

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

6. SDL-Konzepte (Präzisierung)

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

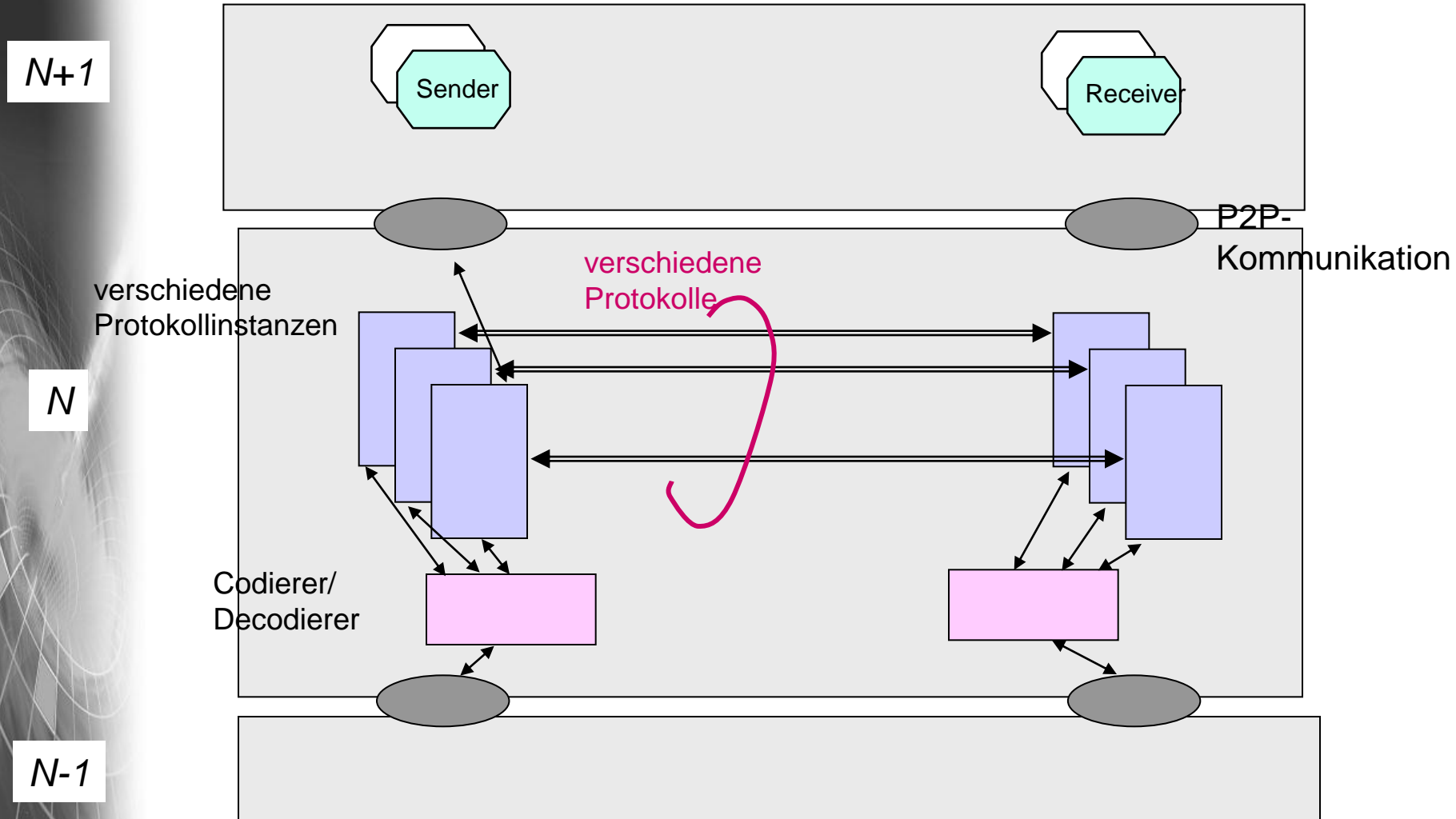
Kap. 7 INRES-Protokoll

