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

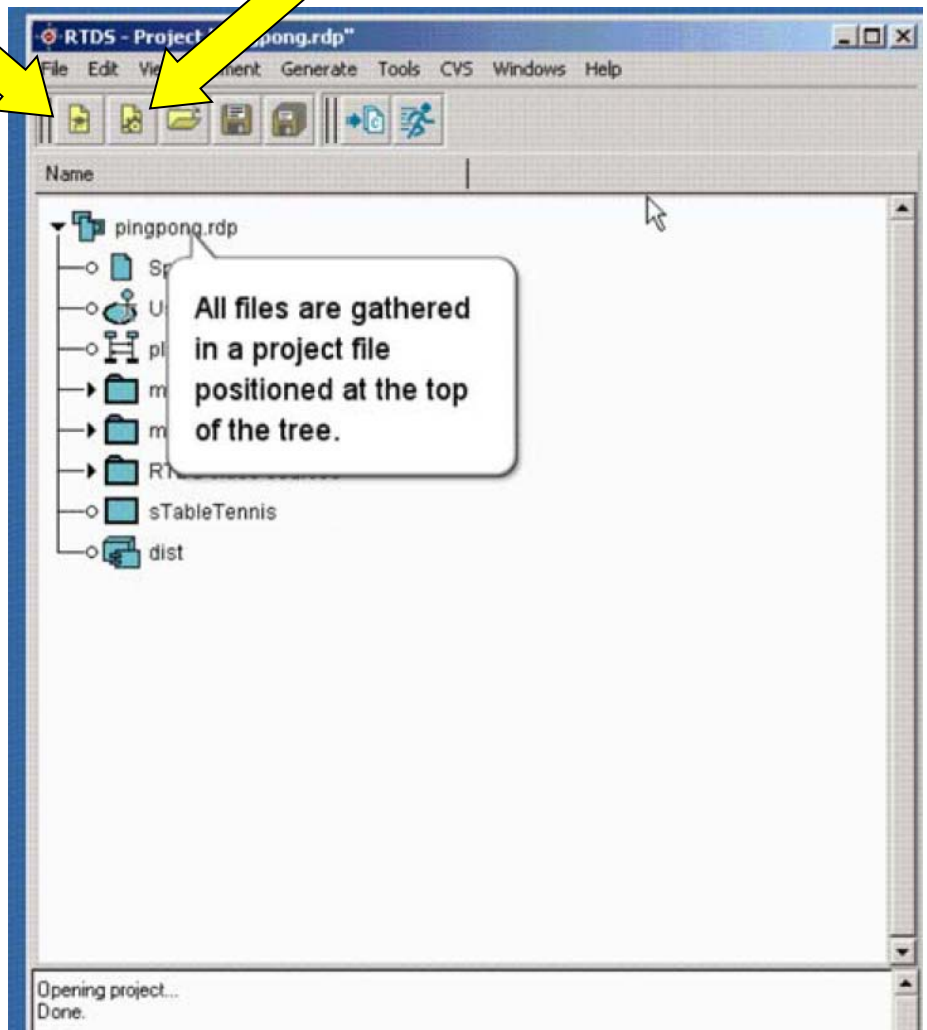
6. SDL-Einführung

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

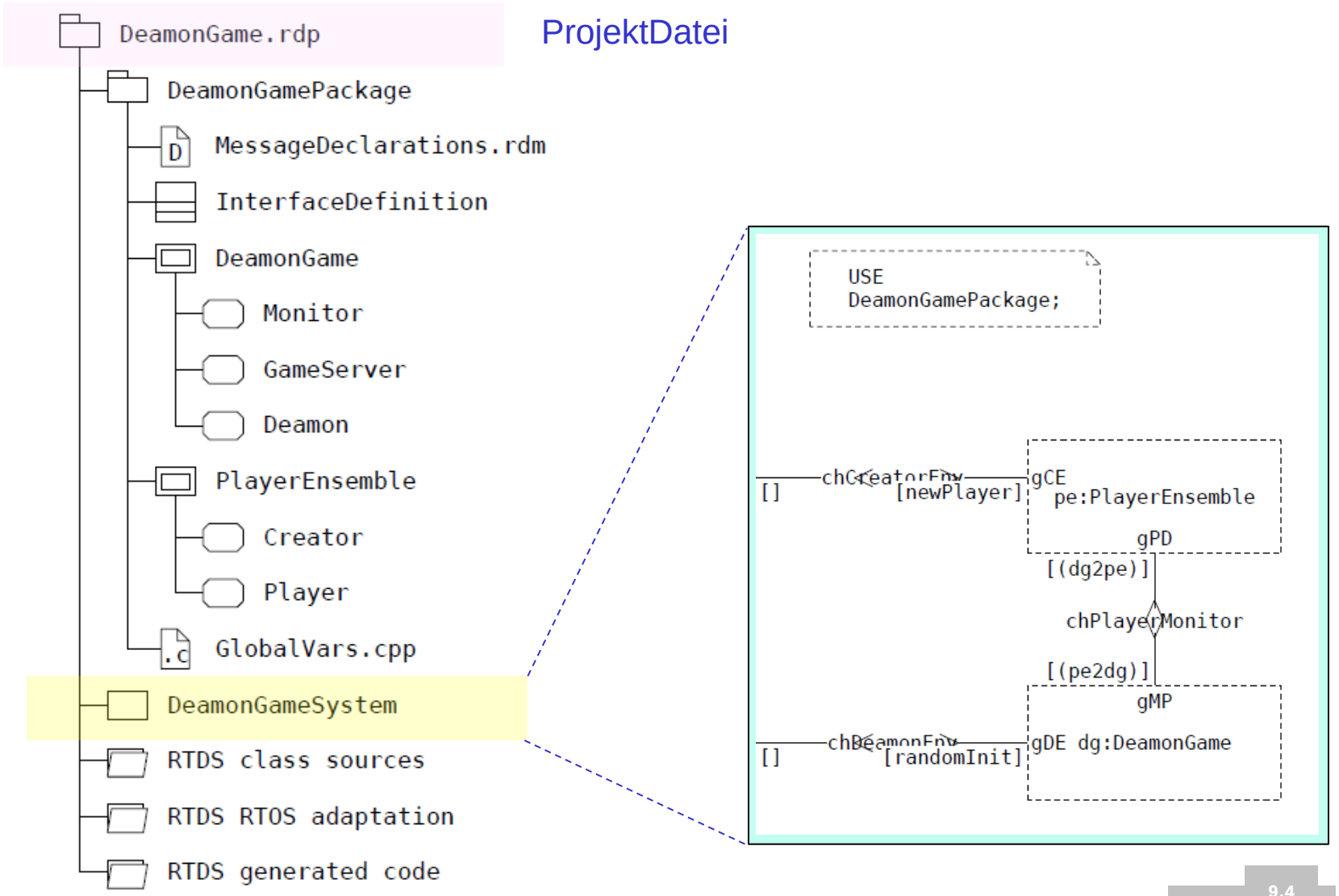
PragmaDev DS- Project Manager

SDL-RT

SDL-Z100



DeamonGame-Projekt



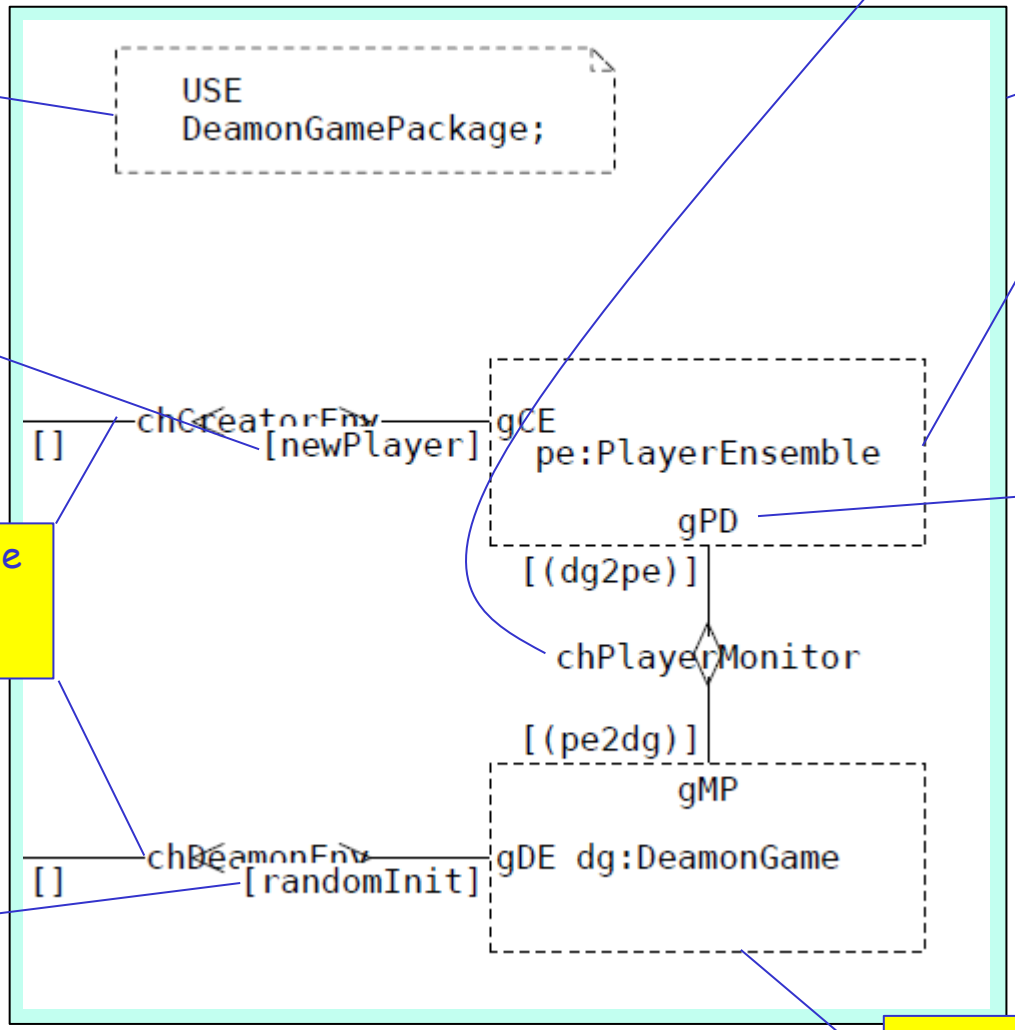
Systemname entspricht Dateinamen/Projektnamen

