Projekt Erdbebenfrühwarnung
im WiSe 2010/11

Entwicklung verteilter eingebetteter Systeme

Prof. Dr. Joachim Fischer
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Frank Kühnlenz

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

5. SDL als UML-Profil

1. ITU-Standard Z.100
2. UML und SDL-Zustandsmaschinen im Vergleich
3. Werkzeuge
4. SDL-Grundkonzepte
5. Musterbeispiel (in UML-Strukturen)
6. Struktur- und Verhaltensbeschreibung in SDL

SDL-Basis (Wdh.)

Agent= aktive Klasse mit Process-Beschreibung
- Classifier-Eigenschaft

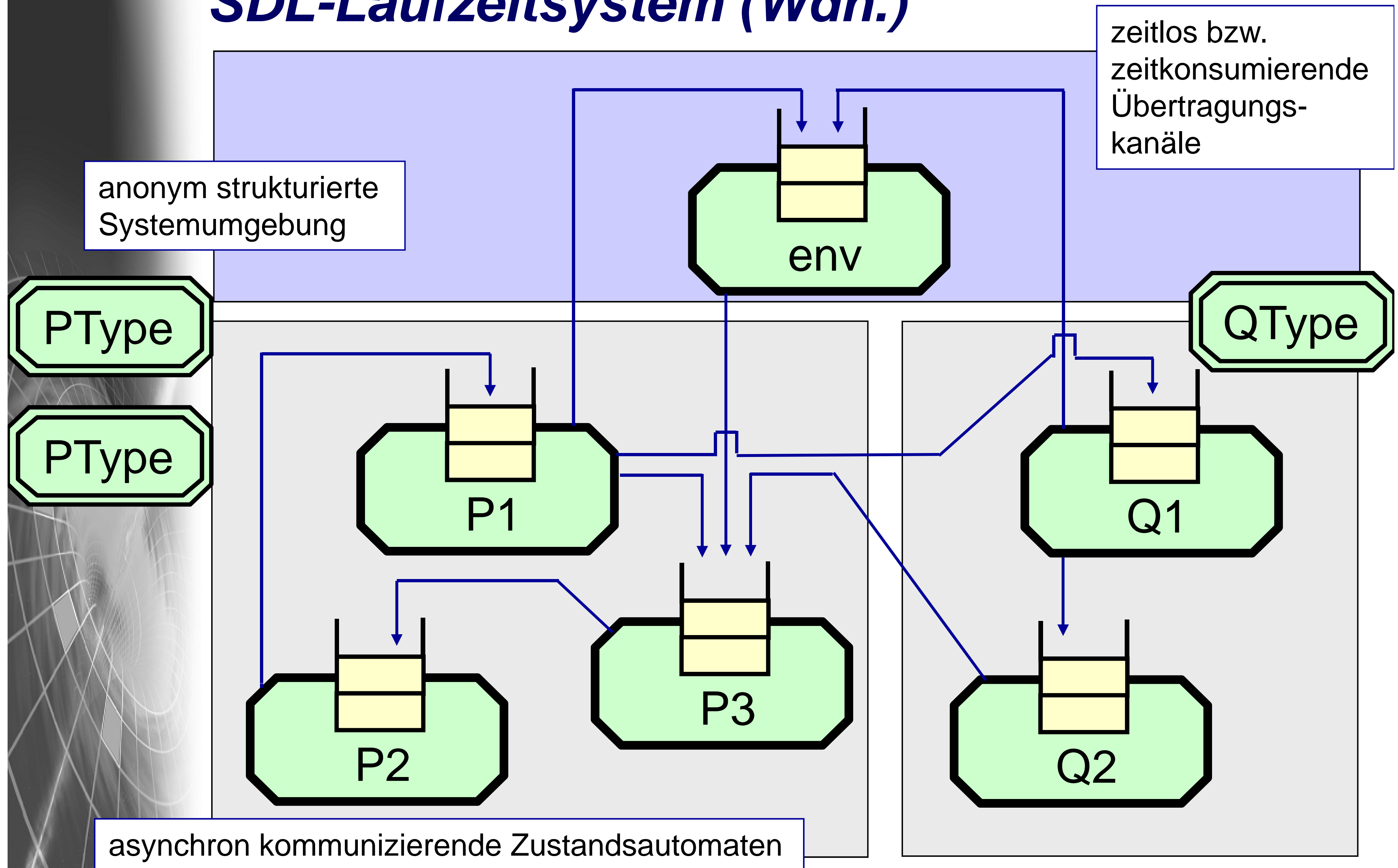
- ein SDL-System besteht zur Laufzeit aus einer Menge von kommunizierenden Zustandsmaschinen
  definiert durch je einen Repräsentanten der festgelegten Process (Agenten)- Instanzmengen

  die in ihrer Wechselwirkung untereinander und mit der Umgebung des Systems das Verhalten erbringen

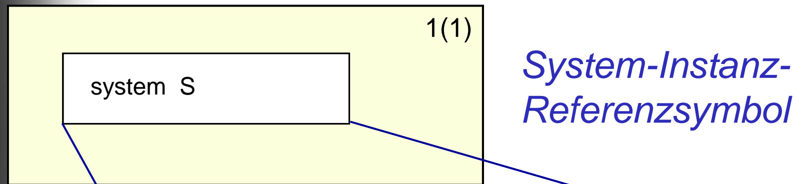
- die Wechselwirkungen werden über einen **asynchronen** Nachrichtenaustausch realisiert
  - Sender und Empfänger sind damit entkoppelt

- jede Prozessinstanz besitzt (genau) einen Empfangspuffer zur Speicherung ankommender Nachrichten
  dieser ist idealerweise a priori unbeschränkt
  - keine Blockierung des Senders aufgrund eines vollen Puffers

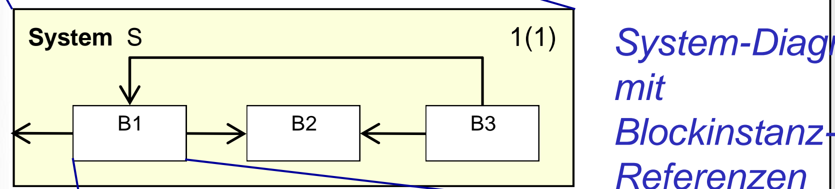
SDL-Laufzeitsystem (Wdh.)



SDL/GR-Diagrammkomposition (Wdh.)

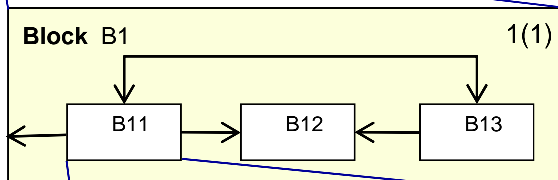


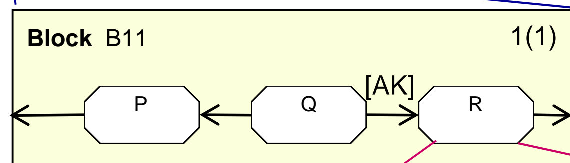
System-Instanz-Referenzsymbol



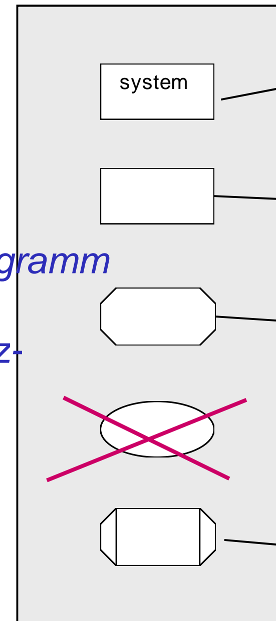
System-Diagramm mit Blockinstanz-Referenzen

Kanäle: Nachrichten-austausch





Referenzsymbole



System-Instanz

Block-Instanz(menge)

Prozess-Instanz(menge)
(Default-Kardinailität=1)

Service-Instanz

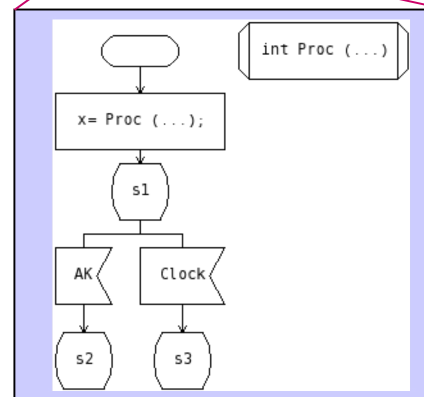
nicht in SDL-RT

Prozedur

SDL-Editor-Anforderung:
Diagramm- und Seitennavigation

Prozess als Zustandsautomat

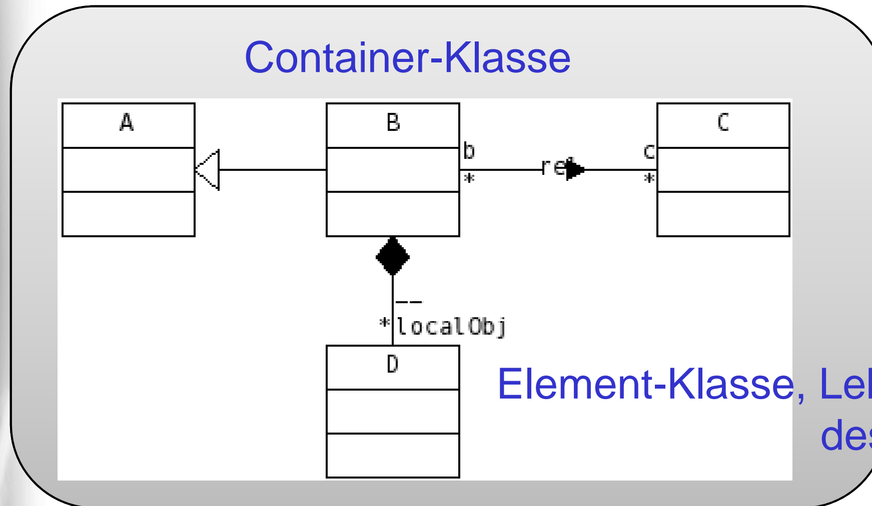
- *kommunizieren per asynchronen Nachrichtenaustausch (impliziter Empfangspuffer)*
- *können andere Prozesse erzeugen*
- *Verhalten: endlich,unendlich*



- *lokale Variablen (auch Assoziationsenden)*
- *lokale Datentypen*
- *lokale Prozeduren/Funktionen*
- *lokale Zustände*
- *Zustandsübergänge als Ereignistrigger*

Weitere SDL-RT-Konzepte (Referenzsymbole)

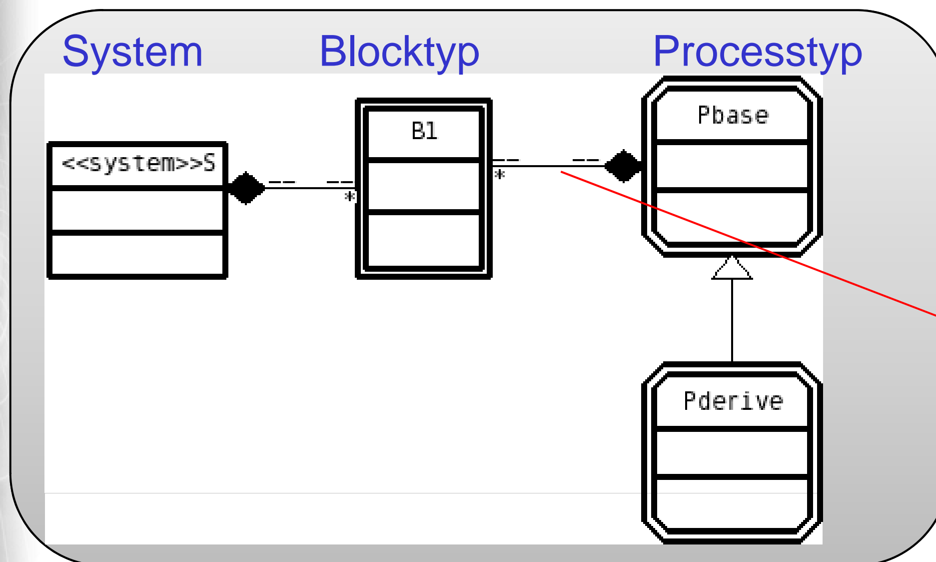
Container-Klasse



Klassendiagramm passiver Klassen

Element-Klasse, Lebenslauf der Elementobjekte ist an Existenz des Objektes der Container-Klasse gebunden

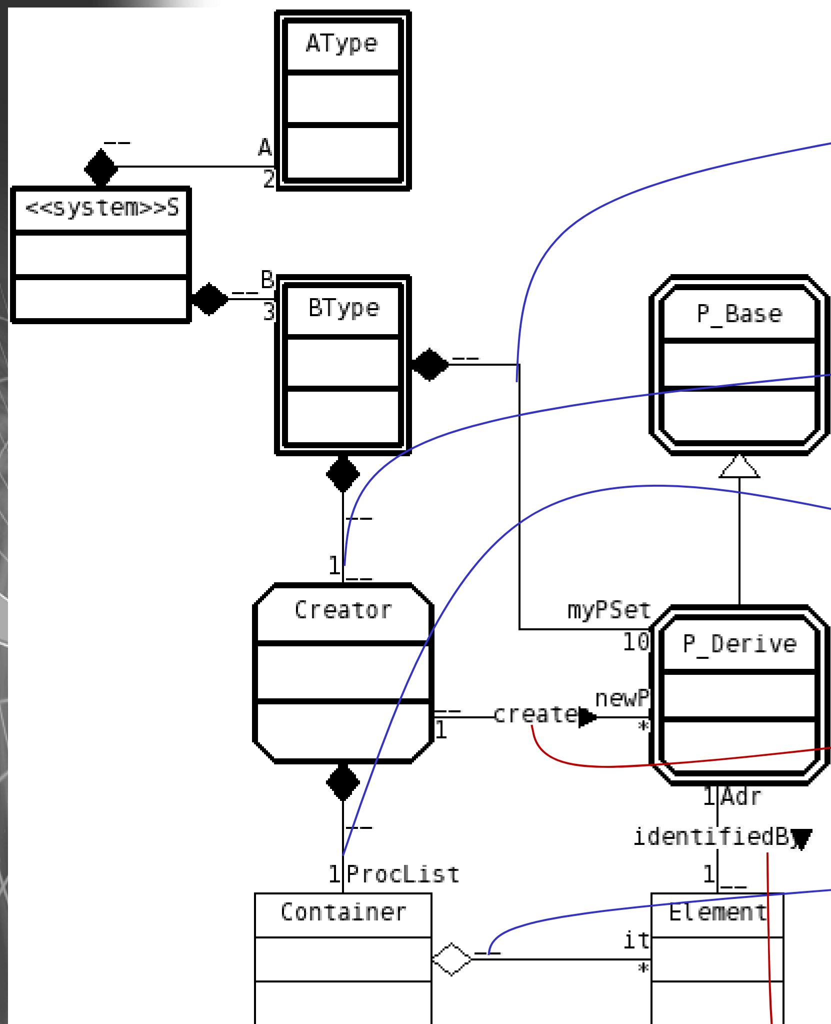
System Blocktyp Processtyp



Klassendiagramm aktiver Klassen

FEHLER:
unzulässige Typen für die Kompositionsbeziehung aus SDL-Sicht

Anwendungsbeispiel



Komposition:
Instanz von P_Base (oder Ableitung)
gehört zu einer Instanz von BType
(zu deren lokalen Prozessmenge myPSet)

Komposition:
BType-Instanz besitzt Creator (als Unikat)

Komposition:
Creator besitzt Container-Objekt ProcList (als Unikat)

Assoziation: creates
Creator erzeugt Instanzen von P_Derive
(beliebig viele, die in myPSet erfasst werden, deren Kardinalität auf 10 beschränkt ist)

Aggregation:
Container-Objekt verwaltet Element-Objekte (0,*)
erreicht die Elemente mit it (Iterator)

Assoziation: identifiedBy
über ProcList→it→Adr gelangt Creator an die Referenz (adr) aller P_Derive-Instanzen von ProcList
über newP gelangt Creator an die Referenz des zuletzt generierten Prozesses

