

# ***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](mailto:fischer|ahrens|eveslage|blunk@informatik.hu-berlin.de)

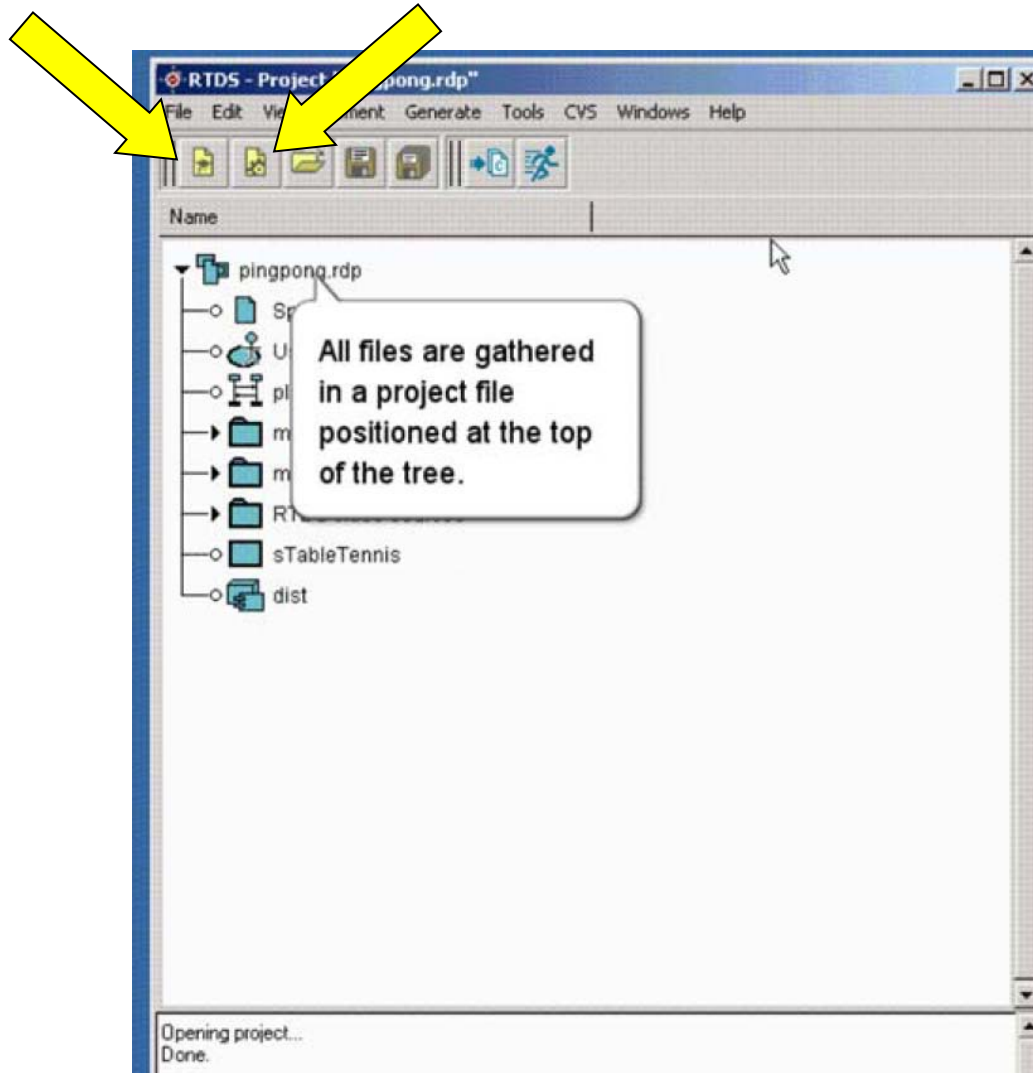
## 6. *SDL-Einführung*

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

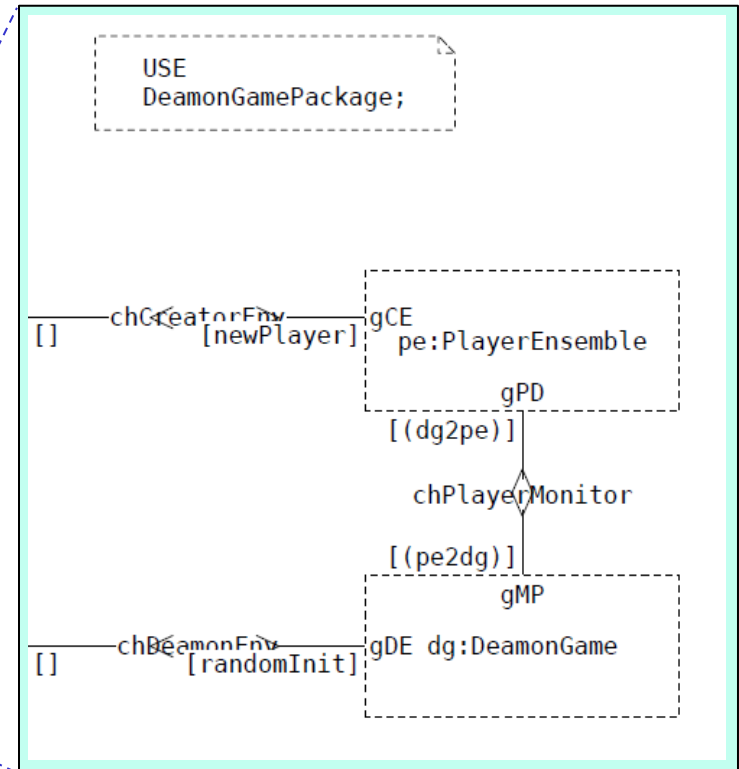
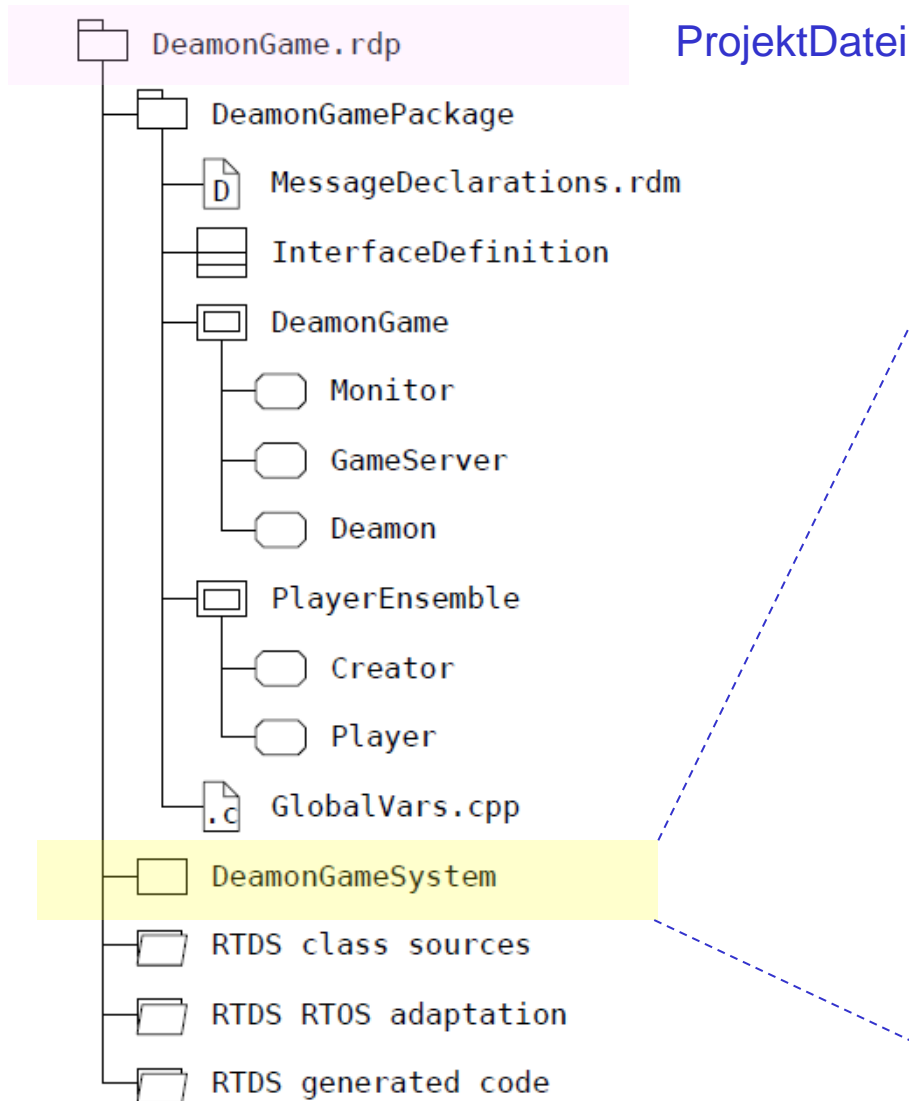
# PragmaDev DS- Project Manager

SDL-RT

SDL-Z100



# DeamonGame-Projekt



Systemname entspricht Dateinamen/Projektname

# Systemspezifikation

bidirektionaler Kanal **chPlayerMonitor**  
mit Angabe zu transportierender Nachrichten

Systemgrenze

Package-Import

```
USE
DeamonGamePackage;
```

Nachricht **newPlayer**  
von der  
Umgebung

Block **pe**  
vom Typ  
**PlayerEnsemble**

Gate **gPD**  
(= UML-Port)  
mit Eingang  
**dg2pe** und  
Ausgang **pe2dg**

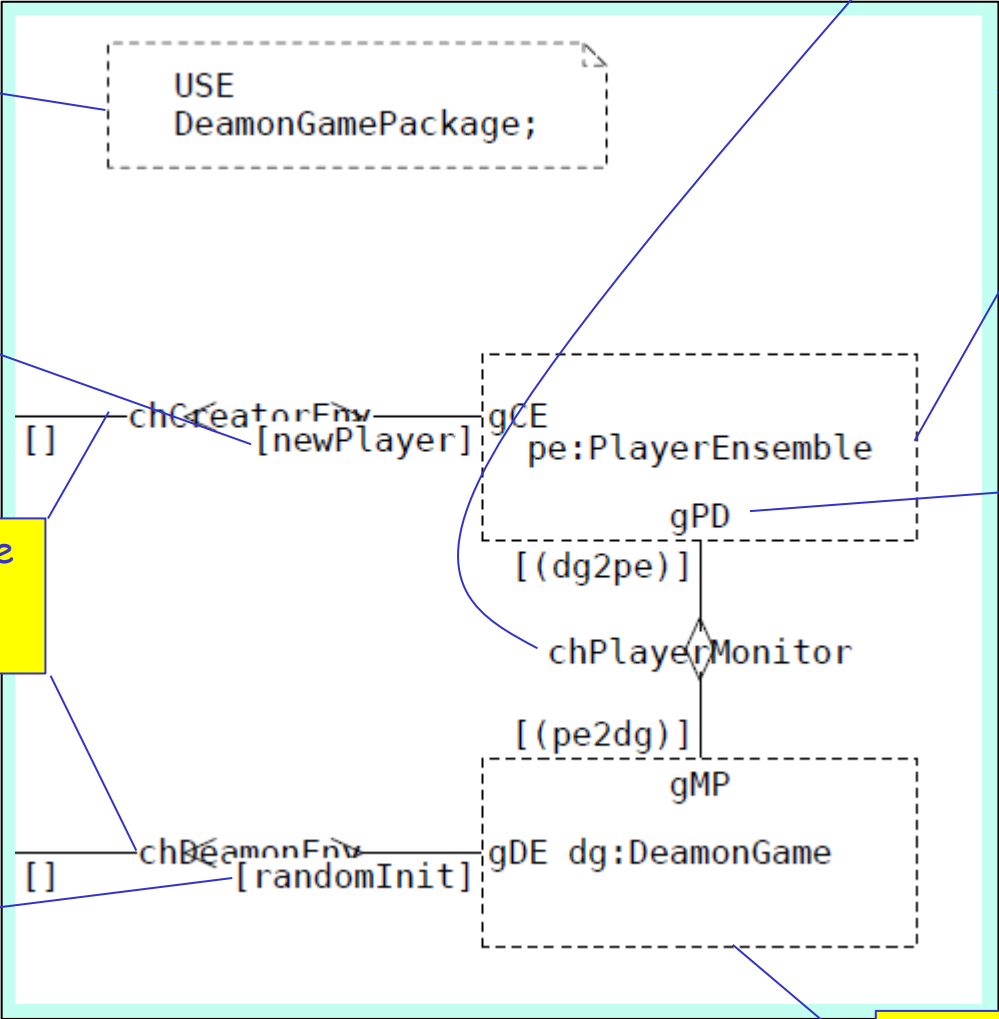
unidirektionale Kanäle  
**chCreatorEnv**  
**chDeamonEnv**

