Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informations-systeme im Katastrophen-management

METRIK

Projekt Erdbebenfrühwarnung
im SoSe 2011

HUMBOLDT-UNIVERSITÄT ZU BERLIN

Entwicklung verteilter echtzeitfähiger Sensorsysteme

Joachim Fischer
Klaus Ahrens
Ingmar Eveslage

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

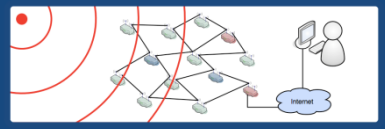
SAFER
Seismic eArly warning For EuRope

EDIM

SOSEWIN-extented

GFZ
Helmholtz-Zentrum
POTSDAM

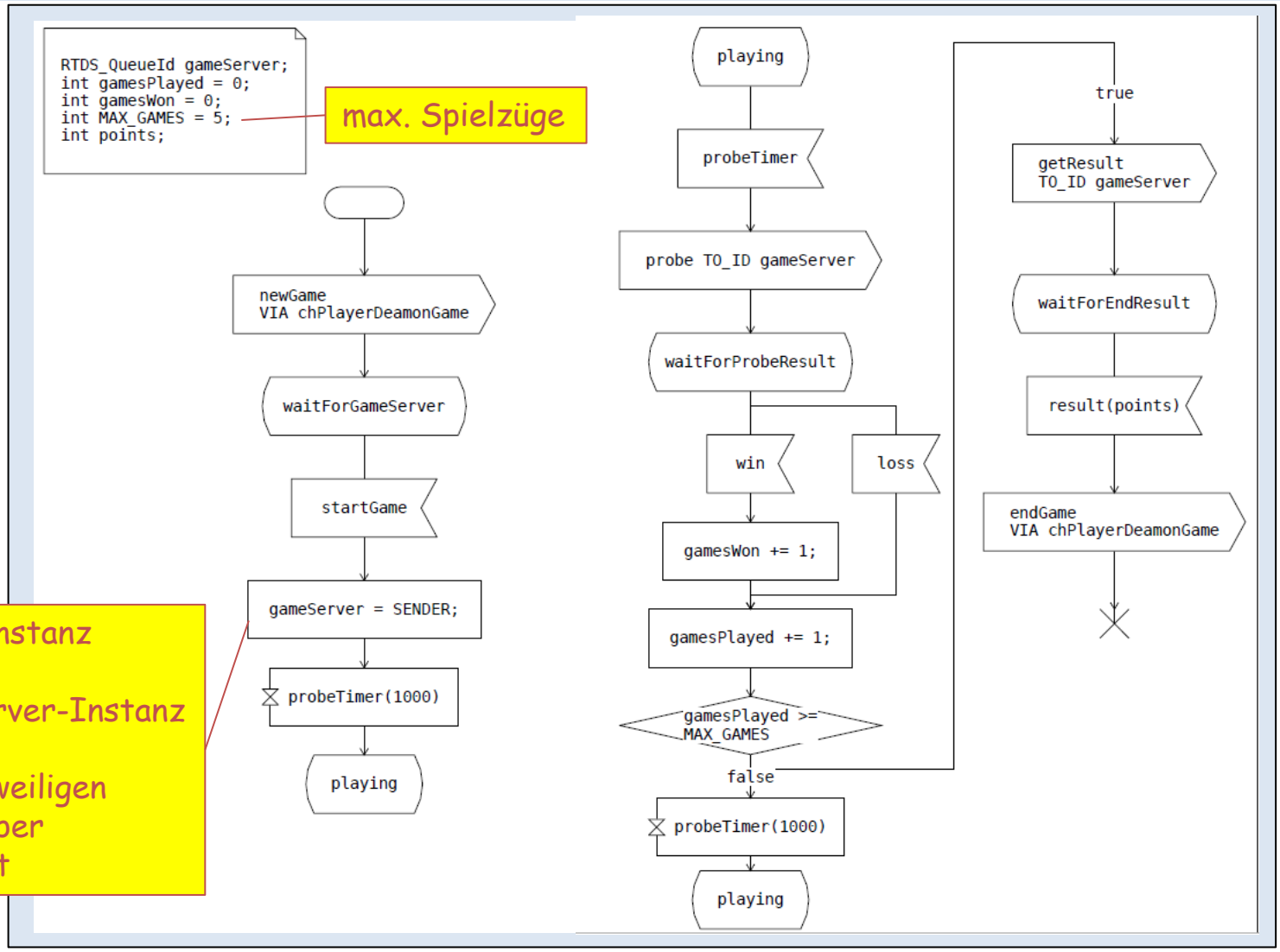
Systemanalyse



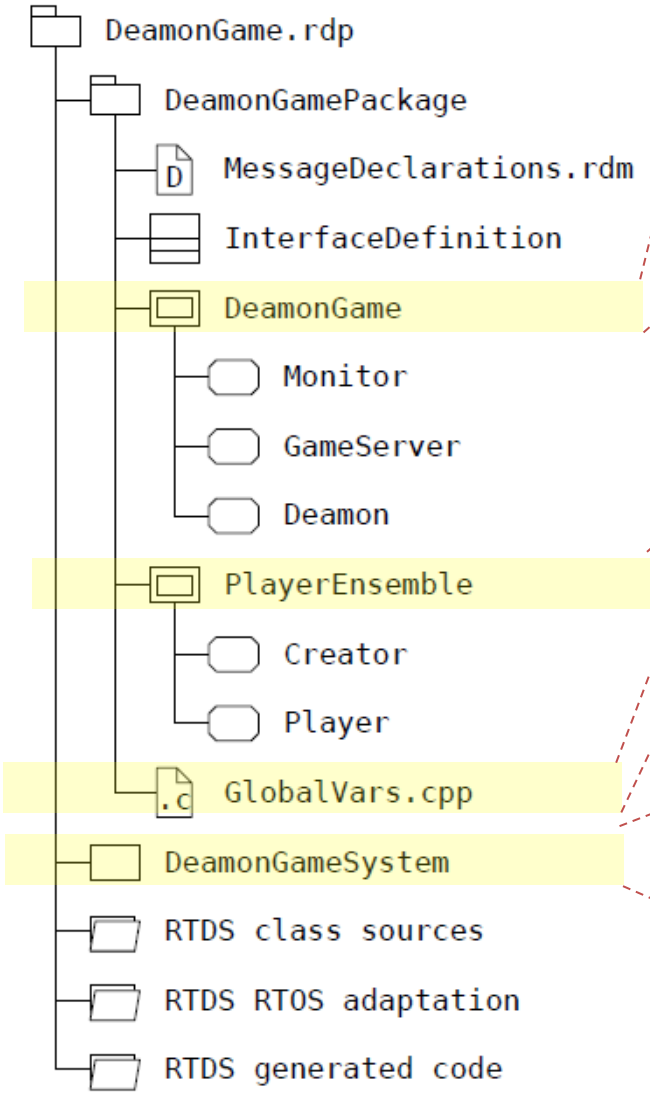
6. SDL

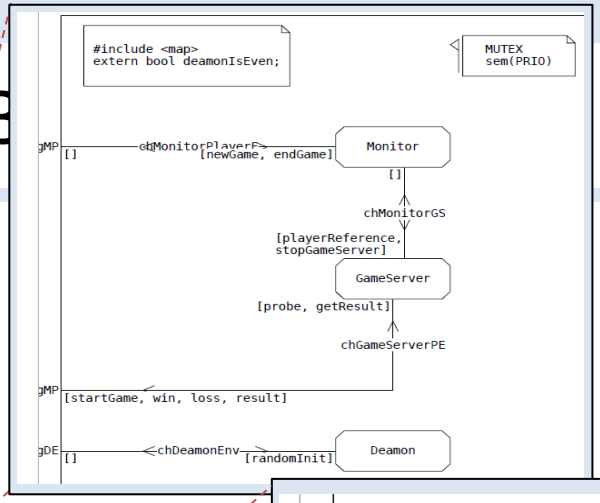
1. Grundphilosophie
2. ITU-Standard Z.100
3. Werkzeuge
4. SDL-Grundkonzepte
5. Musterbeispiel (in UML-Strukturen)
6. Umsetzung in SDL-RT