Prozessidentifikation

- jede Prozessmengen-Instanz hat eine systemweit-eindeutige **Identifikation** vom Typ PId (PId = process identification)

- Vergabe eines PId-Wertes erfolgt implizit per Instanzgenerierung

- keine Kompatibilität von PId zu anderen Typen

- PId-Werte können nur
  - verglichen werden (=, =/) und
  - in Variablen vom Typ PId zur Adressierung von Signalen gespeichert werden

- einziges Literal (explizite Notation für einen Wert): null

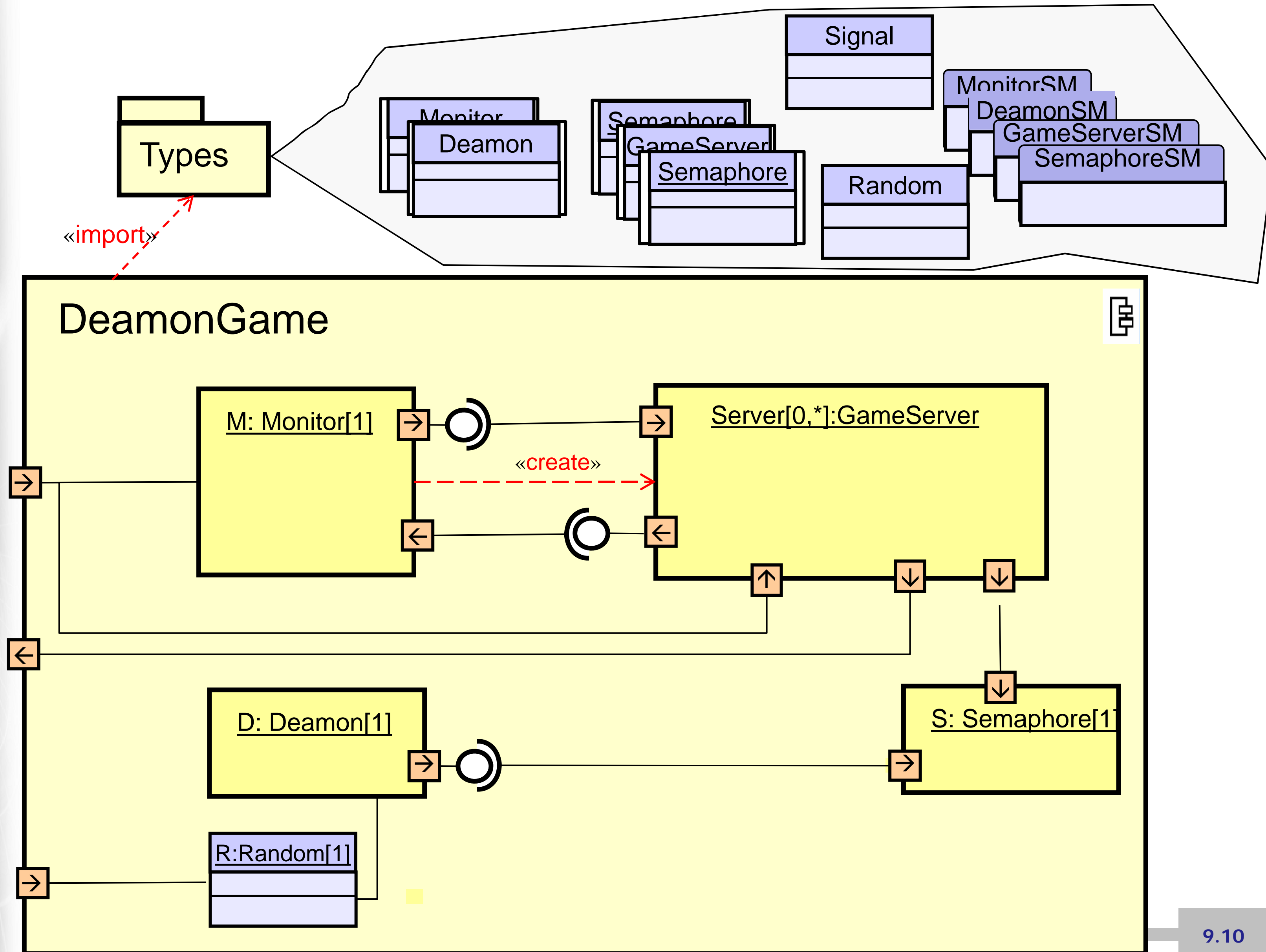
- eine Prozessmengen-Instanz existiert mit Systemstart oder wird zur Laufzeit (durch einen anderen Prozess) explizit generiert

- die Lebensdauer einer Prozessinstanz bestimmt nur die Instanz selbst

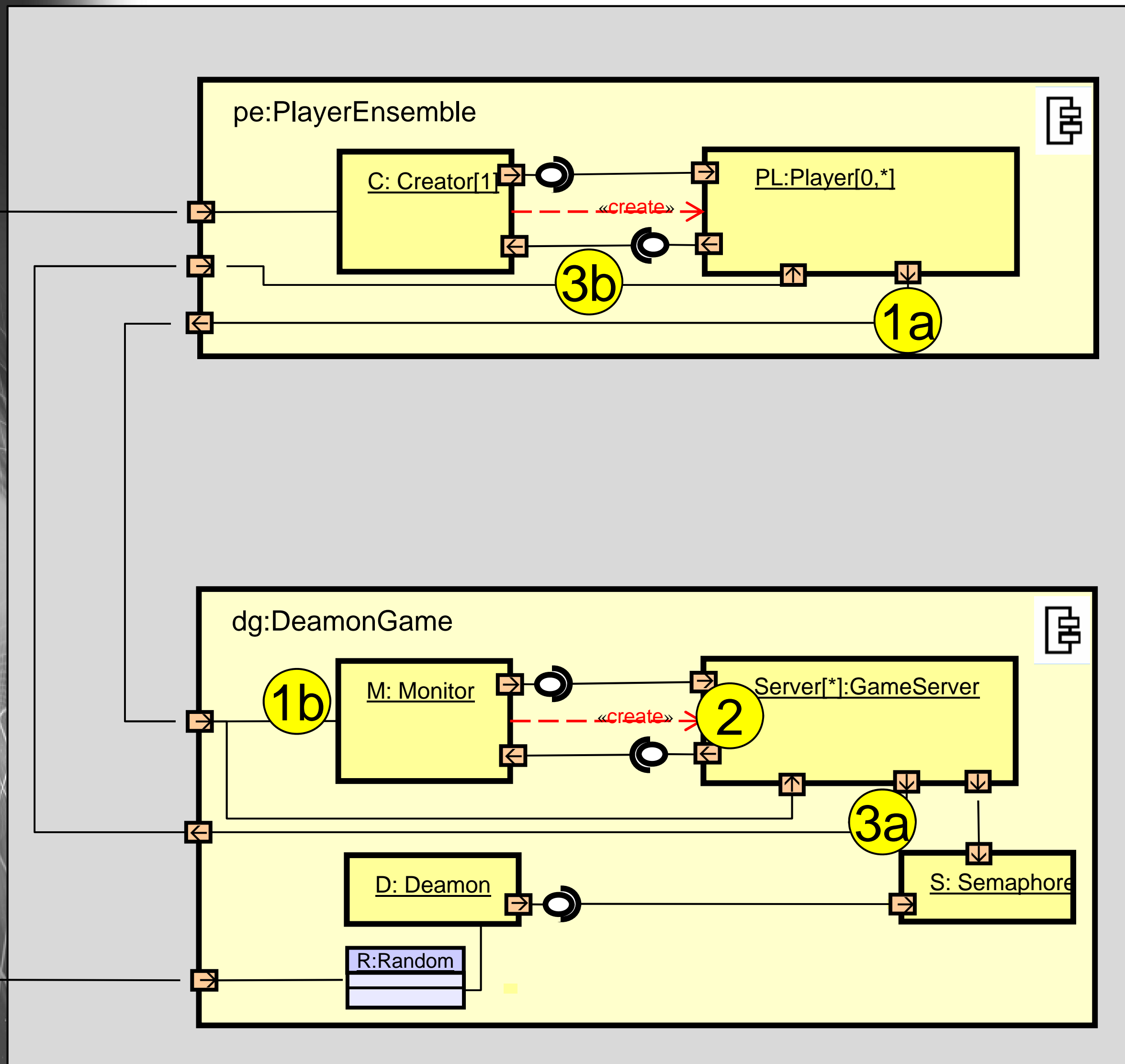
5. SDL als UML-Profil

1. ITU-Standard Z.100
2. Werkzeuge
3. SDL-Grundkonzepte
4. Musterbeispiel (in UML-Strukturen)
5. Struktur- und Verhaltensbeschreibung in SDL

UML-Systemtypdefinition: DeamonGame

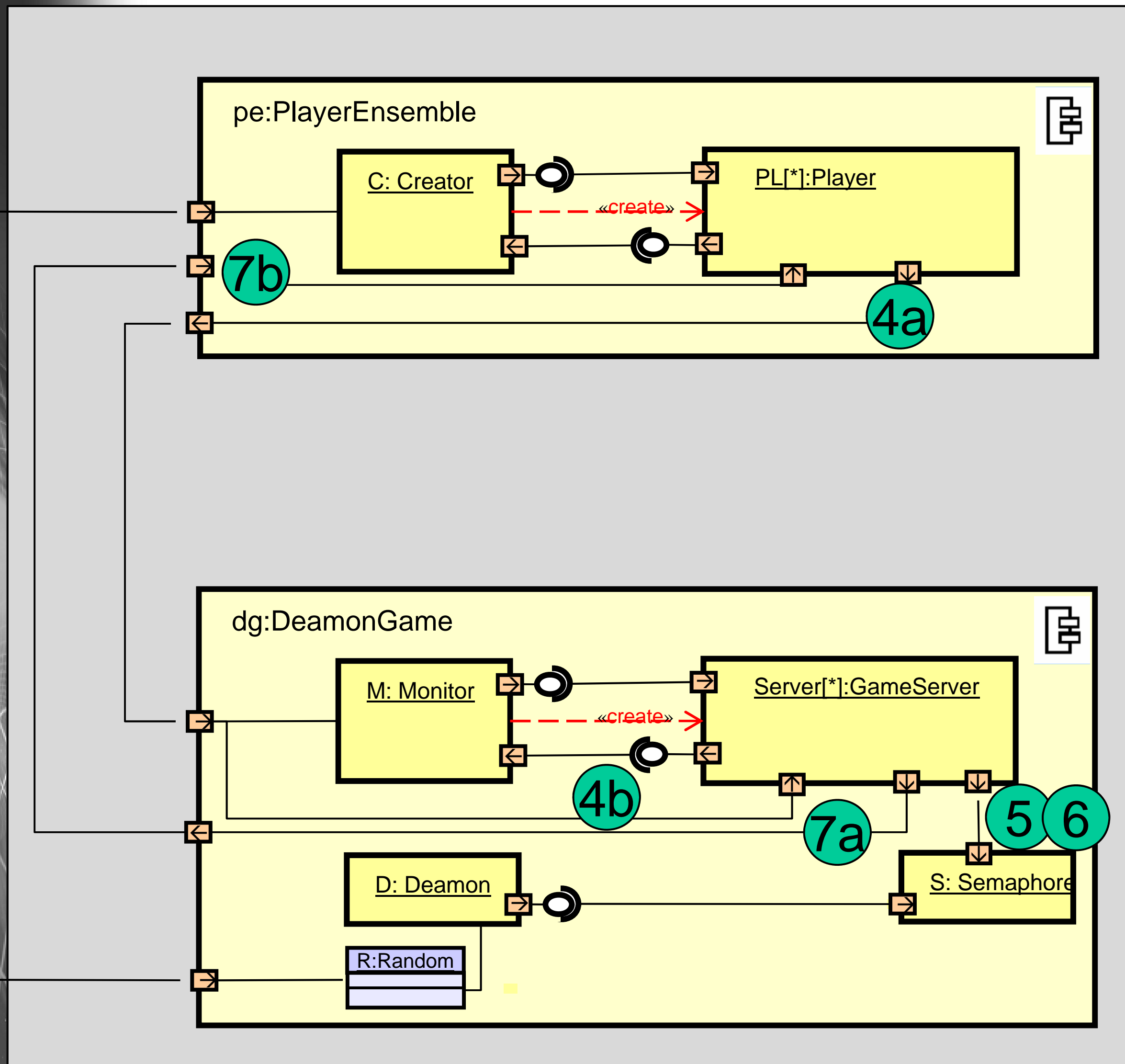


Erweitertes System (Nutzeranmeldung)



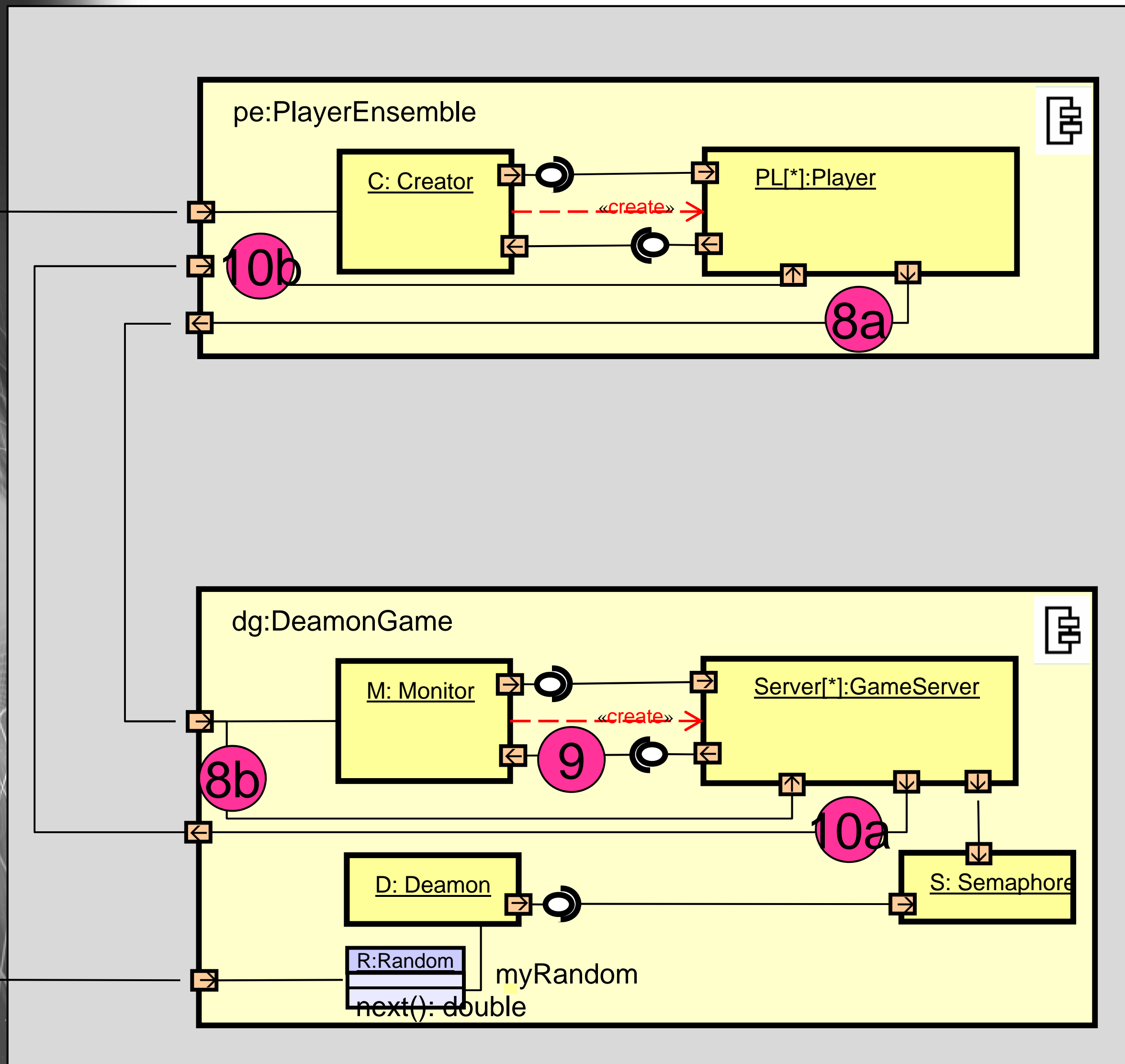
1. Anmeldung des Nutzers
2. Bereitstellung von Ressourcen zum Dienstzugang
3. Zugangsdaten an Nutzer
4. Dienstaktivierung
5. Dienstrealisierung (Start)
6. Dienstrealisierung (Ende)
7. Dienstfinalisierung
8. Abmeldung des Nutzers
9. Ressourcenfreigabe
10. Bestätigung der Abmeldung