Systemspezifikation

bidirektionalerKanal chPlayerMonitor
mit Angabe zu transportierender Nachrichten

Package-Import

Systemgrenze



Block pe
vom Typ
PlayerEnsemble

Nachricht
newPlayer
von der
Umgebung

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

unidirektionale Kanäle
chCreatorEnv
chDeamonEnv

Nachrichten-Listen
dg2pe
pe2dg

entweder lokal
oder im Paket
defniert

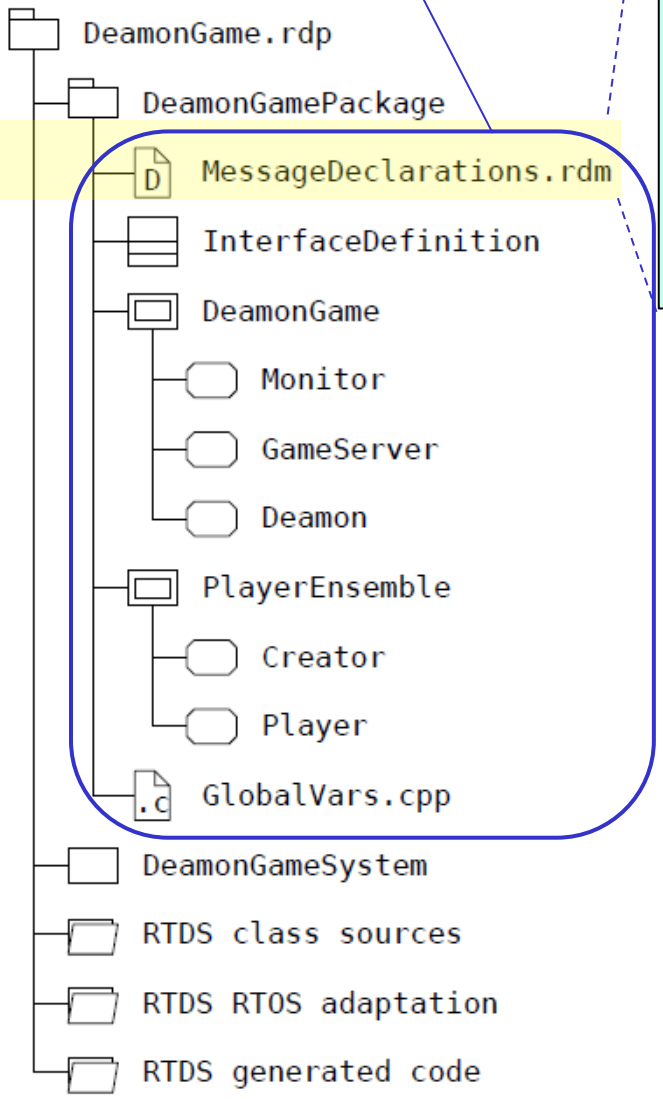
Nachricht
randomInit
von der
Umgebung

entweder lokal
oder im Paket
defniert

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;
```

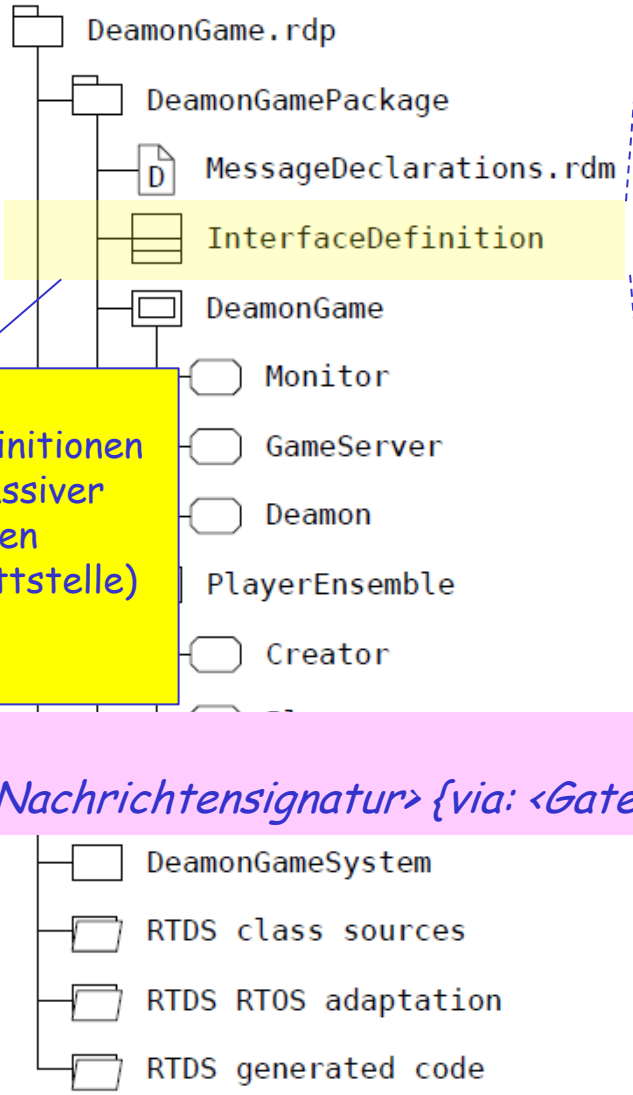
Definition von Nachrichtentypen
und
Definition von Nachrichtenlisten

SDL-RT-Besonderheiten

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

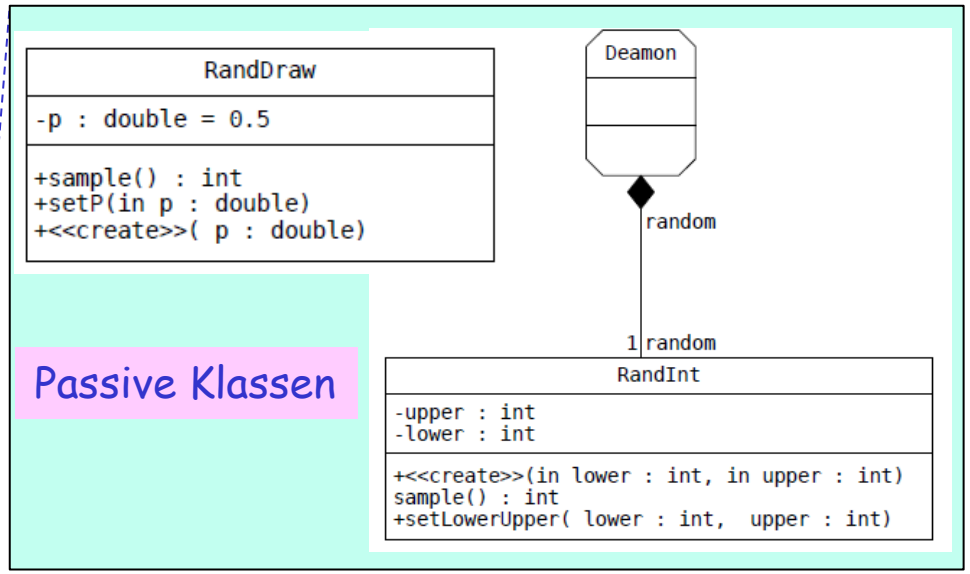
PId = RTDS_QueueId
SIGNAL = MESSAGE

Package

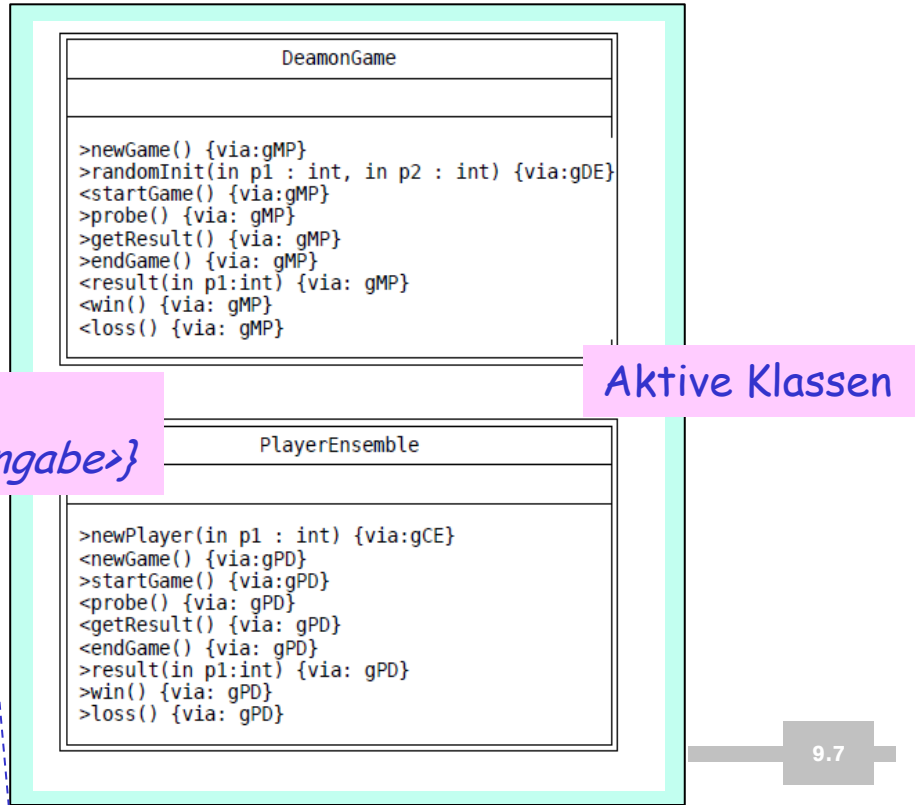


besteht aus UML-like-Definitionen aktiver und passiver SDL-RT-Klassen (äußere Schnittstelle) und deren Assoziationen

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

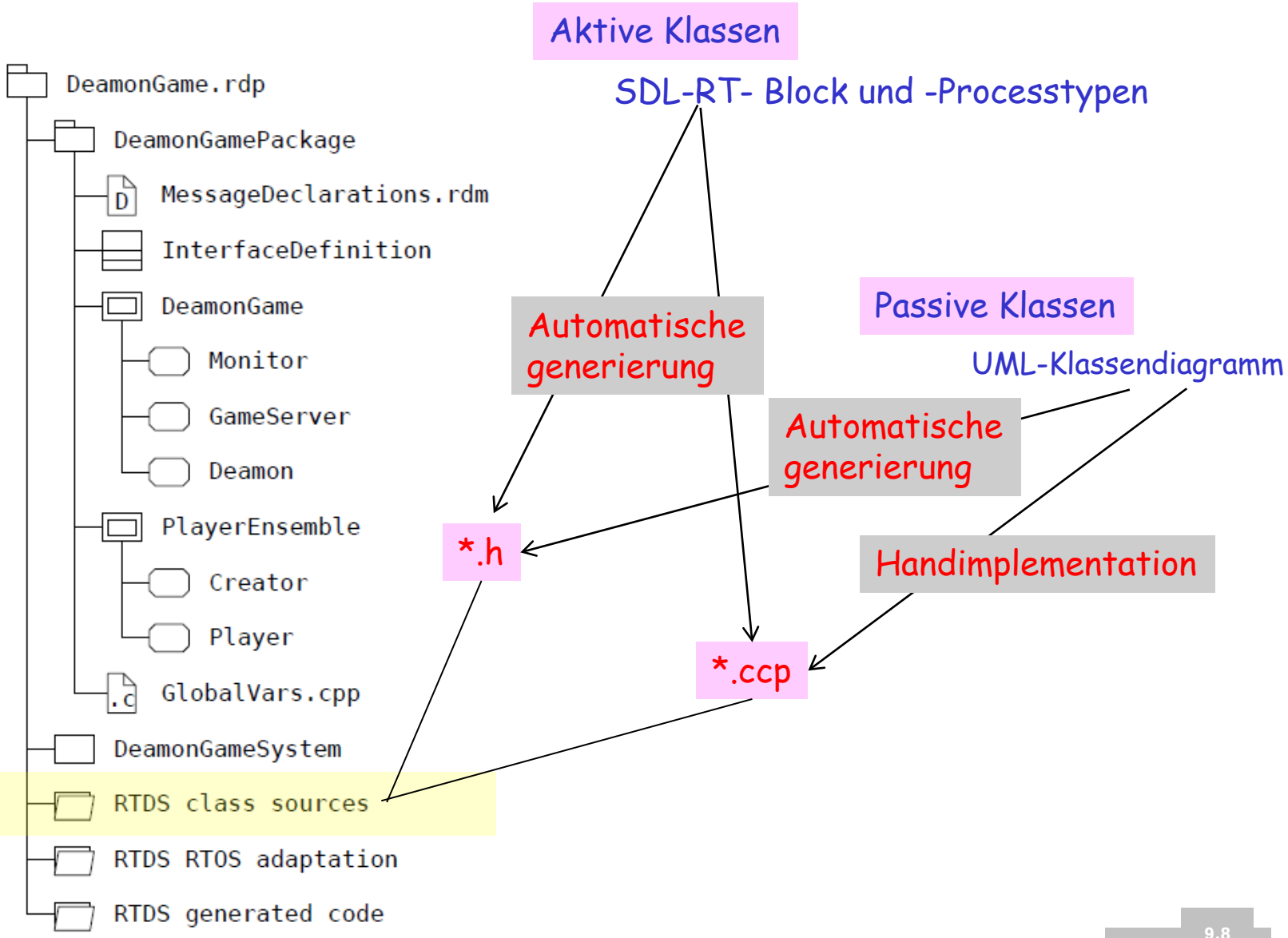


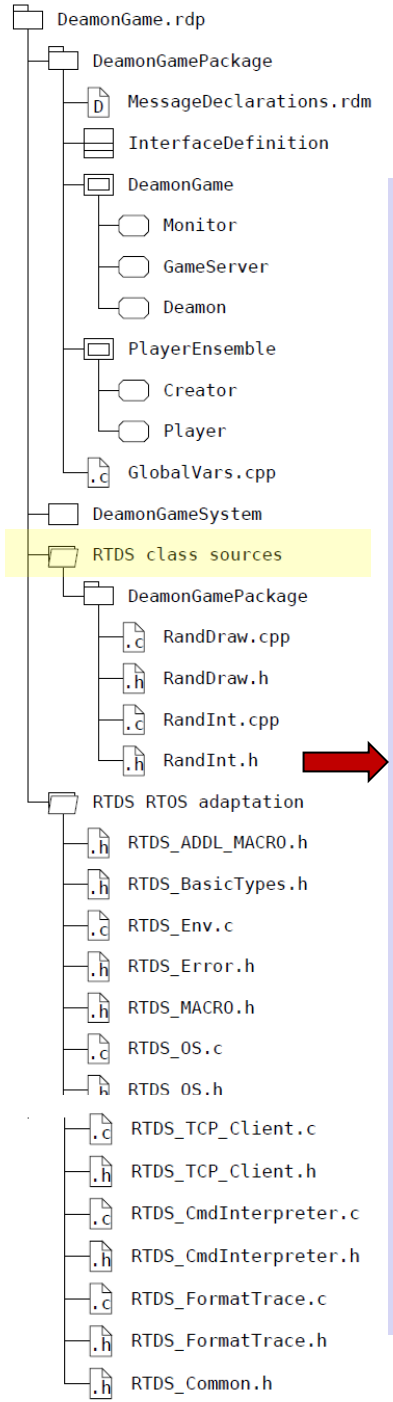
Passive Klassen



Aktive Klassen

Package


DeamonGame.rdp
DeamonGamePackage
MessageDeclarations.rdm
InterfaceDefinition
DeamonGame
Monitor
GameServer
Deamon
PlayerEnsemble
Creator
Player
GlobalVars.cpp
DeamonGameSystem
RTDS class sources
RTDS RTOS adaptation
RTDS generated code
Aktive Klassen
SDL-RT- Block und -Processtypen
Automatische generierung
Passive Klassen
UML-Klassendiagramm
Automatische generierung
*.h
Handimplementation
*.ccp

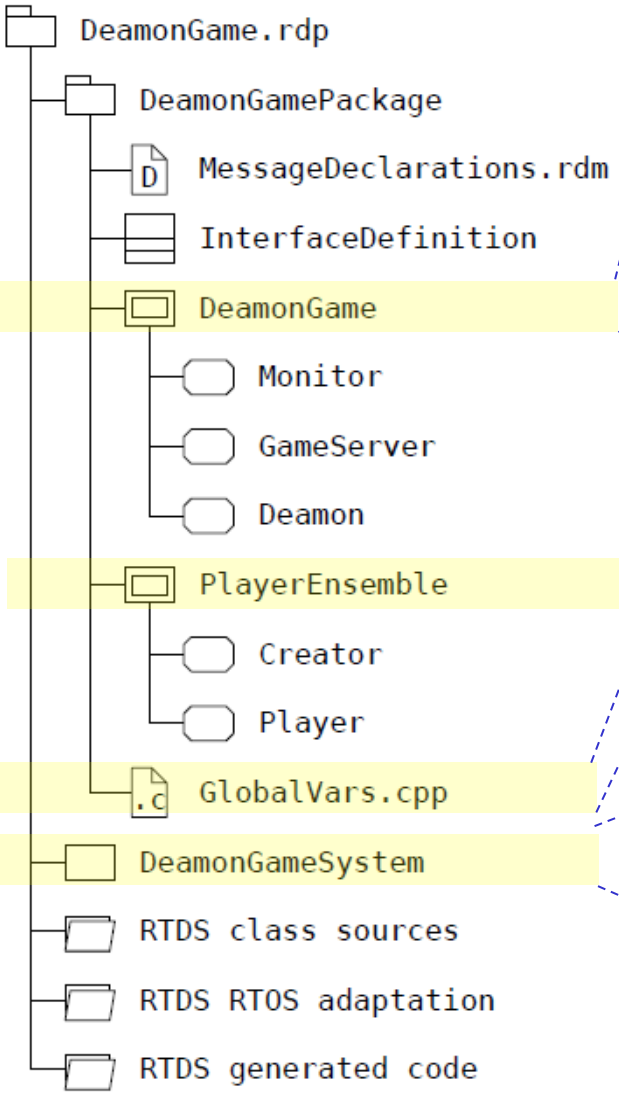


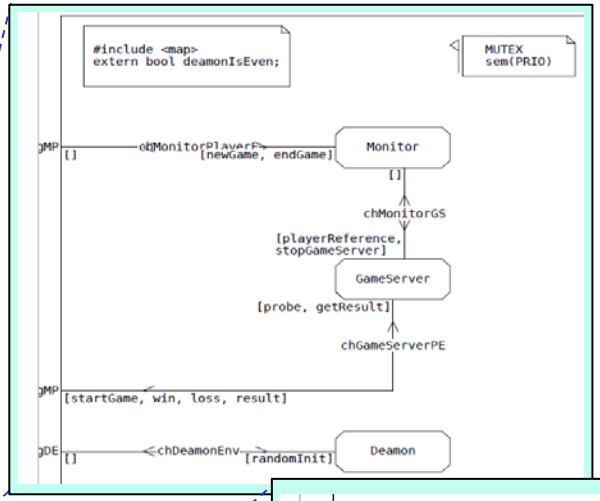
C++ -Quellen

```
#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
```

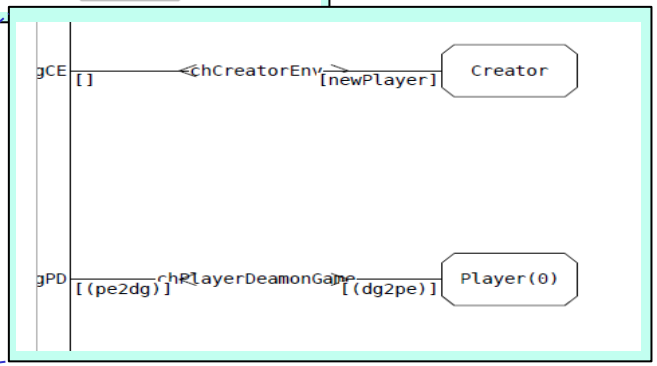
```
#include "DeamonGamePackage\RandInt.h"
#include <cmath>
/*
 * ATTRIBUTES FOR CLASS:
 * ---------------------
 *
 * [From RandInt]
 * - int   lower;
 * - int   upper;
 */