Nachricht **randomInit**  
von der  
Umgebung

Nachrichten-Listen  
**dg2pe**  
**pe2dg**

entweder lokal  
oder im Paket  
definiert

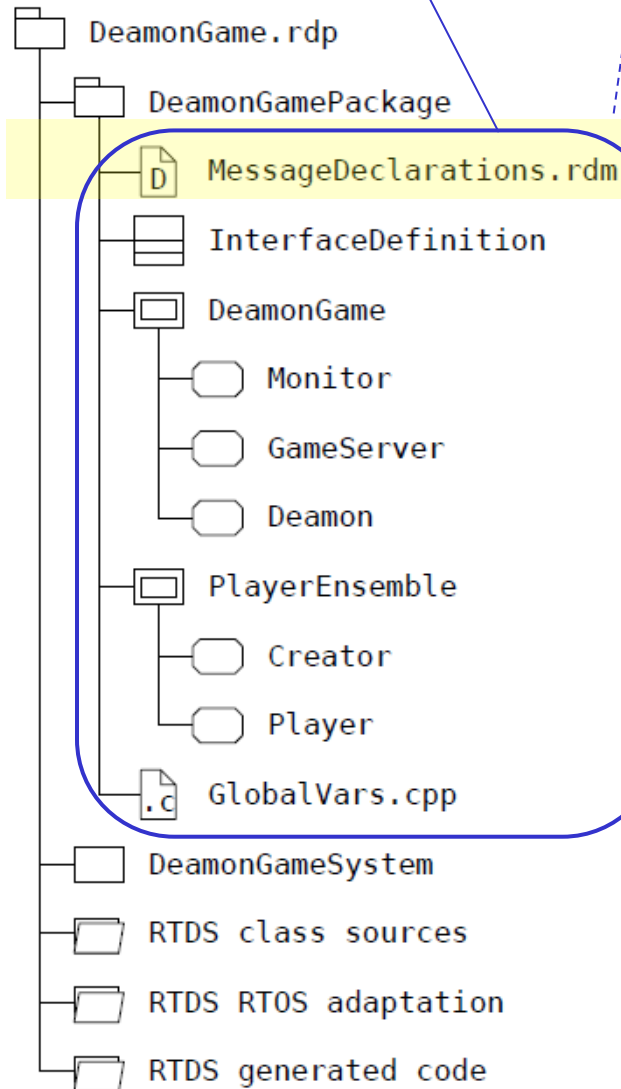
entweder lokal  
oder im Paket  
definiert



Block **dg**  
vom Typ **DeamonGame**

# Package

besteht aus verschiedenen Typdefinitionen



```
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;
```

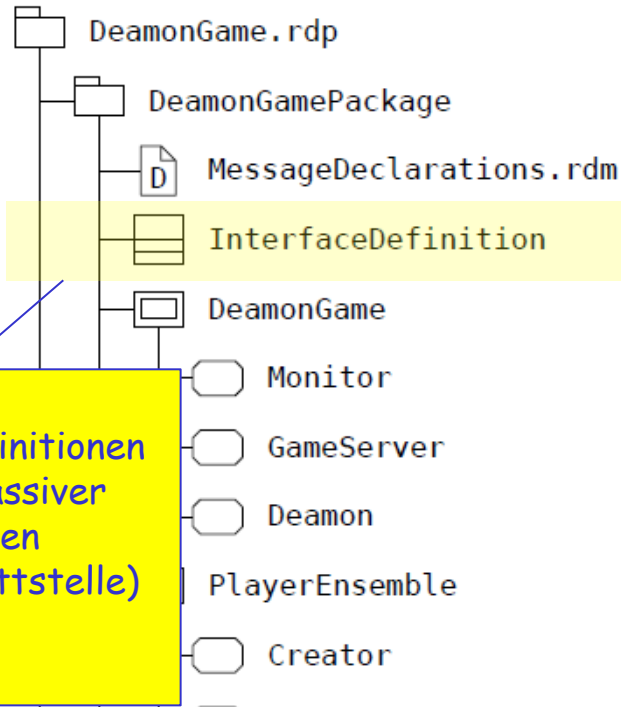
Definition von Nachrichtentypen  
und  
Definition von Nachrichtenlisten

## SDL-RT-Besonderheiten

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

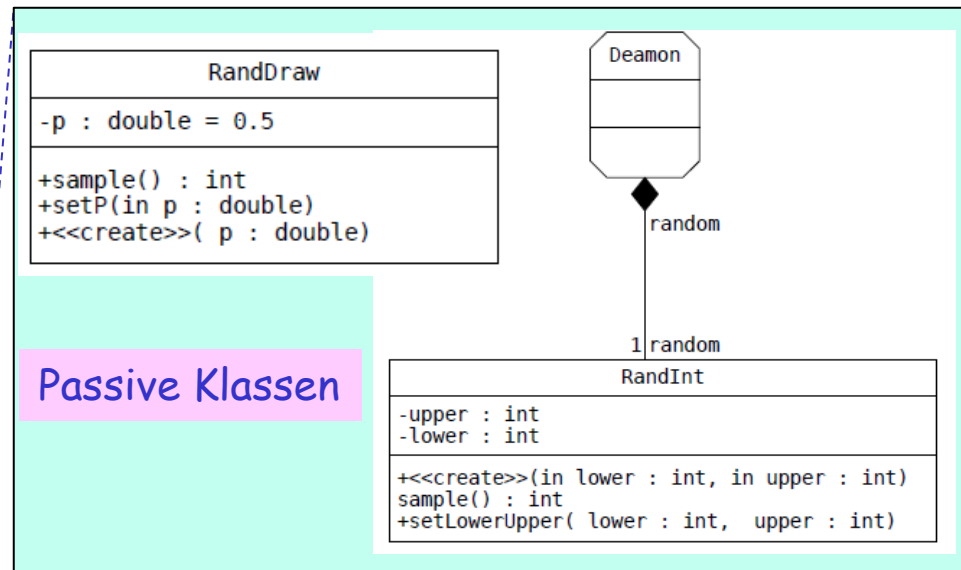
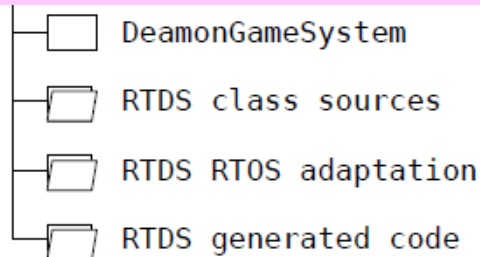
PId = RTDS\_QueueId  
SIGNAL = MESSAGE

# Package

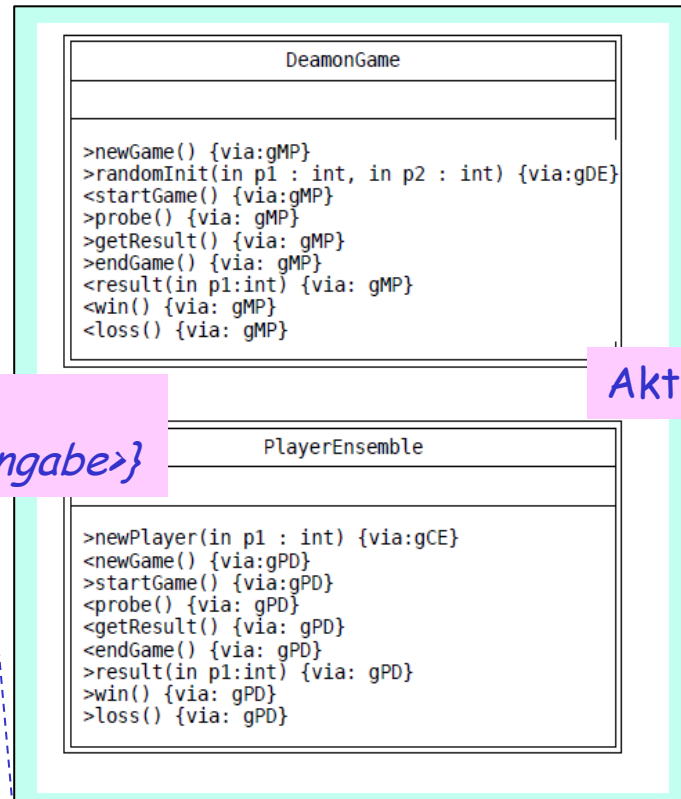


besteht aus UML-like-Definitionen aktiver und passiver SDL-RT-Klassen (äußere Schnittstelle) und deren Assoziationen