Erweitertes System (Dienstnutzung)



1. Anmeldung des Nutzers
2. Bereitstellung von Ressourcen zum Dienstzugang
3. Zugangsdaten an Nutzer
4. Dienstaktivierung
5. Dienstrealisierung (Start)
6. Dienstrealisierung (Ende)
7. Dienstfinalisierung
8. Abmeldung des Nutzers
9. Ressourcenfreigabe
10. Bestätigung der Abmeldung

Erweitertes System (Nutzerabmeldung)



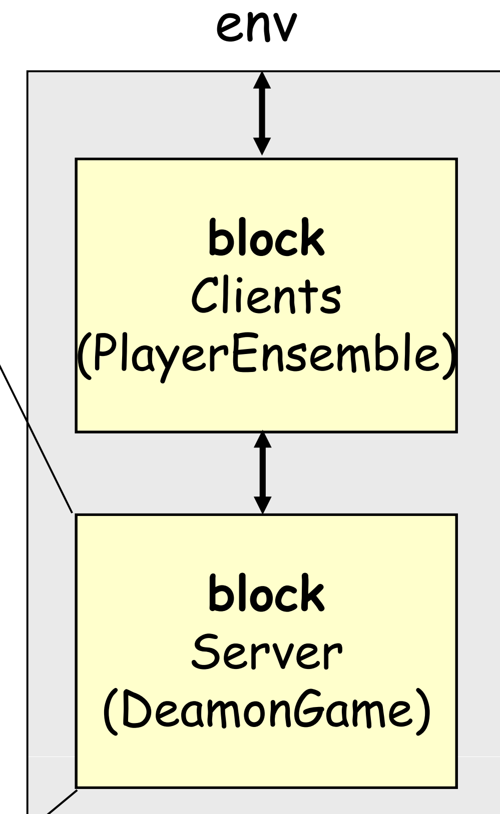
1. Anmeldung des Nutzers
2. Bereitstellung von Ressourcen zum Dienstzugang
3. Zugangsdaten an Nutzer
4. Dienstaktivierung
5. Dienstrealisierung (Start)
6. Dienstrealisierung (Ende)
7. Dienstfinalisierung
8. Abmeldung des Nutzers
9. Ressourcenfreigabe
10. Bestätigung der Abmeldung

Informale Festlegung des Dienstes (1)

Dienst: Computerspiel als reaktive Komponente

- unterstützt unbegrenzte (unbekannte) Anzahl von Spielern
- Spieler treten gegen den Computer an
  - nach Registrierung
  - bis zur Abmeldung
- eigentliche Spiel ist trivial
- Unterstützung
  - mehrerer,
  - von einander unabhängiger Spiele,

  wobei ein Spieler zu einem Zeitpunkt nur bei höchstens einem Spiel angemeldet sein kann und mitwirken darf

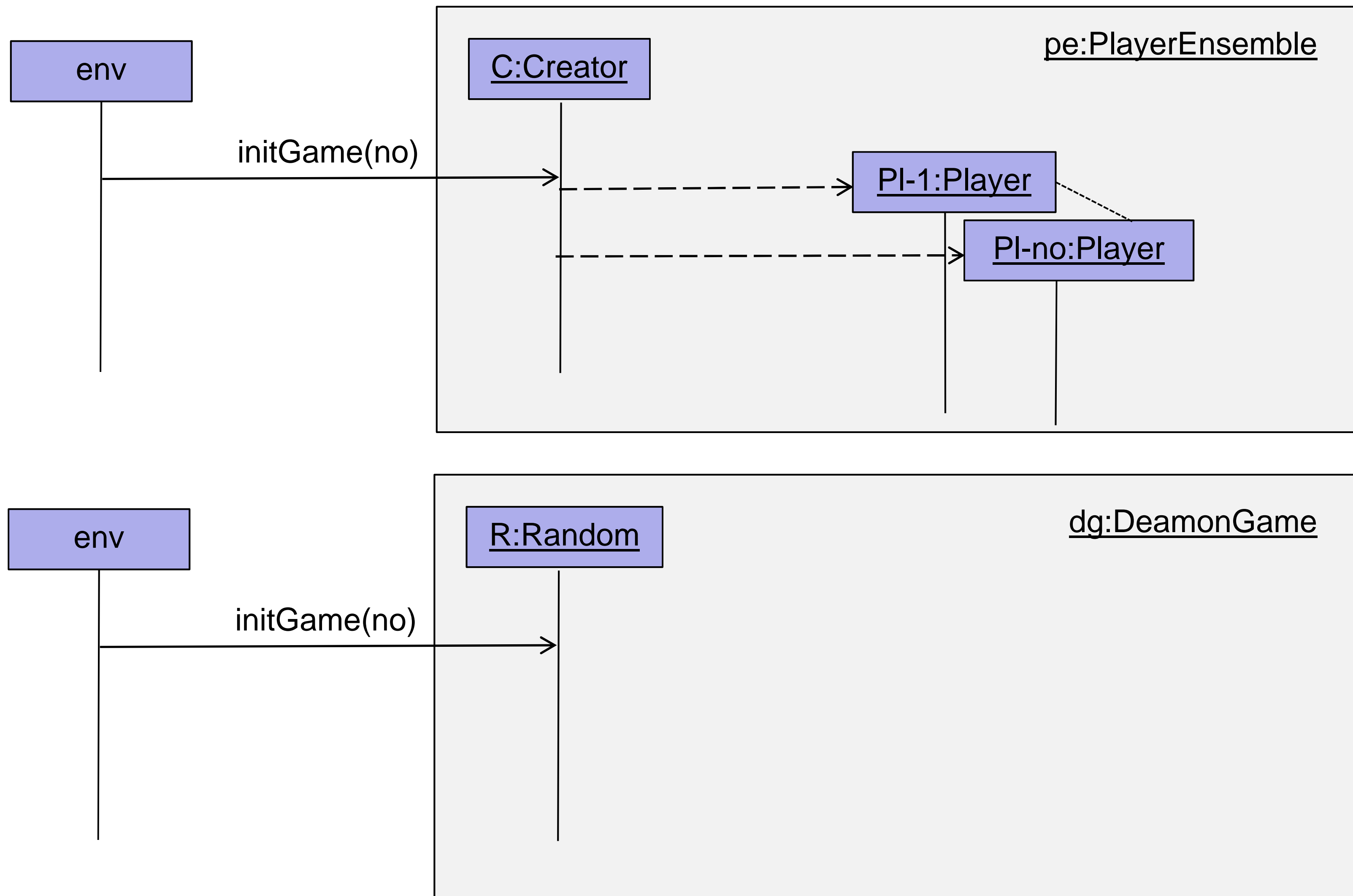


Informale Festlegung des Dienstes (2)

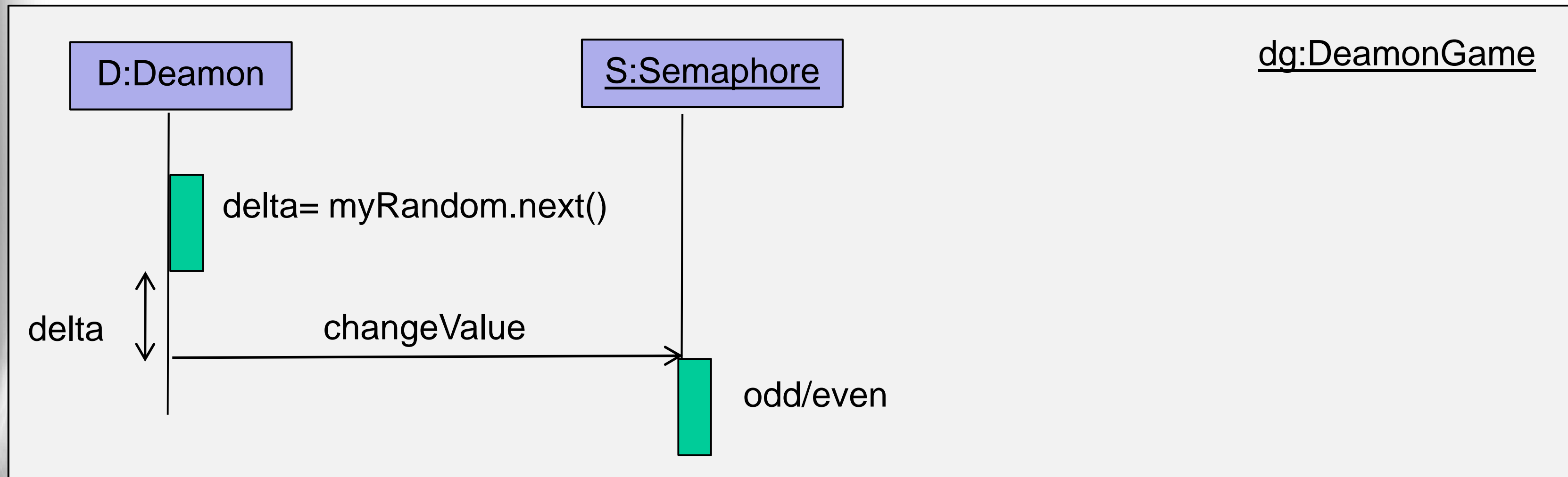
Spielregeln von DeamonGame

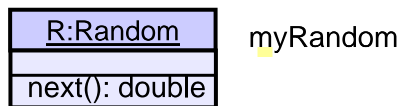
- der Wert **einer** nicht sichtbaren Variable ändert sich nichtdeterministisch: even ←→odd
- zu diskreten Zeitpunkten rät ein Spieler (als Client), ob der Wert ungerade (odd) ist
  - ist das der Fall, **gewinnt** er einen Punkt
  - wenn nicht, **verliert** er einen Punkt
- zu jedem Zeitpunkt kann von den Spielern ihr jeweiliger Punktestand abgefragt werden

Schnittstellenprotokolle: Spielinitialisierung

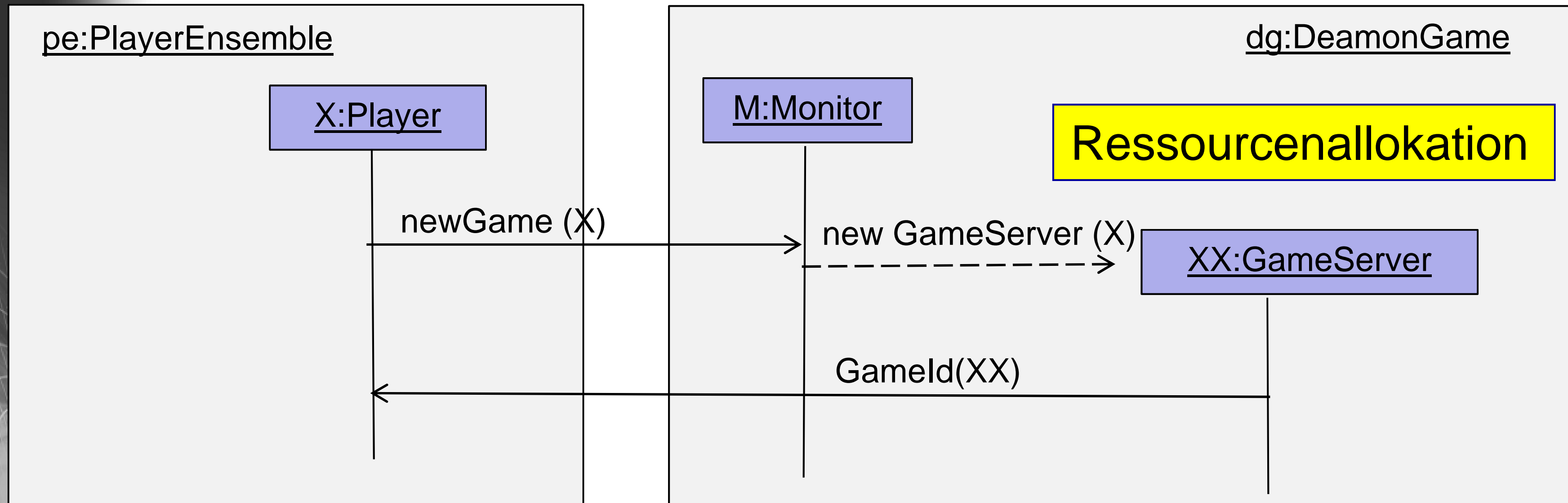


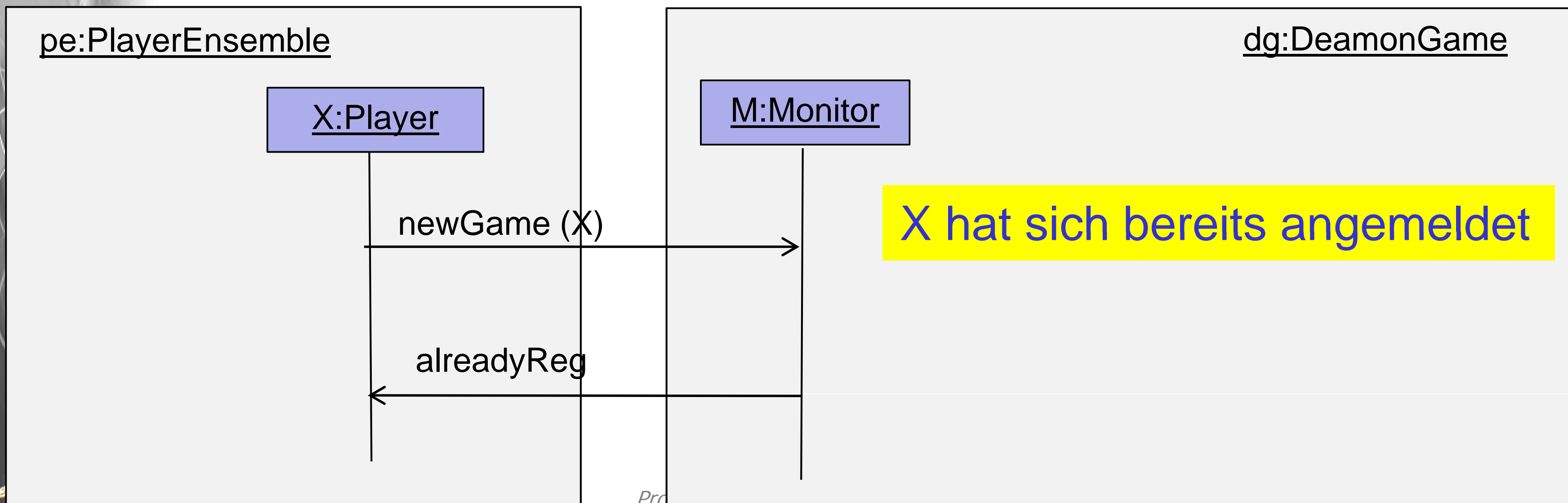
Schnittstellenprotokolle: *Zufällige Variablenänderung*