// 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





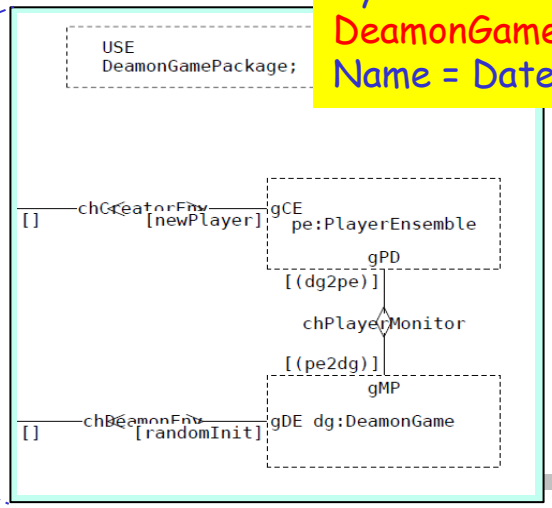
Blocktyp DeamonGame
Name = Dateiname



Blocktyp PlayerEnsemble
Name = Dateiname

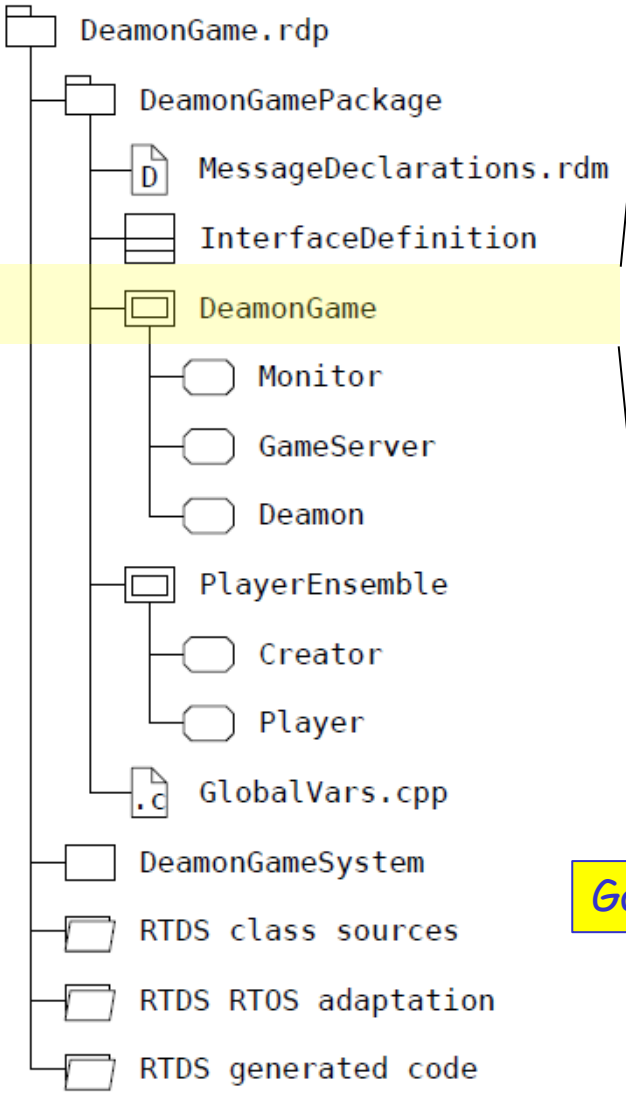
```
bool deamonIsEven = true;
```

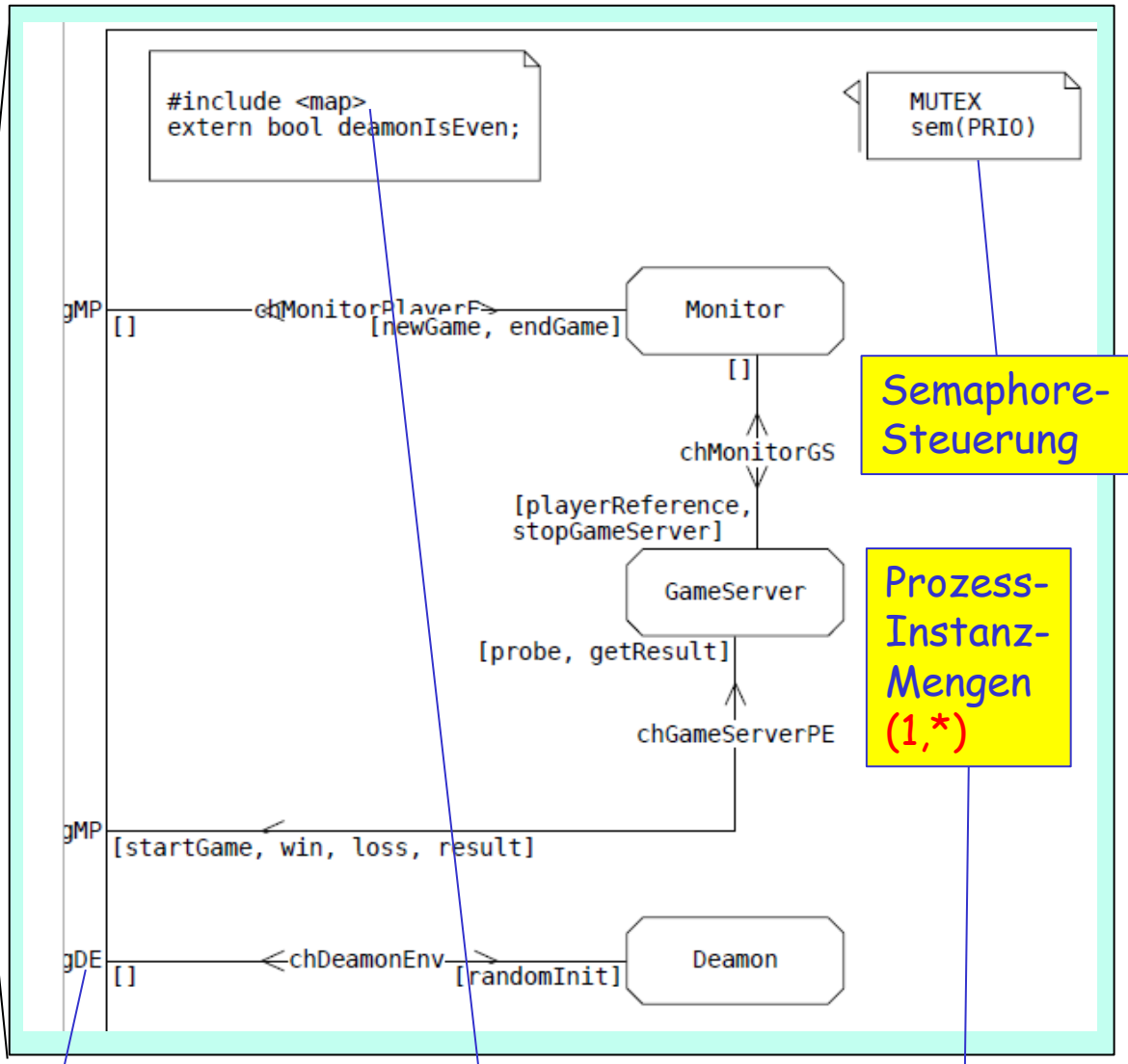
C/C++ Erweiterung
globale Variable



System
DeamonGameSystem
Name = Dateiname

Blocktyp DeamonGame





Semaphore-Steuerung

Prozess-Instanz-Mengen (1,*)

Gate-Angabe

C++ StandardTemplateLibrary

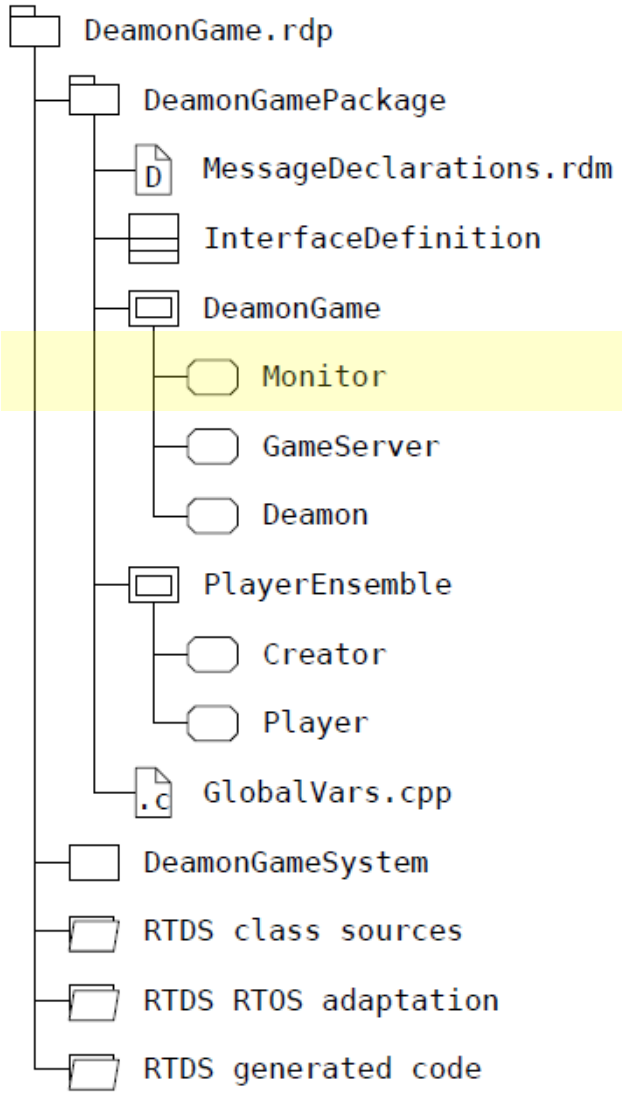
besser:
Monitor(1,1)
GameServer(0,*)
Deamon(1,1)

C++ Standard Template Library

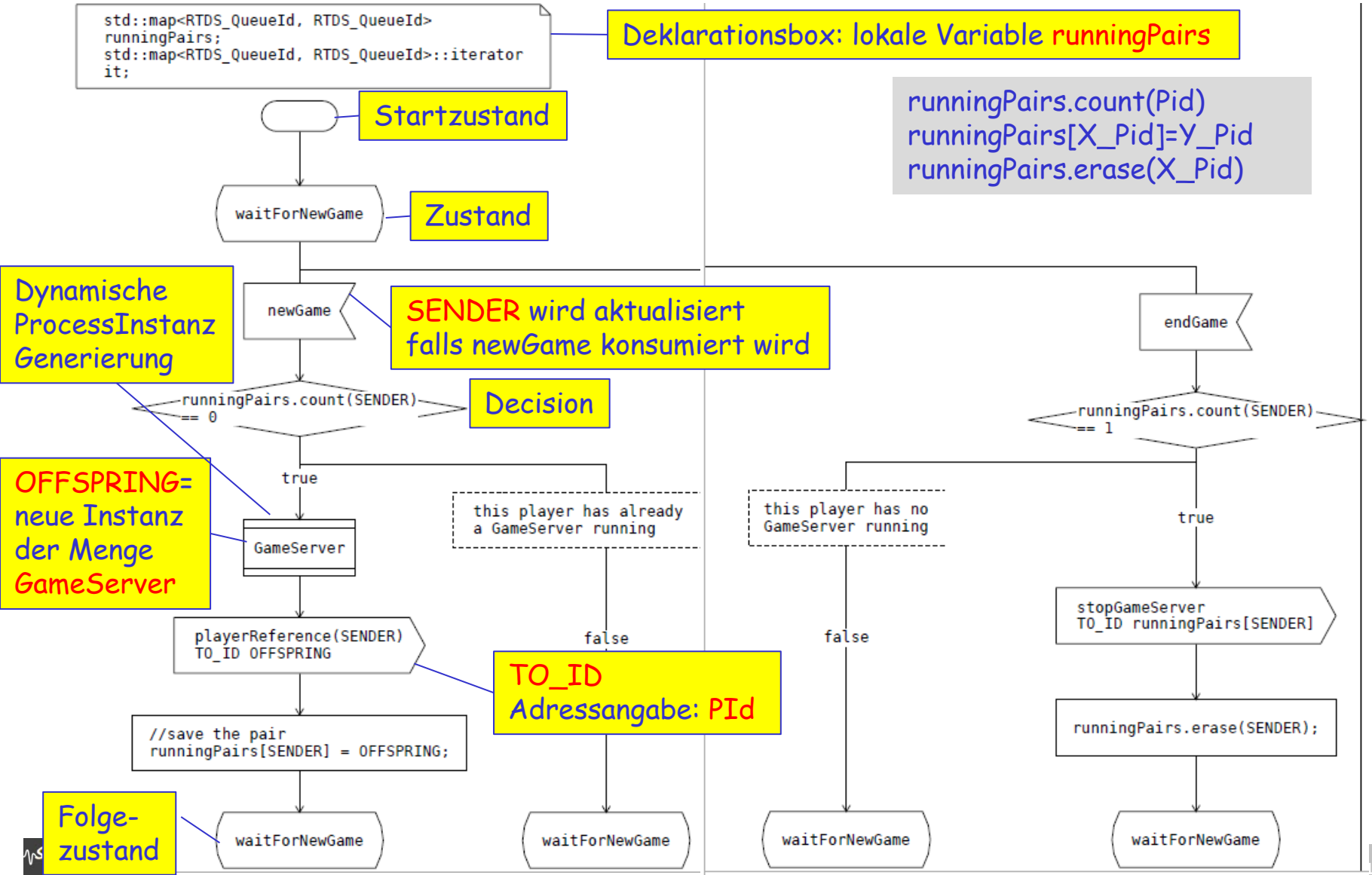
C++ Maps are sorted associative containers
that contain unique key/value pairs.
Maps are sorted by their keys.

