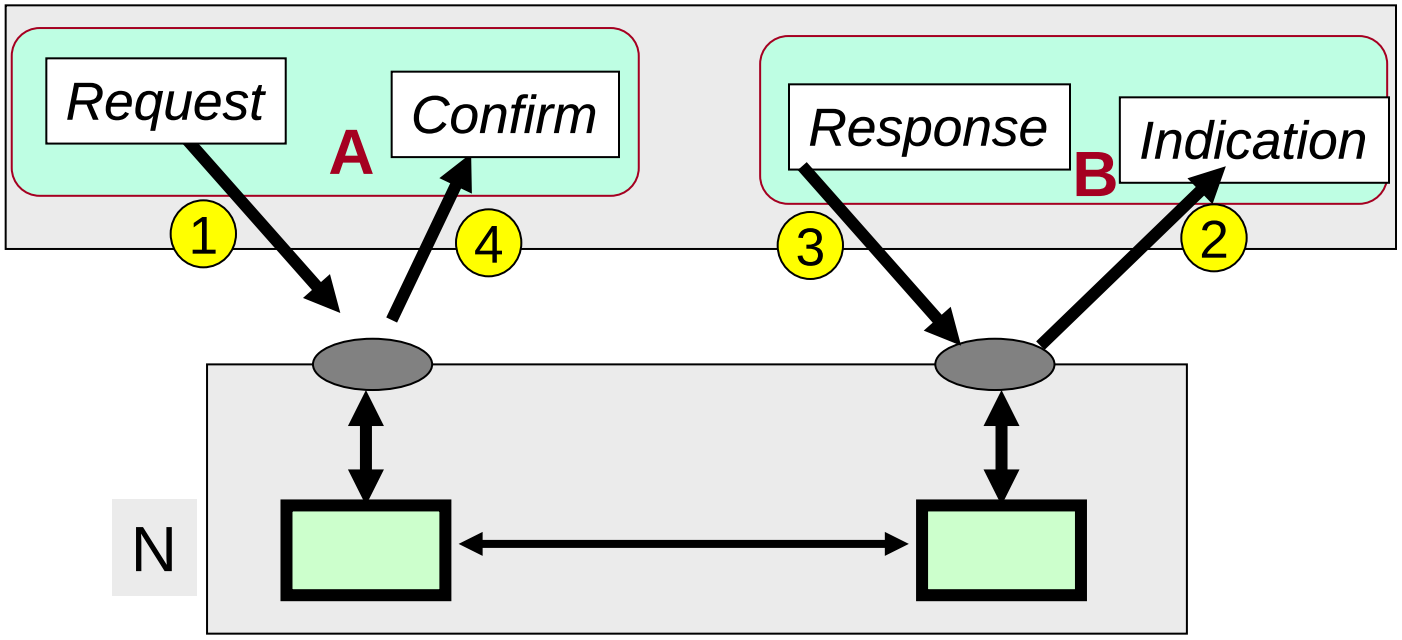


- A und B sind Kommunikationspartner
- Sie haben komplexe Informationen auszutauschen
  ohne physische Verbindung – sie vereinbaren ein Protokoll
- Sie sind Nutzer von Diensten der Schicht N, die physische Übertragung elementarer Informationen unterstützt

Identifikation von Dienstprimitiv-Klassen

vier Grundtypen von Aktionen bei **einer** gerichteten Kommunikation zwischen **zwei** Partnern

Beispiel: A ist Auslöser, sein Anliegen wird von B bestätigt



Identifikation von Dienstprimitiv-Klassen

Beispiel: B ist Auslöser, sein Anliegen soll von A nicht bestätigt werden