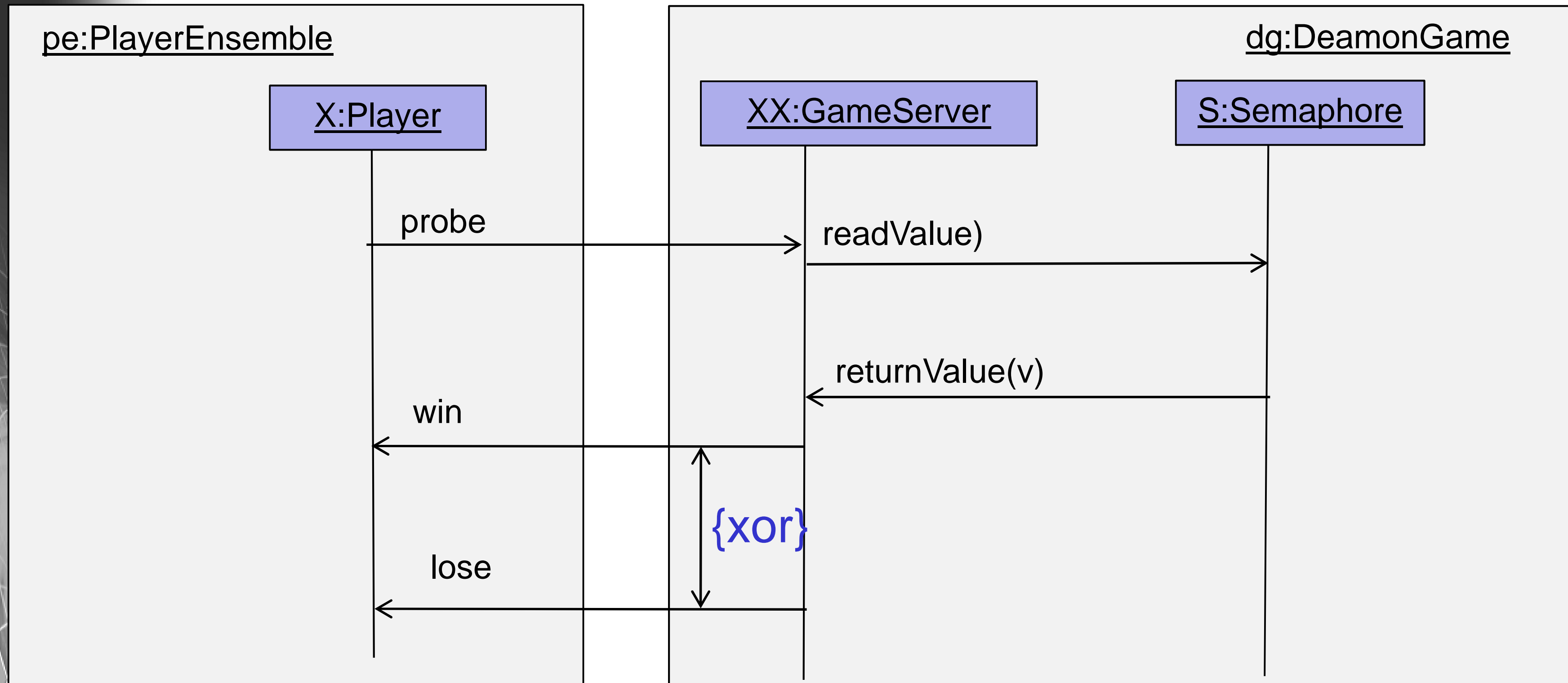


Schnittstellenprotokolle: Anmeldung

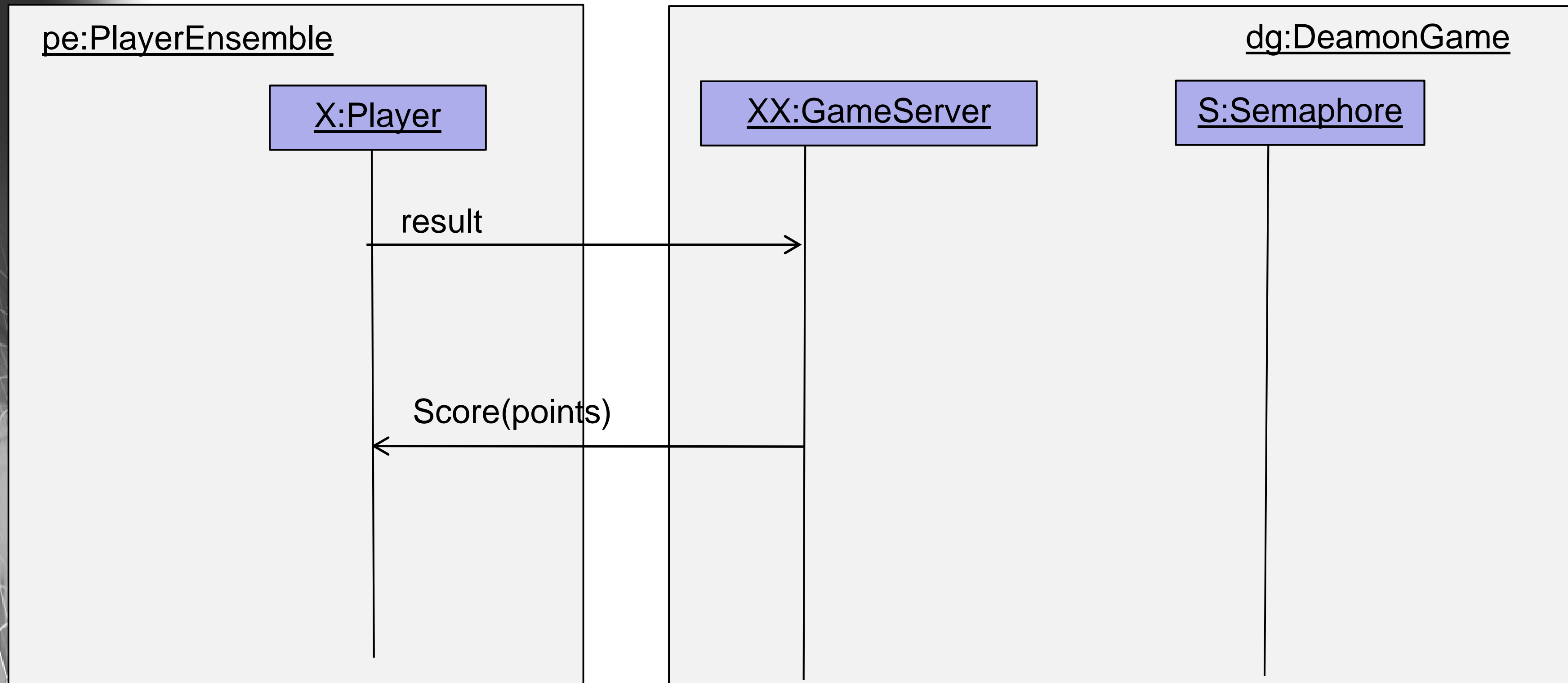




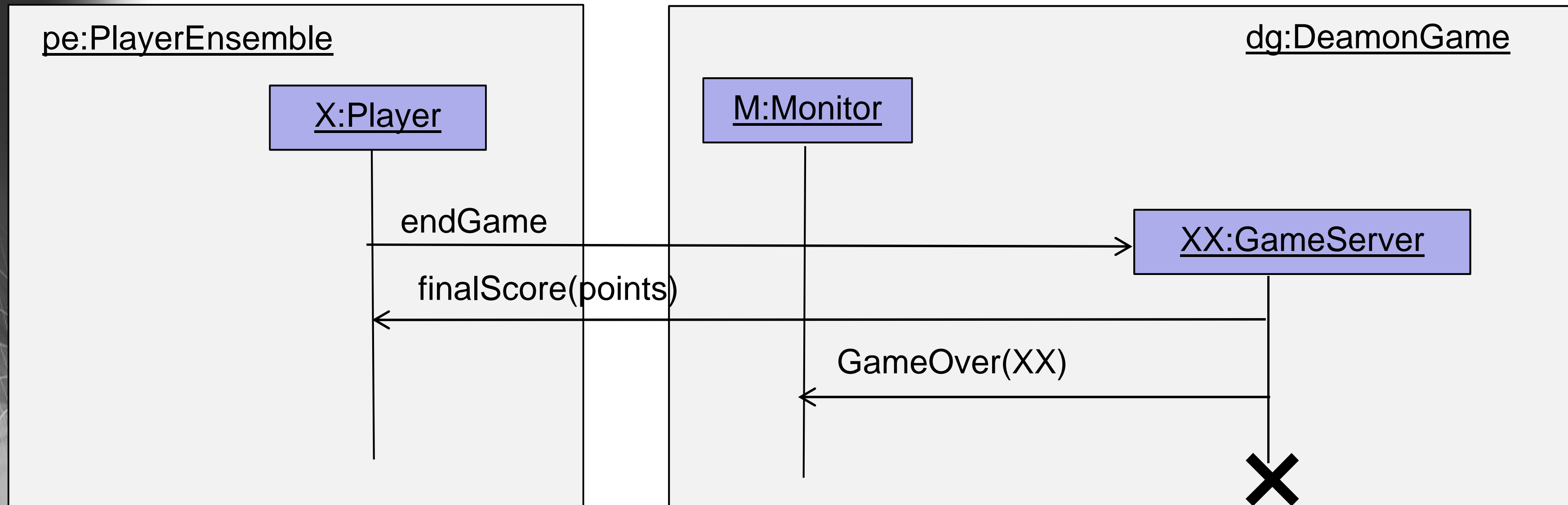
Schnittstellenprotokolle: Spielzug



Schnittstellenprotokolle: Spielstandabfrage

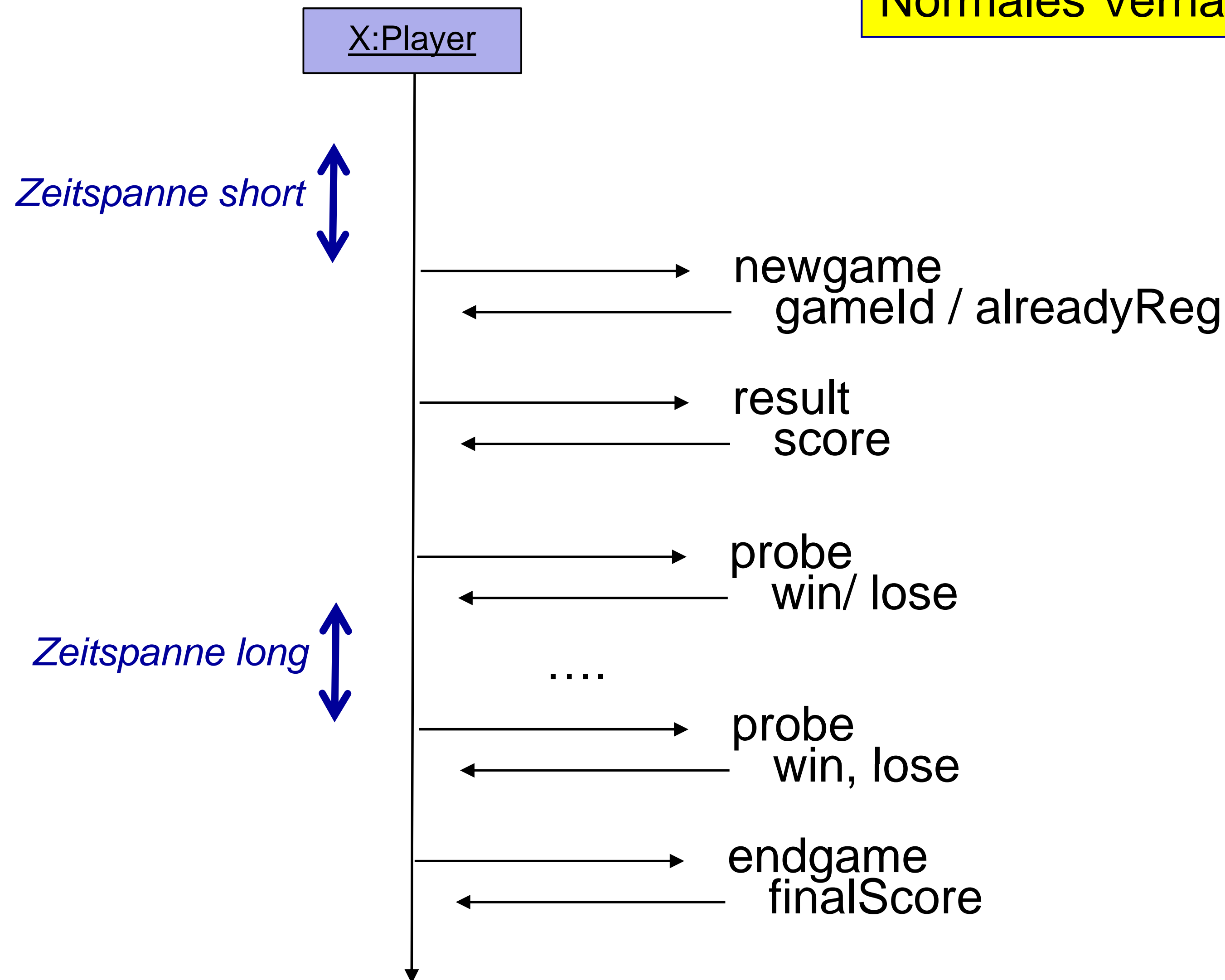


Schnittstellenprotokolle: Abmeldung

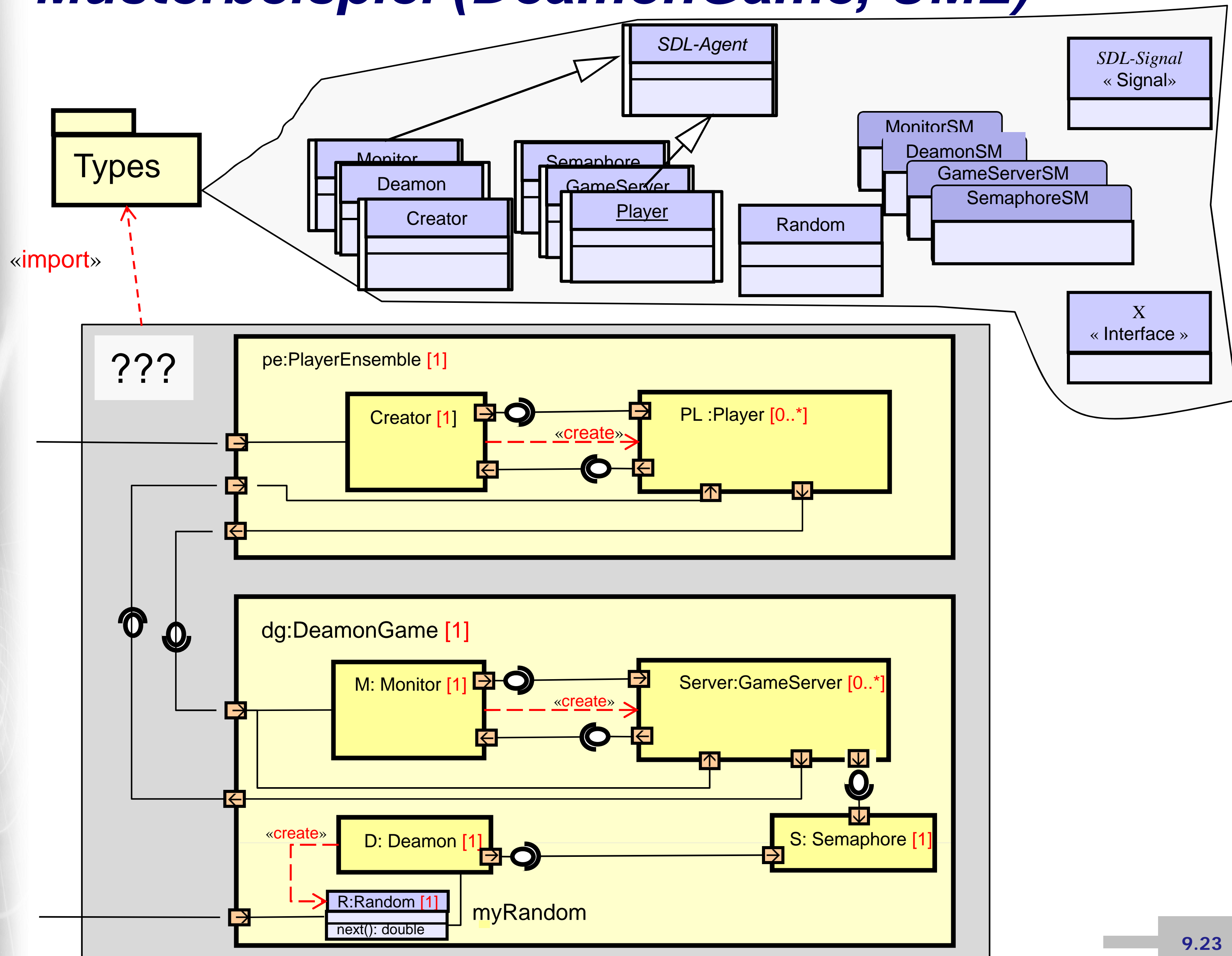


Ressourcenfreigabe

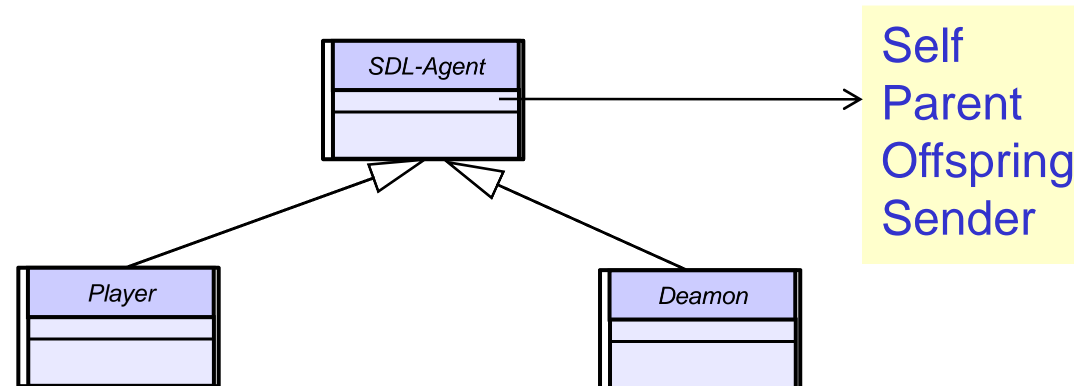
A-4: Nutzerverhalten



Musterbeispiel (DeamonGame, UML)

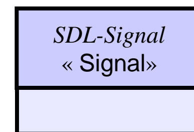


Erste Präzisierungen (SDL als UML-Profil)



Self
Parent
Offspring
Sender

Existenz verbunden mit Vergabe systemeindeutiger PId-Werte



jedes Signal überträgt die PId des sendenden Prozesses

Gesamtspezifikation, inkl. Verhalten in UML bleibt dennoch schwierig (ohne weitere Präzisierungen unmöglich) !!!