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
…	

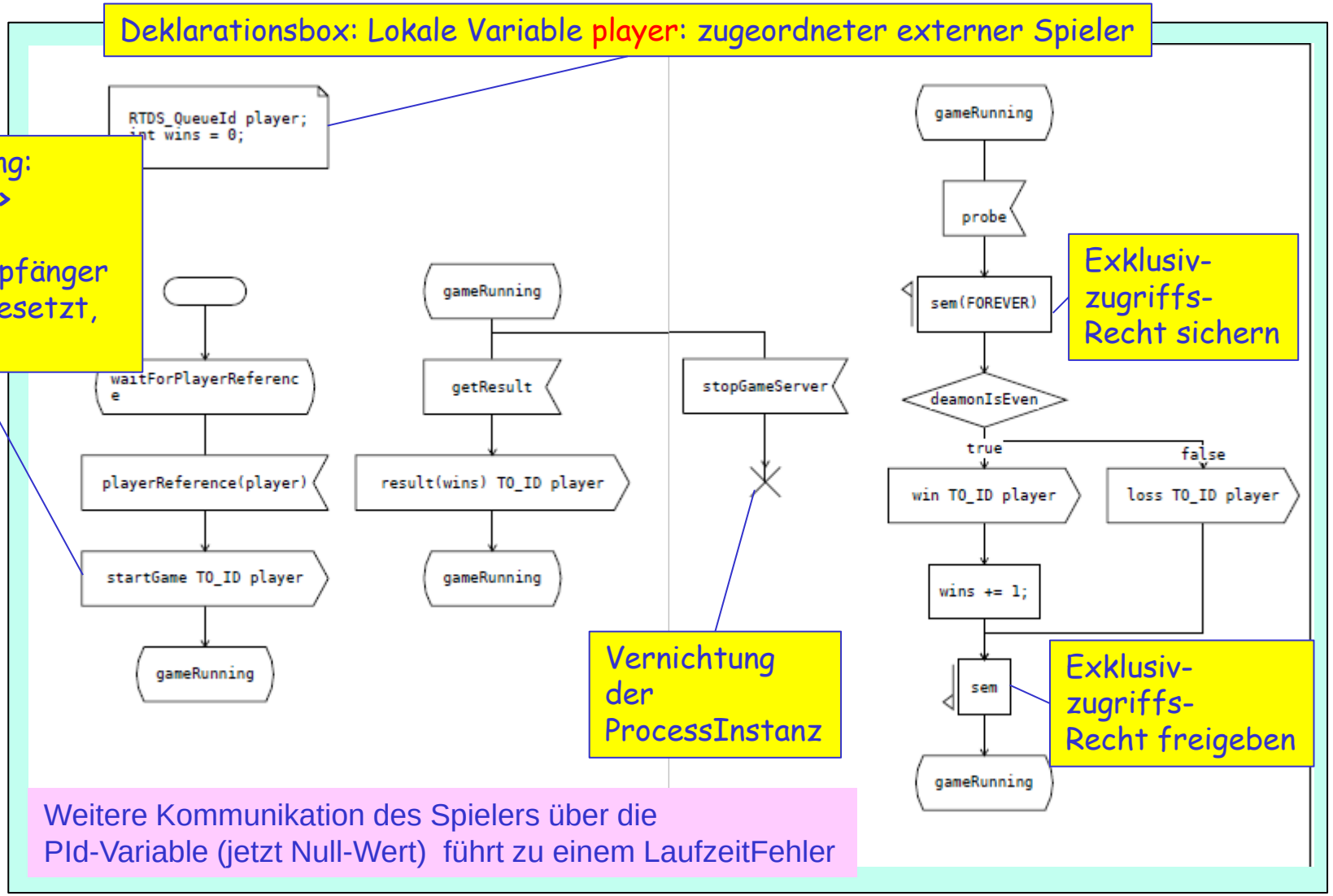
Package


DeamonGame.rdp
DeamonGamePackage
MessageDeclarations.rdm
InterfaceDefinition
DeamonGame
Monitor
GameServer
Deamon
PlayerEnsemble
Creator
Player
GlobalVars.cpp
DeamonGameSystem
RTDS class sources
RTDS RTOS adaptation
RTDS generated code

Prototypinstanz der Processmenge Monitor

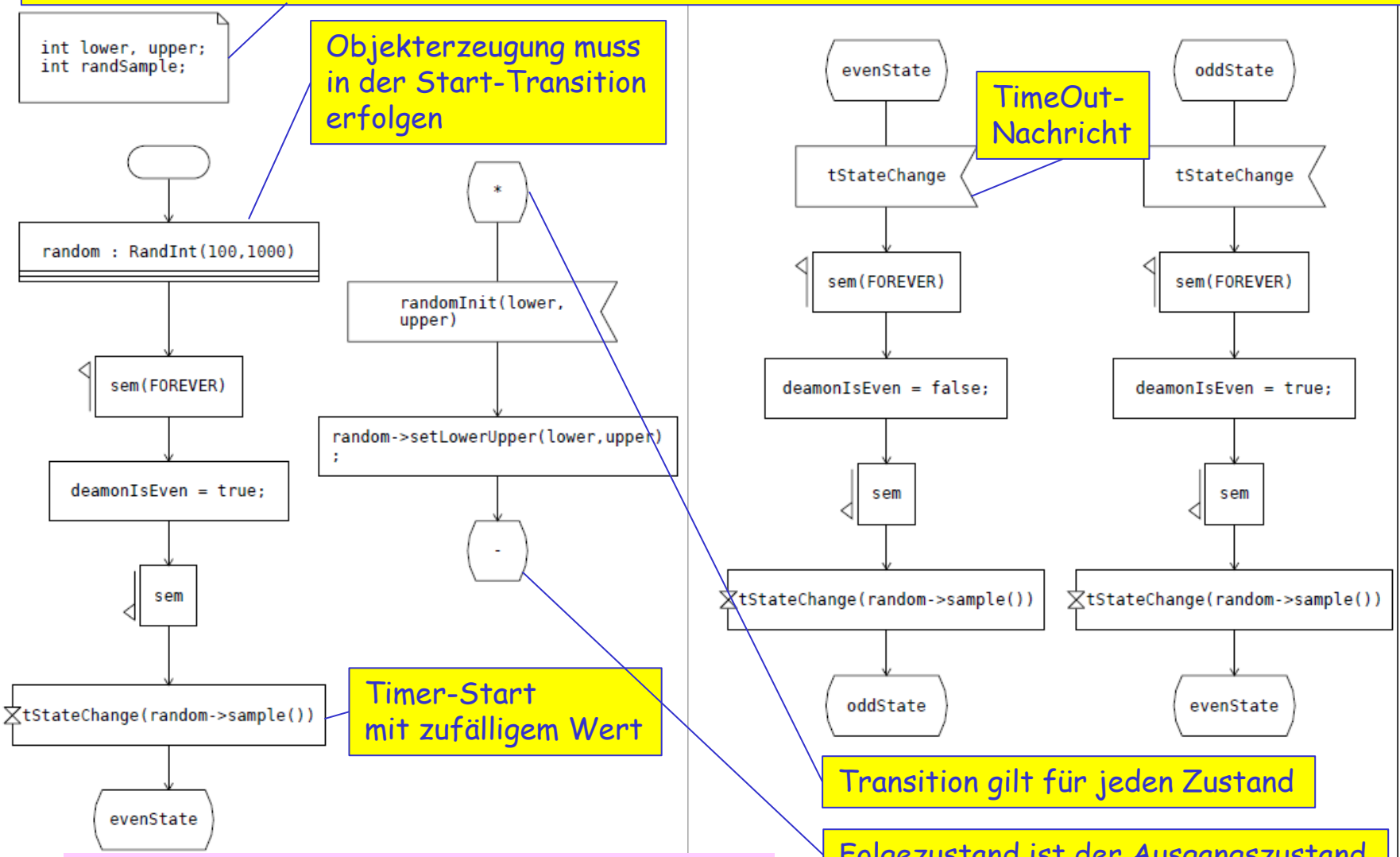


Prototypinstanz der Processmenge GameServer



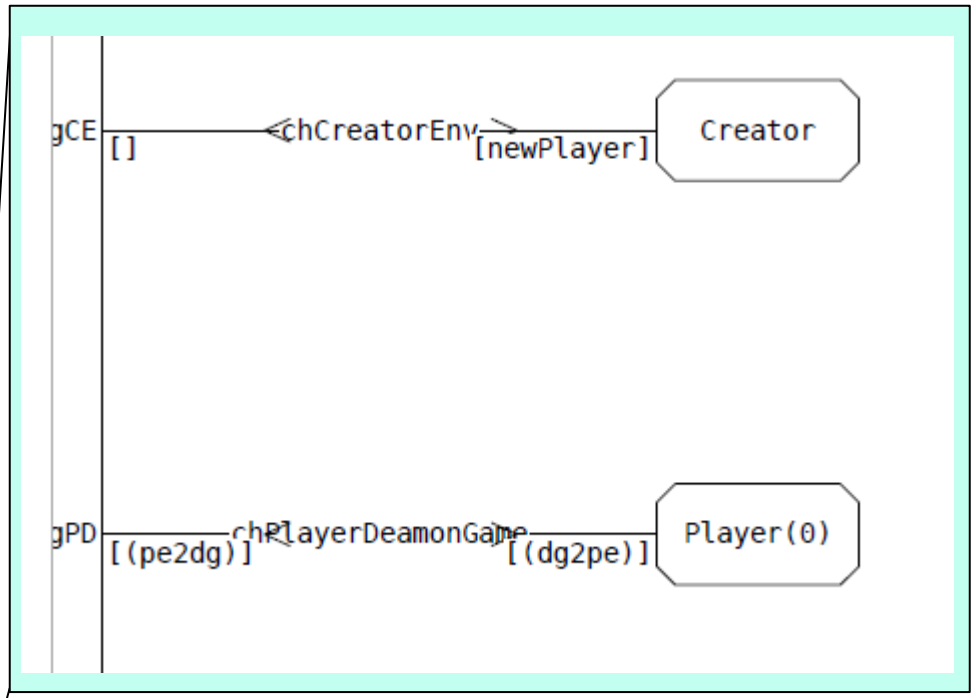
Prototypinstanz der Processmenge Deamon

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



Timervariablen werden in SDL-RT implizit deklariert

Blocktyp PlayerEnsemble

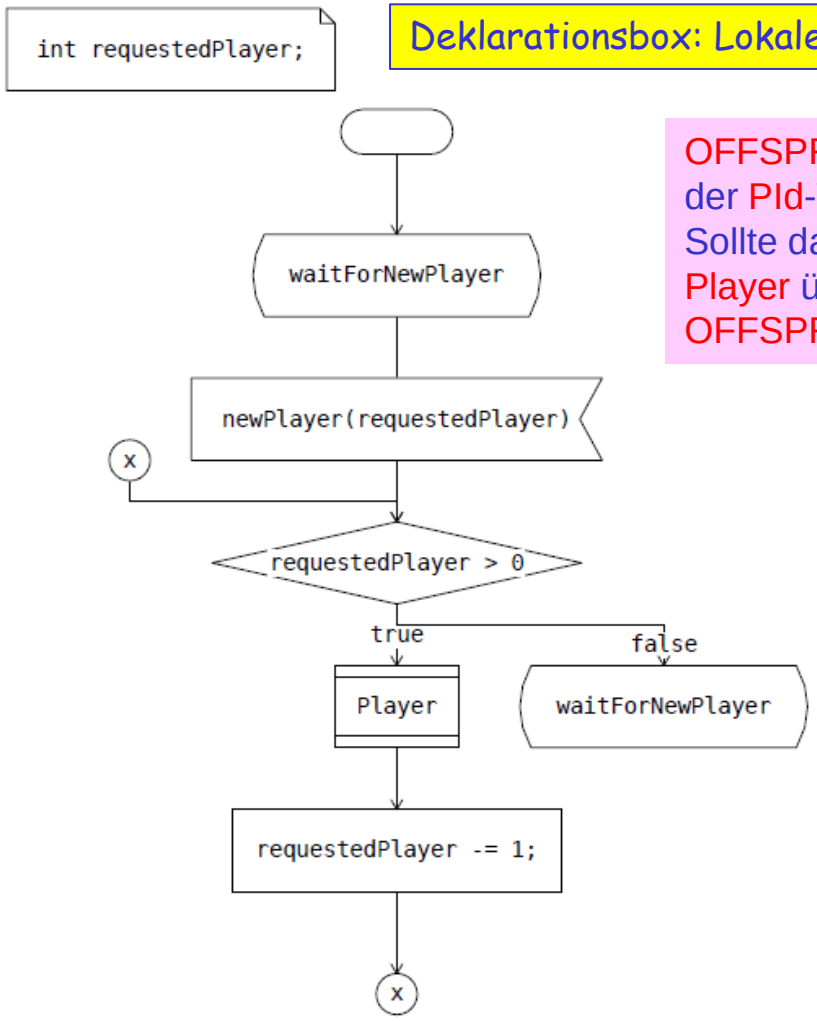


- DeamonGame.rdp
  - DeamonGamePackage
    - MessageDeclarations.rdm
    - InterfaceDefinition
    - DeamonGame
      - Monitor
      - GameServer
      - Deamon
    - PlayerEnsemble
      - Creator
      - Player
    - GlobalVars.cpp
  - DeamonGameSystem
  - RTDS class sources
  - RTDS RTOS adaptation
  - RTDS generated code

```
bool deamonIsEven = true;
```

Verbindung zur Semaphore-Variablen ist nutzerseitig zu organisieren

Prototypinstanz der Processmenge Creator



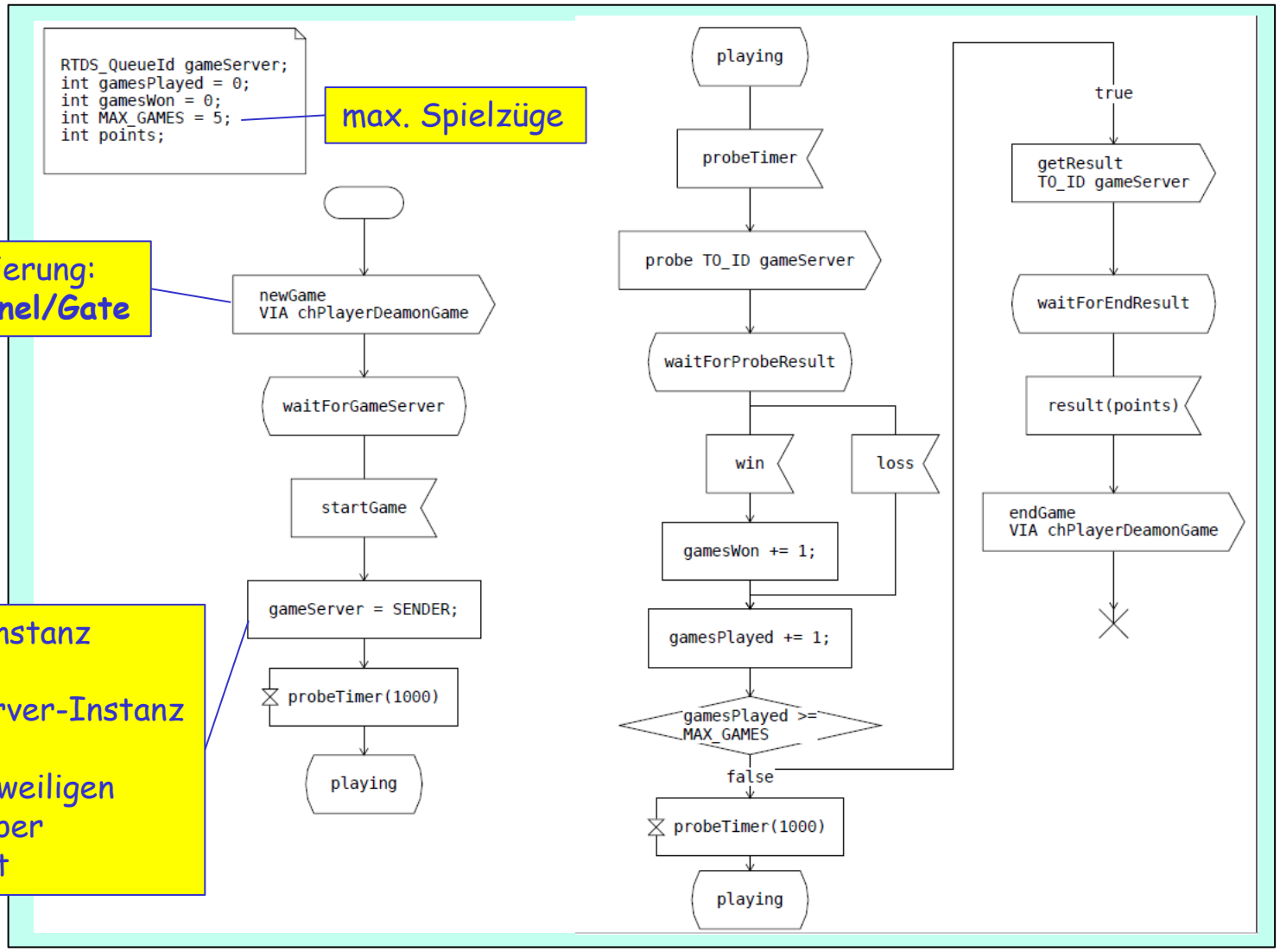
Deklarationsbox: Lokale Variable requestedPlayer