später

7. *Protokollentwicklung in SDL*

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

Allgemeine Protokollarchitektur



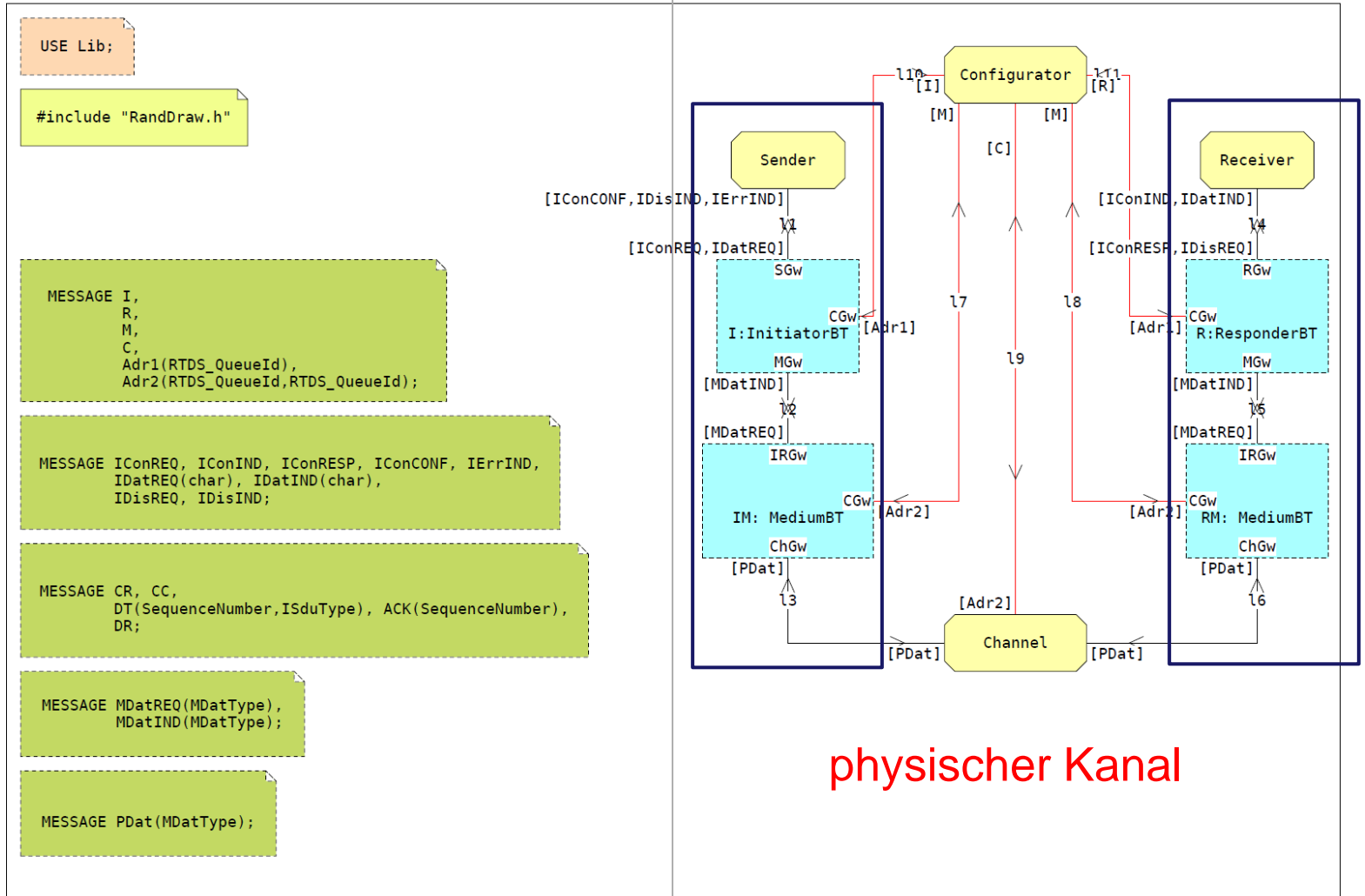