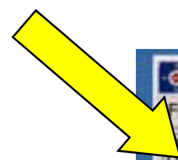
5. SDL als UML-Profil

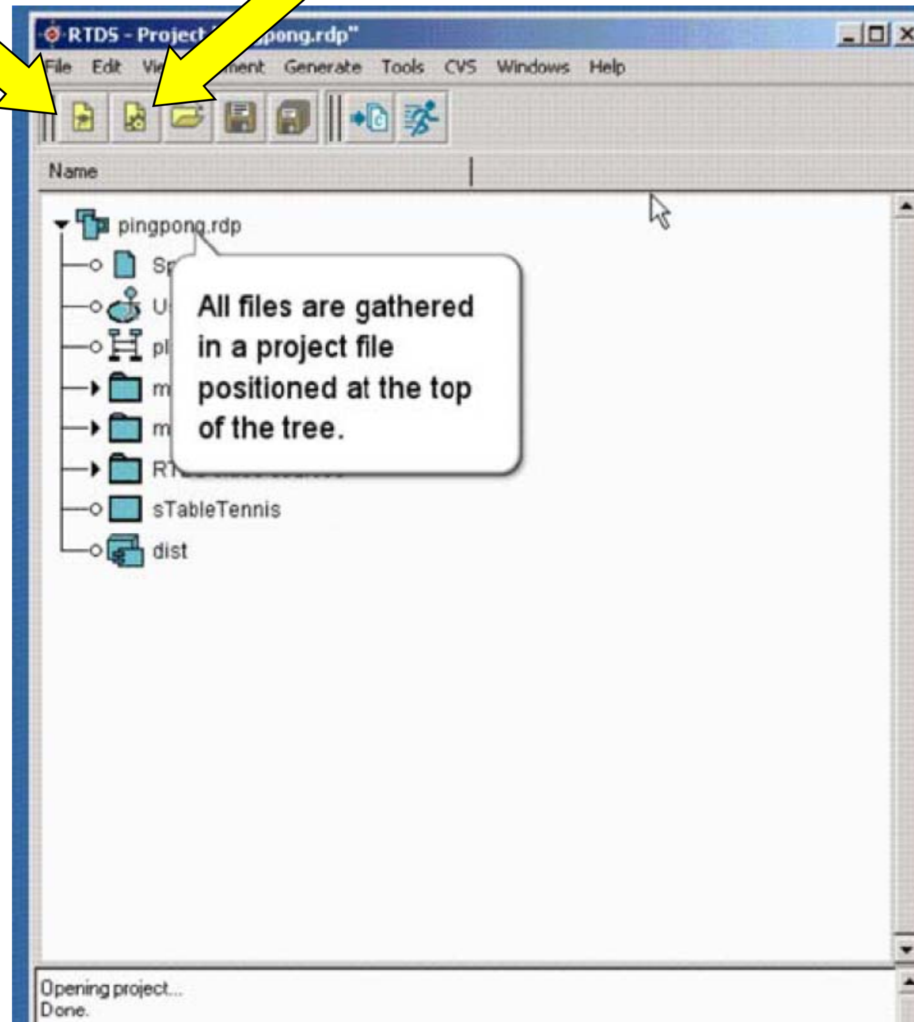
1. ITU-Standard Z.100
2. Werkzeuge
3. SDL-Grundkonzepte
4. Musterbeispiel in UML-Strukturen
5. Musterbeispiel in PragmaDev SDL-RT
6. Struktur- und Verhaltensbeschreibung in SDL (Präzisierung)

PragmaDev DS- Project Manager

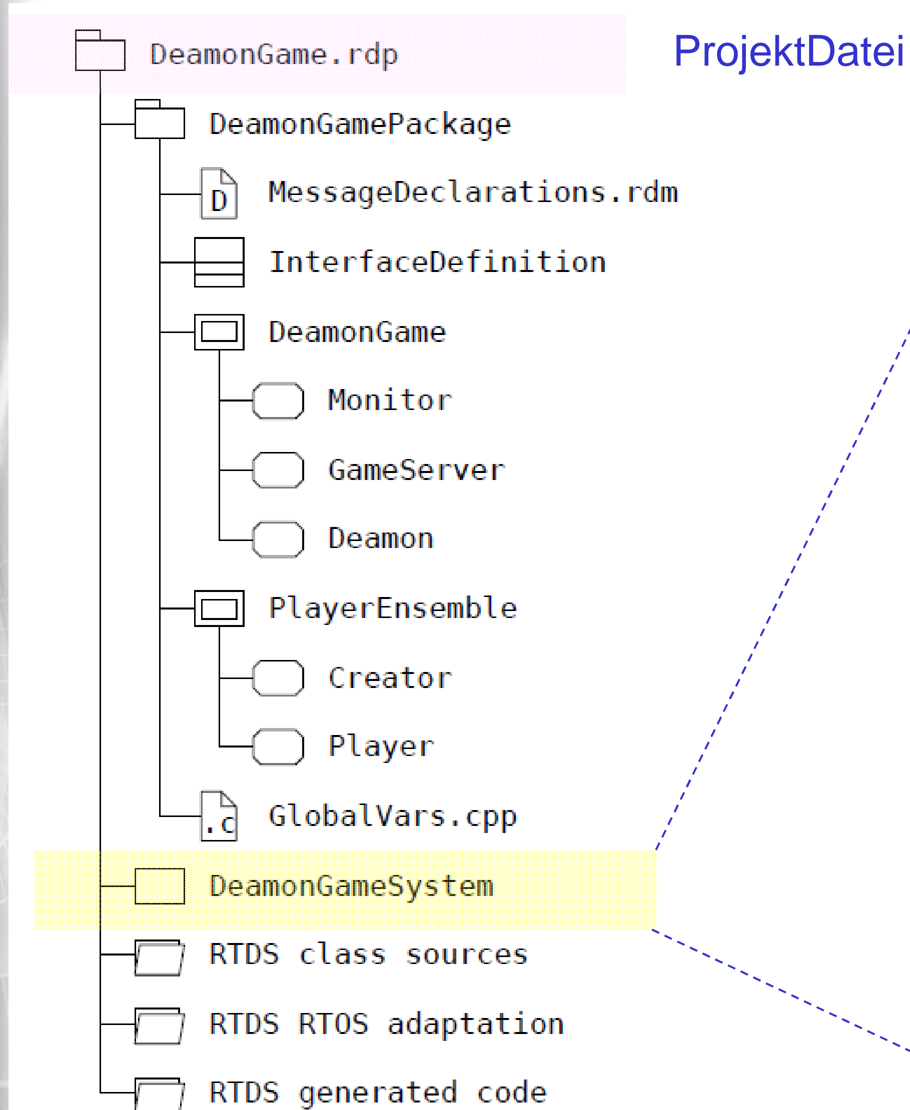
SDL-RT

SDL-Z100

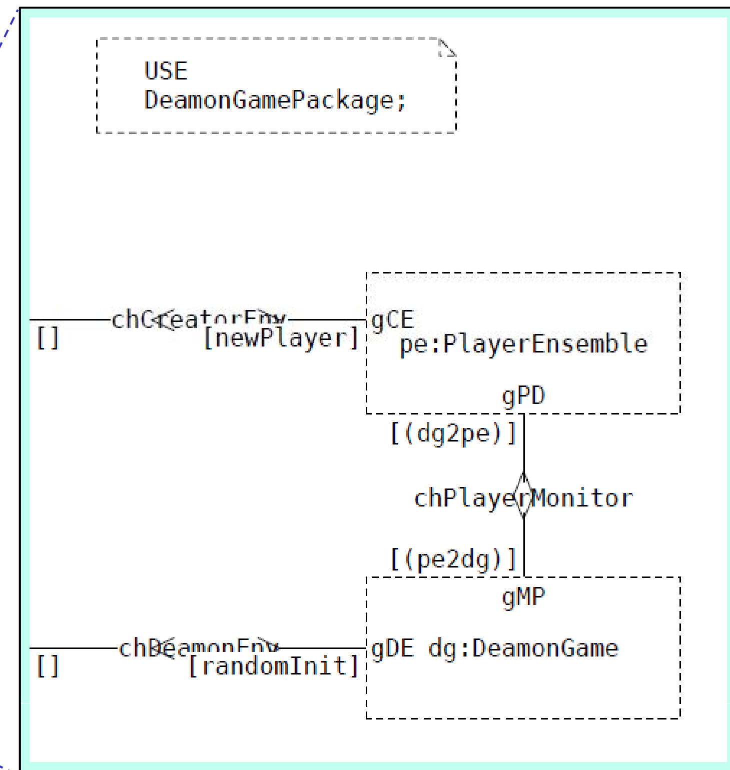




DeamonGame-Projekt



ProjektDatei



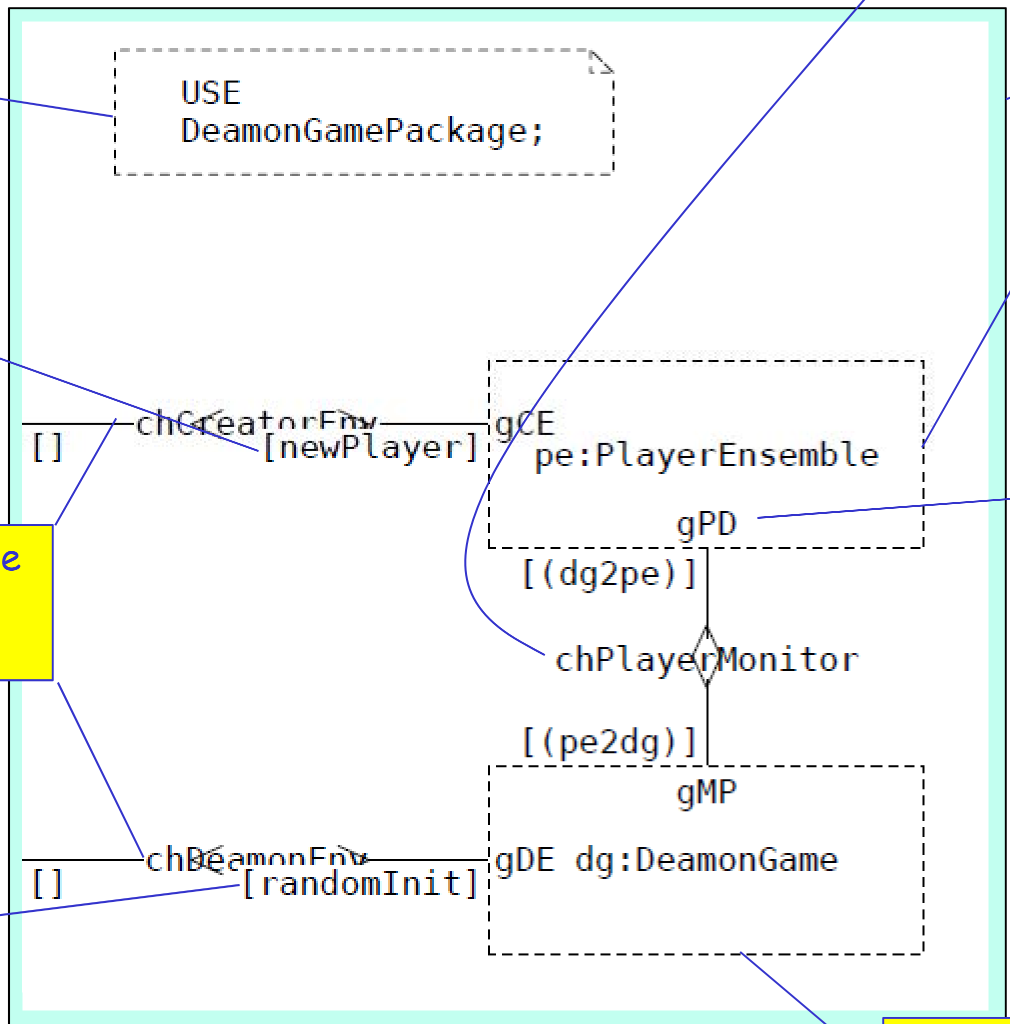
Systemname entspricht Dateinamen/Projektnamen

Systemspezifikation

bidirektionalerKanal chPlayerMonitor
mit Angabe zu transportierender Nachrichten

Package-Import

Systemgrenze



Signal newPlayer von der Umgebung

Block pe vom Typ PlayerEnsemble

Gate gPD (= UML-Port) mit Eingang dg2pe und Ausgang p22dg

unidirektionale Kanäle chCreatorEnv chDeamonEnv

Nachrichten-Listen dg2pe pe2dg

entweder lokal oder im Paket defniert

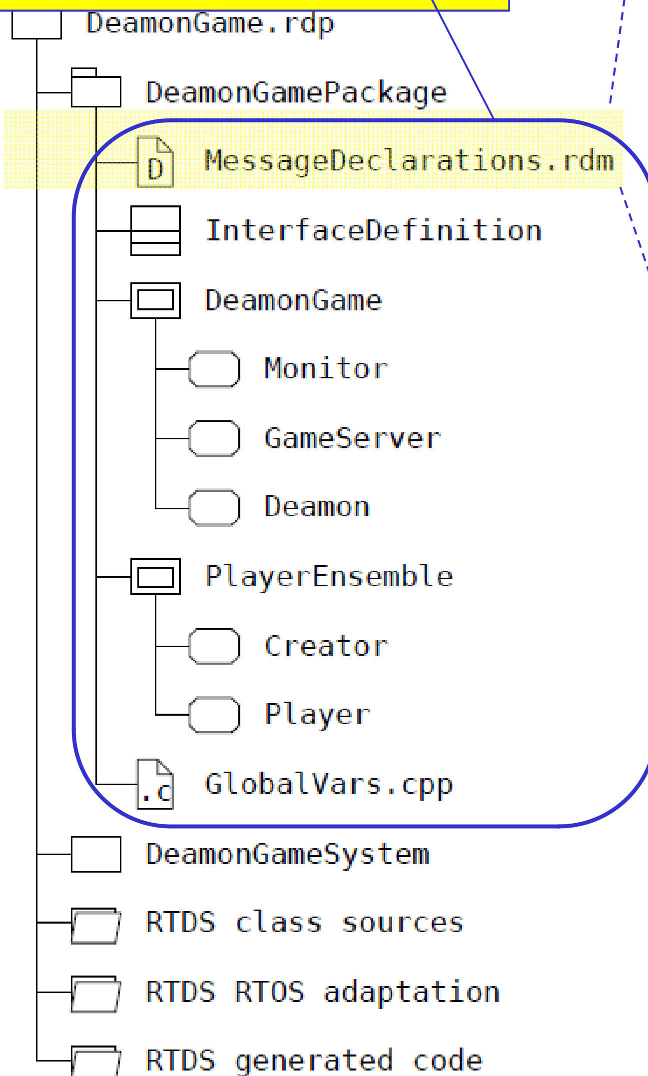
Signal randomInit von der Umgebung

entweder lokal oder im Paket defniert

Block dg vom Typ DeamonGame

Package

besteht aus verschiedenen Typdefinitionen und systemglobalen Variablen




```
MESSAGE newPlayer(int);
MESSAGE newGame();
MESSAGE randomInit(int, int);
MESSAGE startGame();
MESSAGE playerReference(RTDS_QueueId);
MESSAGE endGame();
MESSAGE getResult();
MESSAGE result(int);
MESSAGE probe();
MESSAGE win();
MESSAGE loss();
MESSAGE stopGameServer();
MESSAGE_LIST pe2dg = newGame, probe, getResult, endGame;
MESSAGE_LIST dg2pe = startGame, result, win, loss;
```