Prototypinstanz der Prozessmenge Player

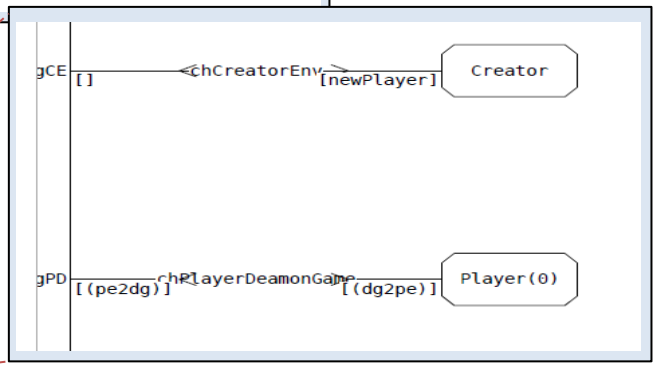


Player-Instanz und GameServer-Instanz kennen ihren jeweiligen Partner per Pid-Wert





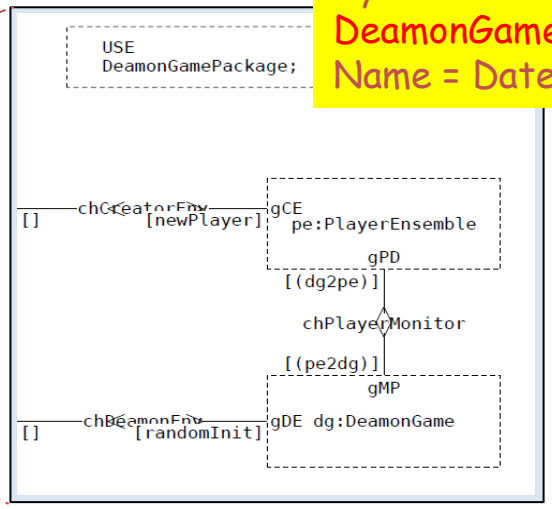
Blocktyp DeamonGame
Name = Dateiname



Blocktyp PlayerEnsemble
Name = Dateiname

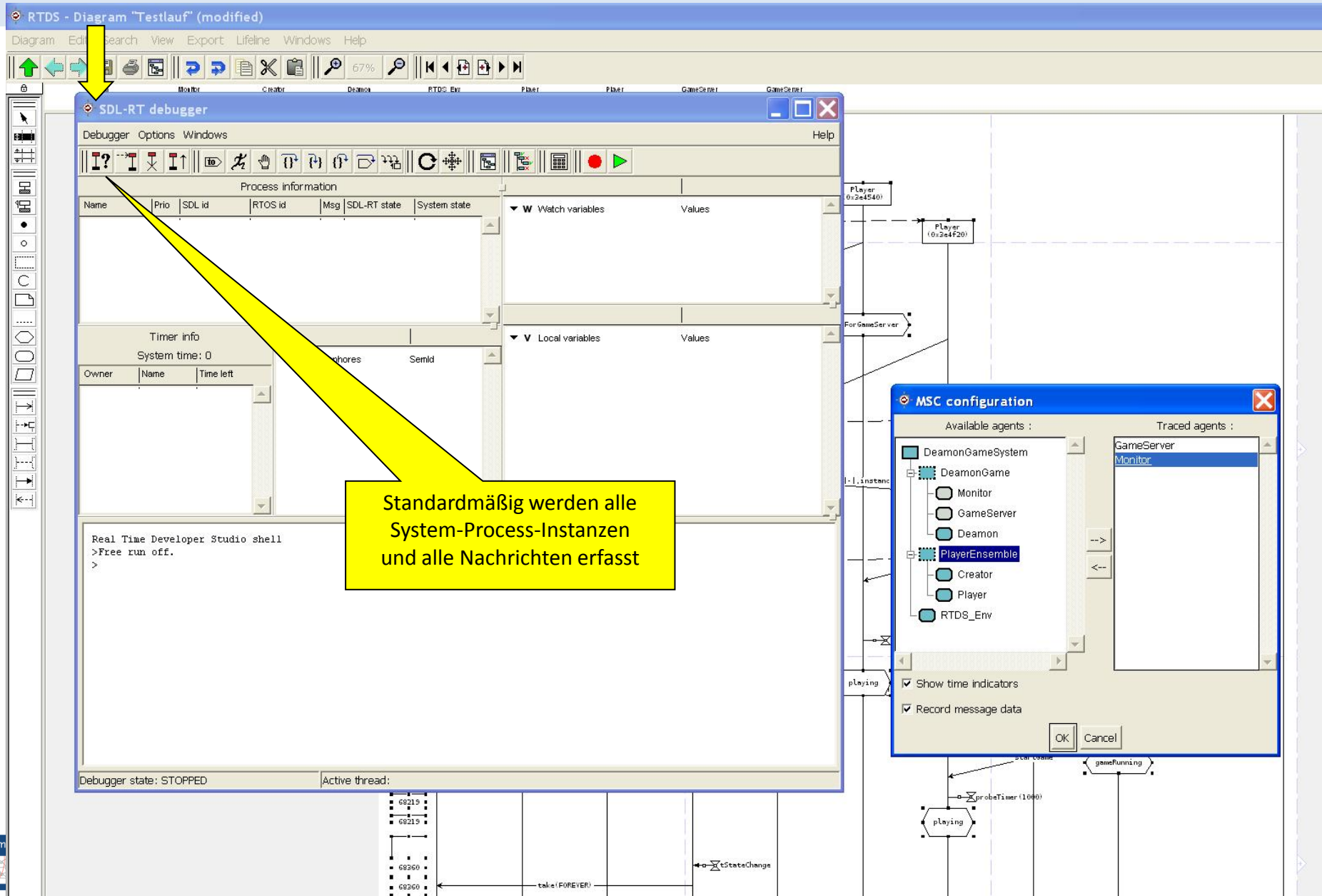
```
bool deamonIsEven = true;
```

C/C++ Erweiterung
globale Variable

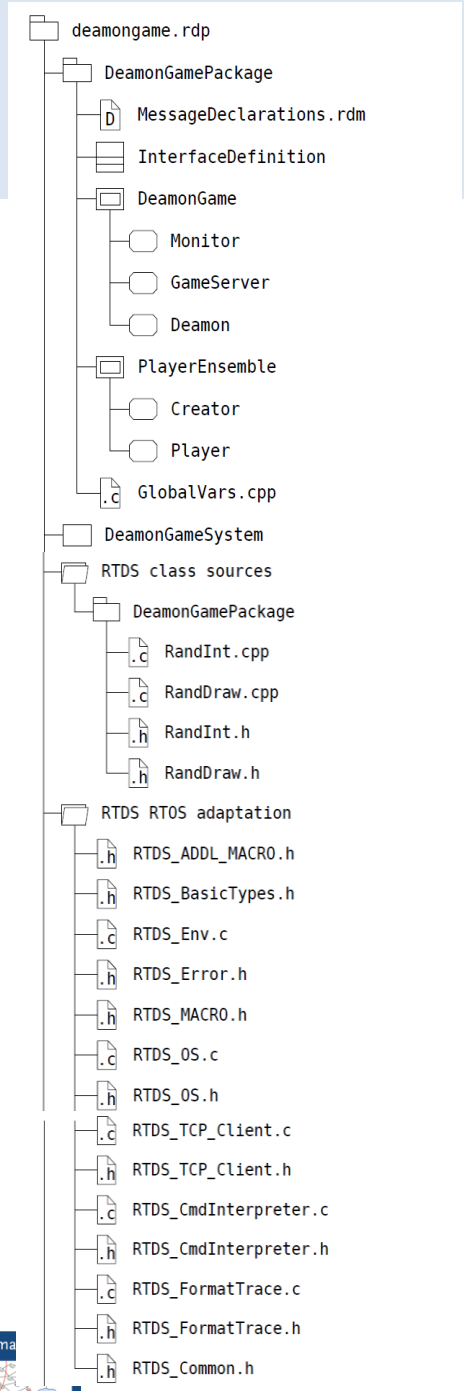


System
DeamonGameSystem
Name = Dateiname