7. *Protokollentwicklung in SDL*

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

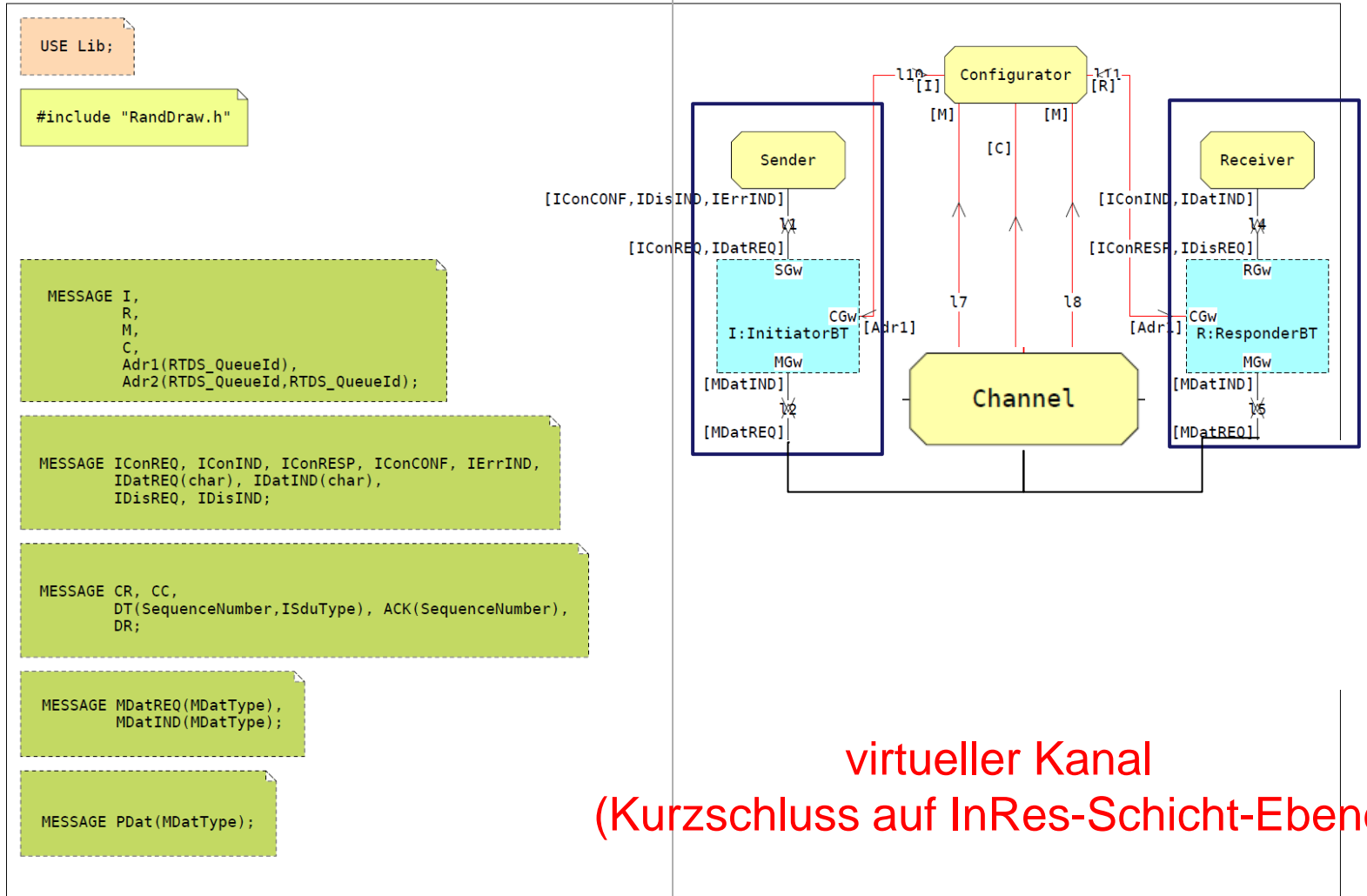
RTDS- InRes-Projektfile: System-Modell



System: Netzkonfiguration (zwei Knoten)