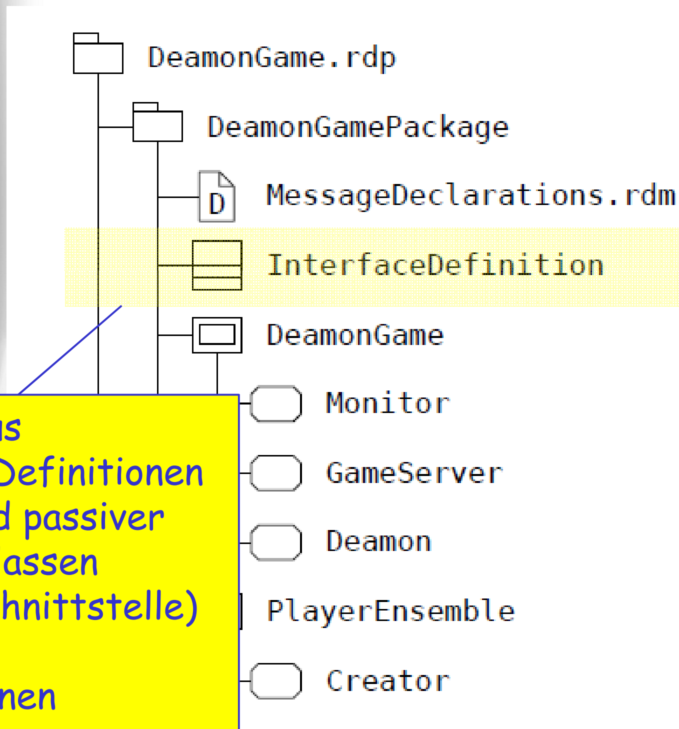
Definition von Nachrichtentypen
und
Definition von Nachrichtenlisten

SDL-RT-Besonderheiten

Nachrichtenparametertypen:
- Standard C-Typen
- SDL-Typ PId
- nutzereigene C++ Typen

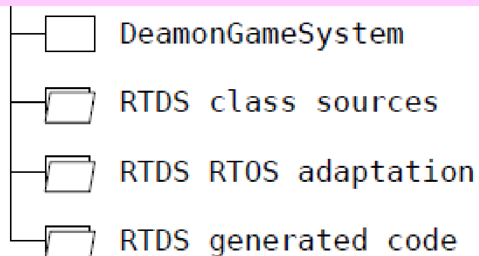
PId = RTDS_QueueId
SIGNAL = MESSAGE

Package

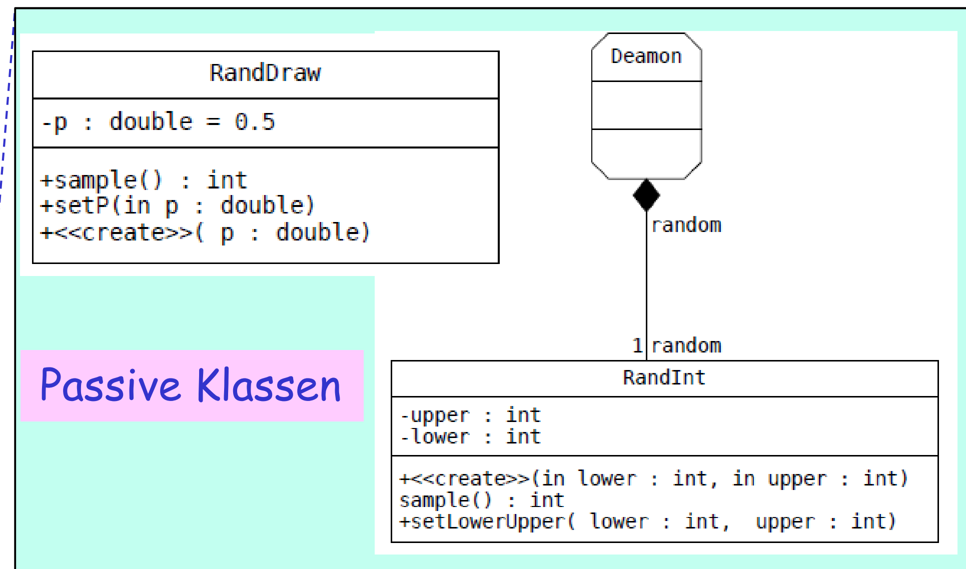


besteht aus UML-like-Definitionen aktiver und passiver SDL-RT-Klassen (äußere Schnittstelle) und deren Assoziationen

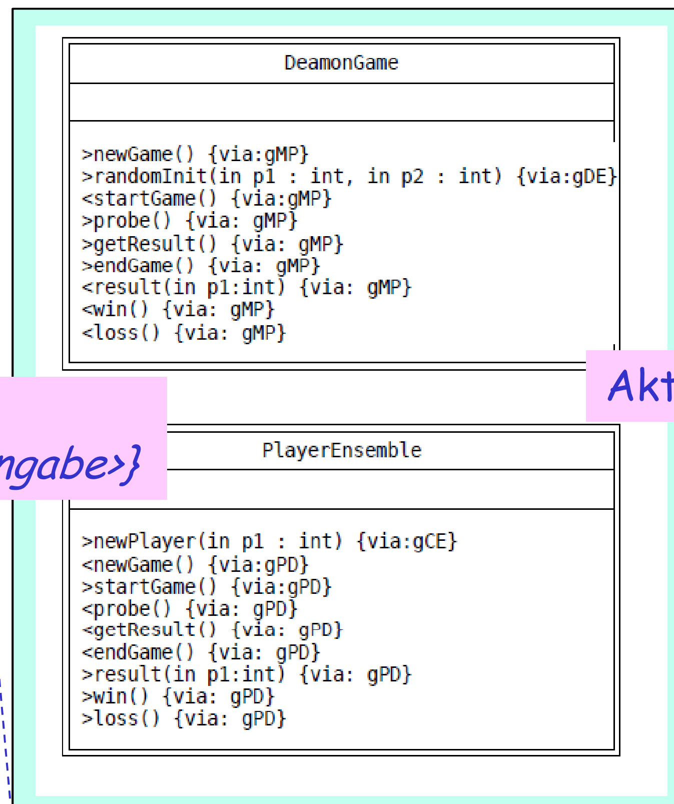
Interface:
<Richtung> <Nachrichtensignatur> {via: <Gate-Angabe>}



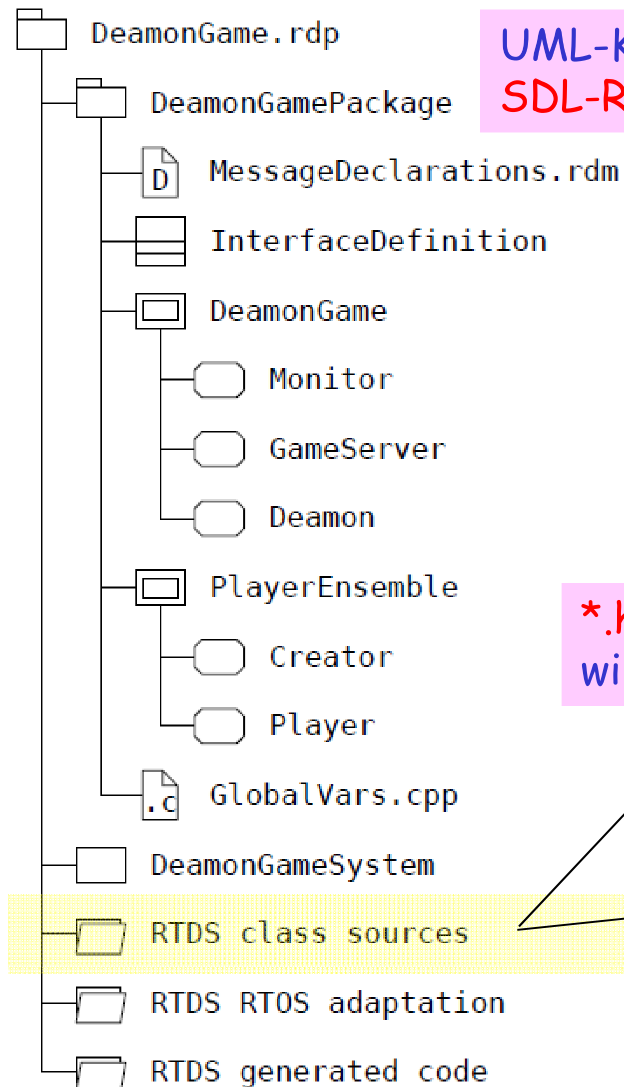
Passive Klassen

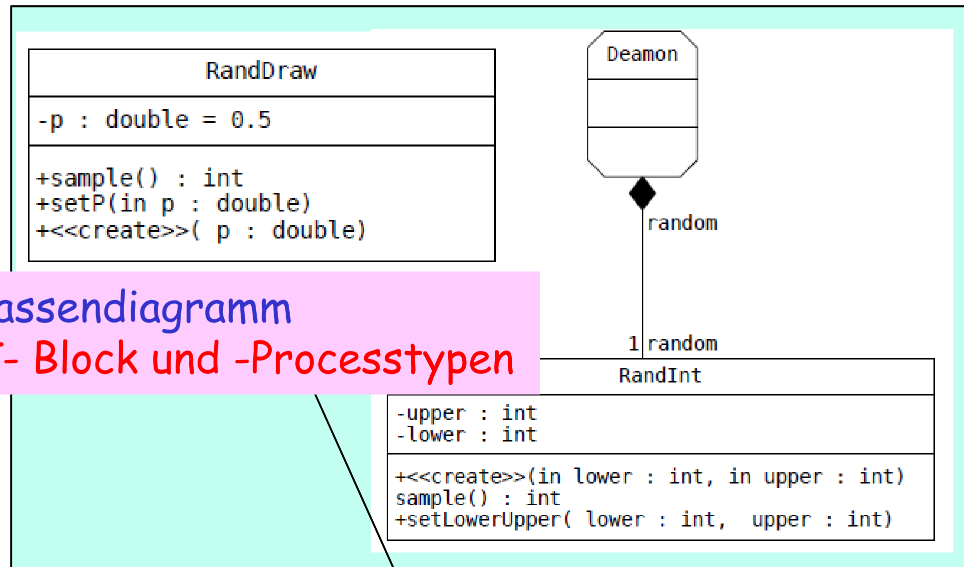


Aktive Klassen



Package



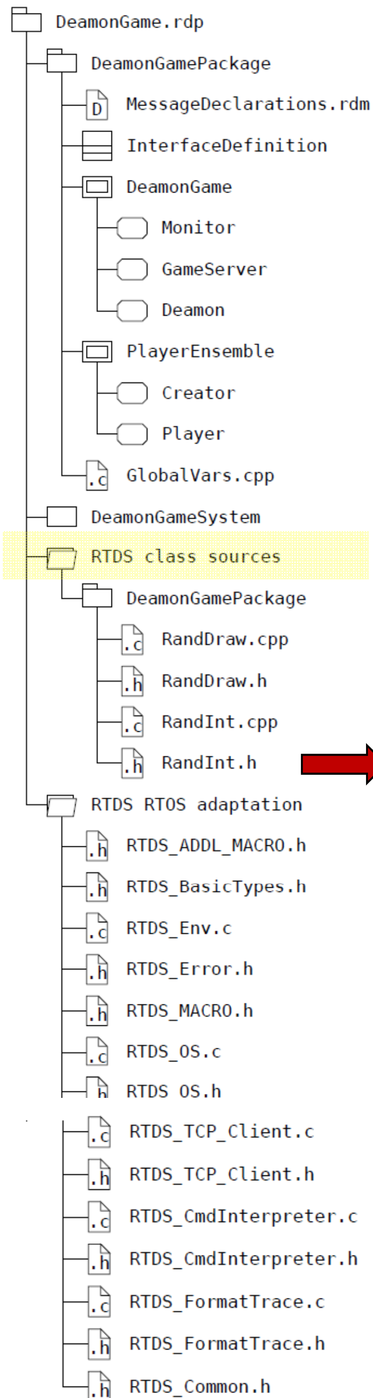


UML-Klassendiagramm
SDL-RT- Block und -Processtypen

Random wird lokale Variable von Deamon (Typ: RandInt)