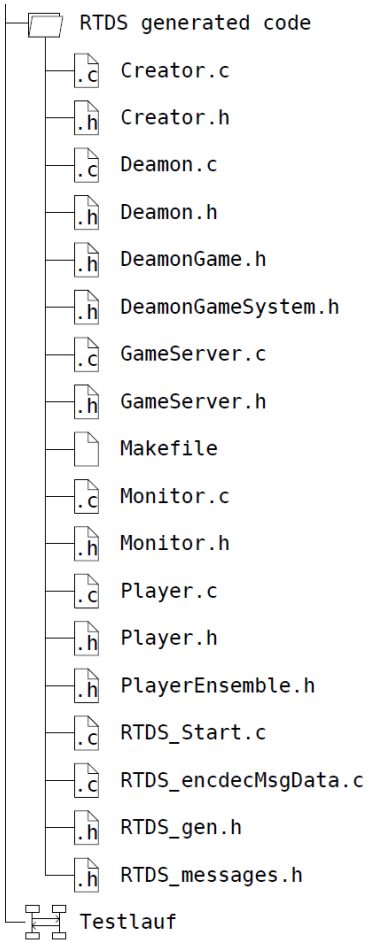
Konfiguration der SD-Ausgabe (MSC)



DeamonGame-Simulator



Quellen



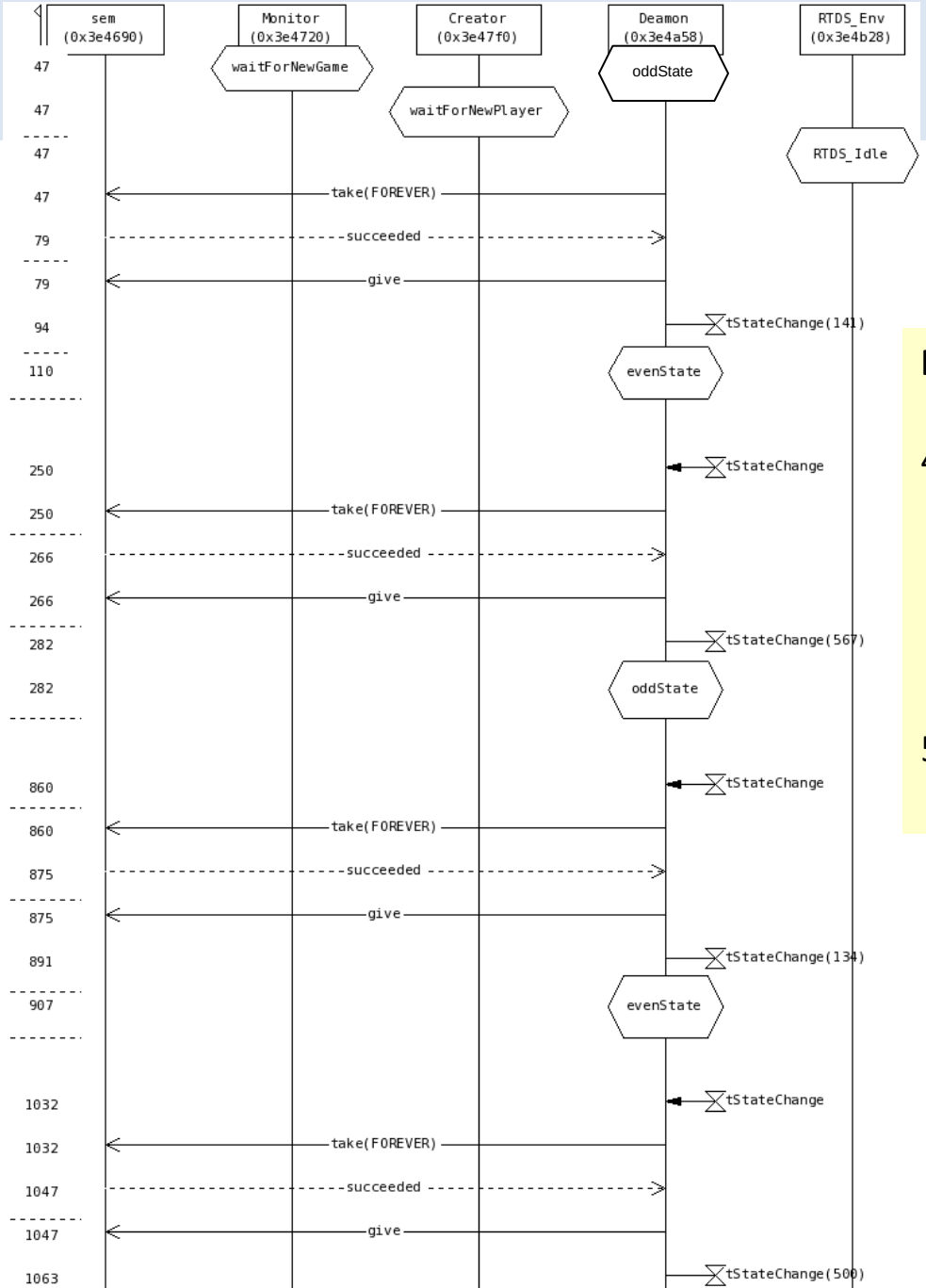
Run

1. Erzeugung des Simulators
   → neue Bedienoberfläche

2. Vorbereitung der Simulatorausgabe
   (SD-Ausgabe:Konfiguration)

3. Start des Simulators
   - alle statischen Prozessinstanzen werden erzeugt (Monitor,Creator,Deamon,Semaphore)
   - arbeiten Starttransitionen ab
   - verharren in Zuständen, wo sie auf Eingabenachrichten warten (bis auf Deamon)