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

vordefinerte 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



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

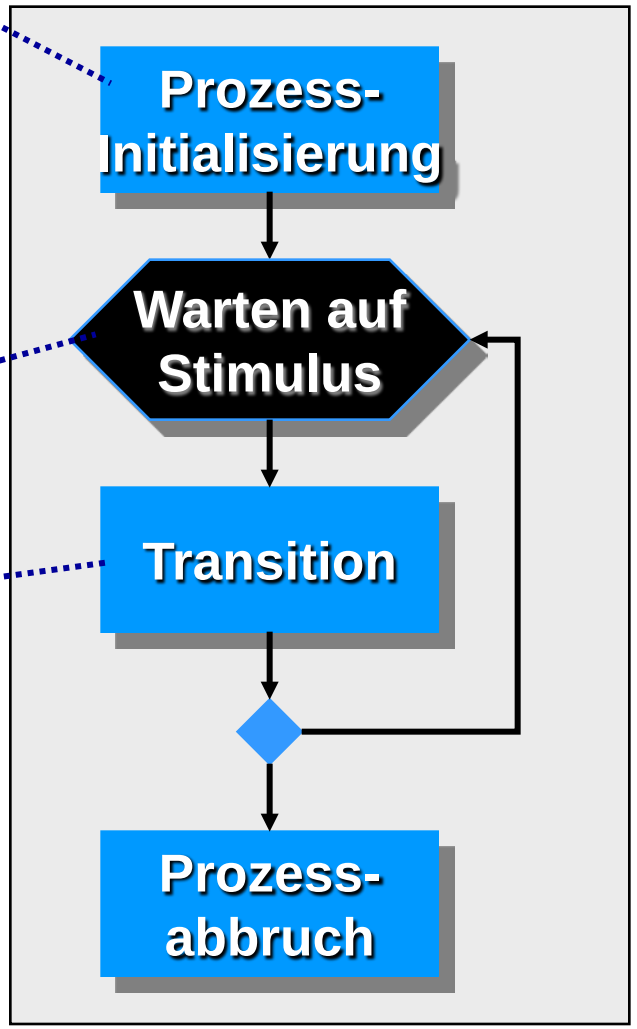
1. Grundphilosophie
2. ITU-Standard Z.100
3. Werkzeuge
4. SDL-Grundkonzepte
5. Deamon-Game: Musterbeispiel (in UML-Strukturen)
6. DeamonGame:
   Struktur- und Verhaltensbeschreibung in SDL-RT
7. DeamonGame: Simulation
8. PragmaDev-Tutorial

Prozess-Lebenszyklus (Schema)

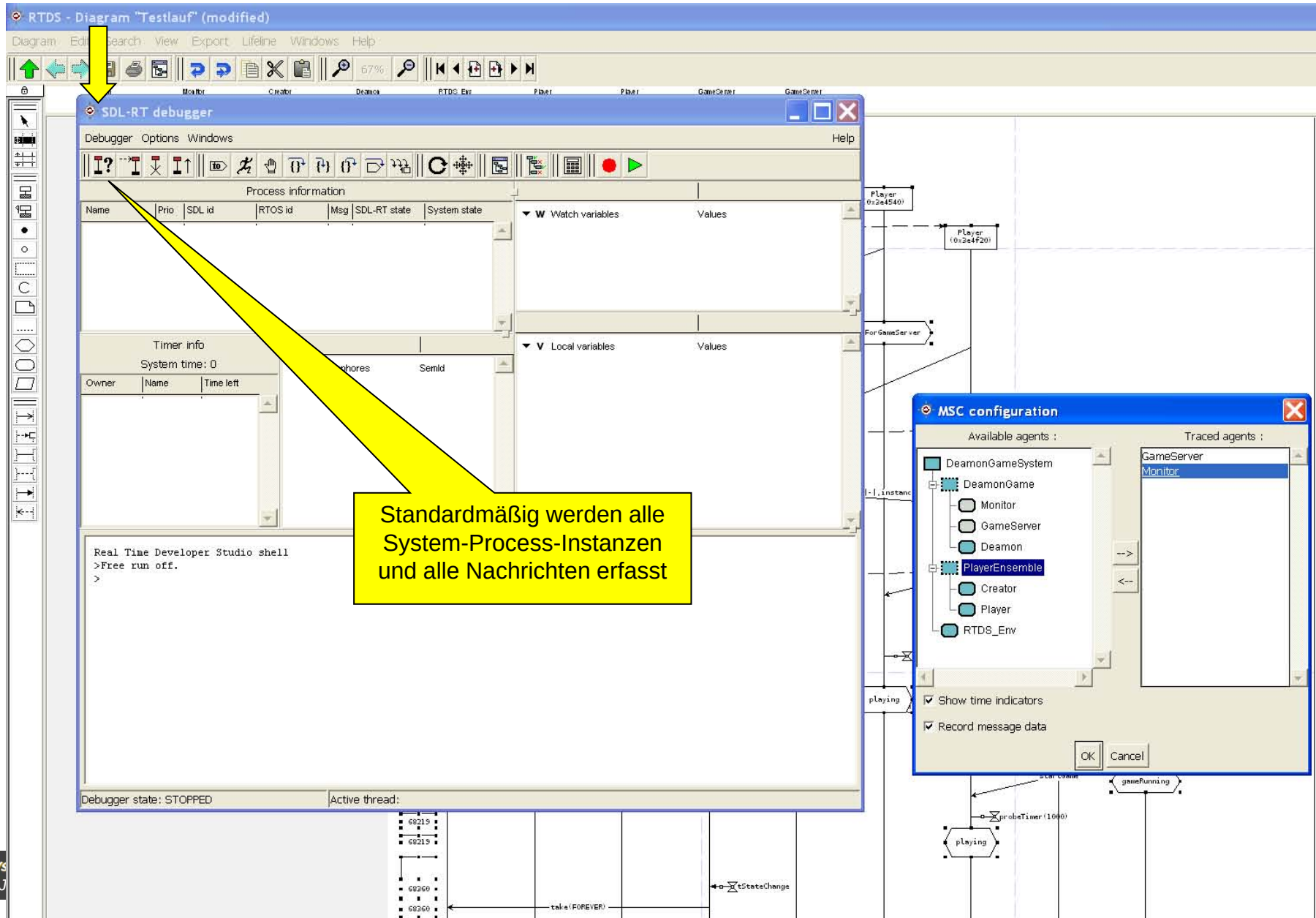
mit Existenzbeginn
- Initialisierung lokaler Variablen
- Ausgabe von Nachrichten
- Aufruf von Prozeduren
- Erzeugung von Prozess-Instanzen
- Übergang in einen echten Zustand

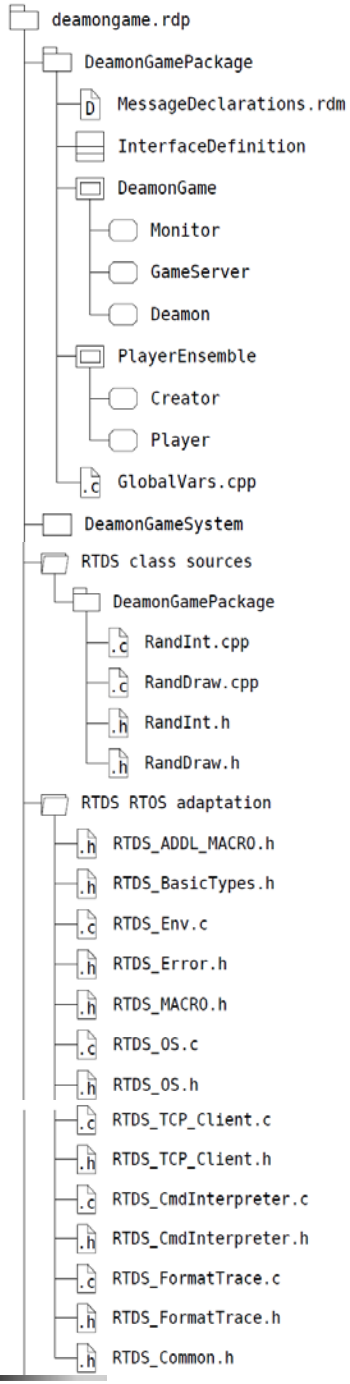
Auswahl erfolgt nach
- Stimulus und
- Zustand

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



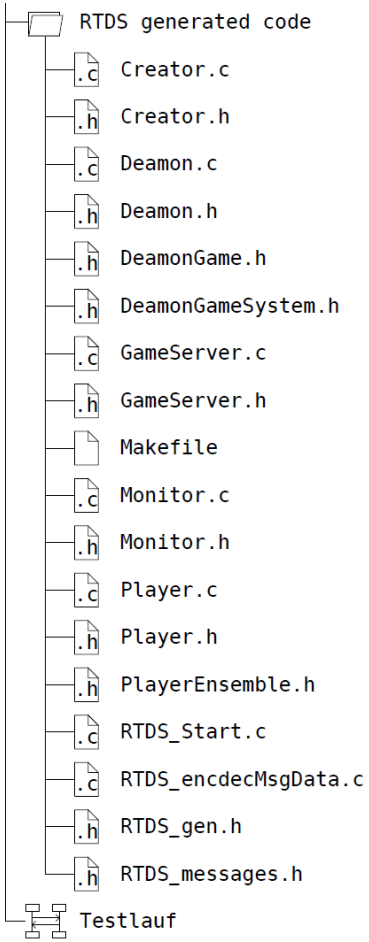
Konfiguration der SD-Ausgabe (MSC)