*.h
wird generiert

*.ccp
ist nutzerseitig bereitzustllen

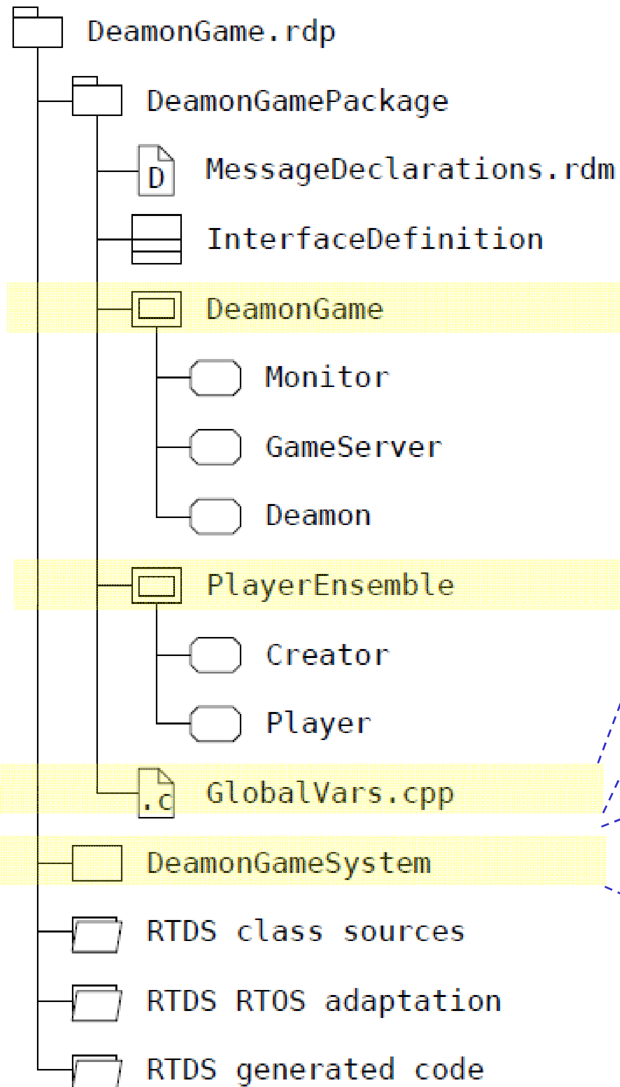


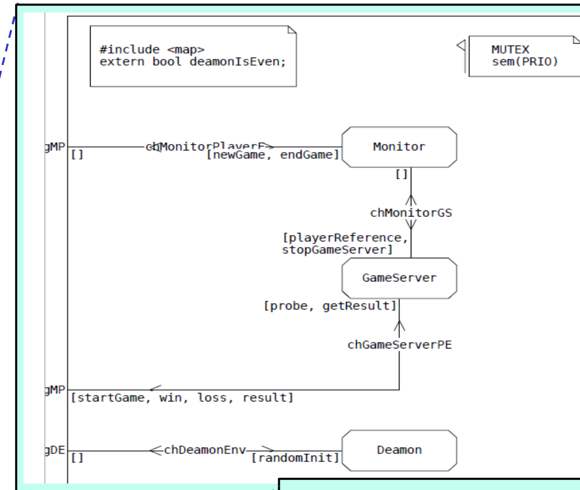
C++ -Quellen

```
#ifndef _RANDINT_H_
#define _RANDINT_H_

// Forward declaration
class RandInt;


// Standard and common includes
#include "RTDS_gen.h"

// Includes for related classes

#include "RTDS_messages.h"




// CLASS RandInt:
// ==============

class RandInt
  {
  // ATTRIBUTES:
  // -----------

   private:
               int    lower;
               int    upper;

   // OPERATIONS:
   // -----------
   public:
               RandInt(int lower, int upper);
               virtual int   sample();
               virtual void  setLowerUpper( int
                             lower, int upper);
   };

#endif
```

```
#include "DeamonGamePackage\RandInt.h"
#include <cmath>
/*
 * ATTRIBUTES FOR CLASS:
 * --------------------
 *
 * [From RandInt]
 * - int   lower;
 * - int   upper;
 */


// PUBLIC OPERATIONS:
// ==================


// Operation RandInt:
// ------------------
RandInt::RandInt(int lower, int upper)
              {
              this->lower = lower;
              this->upper = upper;
              }

// Operation sample:
// -----------------
int RandInt::sample()
              {
              int span = upper -lower;
  return (rand() % span + lower);
              }

// Operation setLowerUpper:
// -----------------------
void RandInt::setLowerUpper(int lower, int upper)
              {
              this->lower = lower;
              this->upper = upper;
              }
```

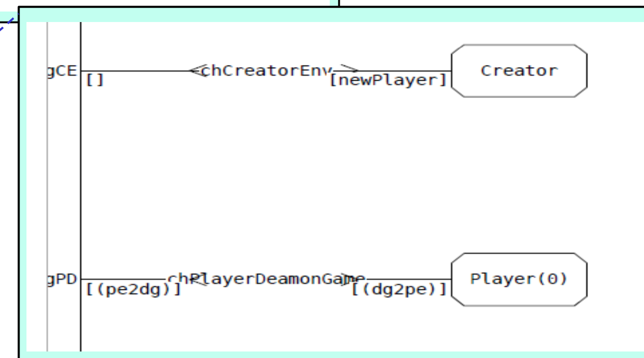
Blocktypen





Blocktyp DeamonGame
Name = Dateiname

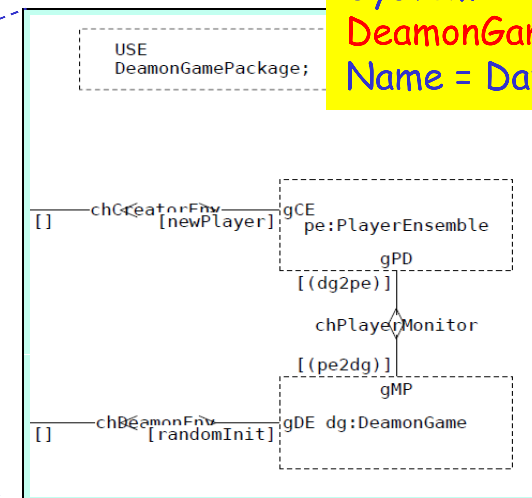
Blocktyp PlayerEnsemble
Name = Dateiname




```
bool deamonIsEven = true;
```

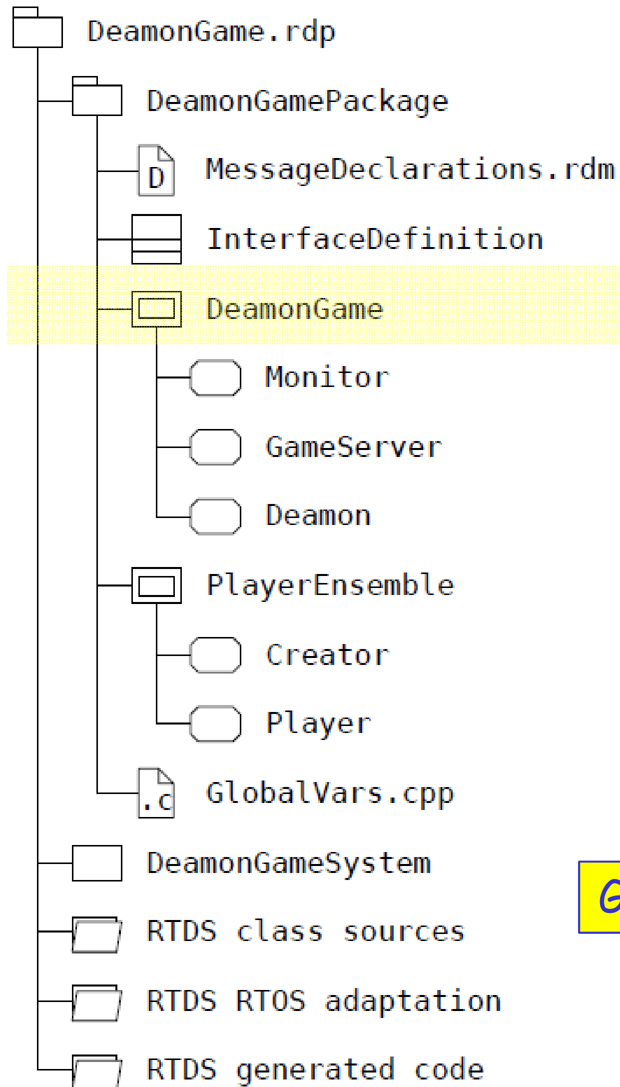
C/C++ Erweiterung
globale Variable

System
DeamonGameSystem
Name = Dateiname



'bebenfrühwarnung

Blocktyp DeamonGame




```
#include <map>
extern bool deamonIsEven;
```

MUTEX
sem(PRIO)

gMP
[]
chMonitorPlayerE
[newGame, endGame]
Monitor
[]
chMonitorGS
[playerReference,
stopGameServer]
GameServer
[probe, getResult]
chGameServerPE
gMP
[startGame, win, loss, result]
gDE
[]
chDeamonEnv
[randomInit]
Deamon

Semaphore-Steuerung

Prozess-Instanz-Mengen (1,*)

Gate-Angabe

C++ StandardTemplateLibrary

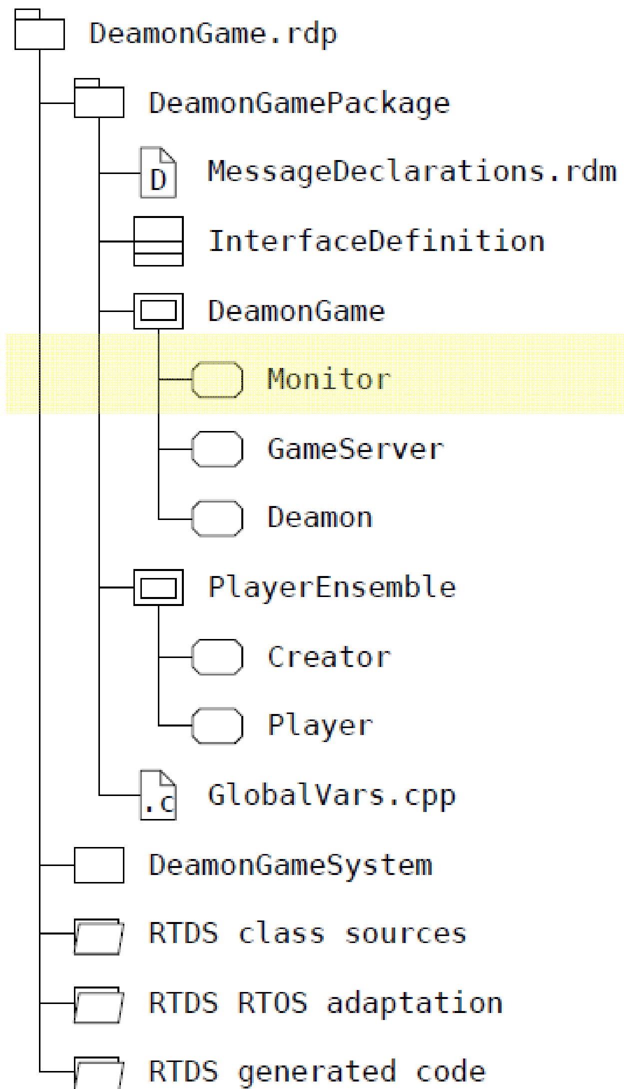
besser:
Monitor(1,1)
GameServer(0,*)
Deamon(1,1)

C++ Standard Template Library

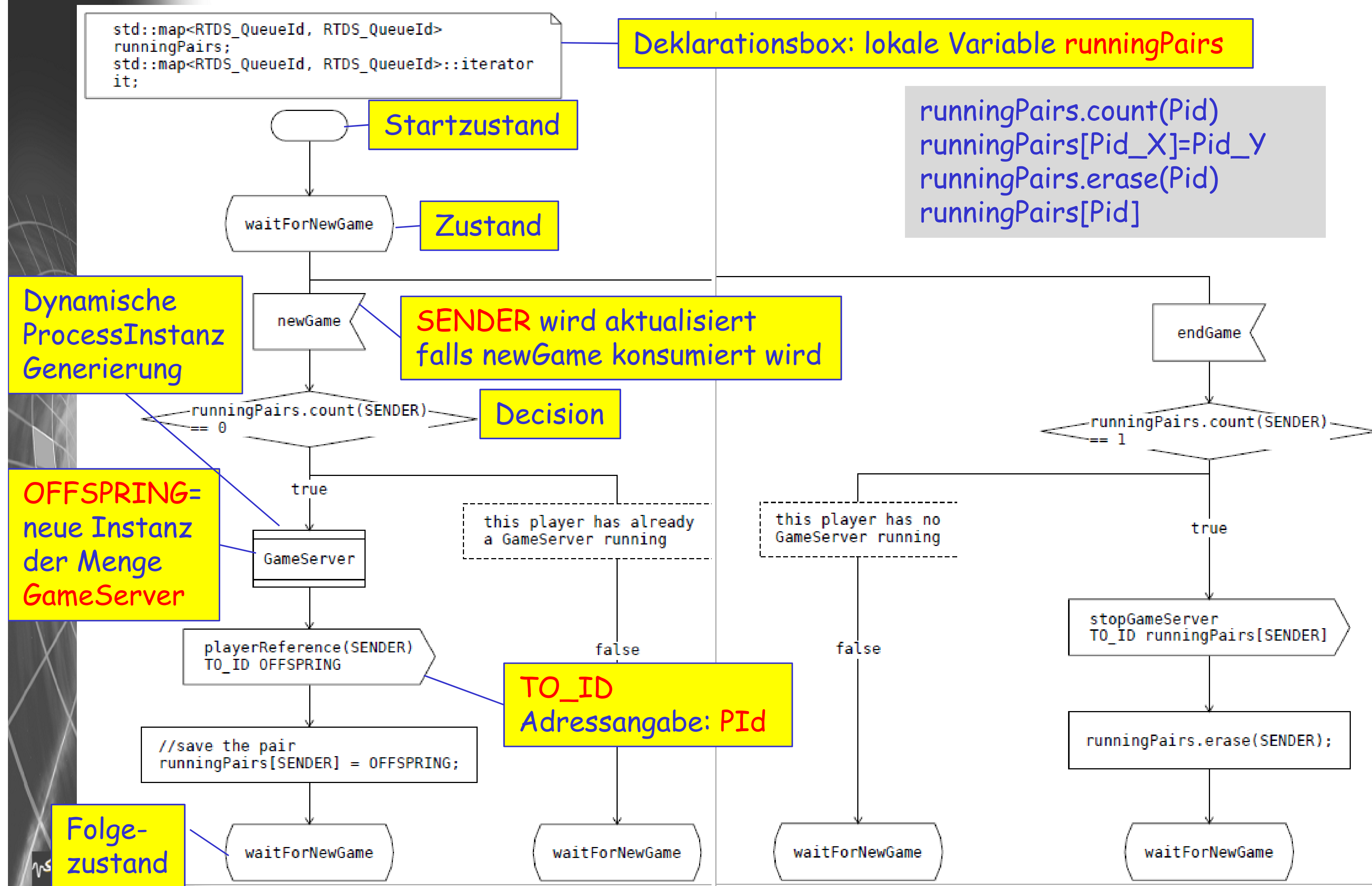
C++ Maps are sorted associative containers
that contain unique key/value pairs.
Maps are sorted by their keys.

Map Constructors & Destructors	**default methods to allocate, copy, and deallocate maps**
Map operators	assign, compare, and access elements of a map
Map typedefs	typedefs of a map
begin	returns an iterator to the beginning of the map
clear	removes all elements from the map
count	returns the number of elements matching a certain key
empty	true if the map has no elements
end	returns an iterator just past the last element of a map
equal_range	returns iterators to the first and just past the last elements matching a specific key
erase	removes elements from a map
find	returns an iterator to specific elements
…	

Package



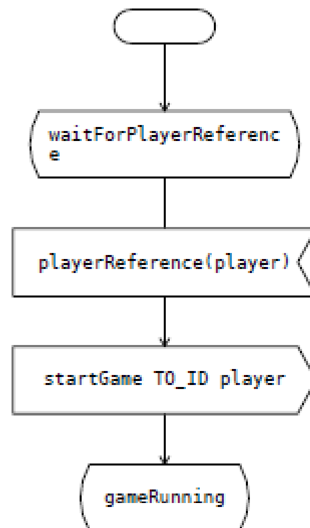
Prototypinstanz der Processmenge Monitor

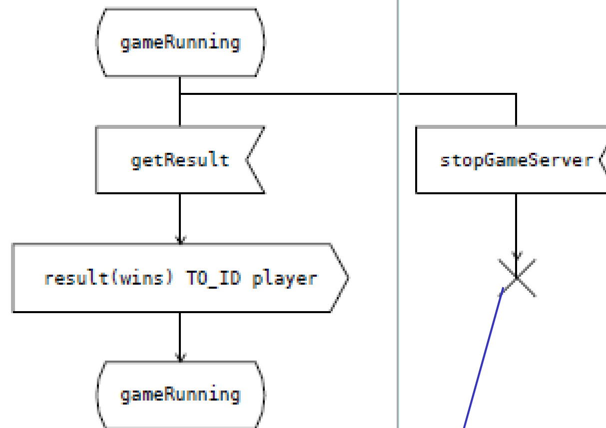


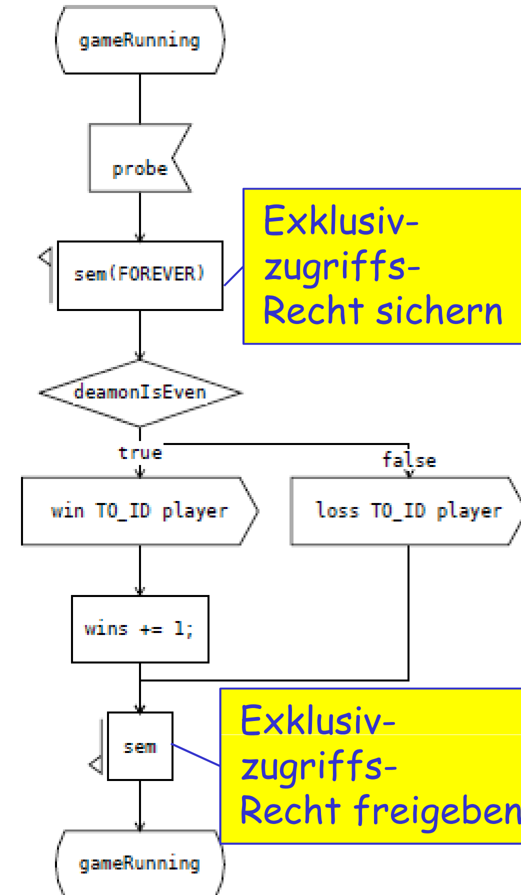
Prototypinstanz der Processmenge GameServer