4. Unterbrechung mit Pause



Noch keine Player- und GameServer-Instanzen

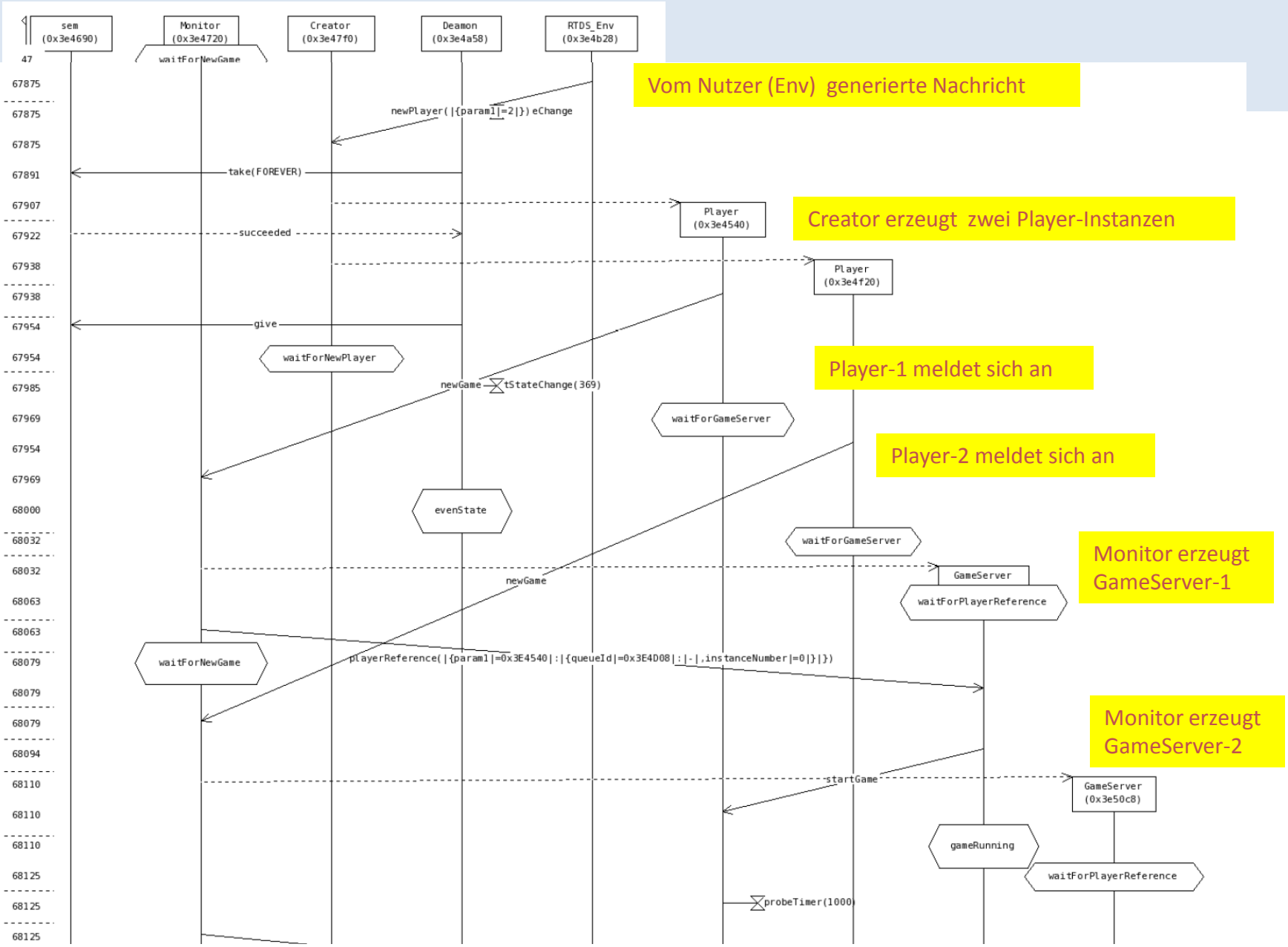
Monitor und Creator verharren

Deamon wechselt stochastisch seinen Grundzustand

Run (Fortsetzung)

4. Erzeugung einer Nachricht newPlayer(2) mit Creator als Empfänger-Prozessinstanz
   → 2 Spieler nehmen damit am Spiel teil

5. Unterbrechung mit Pause


sem (0x3e4690)
Monitor (0x3e4720)
Creator (0x3e47f0)
Deamon (0x3e4a58)
RTDS_Env (0x3e4b28)
47
waitForNewGame
67875
67875
newPlayer(|{param1|=2|})eChange
67875
67891
take(FOREVER)
67907
Player (0x3e4540)
67922
succeeded
67938
Player (0x3e4f20)
67938
67954
give
67954
waitForNewPlayer
67985
newGame
tStateChange(369)
67969
waitForGameServer
67954
67969
68000
evenState
68032
waitForGameServer
68032
newGame
GameServer
68063
waitForPlayerReference
68063
68079
waitForNewGame
playerReference(|{param1|=0x3E4540|:|{queueId|=0x3E4D08|:|-|,instanceNumber|=0|}|})
68079
68079
68094
68110
startGame
GameServer (0x3e50c8)
68110
68110
gameRunning
68125
waitForPlayerReference
68125
probeTimer(1000)
68125
Vom Nutzer (Env) generierte Nachricht
Creator erzeugt zwei Player-Instanzen
Player-1 meldet sich an
Player-2 meldet sich an
Monitor erzeugt GameServer-1
Monitor erzeugt GameServer-2

7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Dynamische Prozessgenerierung
6. Prozeduren
7. Lokale Objekte
8. Semaphore
9. Spezialisierung von Zustandsautomaten

Zustand und Zustandsübergang

Ann.: betrachten ein System zu einem beliebigen Zeitpunkt

- jede (existierende) Prozessinstanz
  - verharrt entweder in einem ihrer Grundzustände
    und wartet dabei auf einen Zustandsübergangsauslöser
    (z.B. auf die Ankunft eines bestimmten Signals)

  oder
  - führt einen Zustandsübergang aus (nicht unterbrechbar)

- im Zustandsgraphen eines Prozesses sind i. Allg.
  - pro Grundzustand alternative Varianten für die Auslösung eines Zustandsübergangs vorgesehen,
    wobei
  - die jeweils aktuelle Nachricht("älteste" im Puffer) in der Regel entscheidet, ob und welche der möglichen Alternativen zur Ausführung
    (d.h. auch ein weiteres Verharren im Zustand ist möglich)

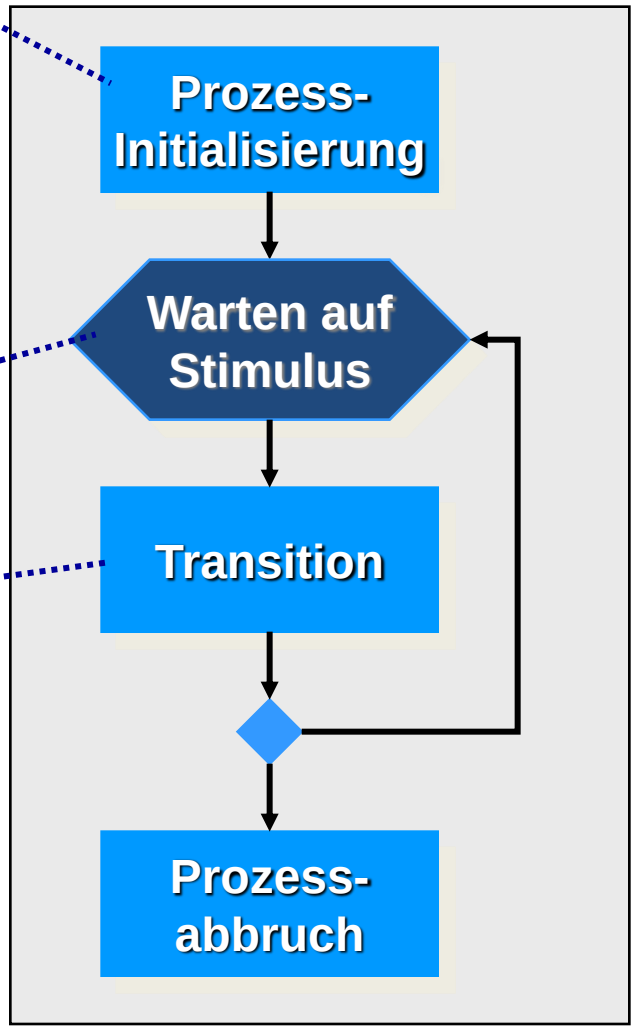
- die Auslösung eines Zustandsübergangs hat zur Folge:
  - die Konsumtion der Auslöser-Nachricht bei optionaler Übernahme der Parameter in lokale Variablen
  - Ausführung sequentieller Aktionen (Variablenänderungen, Nachrichtenausgaben, Prozessgenerierungen, Remote-Prozeduren-Rufe, Stop…)
  - Annahme eines neuen oder des gleichen Zustandes (vollzogener Zustandsübergang)