DeamonGame-Simulator

Quellen



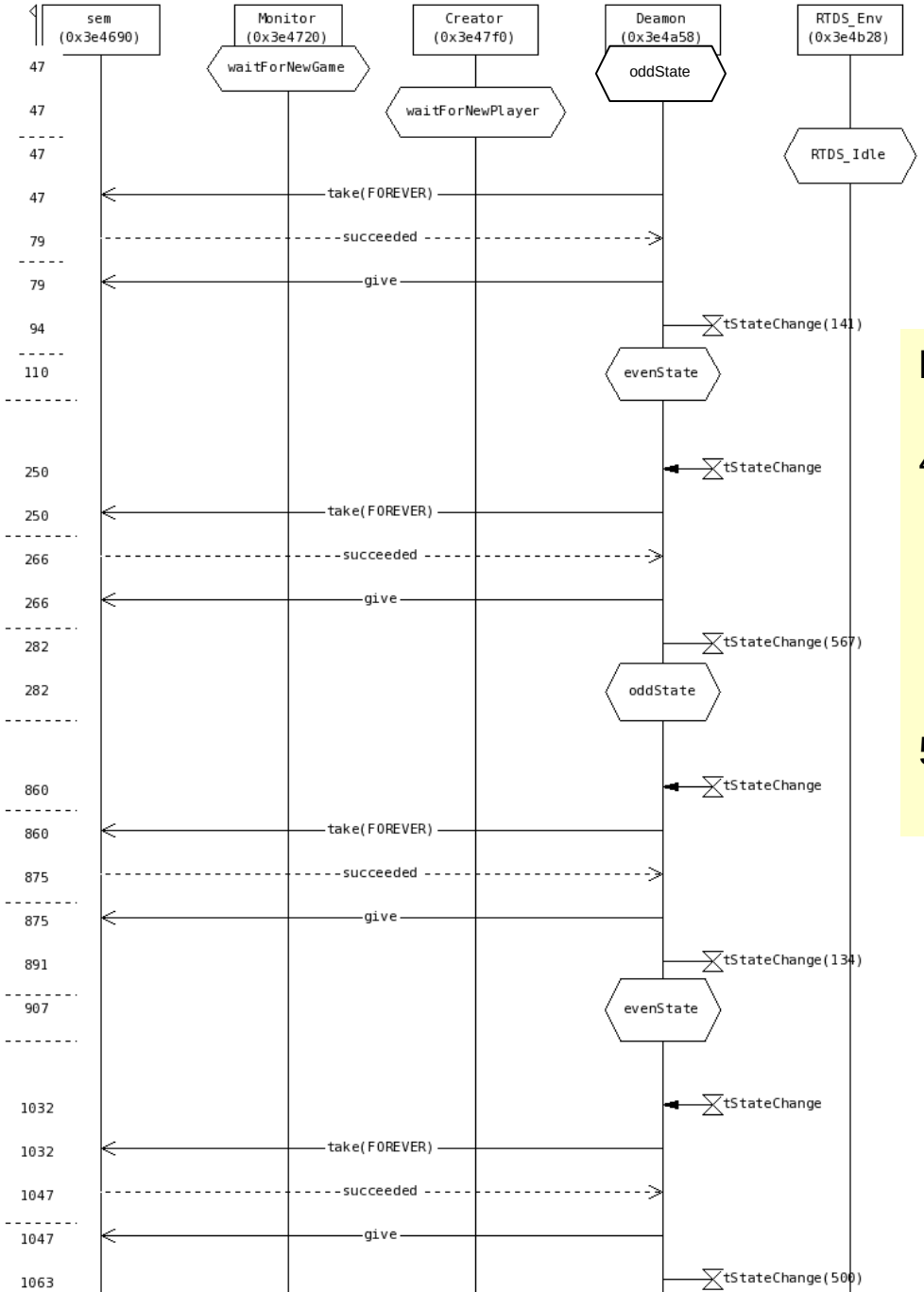
Run

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

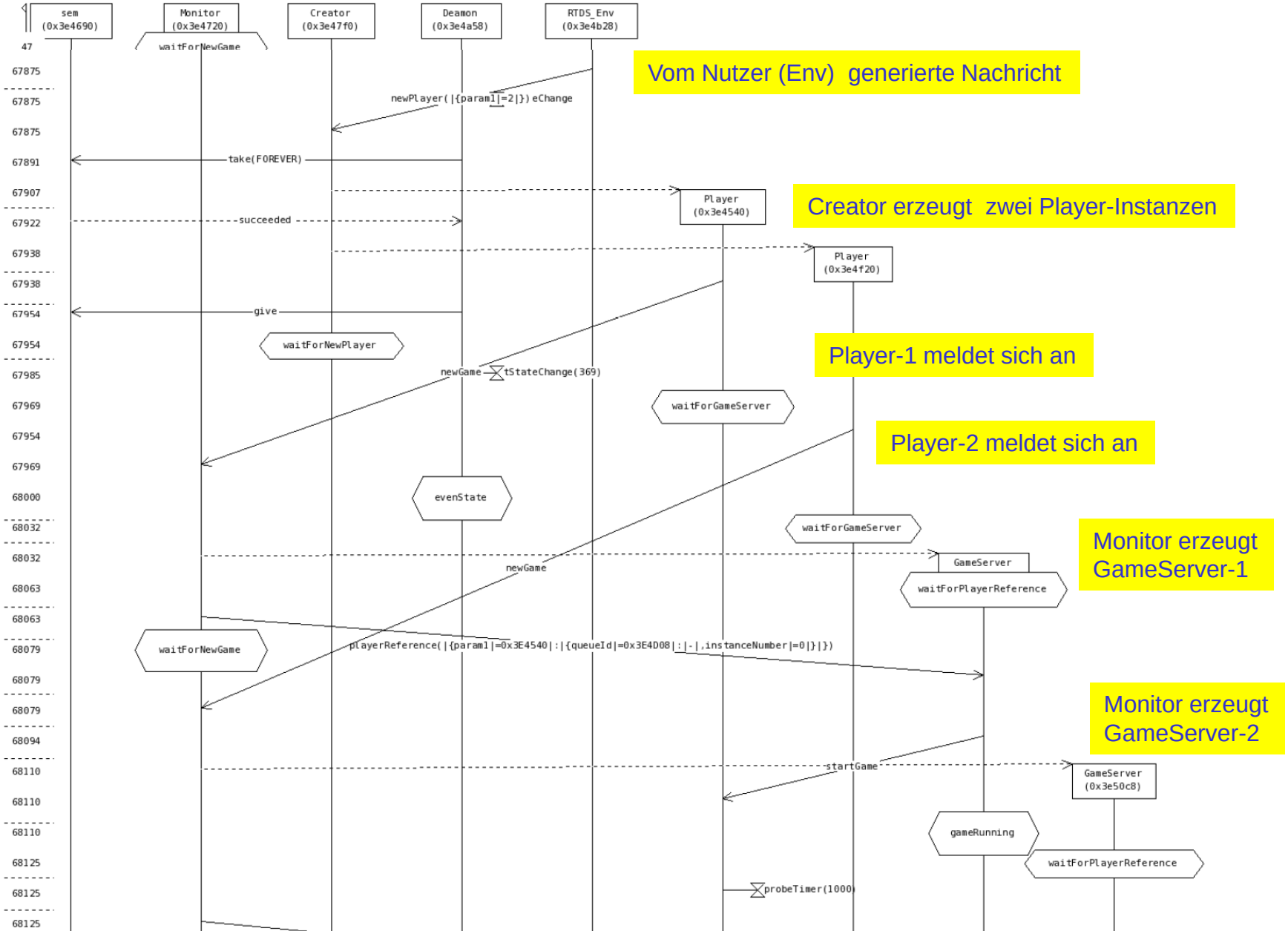
Monitor und Creator verharren

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

6. SDL-Einführung

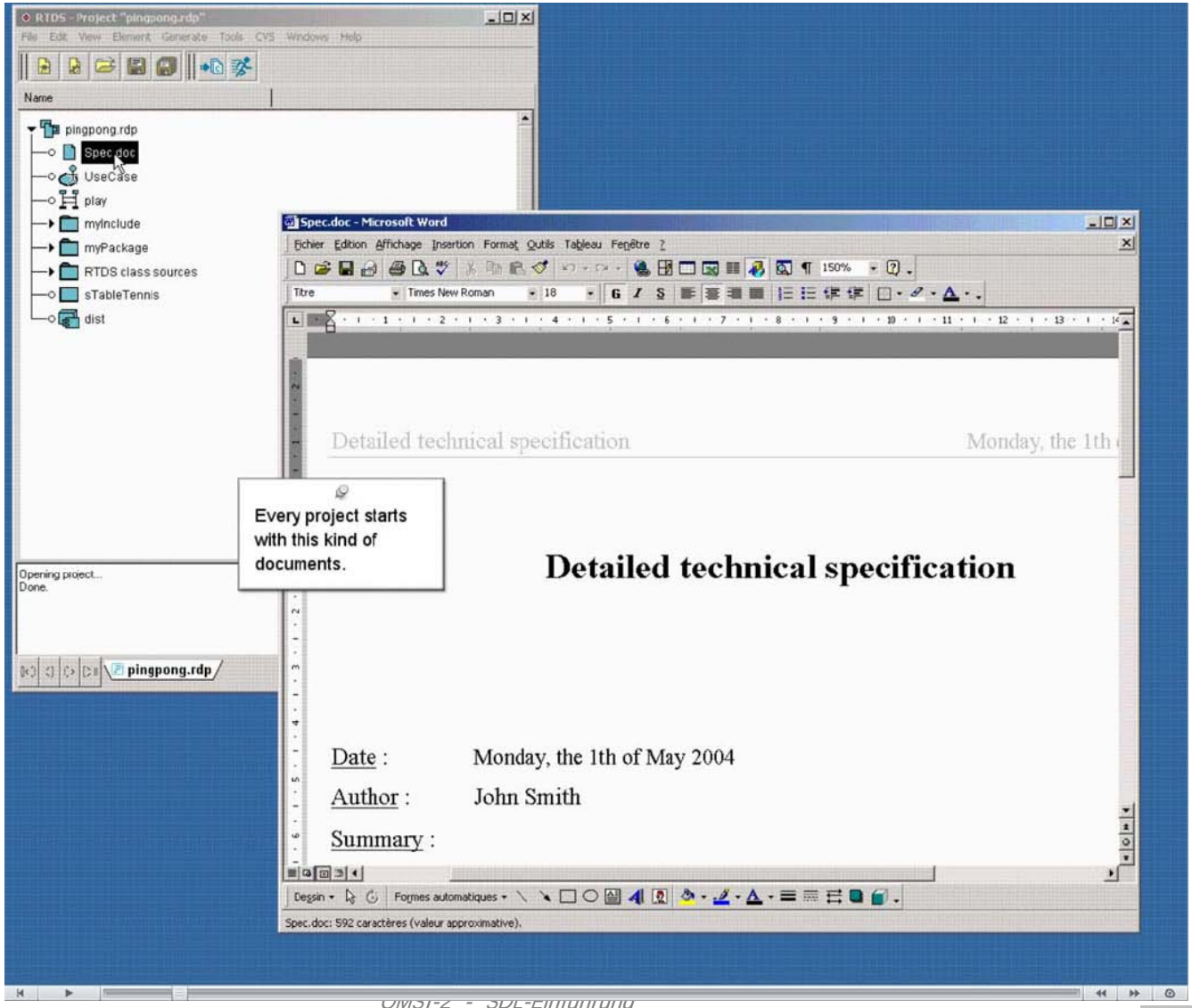
1. Grundphilosophie
2. ITU-Standard Z.100
3. Werkzeuge
4. SDL-Grundkonzepte
5. Deamon-Game: Musterbeispiel (in UML-Strukturen)
6. DeamonGame:
   Struktur- und Verhaltensbeschreibung in SDL-RT
7. DeamonGame: Simulation
8. PragmaDev-Tutorial

Einsatzfelder von PragmaDev

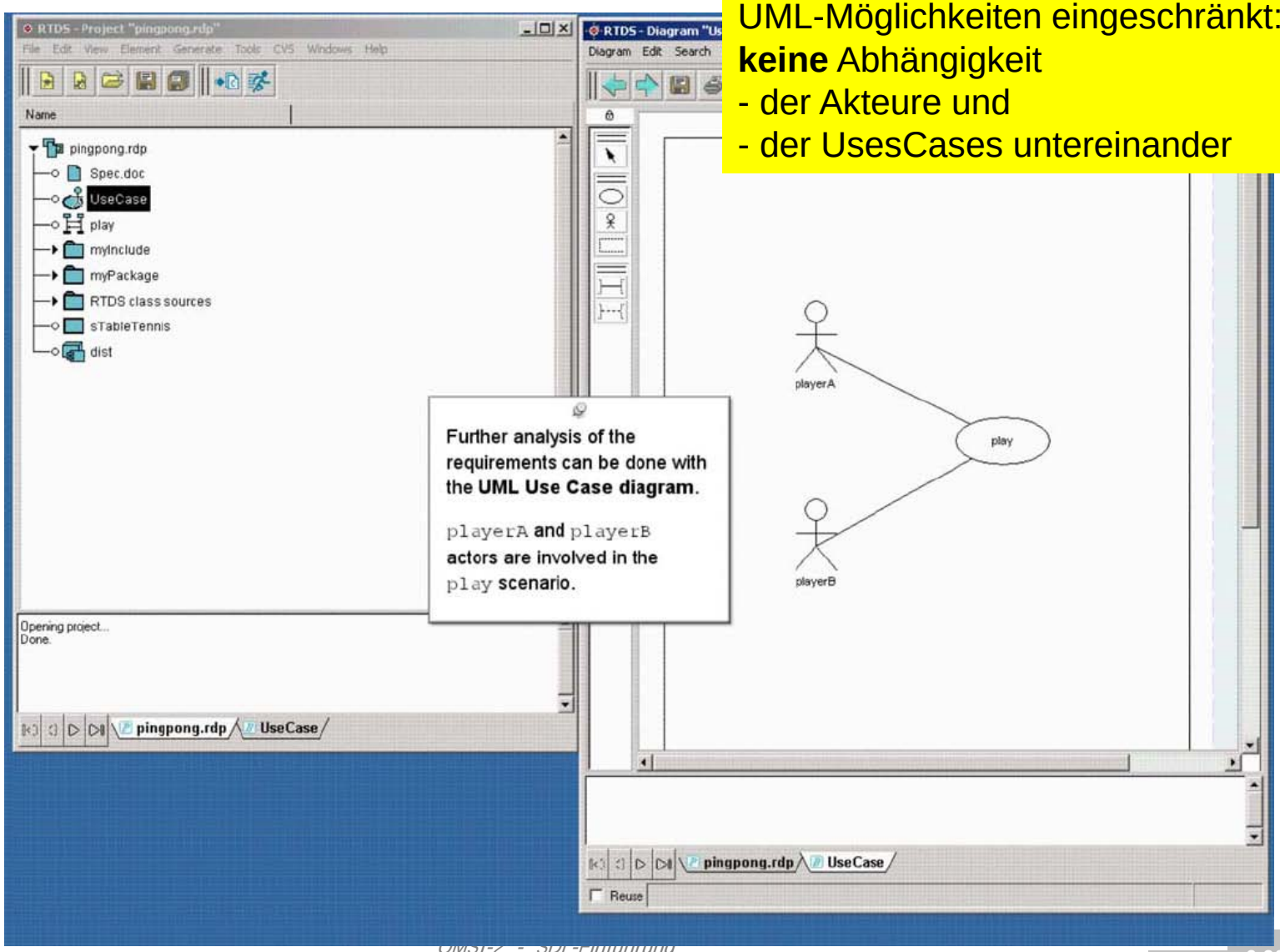
- Anforderungsspezifikation
- Third-Party-Tool-Integration (z.B. CVS)
- OOA
- Systemarchitektur
- Spezifikation eines verteilten Systems
- Code-Generierung
- Debugger (SDL, generierter Code)
- Graphische Trace-Ausgaben
- Konformität von Simulation und Anforderungsspezifikation