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

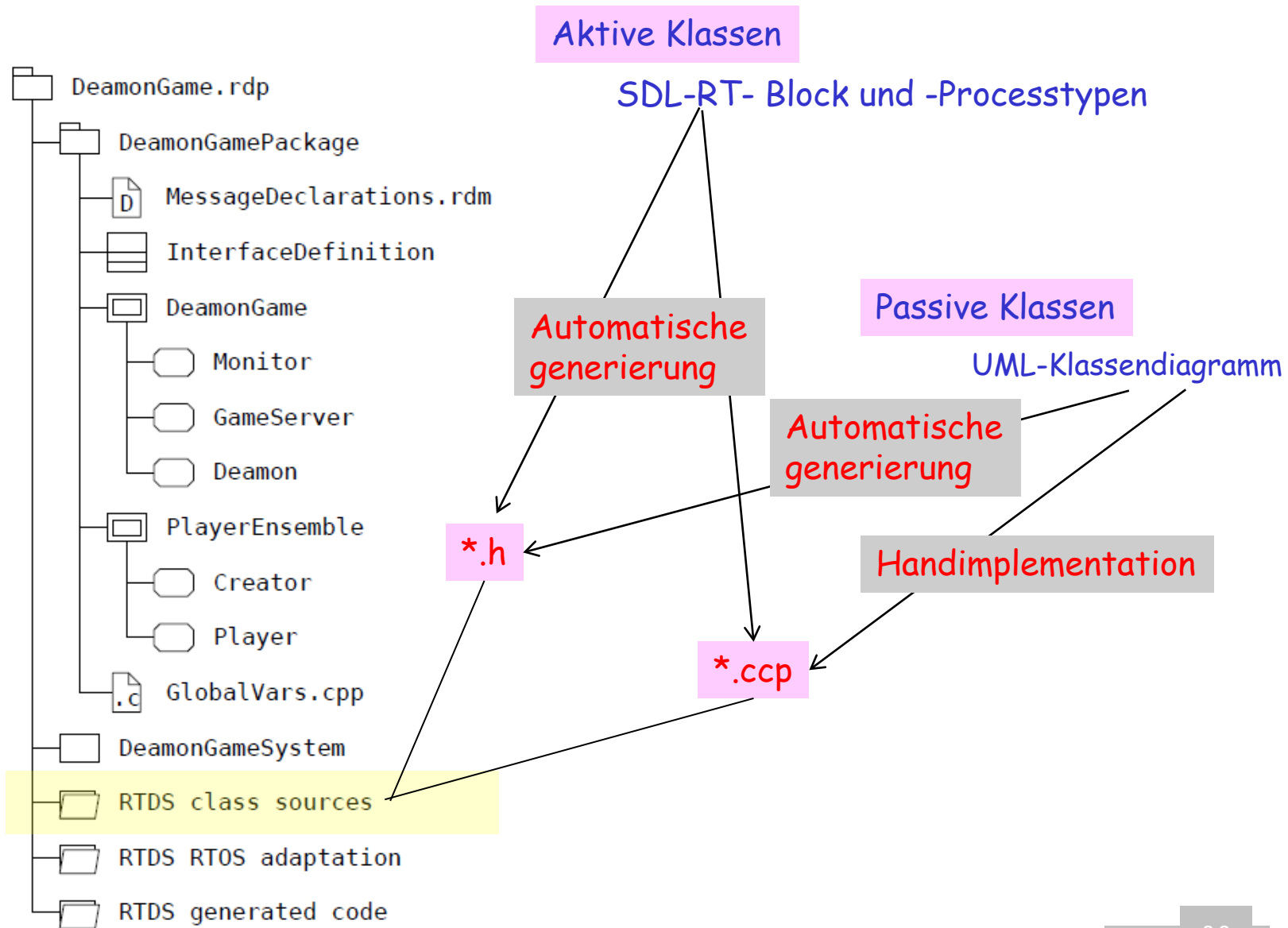


Passive Klassen



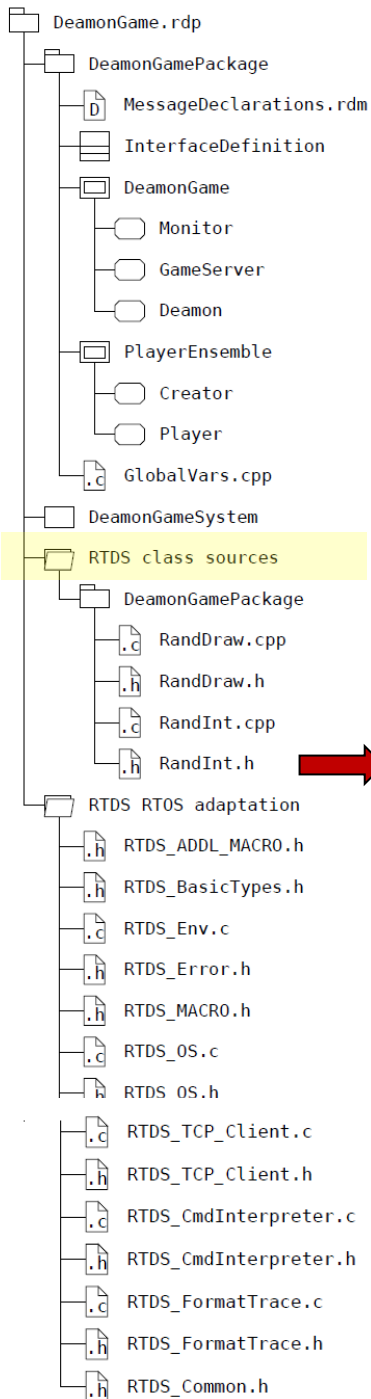
Aktive Klassen

# Package





# 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;
 */

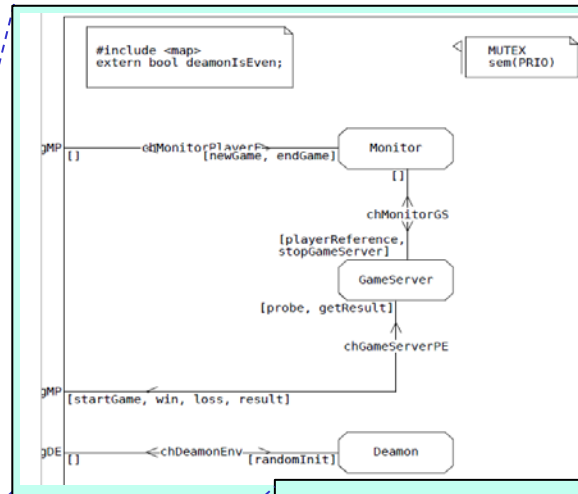
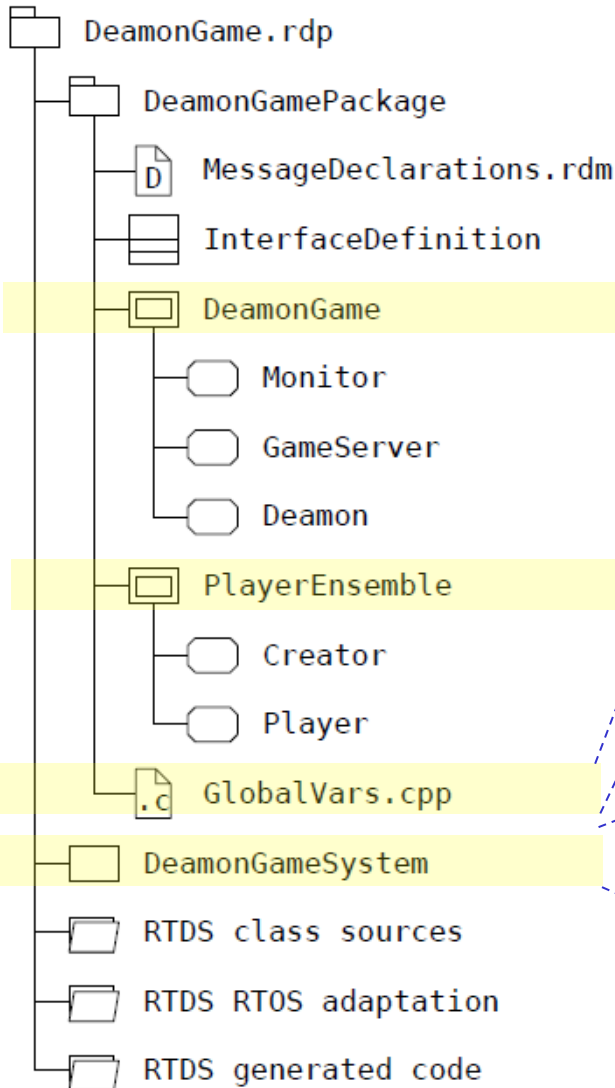
// PUBLIC OPERATIONS:
// =====