Prozess-Lebenszyklus (Schema)

mit Existenzbeginn
- Initialisierung lokaler Variablen
- Ausgabe von Nachrichten
- Aufruf von Prozeduren
- Erzeugung von Prozess-Instanzen
- Übergang in einen echten Zustand

Auswahl erfolgt nach
- Stimulus und
- Zustand

- Wertänderung lokaler Variablen,
- Ausgabe von Nachrichten,
- Aufruf von Prozeduren,
- Erzeugung von Prozess-Instanzen
- etc.



Vorschlag einer Namenskonvention

… für erstes Zeichen von 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)

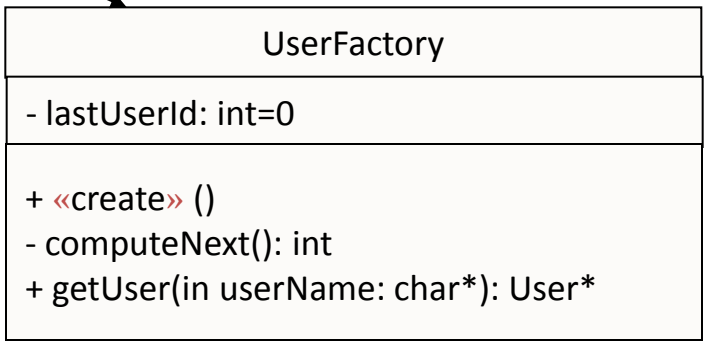
- Agent ist Element eines Systems
  - zwei Ausprägungen:
    - Block (ohne besondere Ausprägung im Zielcode),
    - Process (OS-Prozesse/Thread)
  - Mix von Prozessen und Blöcken auf einer Ebene des SDL-Programms ist erlaubt

- Prozessinstanzmengen-Kardinalität: Default (1,)

Klassenbeschreibung

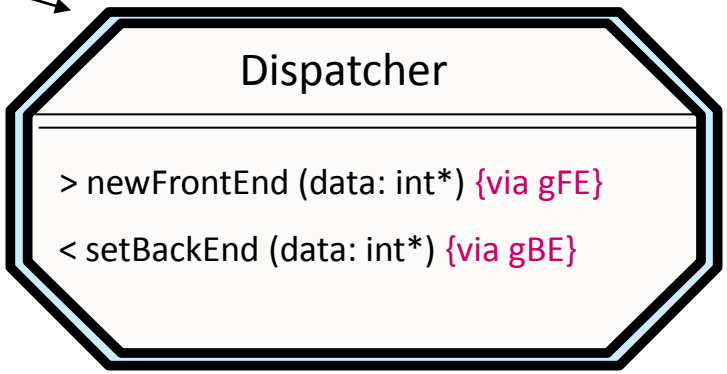
- 4 Formen
  - Interface-Klasse «interface»
  - System-Klasse «system»
  - Passive Klasse
  - Aktive Klasse : für Block-Typen, Process-Typen

In UML-Syntax



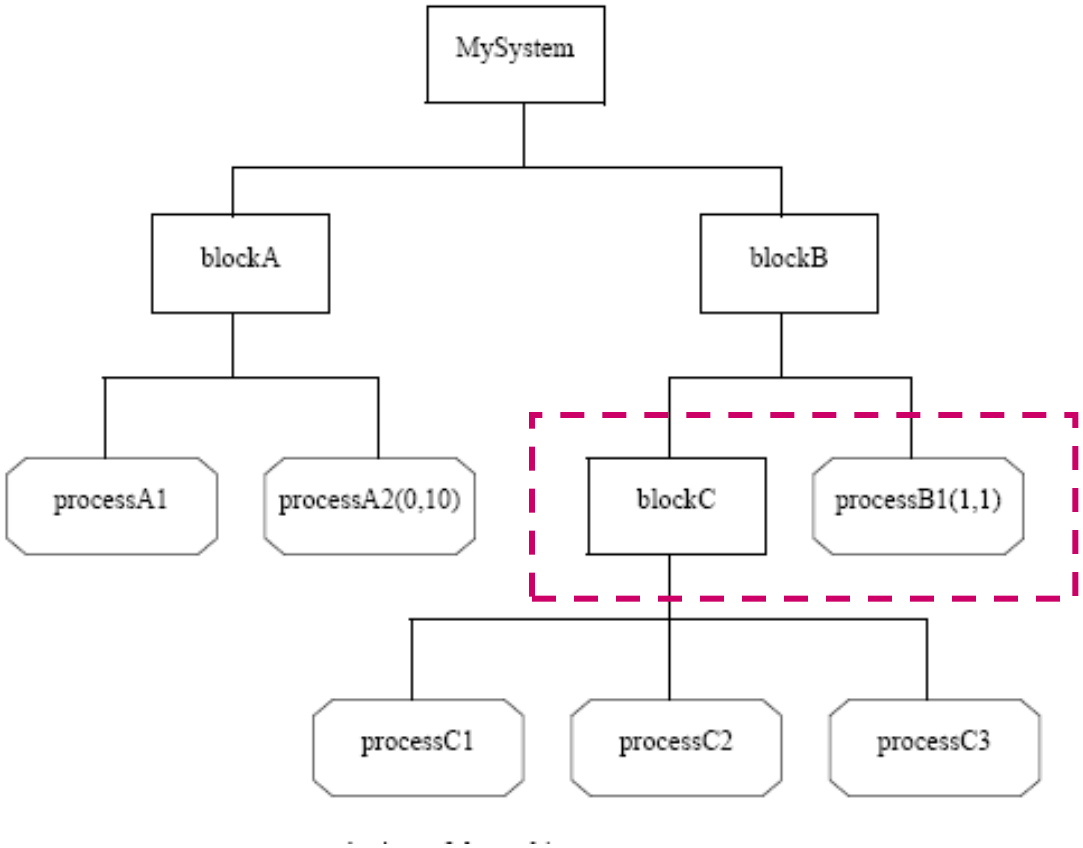
mehrere Konstruktoren (kein Return) möglich
nur ein Destruktor (<<delete>>)
In-, Out-, inOut-Parameter