Demo-Video: www.pragmadev.com

Projektbeschreibung

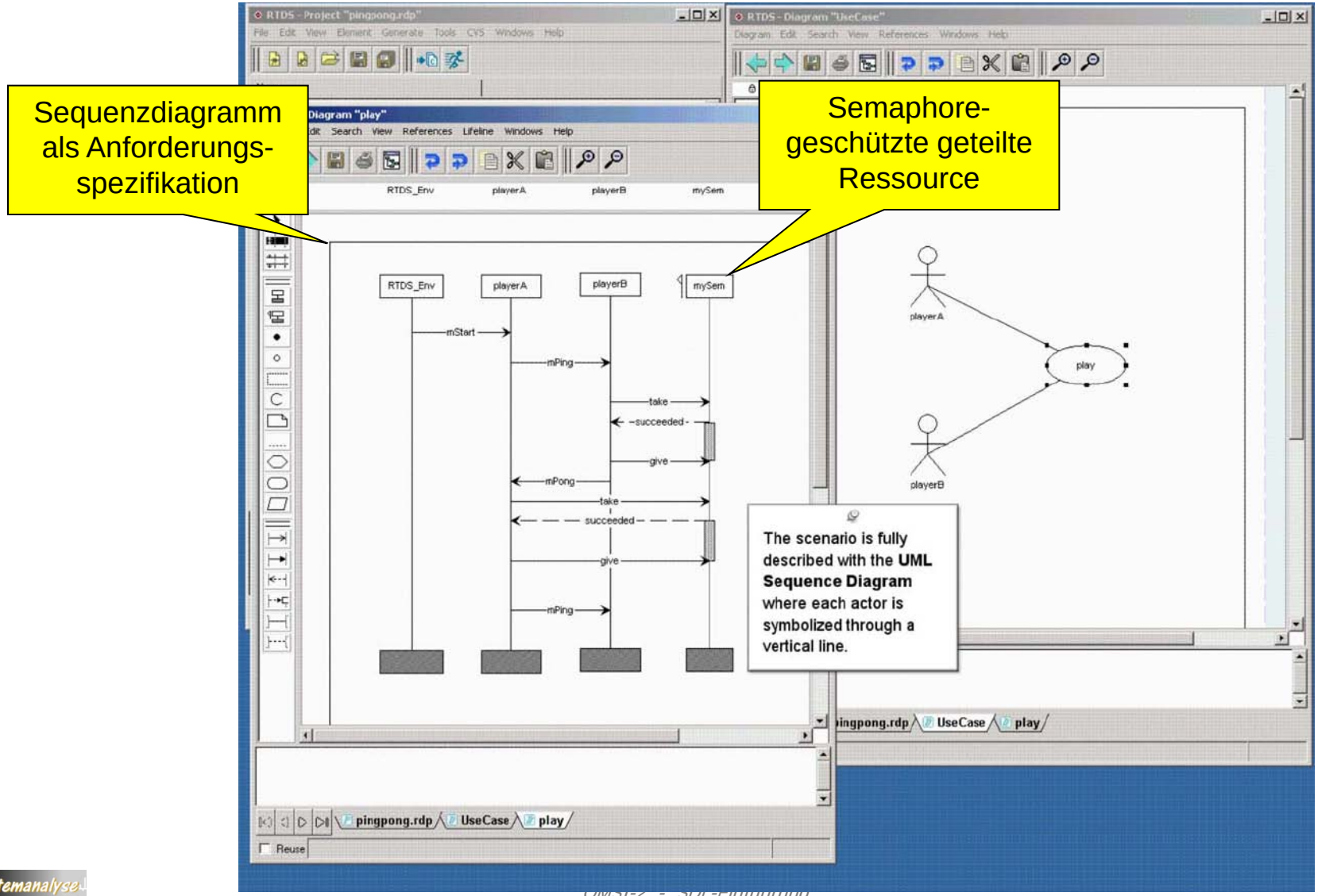


Use-Case-Diagramme

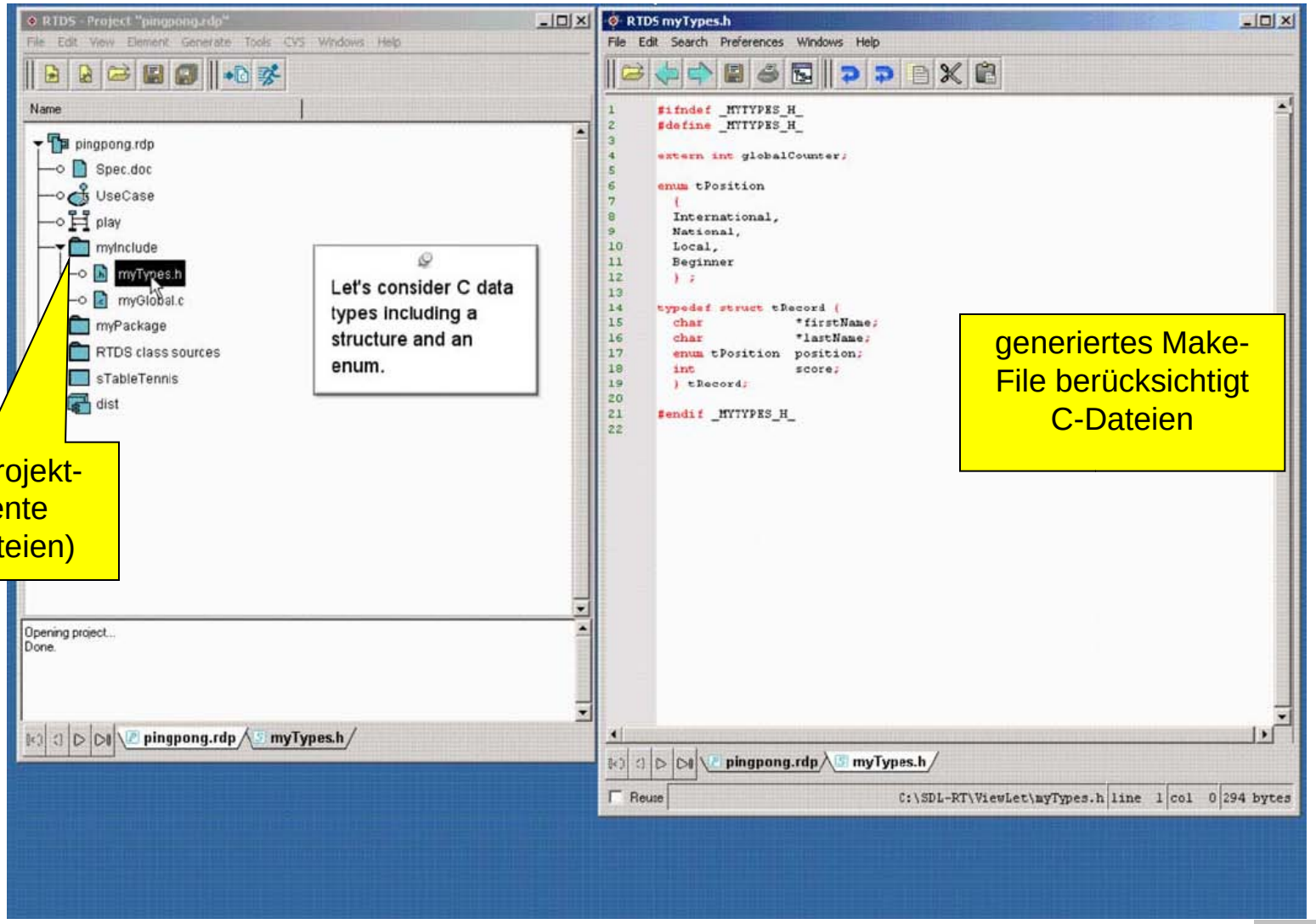


UML-Möglichkeiten eingeschränkt:
keine Abhängigkeit
- der Akteure und
- der UsesCases untereinander

Sequenzdiagramme



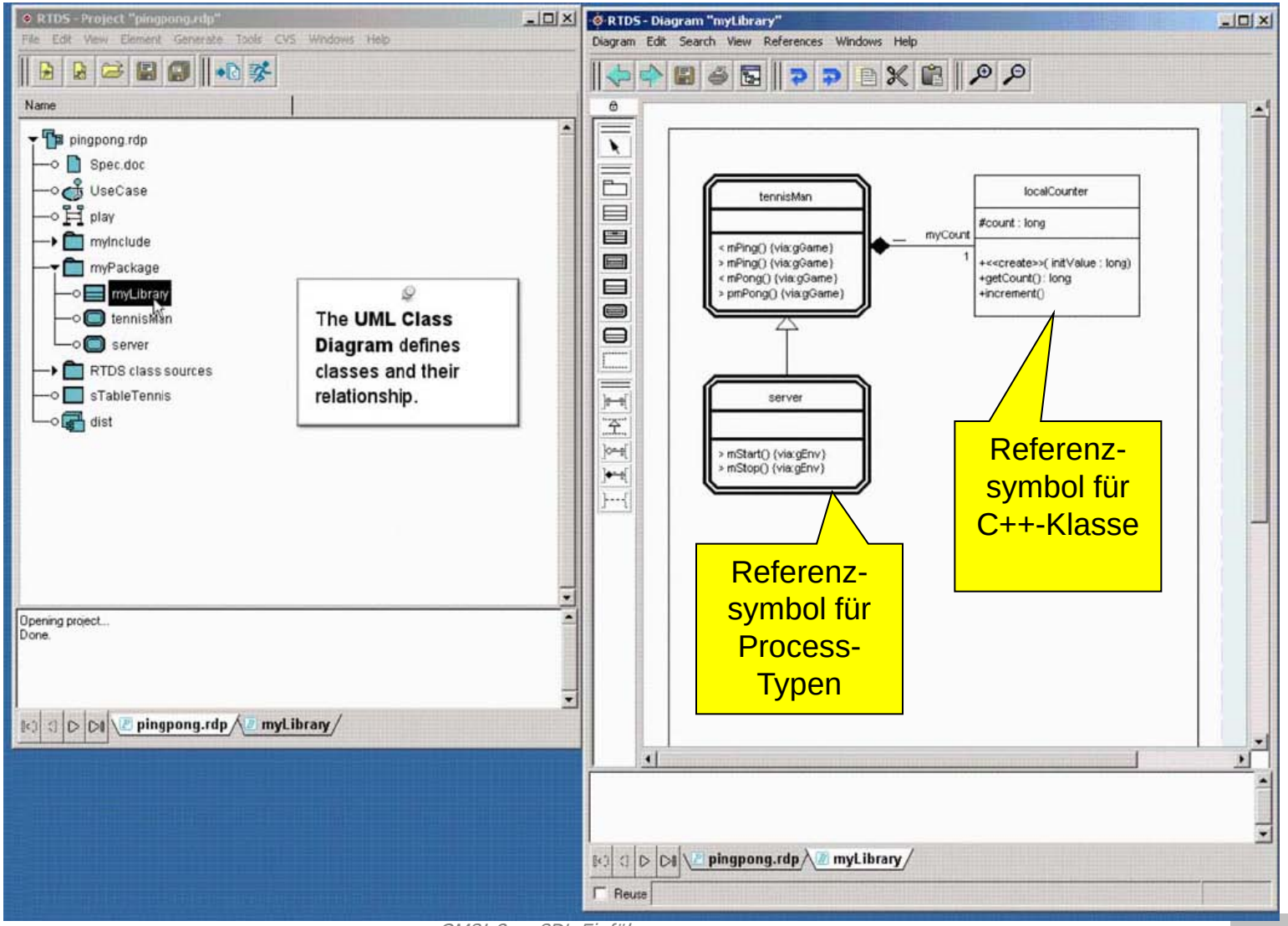
Externe Datentypen



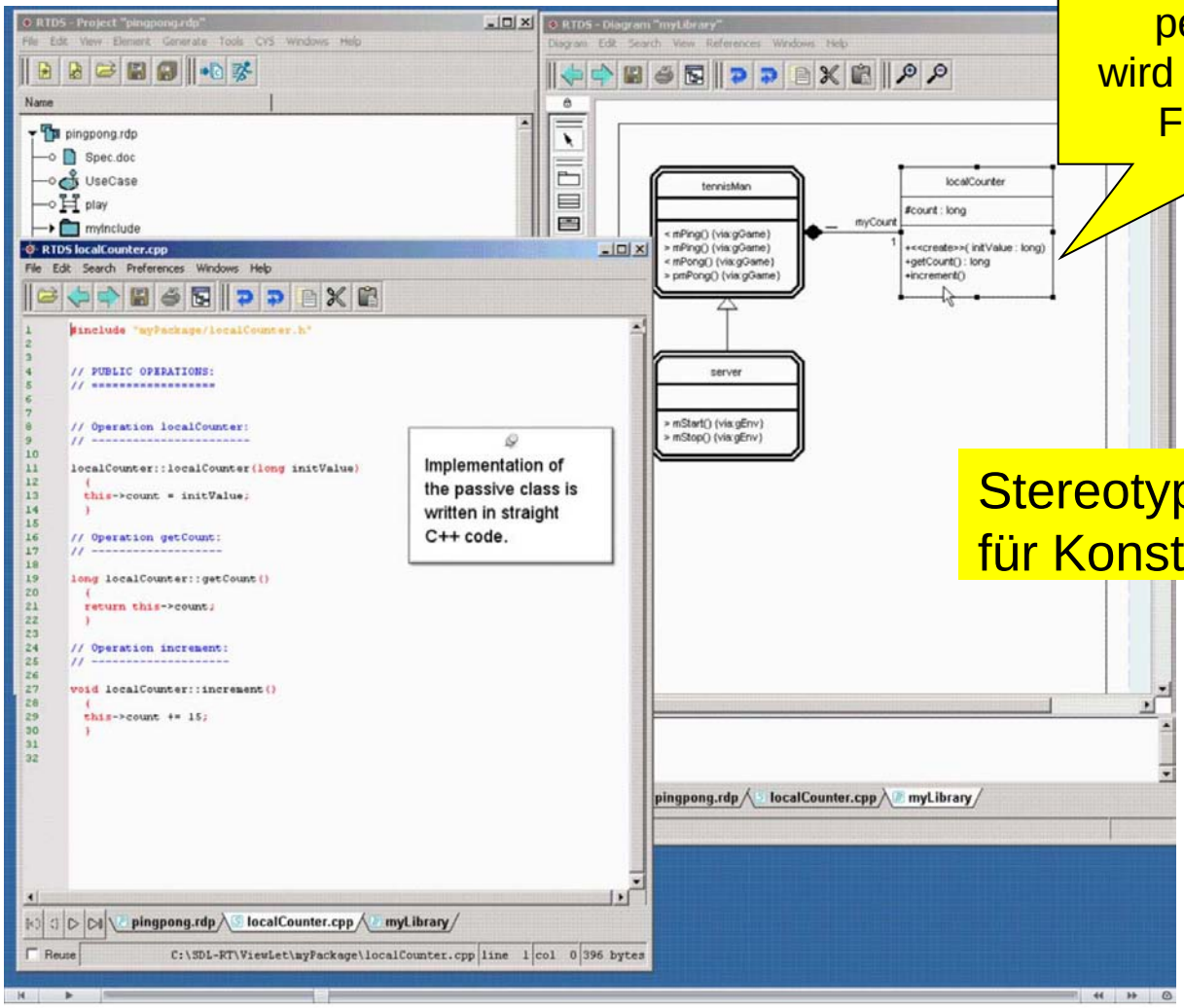
Paket als Projekt-Komponente (hier: C-Dateien)