// 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);
}

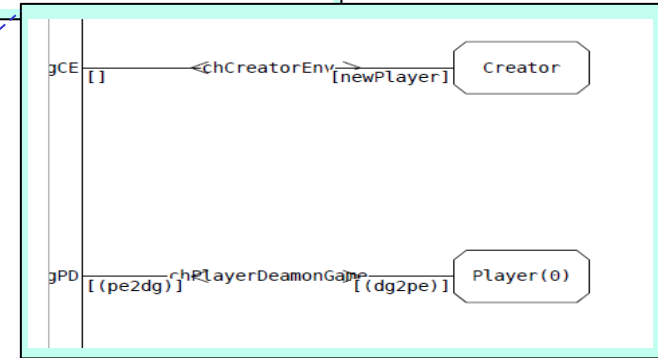
// Operation setLowerUpper:
// -----
void RandInt::setLowerUpper(int lower, int upper)
{
    this->lower = lower;
    this->upper = upper;
}
```

# Blocktypen



Blocktyp DeamonGame  
Name = Dateiname

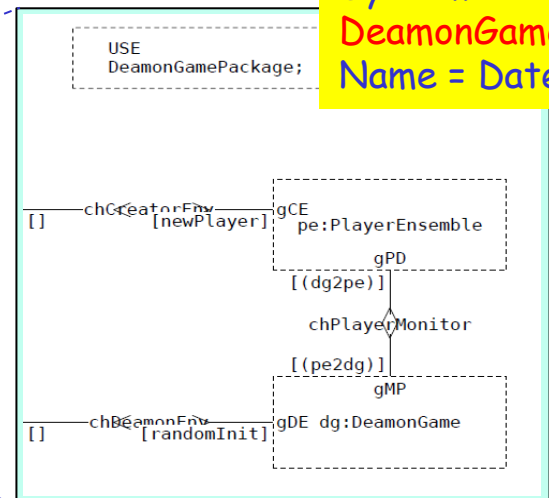
Blocktyp PlayerEnsemble  
Name = Dateiname



System DeamonGameSystem  
Name = Dateiname

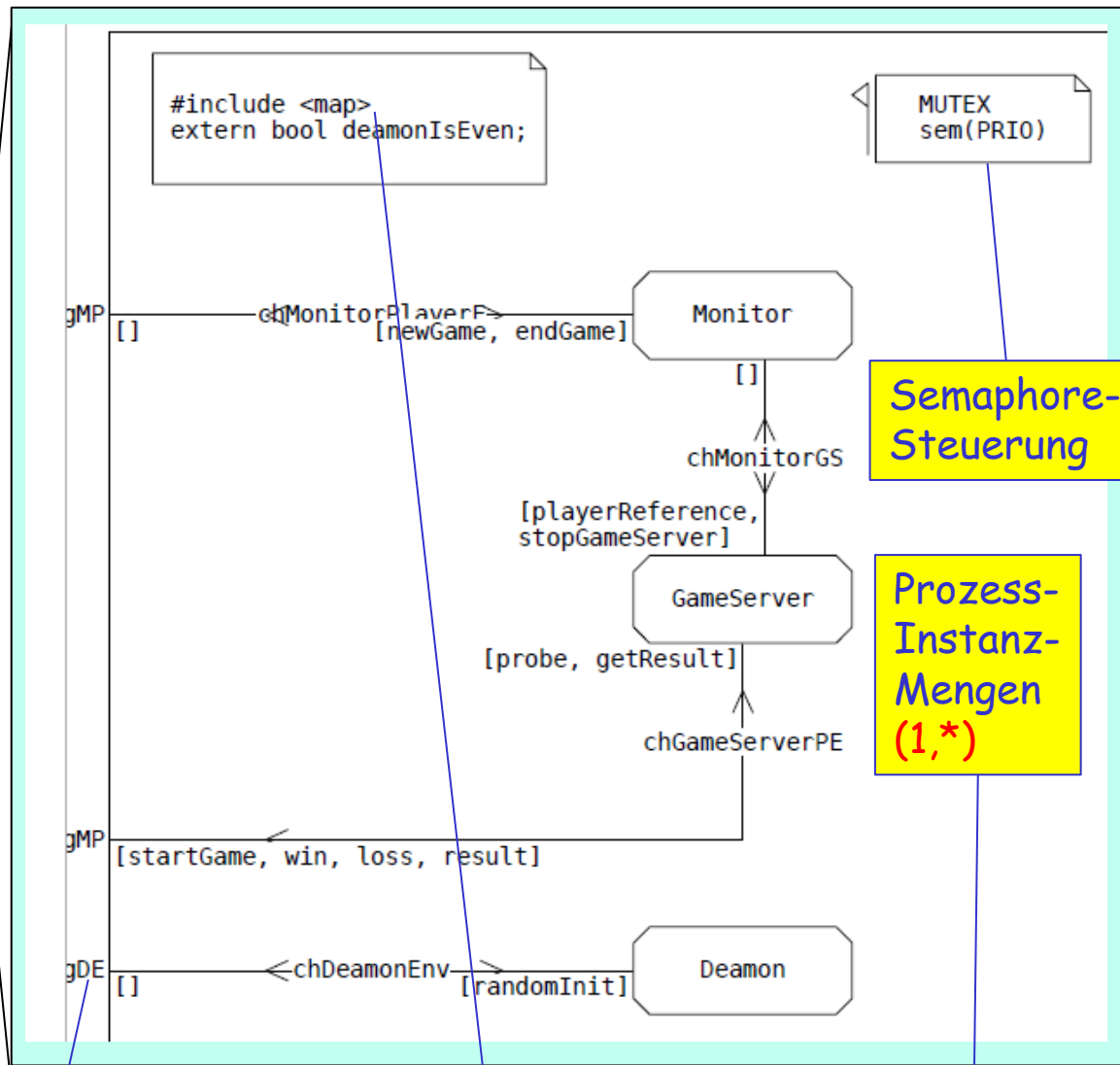
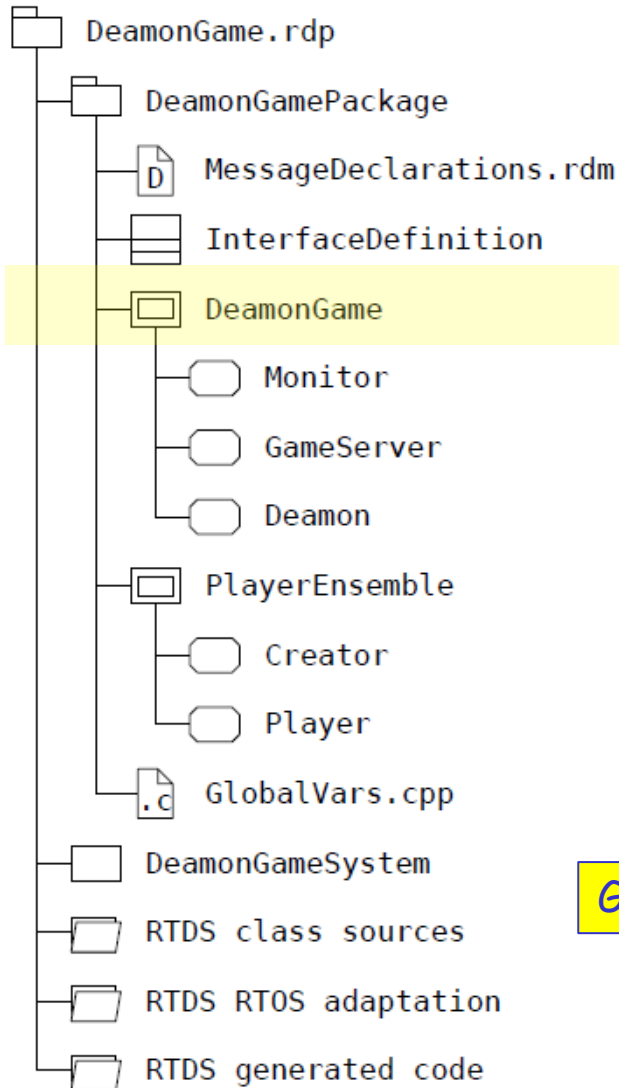
```
bool daemonIsEven = true;
```

C/C++ Erweiterung globale Variable



SDL-Einführung

# Blocktyp DeamonGame



Semaphore-Steuerung

Prozess-Instanz-Mengen (1,\*)

Gate-Angabe

C++ StandardTemplateLibrary

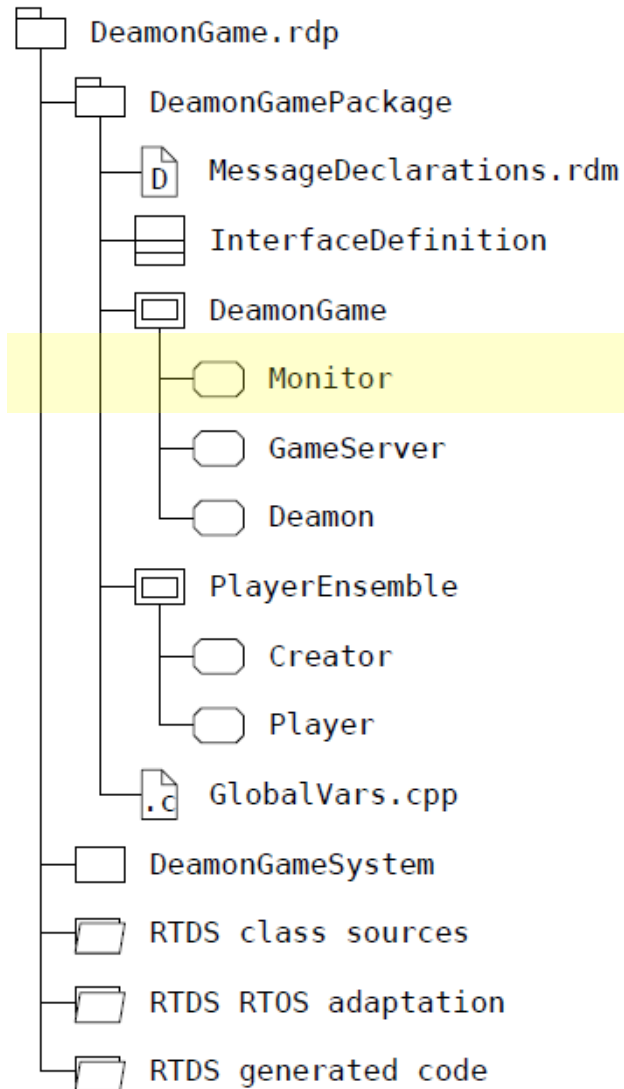
besser:  
 Monitor(1,1)  
 GameServer(0,\*)  
 Daemon(1,1)

# C++ *Standard Template Library*

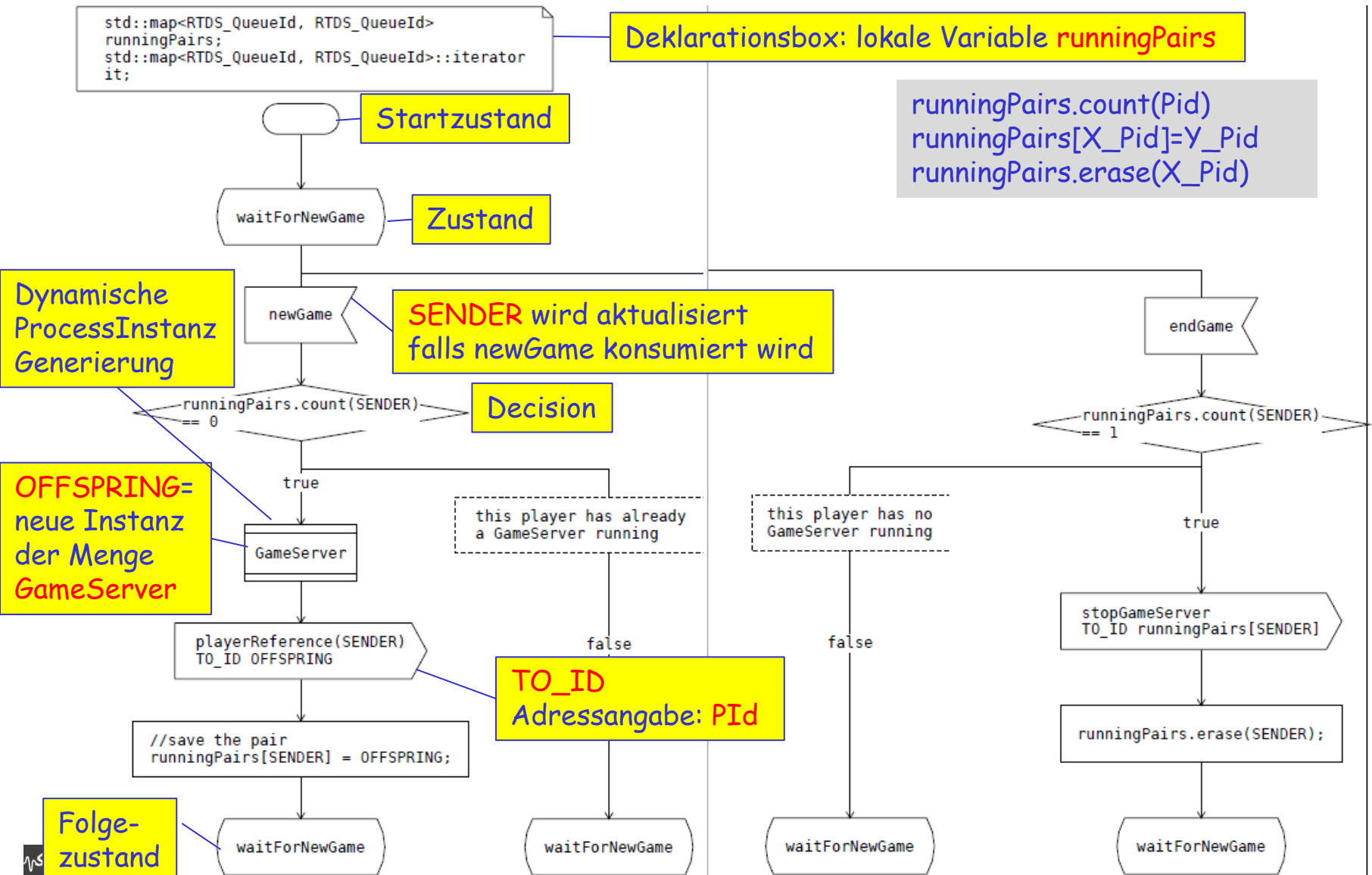
C++ Maps are sorted associative containers that contain unique key/value pairs. Maps are sorted by their keys.

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



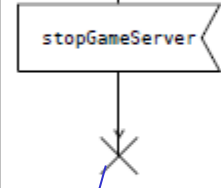