SDL-angepasste-Syntax

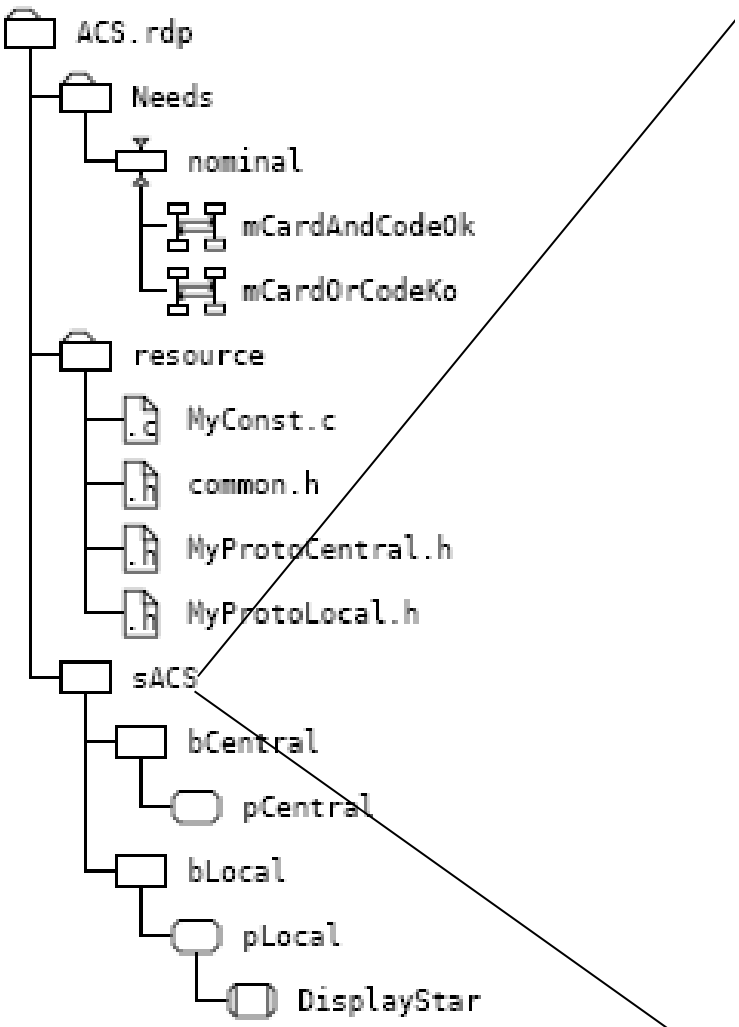


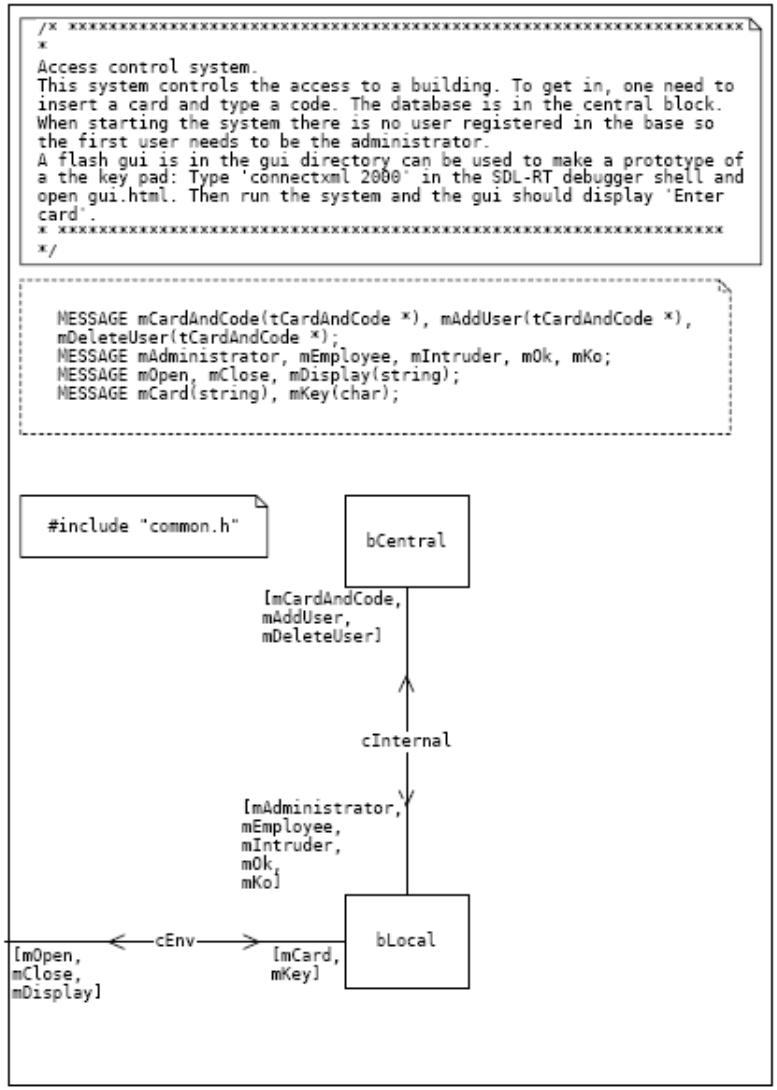
keine Attribute
Operations → PseudoNachrichten (nurEinParameter)
Property für Gate-Angabe