generiertes Make-File berücksichtigt C-Dateien

Definition von aktiven und passiven Klassen



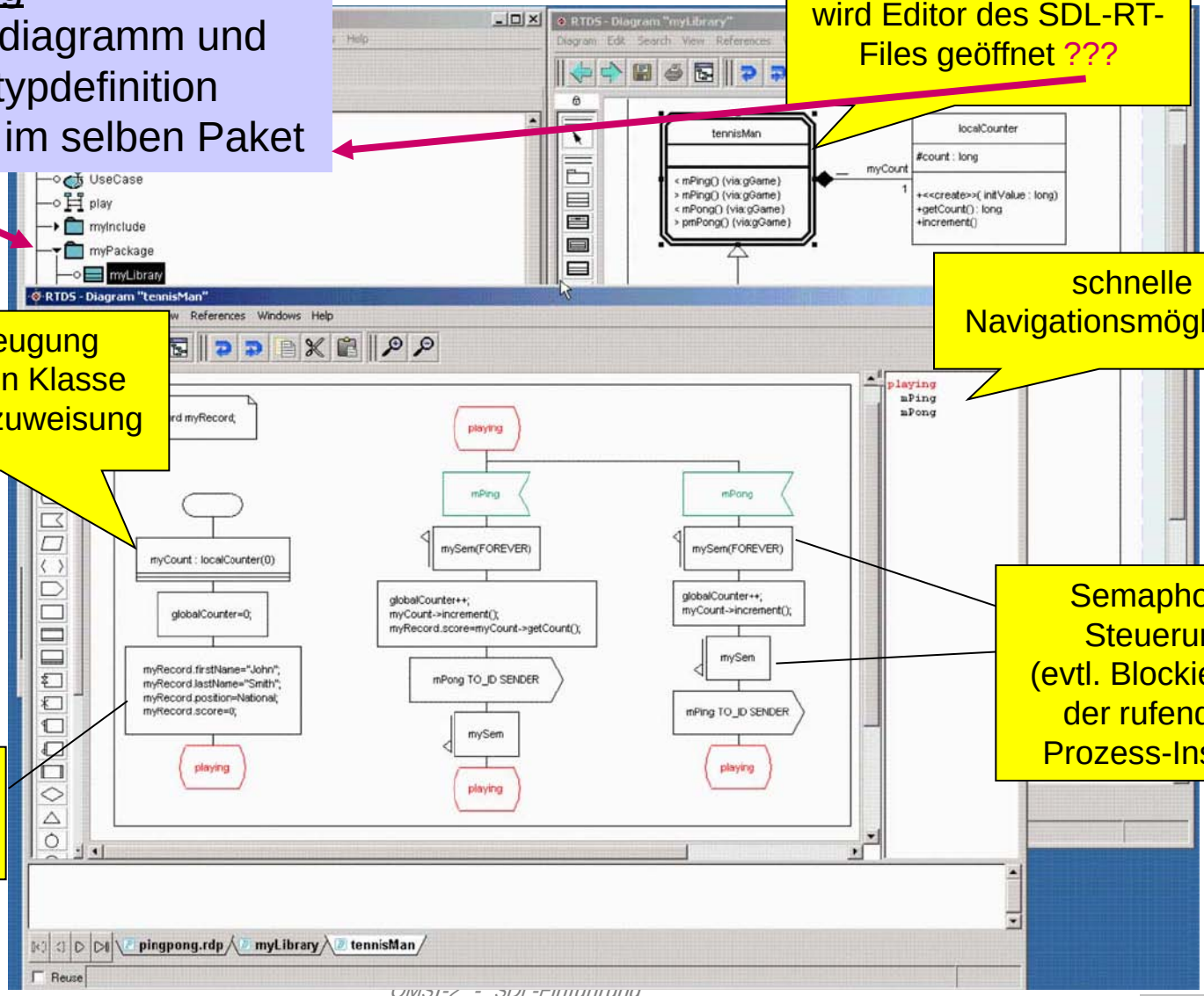
Implementation der Klassenmethoden



Processtyp-Definition

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

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



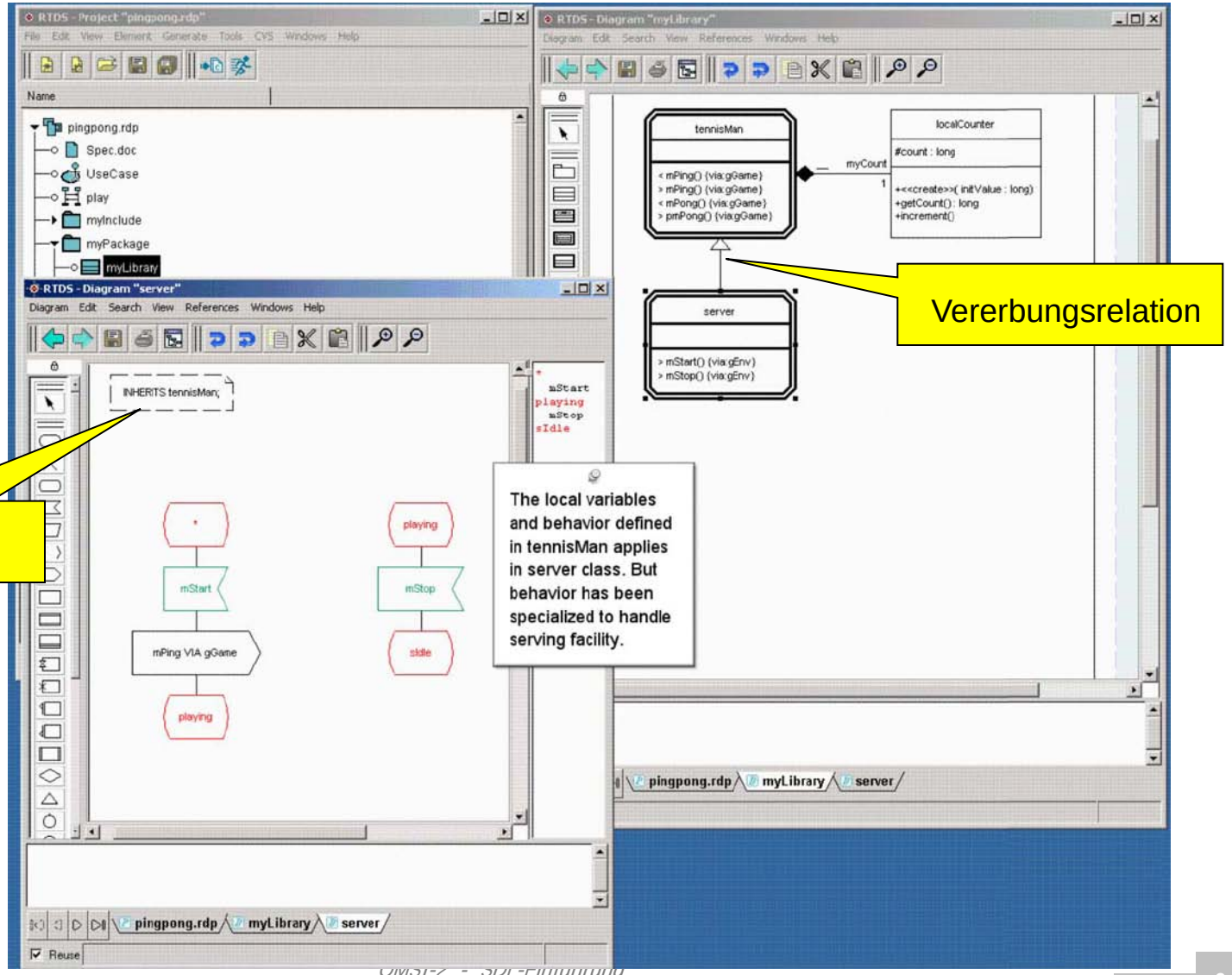
Objekt-Erzeugung einer passiven Klasse und Referenzzuweisung

schnelle Navigationsmöglichkeit

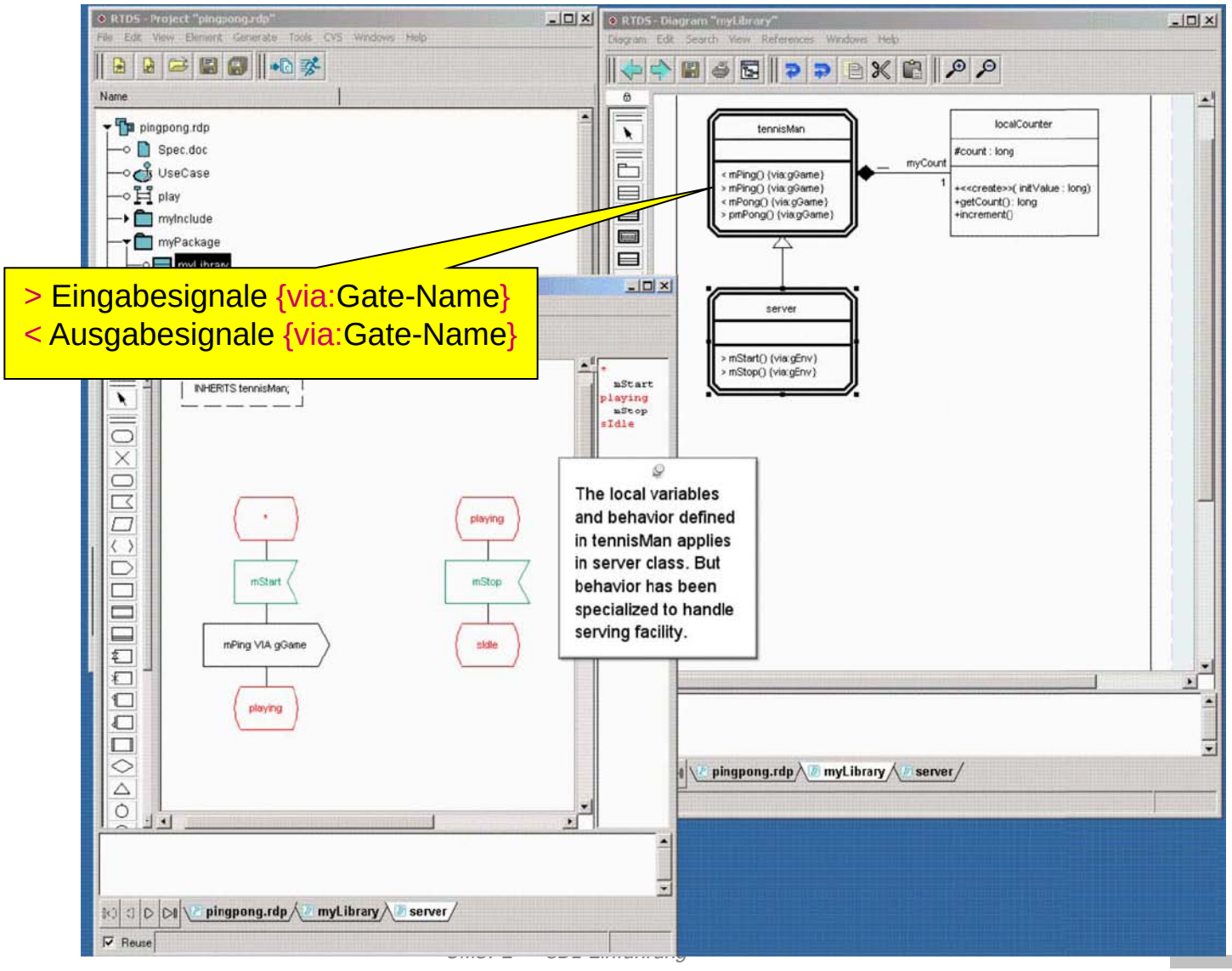
Semaphore-Steuerung (evtl. Blockierung der rufenden Prozess-Instanz

Task:
C++ Code

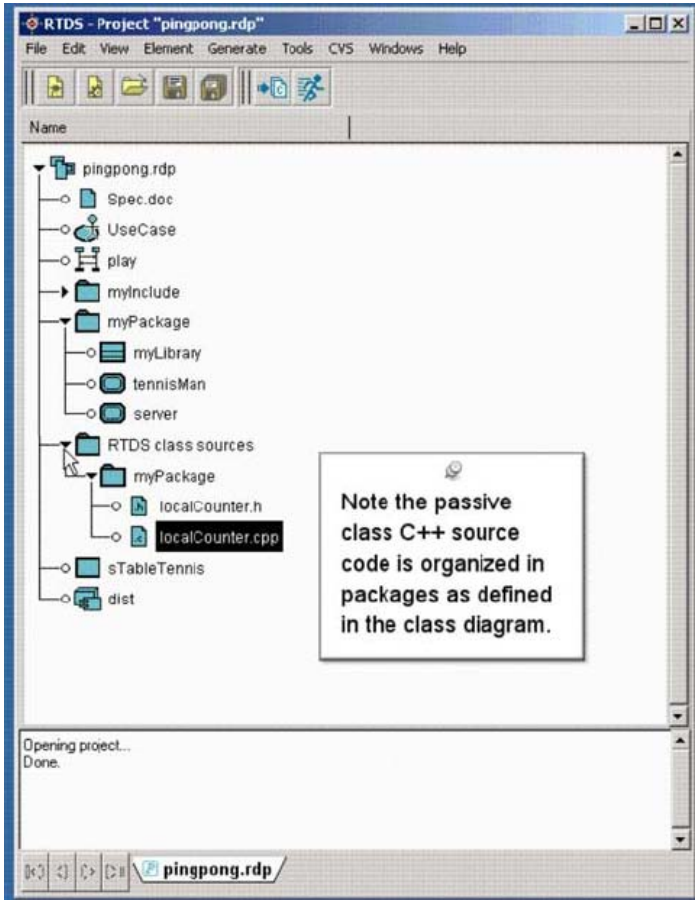
Vererbung von Process-Typen



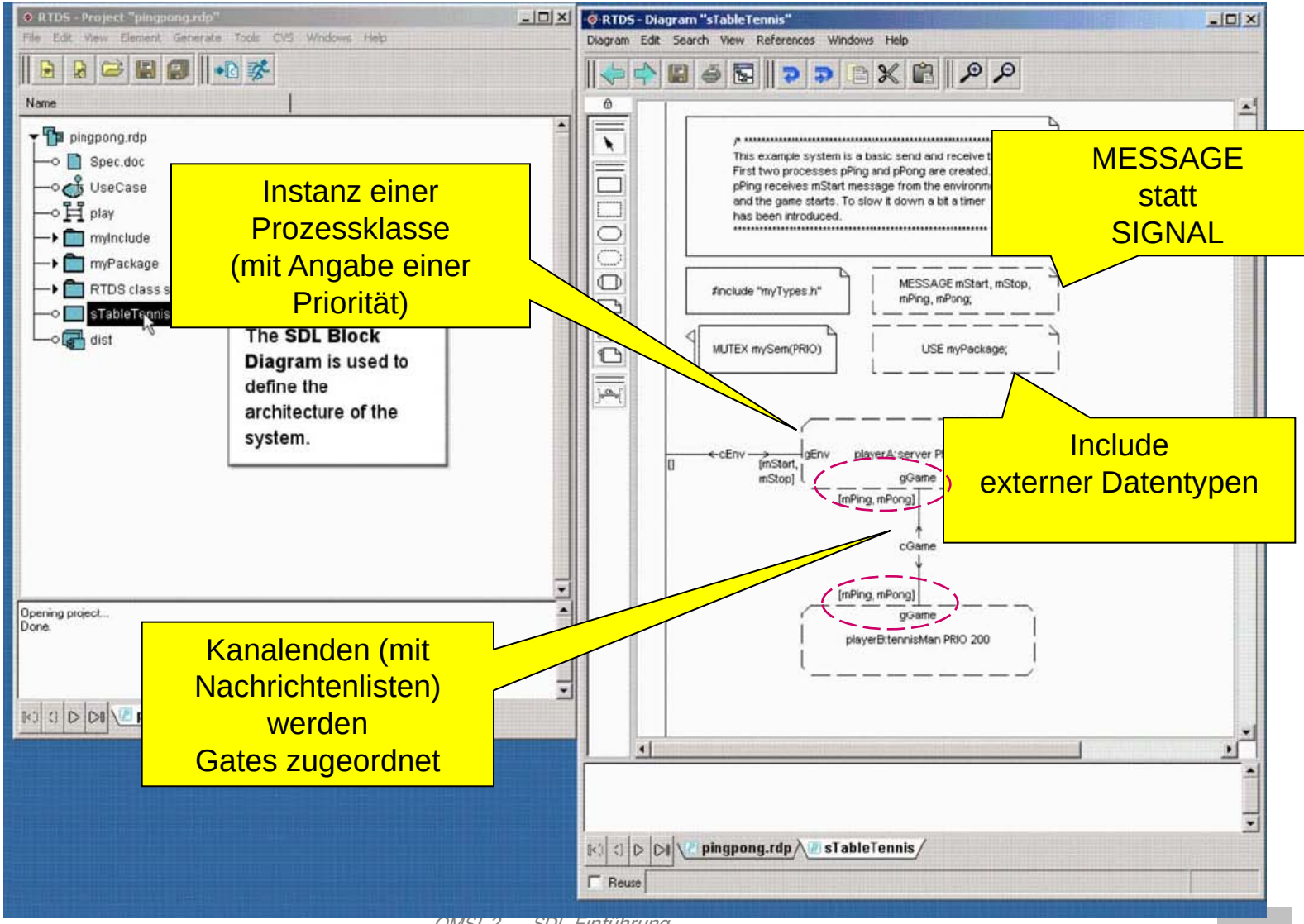
Gates: Interfacedefinition aktiver Klassen



Quelldateien der passiven Klassen



Definition der Systemarchitektur



Verteilungsdiagramm: Knoten, Softwarekomponenten