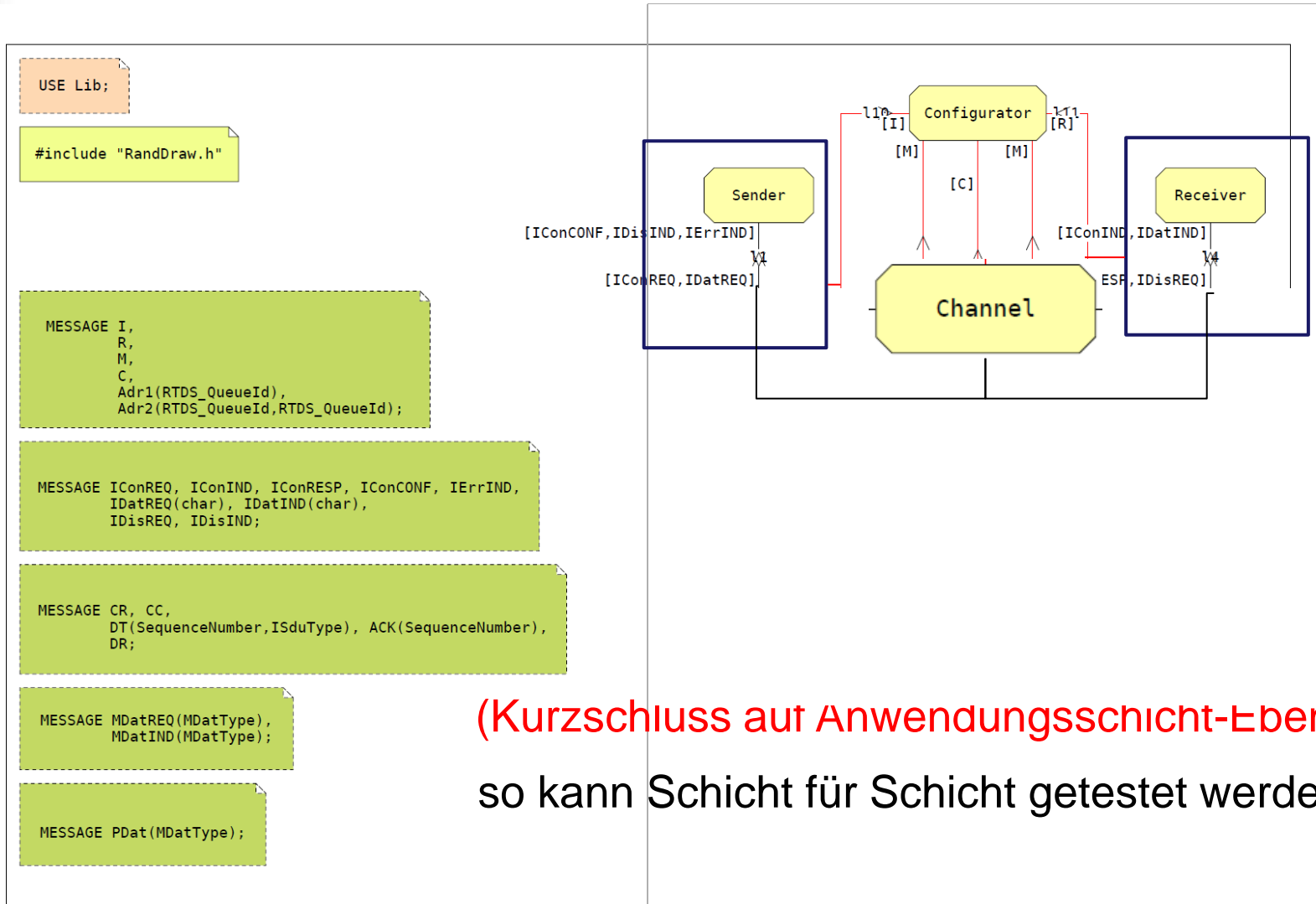


System: Netzkonfiguration (virtueller Kanal)



virtueller Kanal
(Kurzschluss auf InRes-Schicht-Ebene)