Beispiel-1: Systemarchitektur

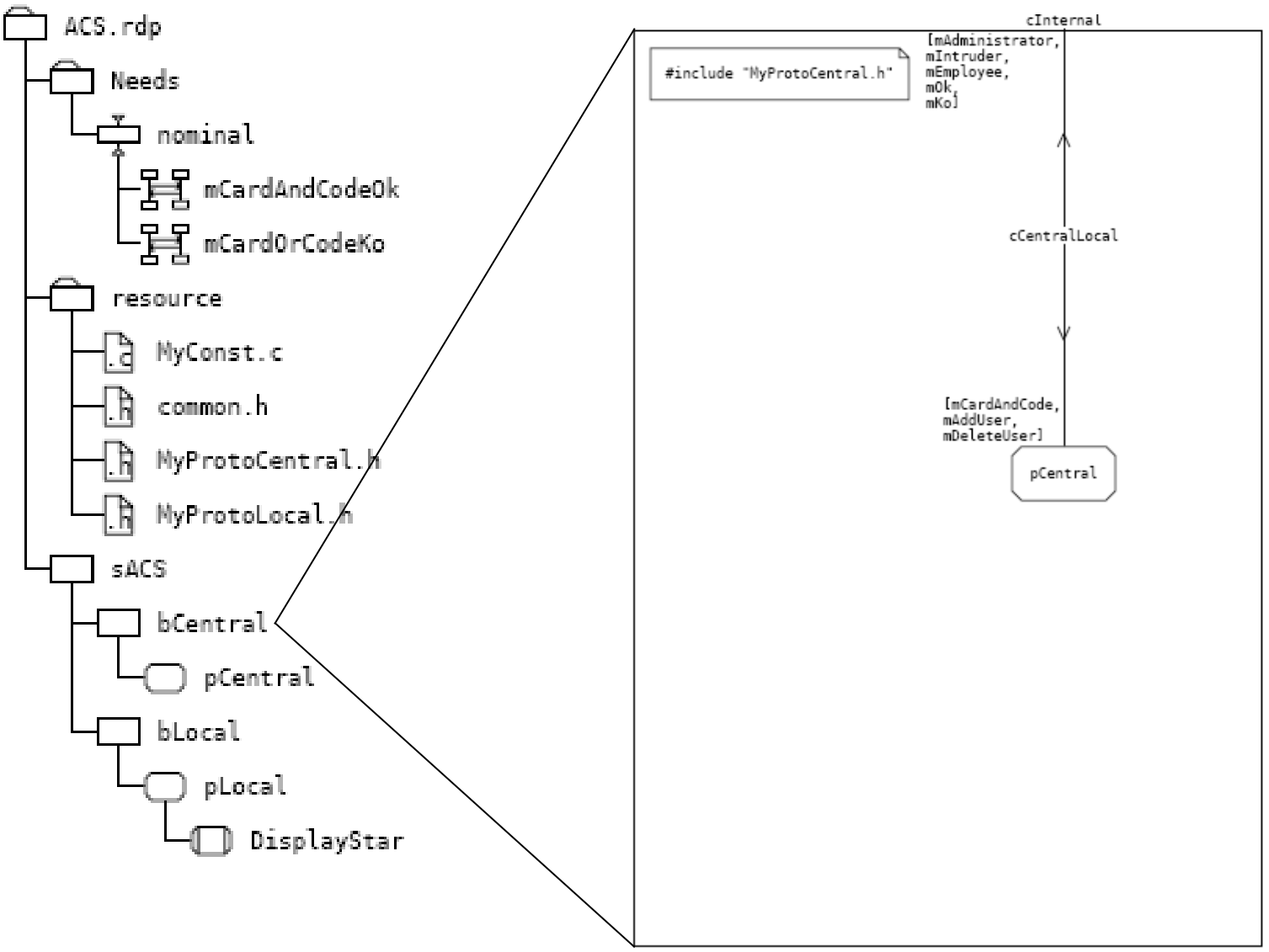


Beispiel-2: System- im SDL-RT-Projektbaum

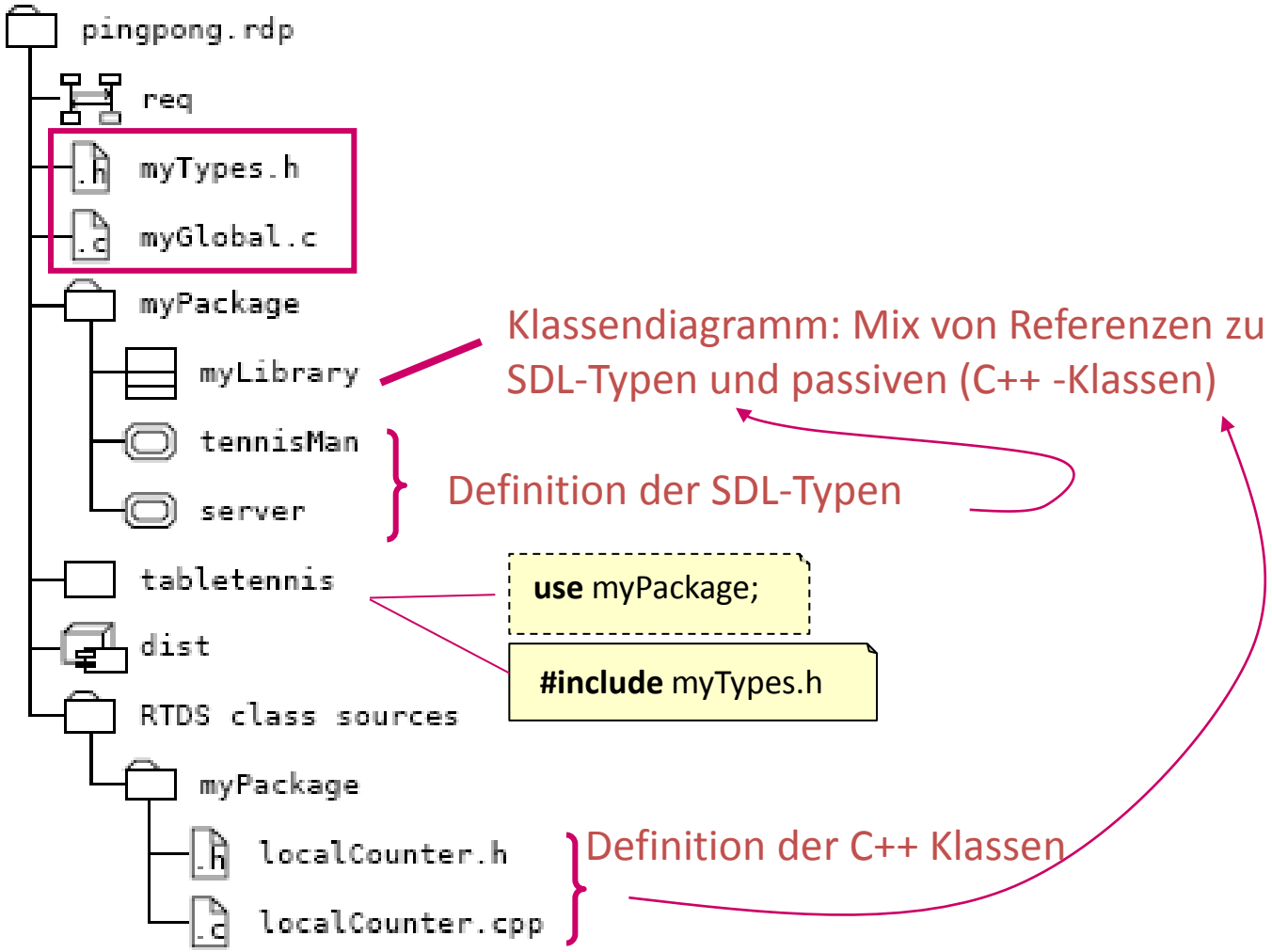




Beispiel-2: System- im SDL-RT-Projektbaum

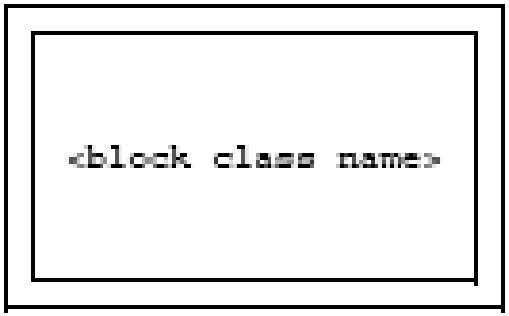


Beispiel-3: System im Projektbaum

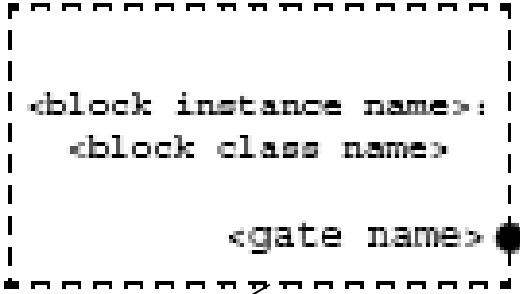


SDL-RT: Block-Typen (Block-Klassen)

Einschränkungen gegenüber Z.100



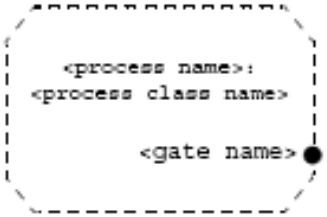
keine Vererbung
keine Virtualität
keine Kontextparameter



Instanzmengen sind nicht zulässig

Gates ohne Atleast-Typ-Angabe

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



Einschränkungen für Vererbung gegenüber Z.100
- Spezialisierung von Transitionen wie Z.100,
- aber keine Kontextparameter

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

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

Präfix: ABSTRACT

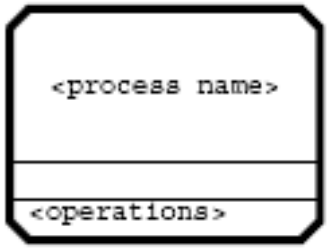
Präfix: VIRTUAL
Wirkung noch nicht überprüft