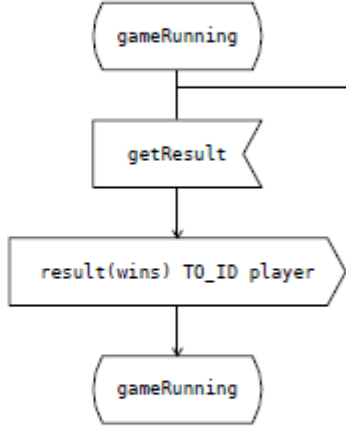
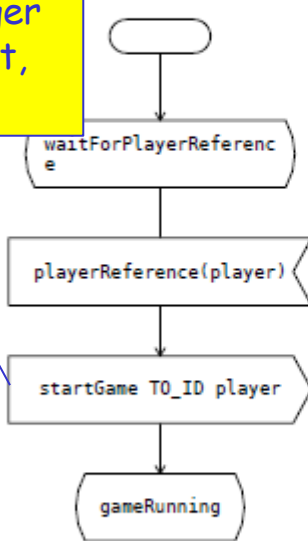
# Prototypinstanz der Processmenge GameServer

Deklarationsbox: Lokale Variable **player**: zugeordneter externer Spieler

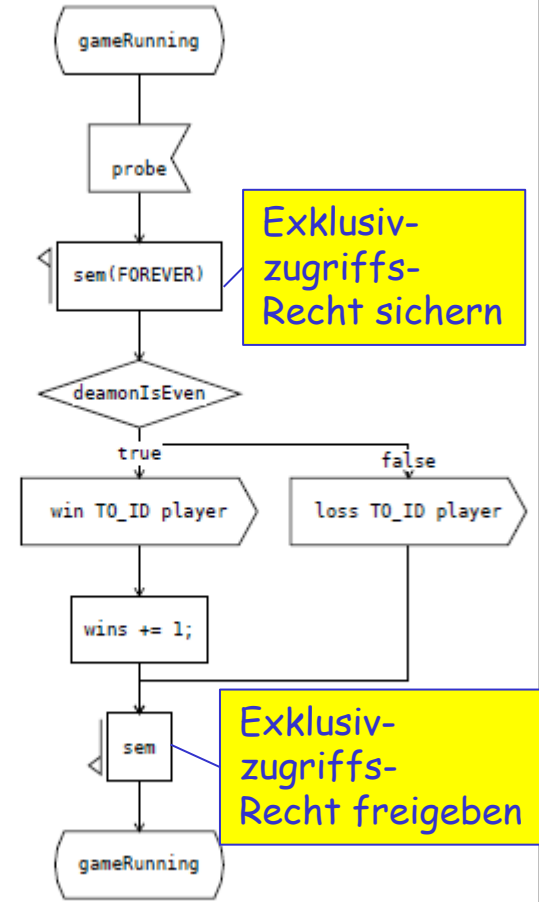
```
RTDS_QueueId player;
int wins = 0;
```

Addressierung:  
TO\_ID <Pid>

Pfad zum Empfänger  
wird vorausgesetzt,  
sonst Fehler



Vernichtung  
der  
ProcessInstanz



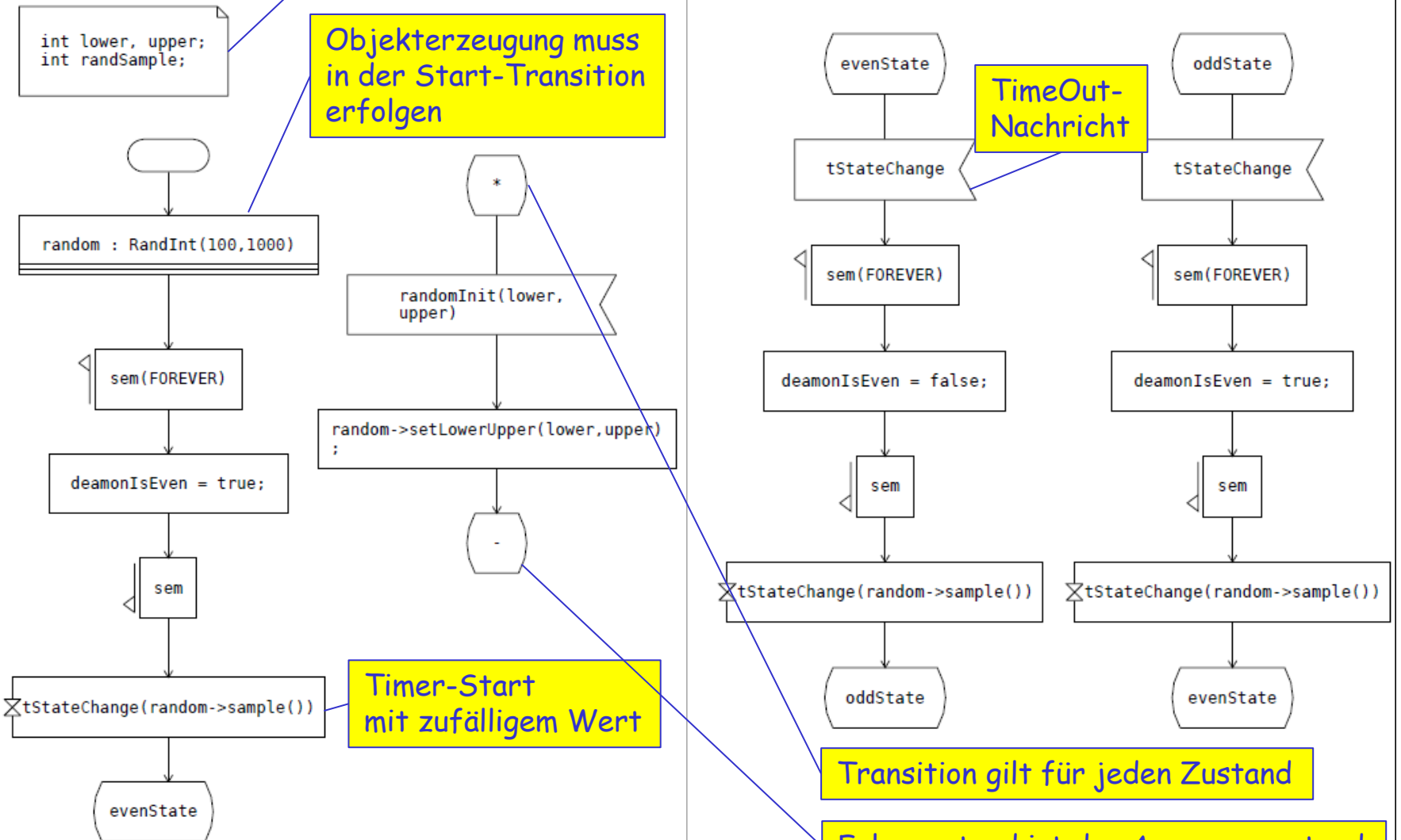
Exklusiv-  
zugriffs-  
Recht sichern

Exklusiv-  
zugriffs-  
Recht freigeben

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

# Prototypinstanz der Processmenge Deamon

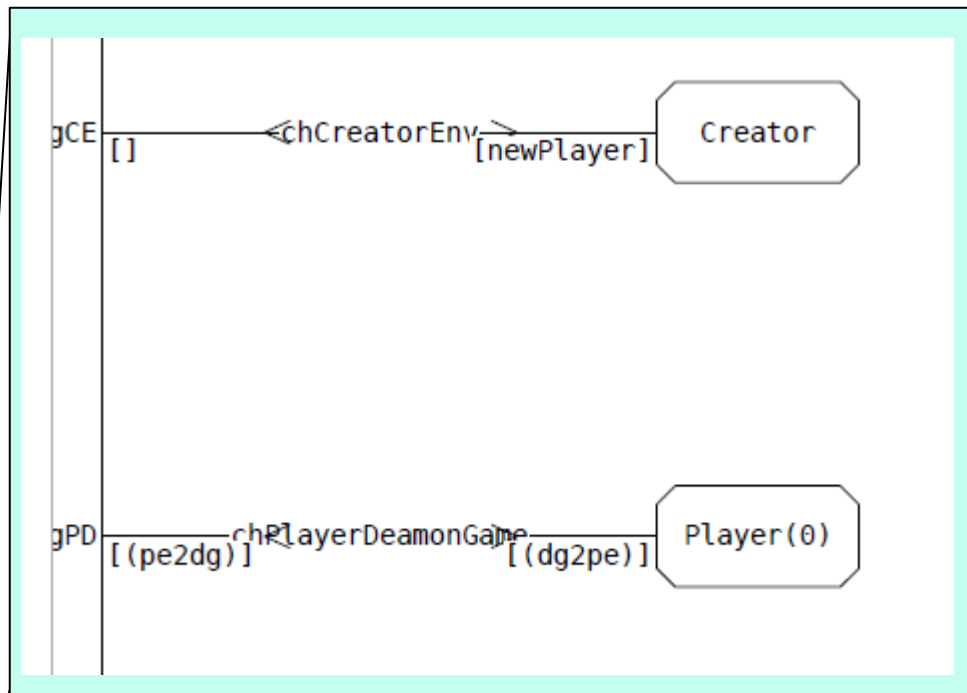
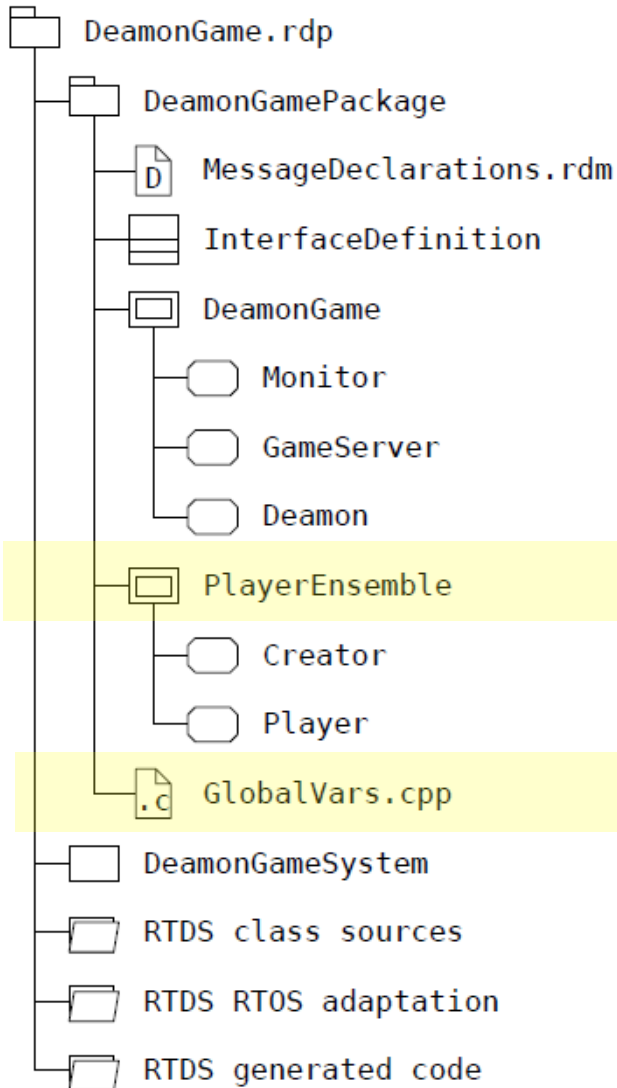
Deklarationsbox: Lokale Variable `lower`, ...`random` sind als Assoziationsende bereits schon Attribute



Timervariablen werden in SDL-RT implizit deklariert



# Blocktyp PlayerEnsemble



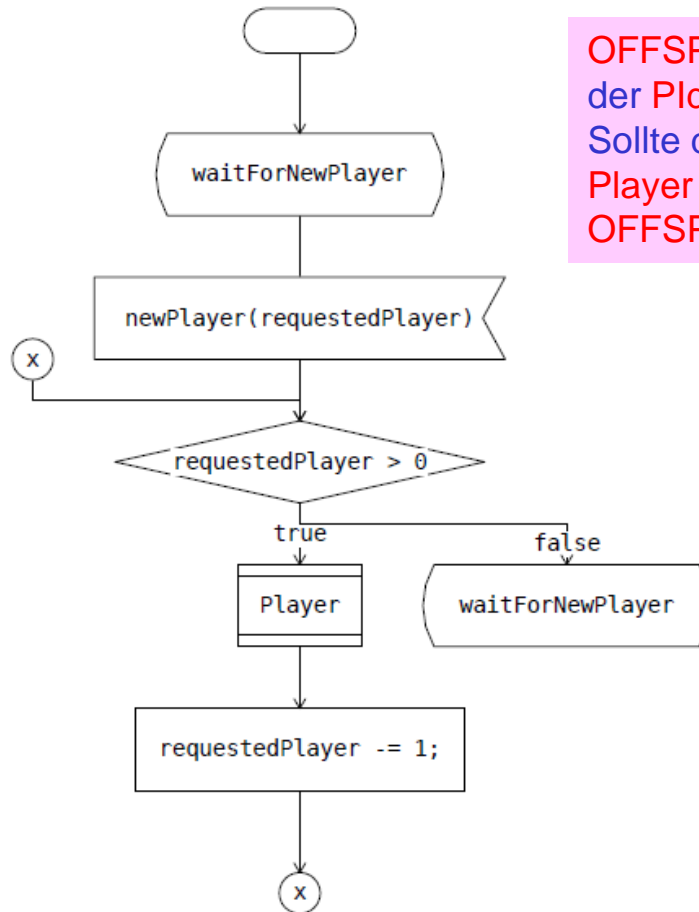
`bool daemonIsEven = true;`

Verbindung zur Semaphore-Variablen  
ist nutzerseitig zu organisieren

# Prototypinstanz der Processmenge Creator