Deklarationsbox: Lokale Variable player: zugeordneter externer Spieler

```
RTDS_QueueId player;
int wins = 0;
```







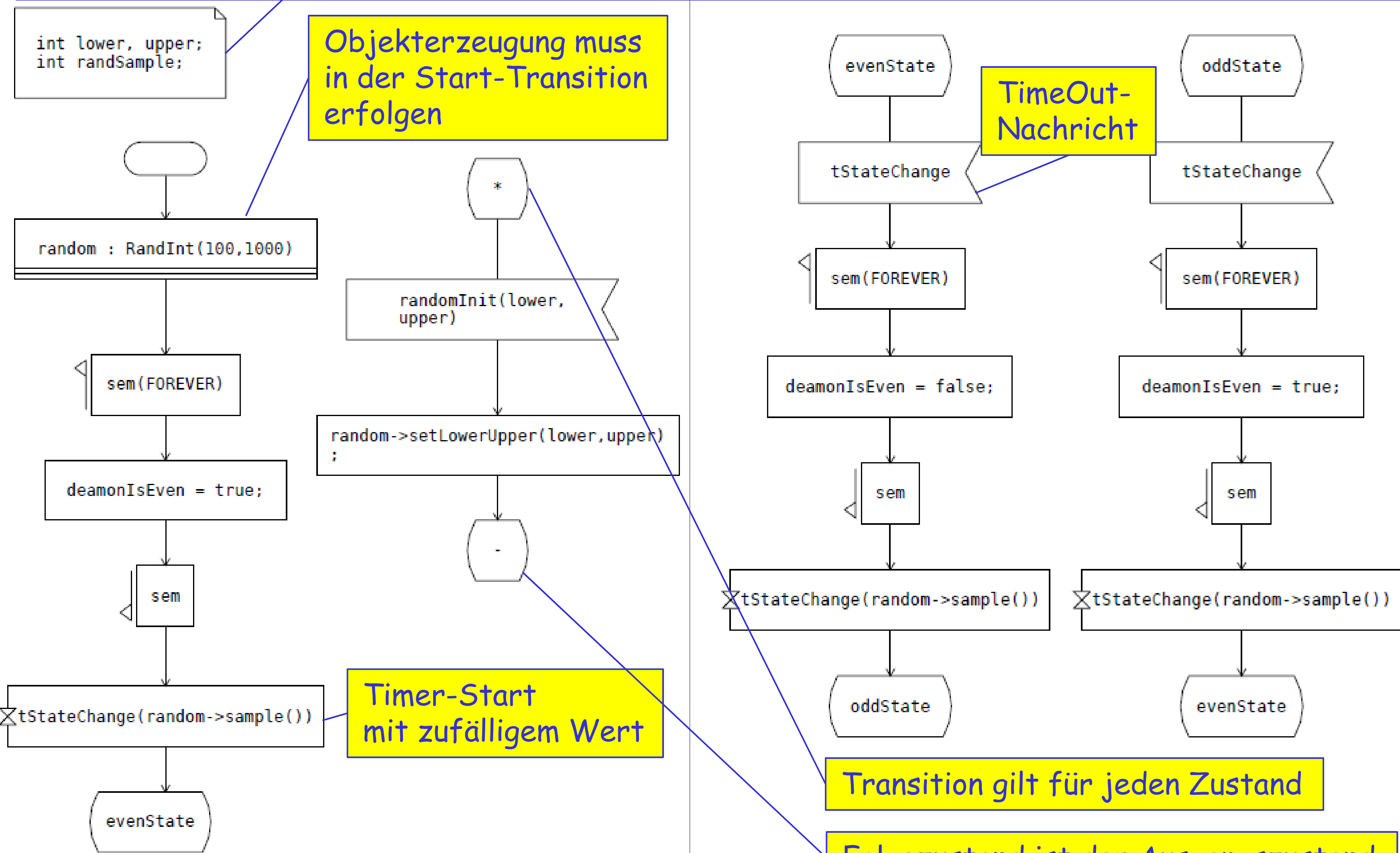
Exklusiv-zugriffs-Recht sichern

Vernichtung der ProcessInstanz

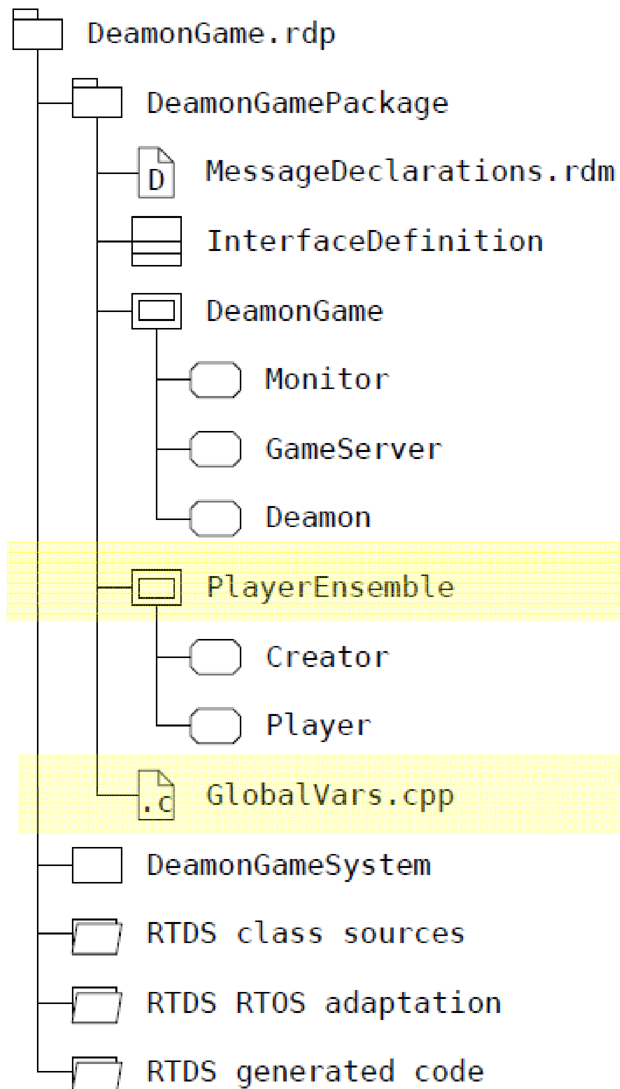
Exklusiv-zugriffs-Recht freigeben

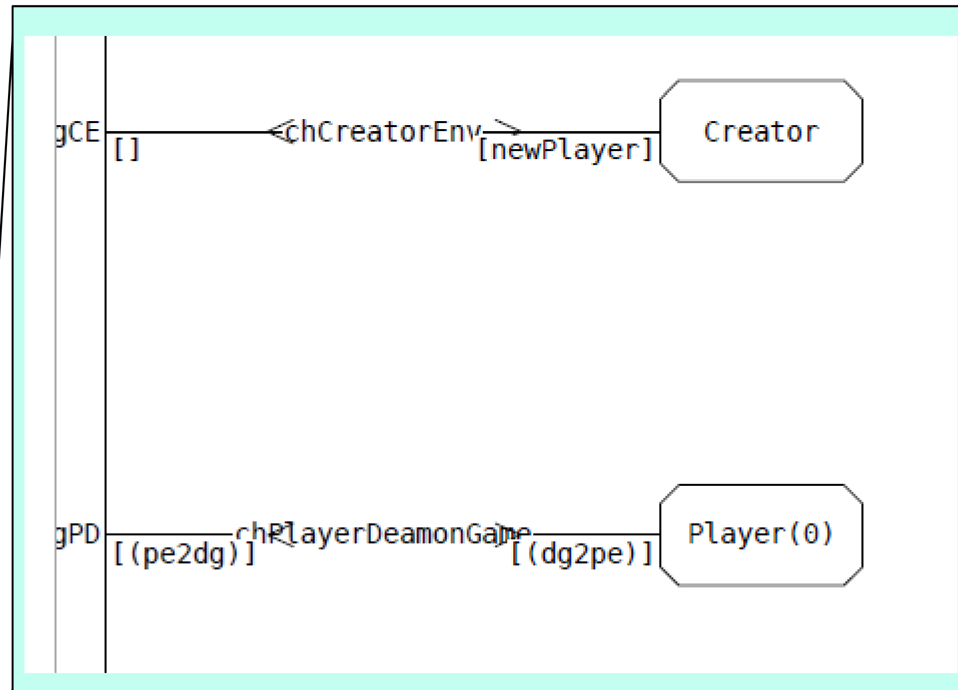
Weitere Kommunikation des Spielers über die PId-Variable (jetzt Null-Wert) führt zu einem LaufzeitFehler

Prototypinstanz der Processmenge Deamon



Blocktyp PlayerEnsemble

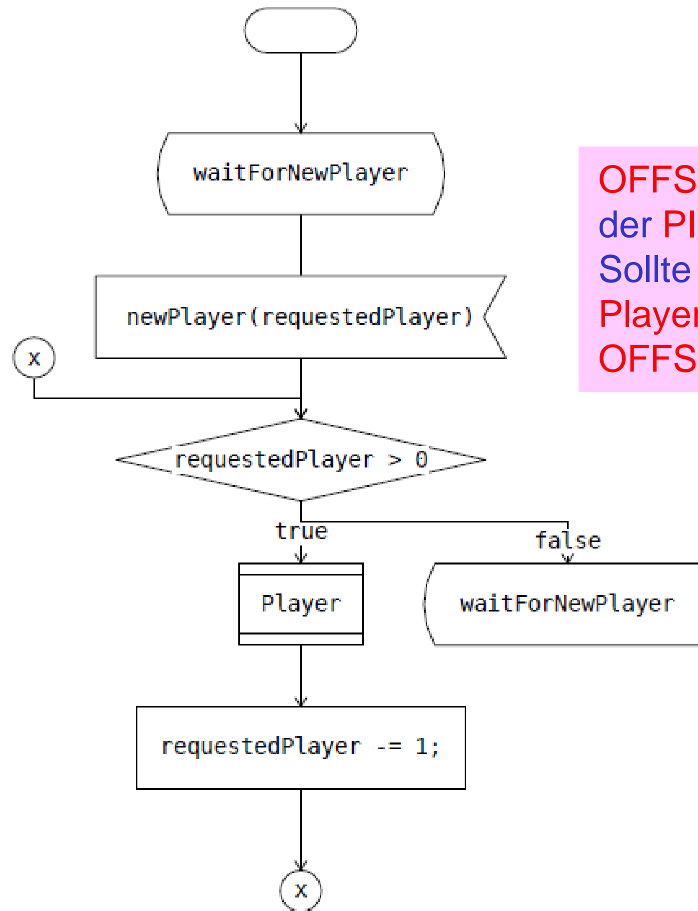





```
bool deamonIsEven = true;
```

Verbindung zur Semaphore-Variablen ist nutzerseitig zu organisieren

Prototypinstanz der Prozessmenge Creator

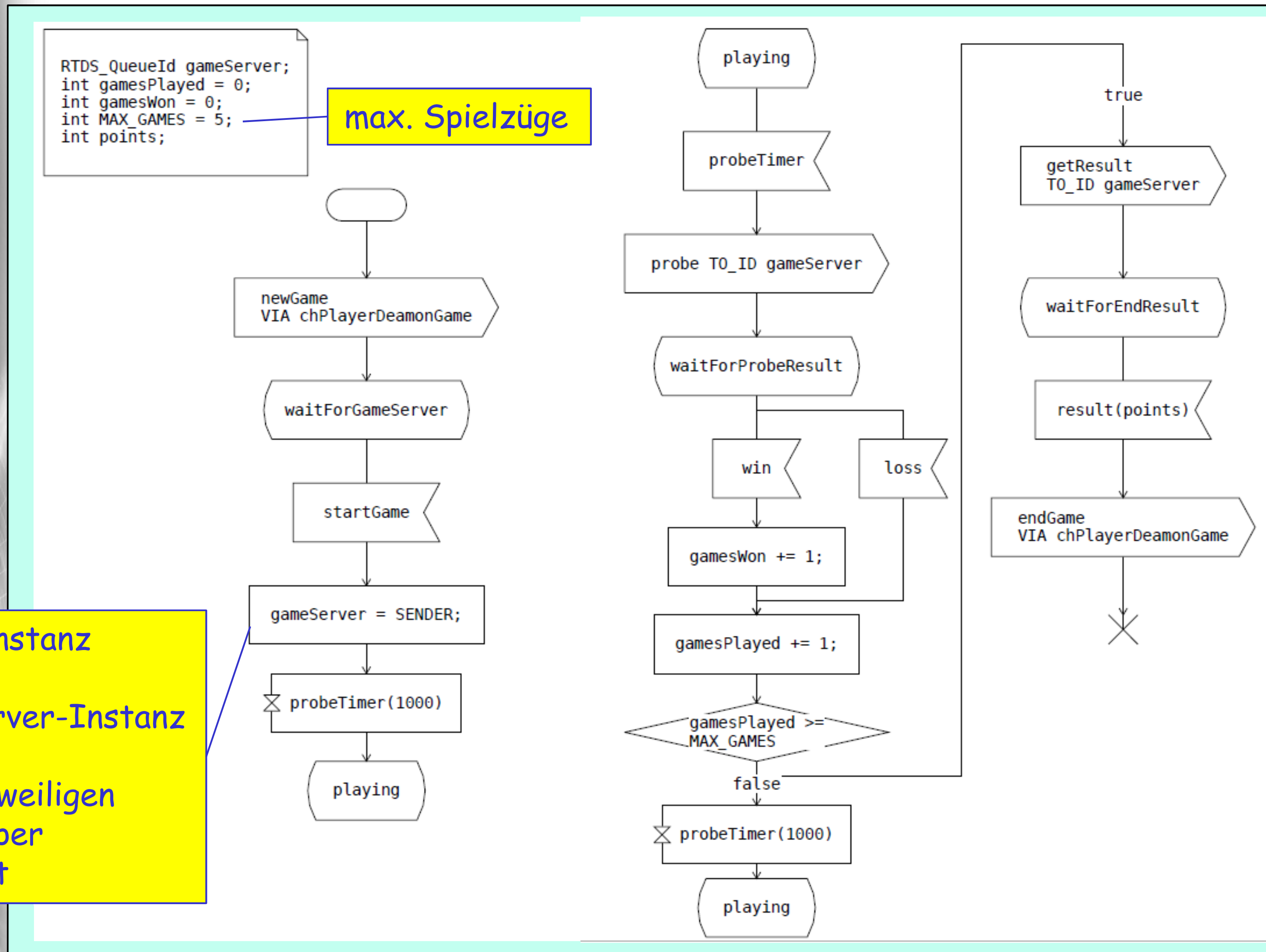


Deklarationsbox: Lokale Variable requestedPlayer

OFFSPRING wird bei jeder erfolgreichen Instanziierung der PId-Wert der Instanz zugewiesen
Sollte dabei die maximale Kardinalität der Instnzmenge Player überschritten werden, wird keine Instanz erzeugt, OFFSPRING erhält den Wert Null

Darstellung: geschlossene Verbindung wäre hier Konnektoren vorzuziehen

Prototypinstanz der Prozessmenge Player



Player-Instanz
Und
GameServer-Instanz
Kennen
Ihren jeweiligen
Partner per
Pid-Wert

5. SDL als UML-Profil

1. ITU-Standard Z.100
2. Werkzeuge
3. SDL-Grundkonzepte
4. Musterbeispiel in UML-Strukturen
5. Musterbeispiel in PragmaDev SDL-RT
6. Struktur- und Verhaltensbeschreibung in SDL (Präzisierung)

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.