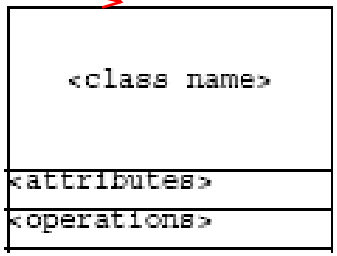
Präfix: VIRTUAL

UML-Klassendiagramme

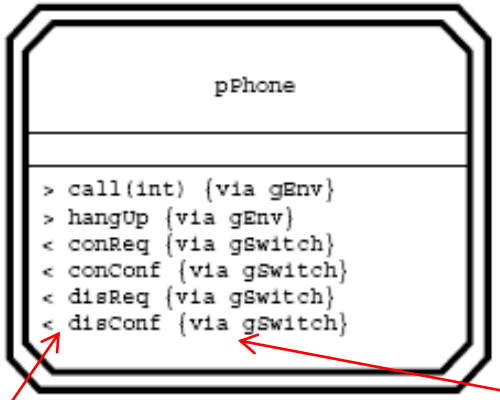
- aktive Klassen, passive Klassen

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





aktive Klassen haben keine Attribut-Compartments (Attribute werden woanders definiert ?)
Operationen sind ein- und ausgehende Nachrichten (asynchron) mit Gate-Angabe
leider: keine Angaben von Nachrichtenlisten möglich

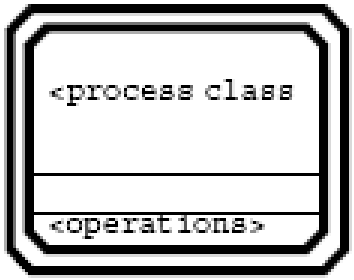


kein Leerzeichen

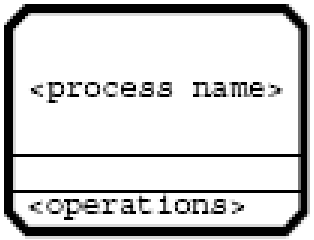
Doppelpunkt statt Leerzeichen

SDL-RT-Doko fehlerhaft !

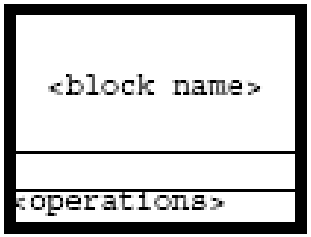
Vordefinierte graphische Symbole von stereotypisierter Klassen



Class stereotyped as a class of process

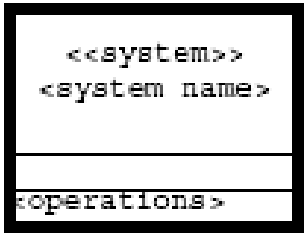


Class stereotyped as a process



Class stereotyped as a block

Class stereotyped as a class of block



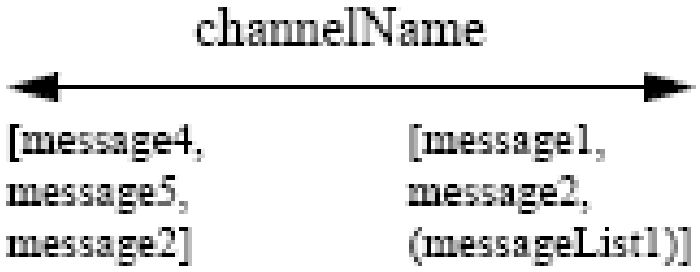
Class stereotyped as a system

Kommunikation

- Channel:= Zusammenfassung von Channel und Signalroute

Z.100: zeitbehaftet

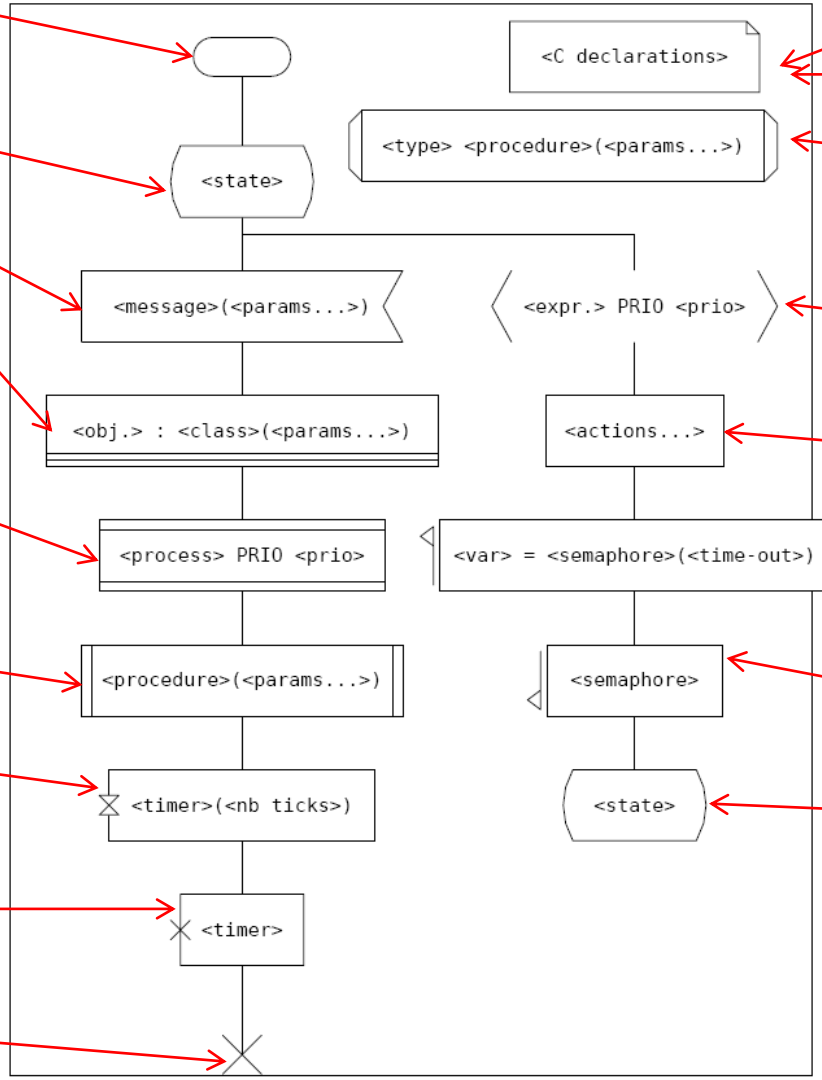
Z.100: zeitlos



Datentypen

- zwei Möglichkeiten für Nutzung von C-Datentypen
  - externes Header-File (Projekt-Manager)
    #include dann auf Block- oder Process-Level
  - SDL-Text-Box
    - Block-Level → wird separates C-Header-File
      (globale Typen, globale Variablen)
    - Process-Level → Process-lokal (Variablen)
    - Procedure-Level → Prozedur-lokal (Variablen)

Process-lokale Deklarationen



Startzustand

Zustand

Trigger: Signal

Objekterzeugung

Prozessinstanz-erzeugung

Prozedurruf

Timer-Start

Timer-Stop

Process-Stop

gestrichelt: SDL-Deklaration

voll: C-Deklaration

Prozedur-Deklaration

Trigger: Continuous-Signal

Task: Aktionssequenz

Semaphore-Ein mit Rückkehrcode

Semaphore-Aus

neuer Zustand

SDL-Signale und Signallisten

Änderung gegenüber Z.100:

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 Timer-Deklaration