```
int requestedPlayer;
```

Deklarationsbox: Lokale Variable `requestedPlayer`



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

## vordefinierte Readonly-Attribute

- SELF (Selbstreferenz, gesetzt mit Prozessinstanz-Existenz)
- PARENT (Erzeuger-Prozess: Null, falls statisch vorhanden)
- OFFSPRING (letzte generierte Process-Instanz oder Null)
- SENDER (Sender-Referenz, der zuletzt konsumierten Empfangsnachricht)

geschlossene Verbindung wäre (falls möglich) einer Konnektoren vorzuziehen

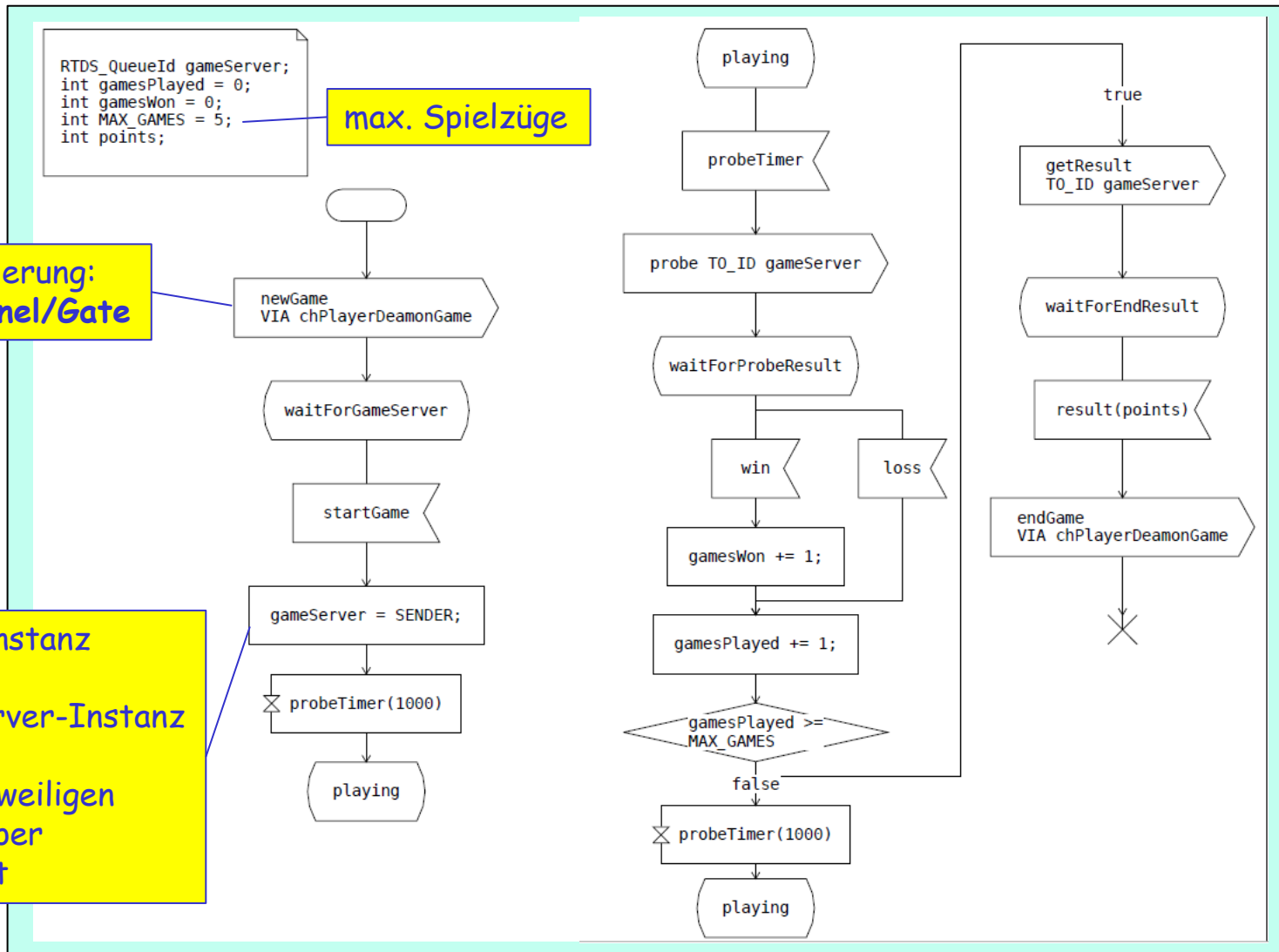
# Prototypinstanz der Processmenge Player

```
RTDS_QueueId gameServer;
int gamesPlayed = 0;
int gamesWon = 0;
int MAX_GAMES = 5;
int points;
```

max. Spielzüge

Addressierung:  
via Channel/Gate

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



# 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**)

## 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

# Prozess-Lebenszyklus (Schema)

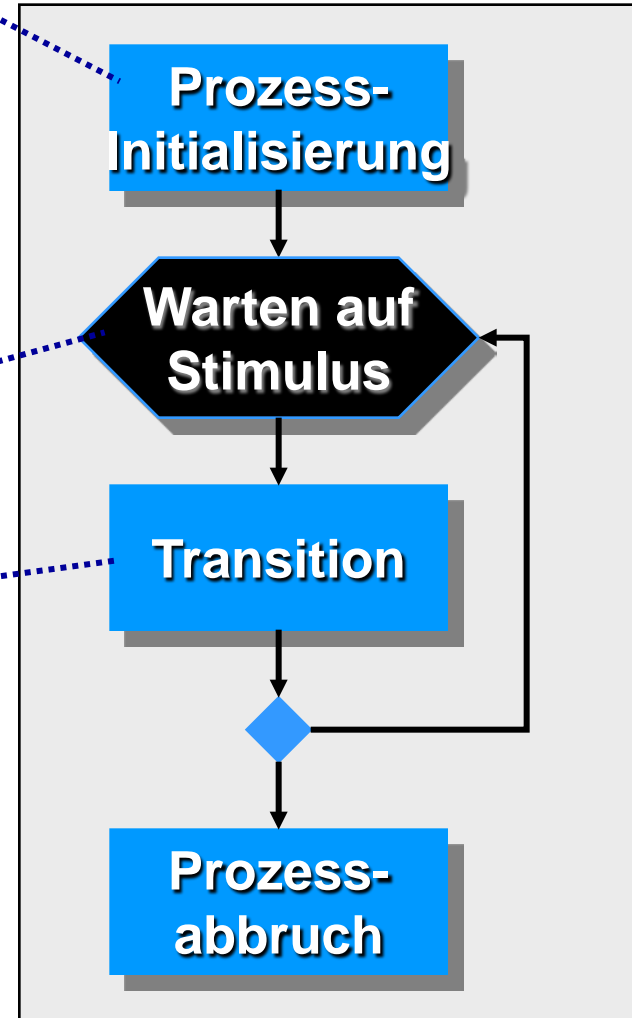
mit Existenzbeginn

- 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.



# Konfiguration der SD-Ausgabe (MSC)

The screenshot displays the RTDS (Real Time Developer Studio) interface. The main window is titled "RTDS - Diagram 'Testlauf' (modified)". Overlaid on this are two windows:

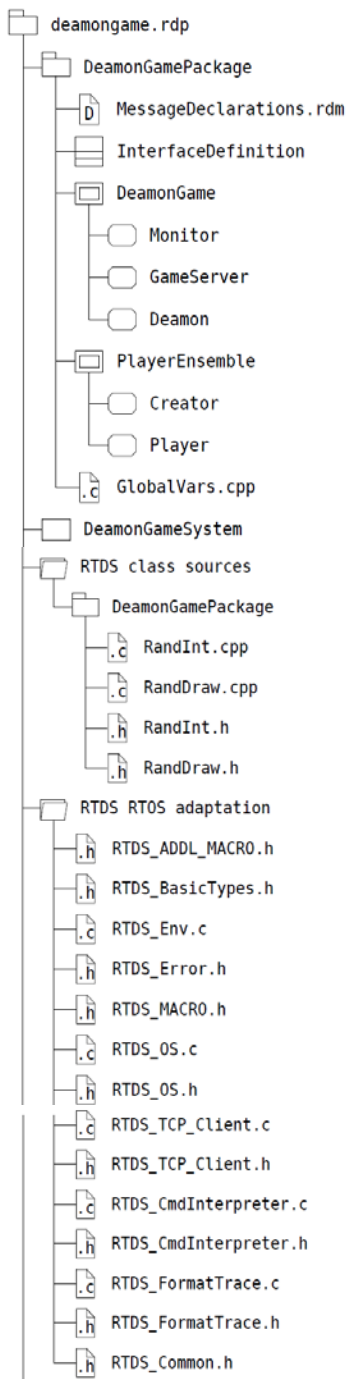
- SDL-RT debugger:** This window is used for debugging. It features a toolbar with various icons, a "Process information" table, and sections for "Watch variables" and "Local variables". A yellow arrow points from the "Process information" table to a yellow callout box.
- MSC configuration:** This window allows for configuring Message Sequence Charts (MSC). It lists "Available agents" and "Traced agents". The "Available agents" list includes: DeamonGameSystem, DeamonGame, Monitor, GameServer, Deamon, PlayerEnsemble (highlighted in blue), Creator, Player, and RTDS\_Env. The "Traced agents" list includes: GameServer and Monitor. Checkboxes for "Show time indicators" and "Record message data" are checked. "OK" and "Cancel" buttons are at the bottom.

A yellow callout box with the text "Standardmäßig werden alle System-Process-Instanzen und alle Nachrichten erfasst" (By default, all system process instances and all messages are captured) points to the "Process information" table in the debugger window.

Name	Prio	SDL id	RTOS id	Msg	SDL-RT state	System state

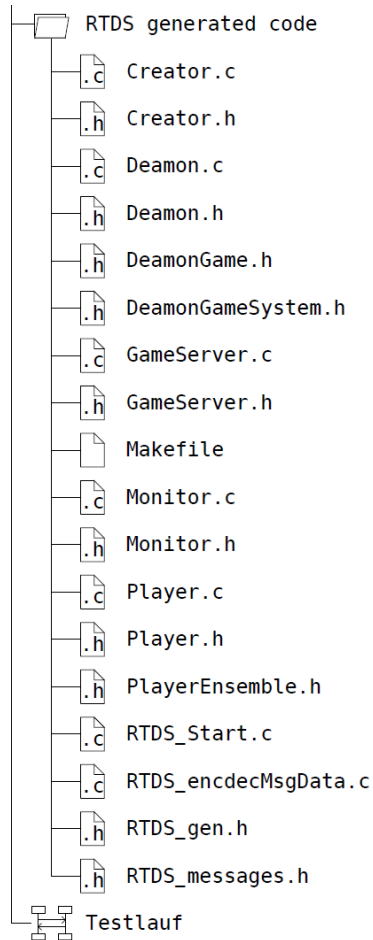
```
Real Time Developer Studio shell
>Free run off.
>
```

Debugger state: STOPPED | Active thread:



# DeamonGame-Simulator

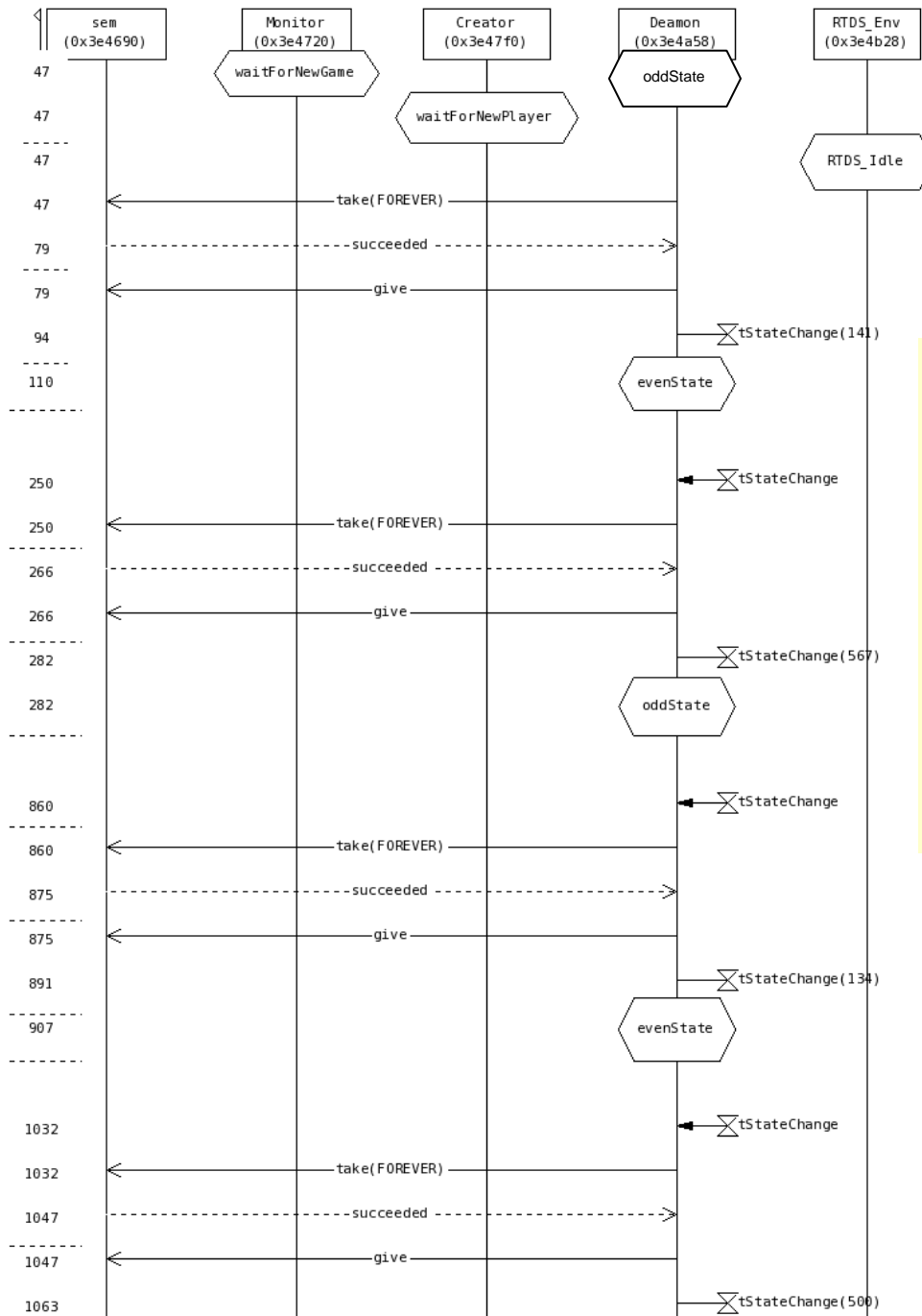
## Quellen



## Run

1. Erzeugung des Simulators  
→ neue Bedienoberfläche
2. Vorbereitung der Simulatoreingabe  
(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



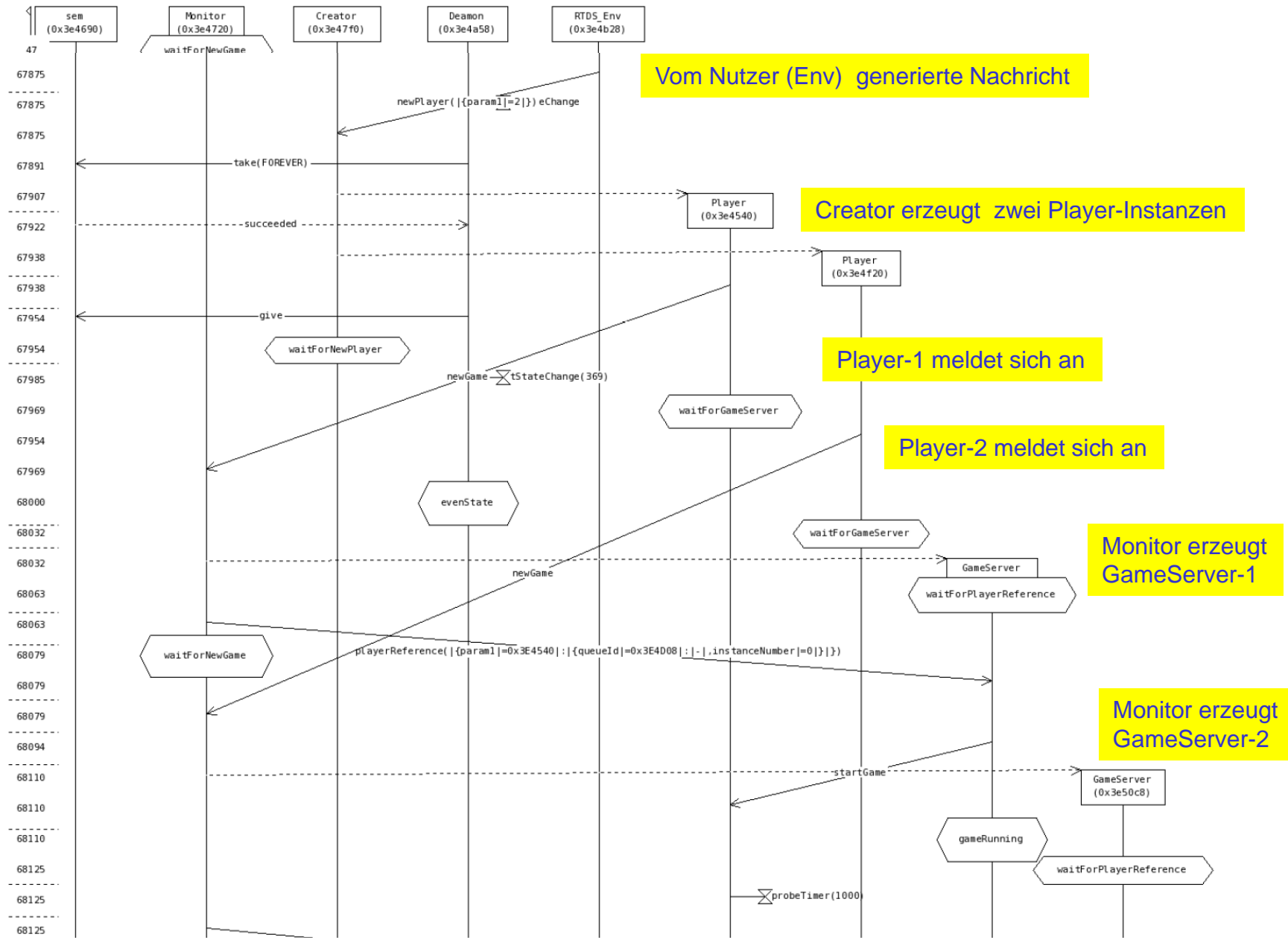


Noch keine Player- und GameServer-Instanzen

Monitor und Creator verharren

Daemon wechselt stochastisch seinen Grundzustand

- ### Run (Fortsetzung)
- Erzeugung einer Nachricht **newPlayer(2)** mit **Creator** als Empfänger-Prozessinstanz → 2 Spieler nehmen damit am Spiel teil
  - Unterbrechung mit Pause



## 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

# *Einsatzfelder von PragmaDev*

- Anforderungsspezifikation
- Third-Party-Tool-Integration (z.B. CVS)
- OOA
- Systemarchitektur
- Spezifikation eines verteilten Systems
- Code-Generierung
- Debugger (SDL, generierter Code)
- Graphische Trace-Ausgaben
- Konformität von Simulation und Anforderungsspezifikation

***Demo-Video: [www.pragmadev.com](http://www.pragmadev.com)***

# Projektbeschreibung

The image shows a screenshot of a software development environment. On the left, a file explorer window titled 'RTDS - Project "pingpong.rdp"' displays a tree view of project files and folders, including 'pingpong.rdp', 'Spec.doc', 'UseCase', 'play', 'myInclude', 'myPackage', 'RTDS class sources', 'sTableTennis', and 'dist'. The 'Spec.doc' file is highlighted. In the foreground, a Microsoft Word window titled 'Spec.doc - Microsoft Word' is open, displaying a document with the following content:

Detailed technical specification Monday, the 1th

**Detailed technical specification**

Date : Monday, the 1th of May 2004  
Author : John Smith  
Summary :

At the bottom of the Word window, a status bar indicates 'Spec.doc: 592 caractères (valeur approximative)'.

A callout box with a speech bubble icon contains the text: 'Every project starts with this kind of documents.'

At the bottom of the overall image, there is a footer: 'UML-2 - SDE-Einführung' and a page number '9.29' in a grey box.

# Use-Case-Diagramme

UML-Möglichkeiten eingeschränkt:

- keine Abhängigkeit
- der Akteure und
- der UsesCases untereinander

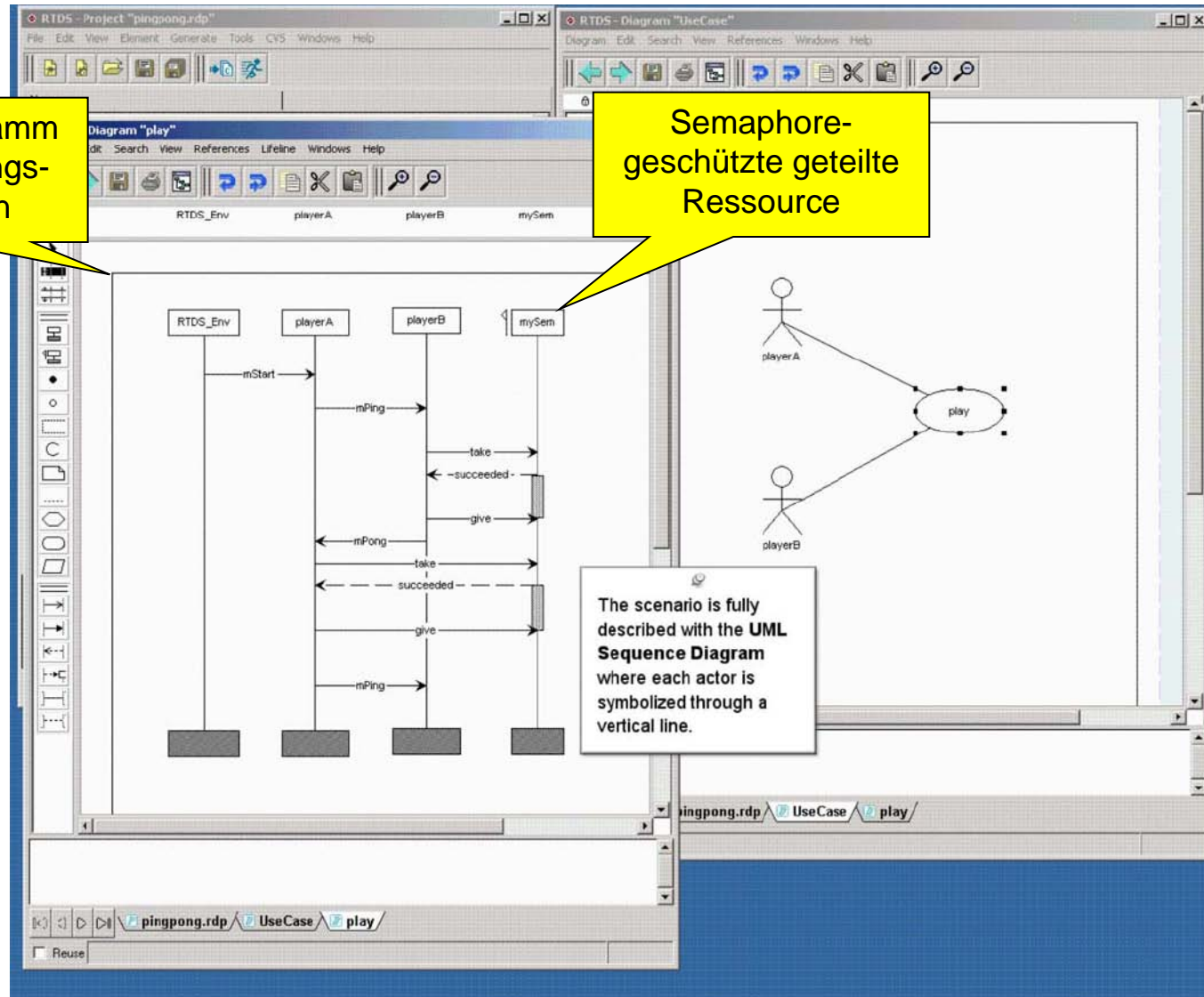
Further analysis of the requirements can be done with the UML Use Case diagram.

playerA and playerB actors are involved in the play scenario.

# Sequenzdiagramme

Sequenzdiagramm  
als Anforderungs-  
spezifikation

Semaphore-  
geschützte geteilte  
Ressource



The scenario is fully  
described with the **UML  
Sequence Diagram**  
where each actor is  
symbolized through a  
vertical line.

# Externe Datentypen

The image shows a screenshot of an IDE with two windows. The left window displays a project tree for 'pingpong.rdp' with folders like 'myInclude' and 'myGlobal.c'. A yellow callout points to 'myInclude' with the text 'Paket als Projekt-Komponente (hier: C-Dateien)'. The right window shows the code for 'myTypes.h', which defines an enum 'tPosition' and a struct 'tRecord'. A yellow callout points to the code with the text 'generiertes Make-File berücksichtigt C-Dateien'. A central text box says 'Let's consider C data types including a structure and an enum.' The status bar at the bottom indicates the file path 'C:\SDL-RT\ViewLet\myTypes.h' and its size '294 bytes'.

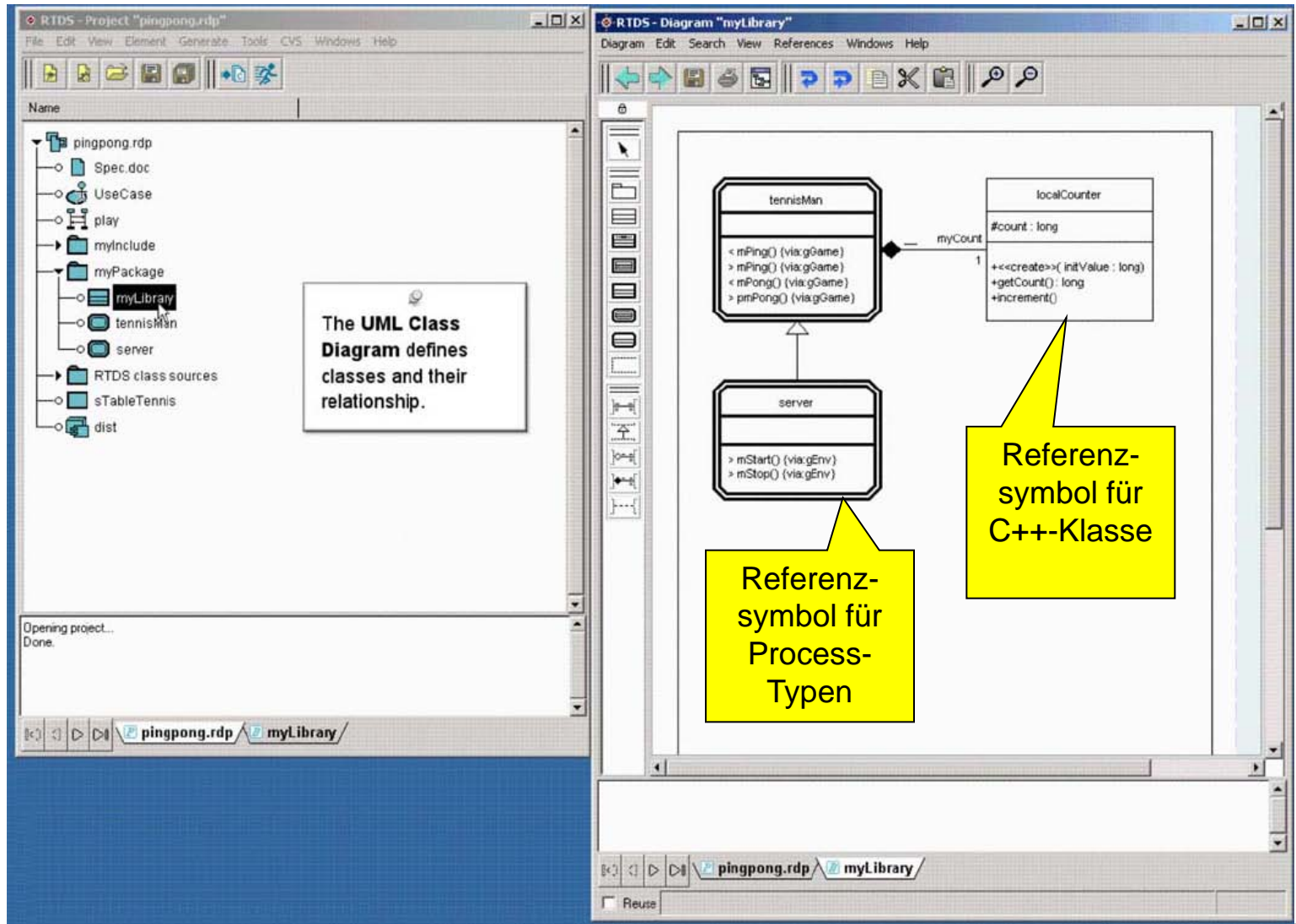
Let's consider C data types including a structure and an enum.

generiertes Make-File berücksichtigt C-Dateien

```
1  #ifndef _MYTYPES_H_
2  #define _MYTYPES_H_
3
4  extern int globalCounter;
5
6  enum tPosition
7  {
8      International,
9      National,
10     Local,
11     Beginner
12  };
13
14  typedef struct tRecord {
15     char *firstName;
16     char *lastName;
17     enum tPosition position;
18     int score;
19  } tRecord;
20
21 #endif _MYTYPES_H_
22
```



# Definition von aktiven und passiven Klassen



# Implementation der Klassenmethoden

per Mausklick wird Editor des cpp-Files geöffnet

Implementation of the passive class is written in straight C++ code.

```
1 #include "myPackage/localCounter.h"
2
3
4 // PUBLIC OPERATIONS:
5 // =====
6
7 // Operation localCounter:
8 // -----
9
10 localCounter::localCounter(long initValue)
11 {
12     this->count = initValue;
13 }
14
15 // Operation getCount:
16 // -----
17
18 long localCounter::getCount()
19 {
20     return this->count;
21 }
22
23 // Operation increment:
24 // -----
25
26 void localCounter::increment()
27 {
28     this->count += 15;
29 }
30
31
32
```

Stereotype <<create>> für Konstruktor

# Processtyp-Definition

Voraussetzung:  
UML-Klassendiagramm und  
SDL-Processtypdefinition  
befinden sich im selben Paket

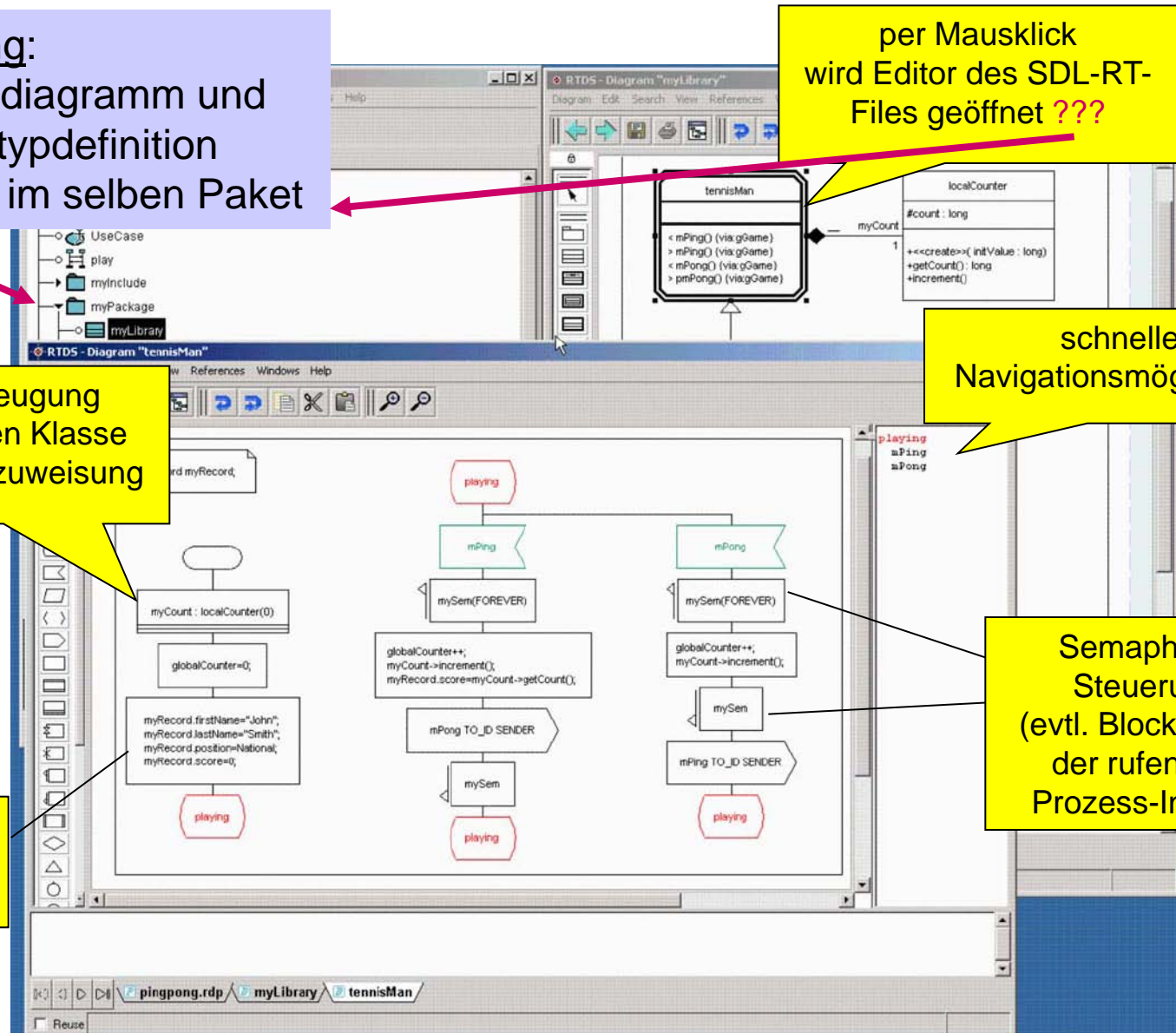
per Mausklick  
wird Editor des SDL-RT-  
Files geöffnet ???

schnelle  
Navigationsmöglichkeit

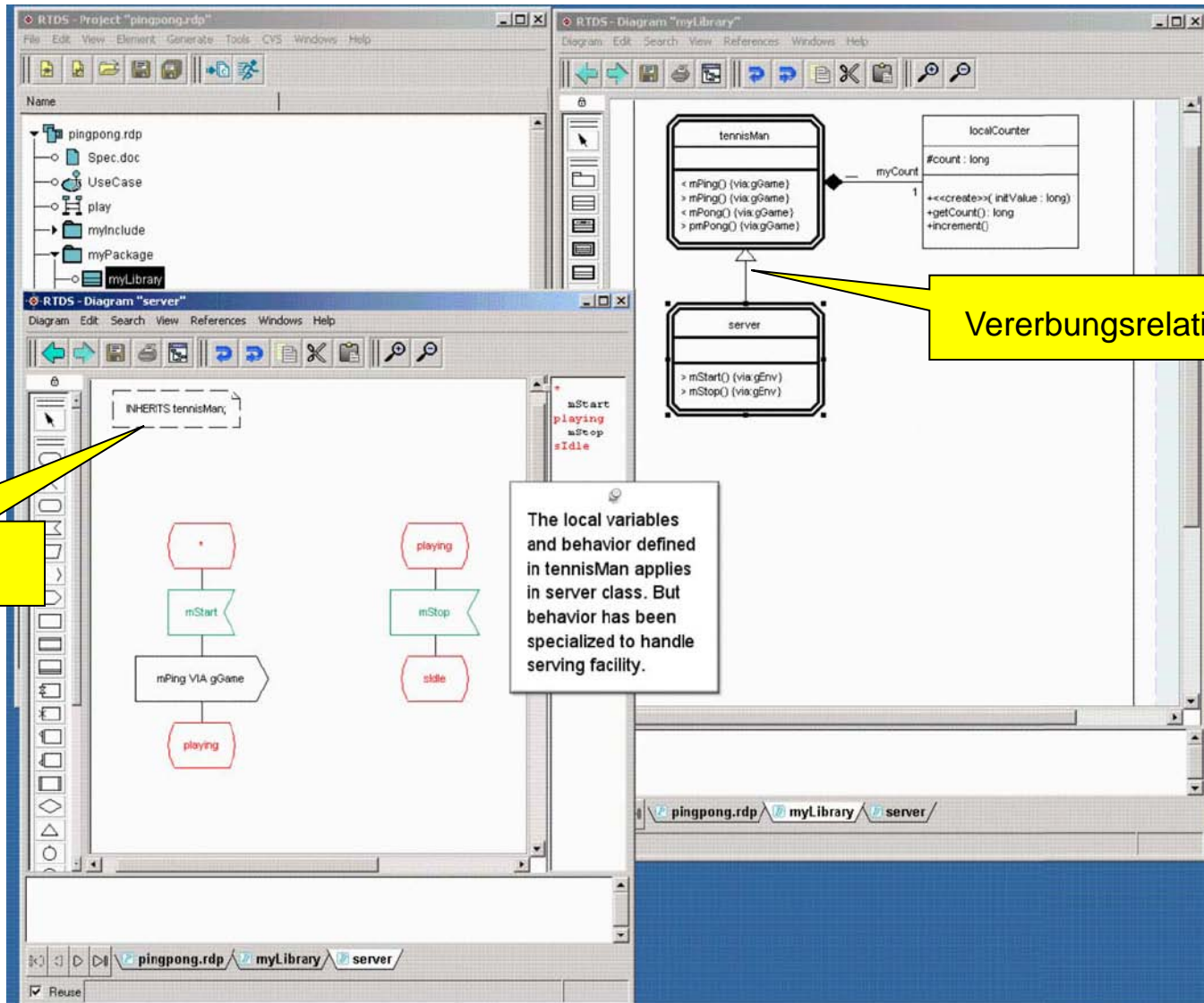
Objekt-Erzeugung  
einer passiven Klasse  
und Referenzzuweisung

Task:  
C++ Code

Semaphore-  
Steuerung  
(evtl. Blockierung  
der rufenden  
Prozess-Instanz)



# Vererbung von Process-Typen

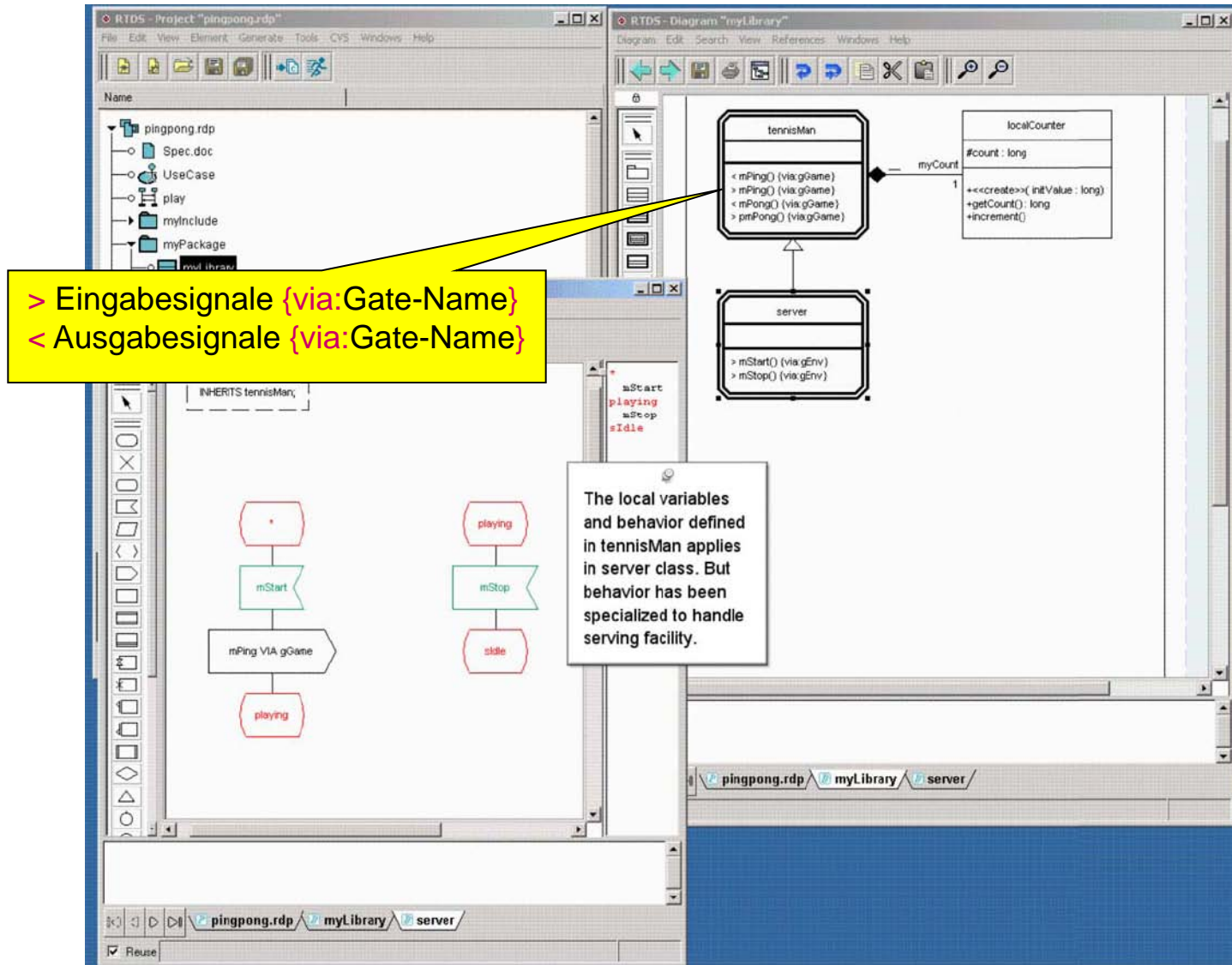


Basistyp

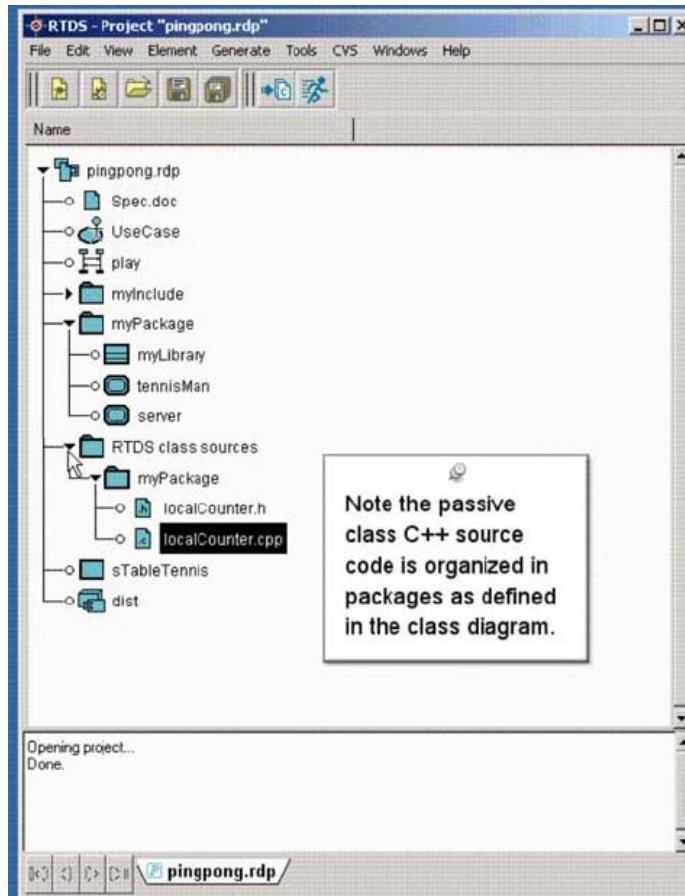
Vererbungsrelation

The local variables and behavior defined in tennisMan applies in server class. But behavior has been specialized to handle serving facility.

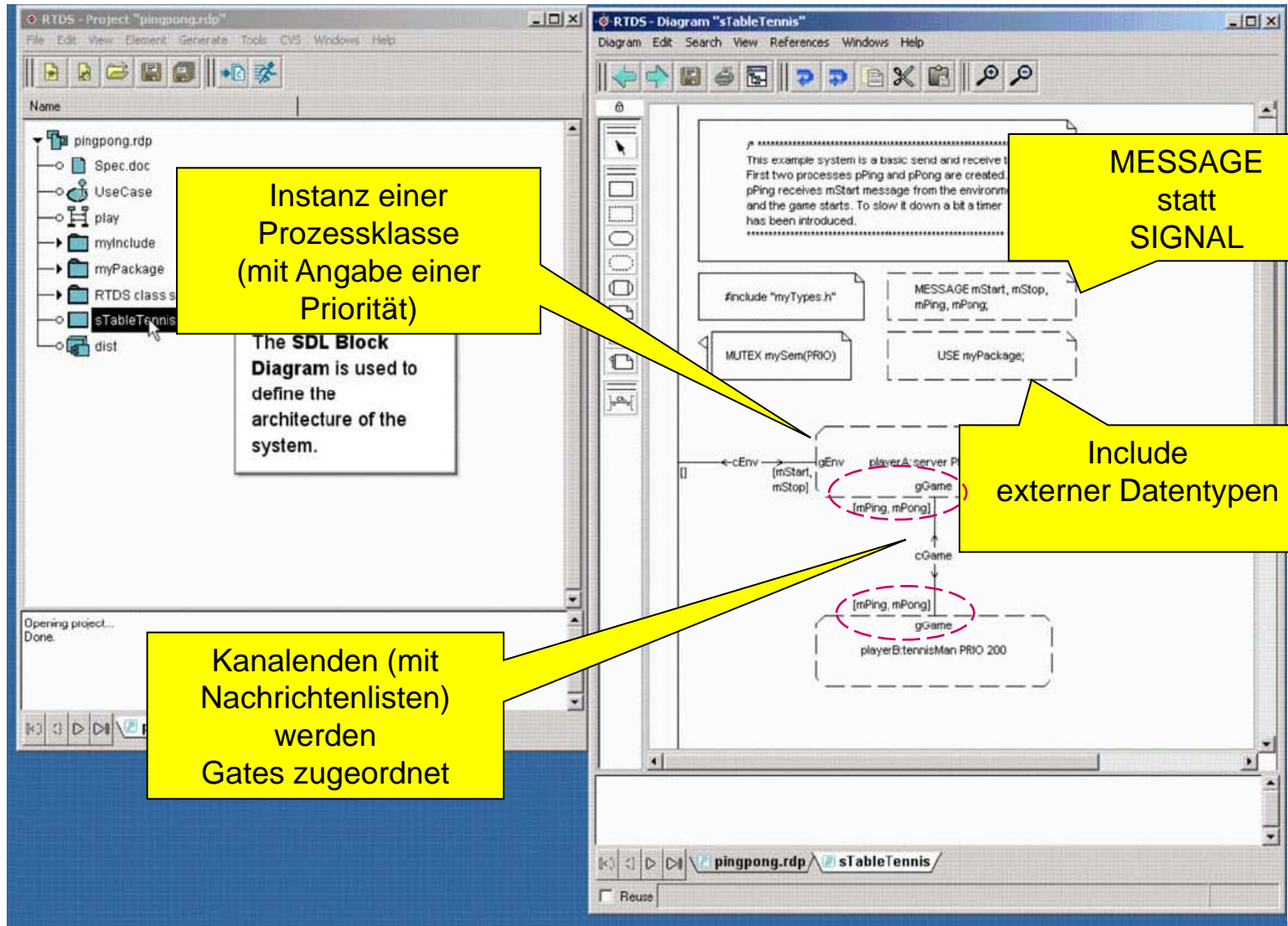
# Gates: Interfacedefinition aktiver Klassen



# Quelldateien der passiven Klassen



# Definition der Systemarchitektur



# Verteilungsdiagramm: Knoten, Softwarekomponenten