System: Netzkonfiguration (virtueller Kanal)



```
USE Lib;
```

```
#include "RandDraw.h"
```

```
MESSAGE I,
R,
M,
C,
Adr1(RTDS_QueueId),
Adr2(RTDS_QueueId,RTDS_QueueId);
```

```
MESSAGE IConREQ, IConIND, IConRESP, IConCONF, IErrIND,
IDatREQ(char), IDatIND(char),
IDisREQ, IDisIND;
```

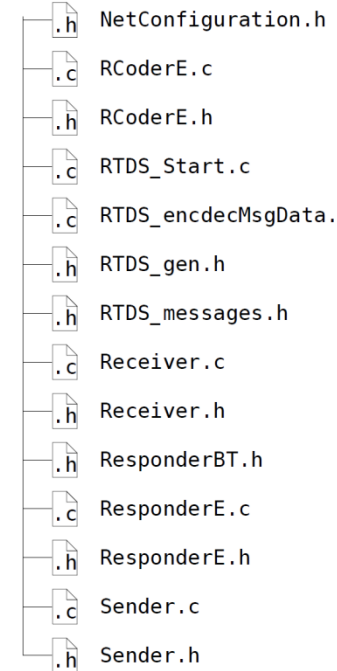
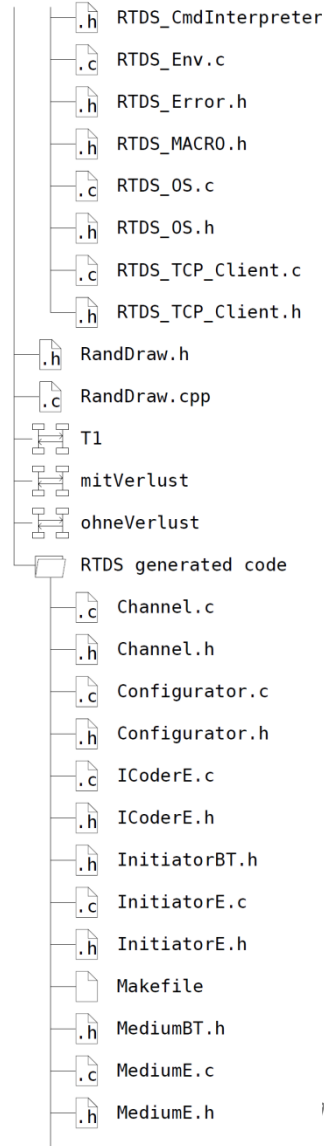
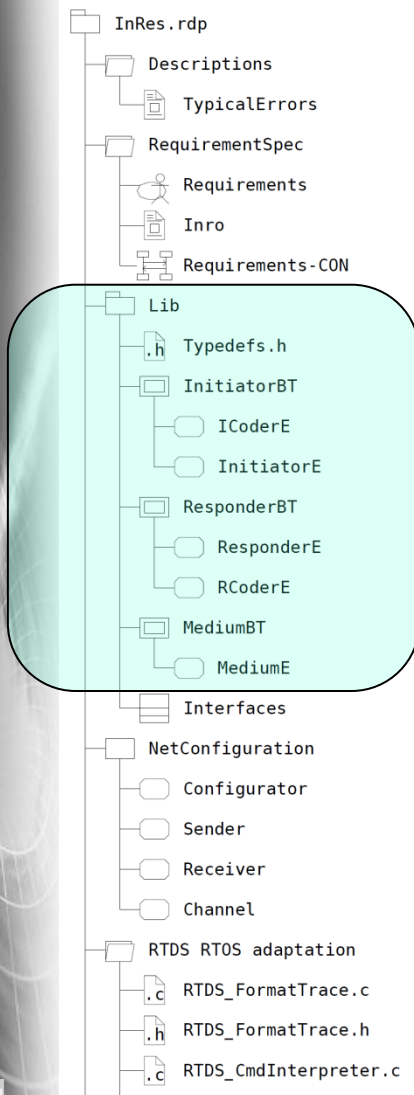
```
MESSAGE CR, CC,
DT(SequenceNumber,ISduType), ACK(SequenceNumber),
DR;
```

```
MESSAGE MDatREQ(MDatType),
MDatIND(MDatType);
```

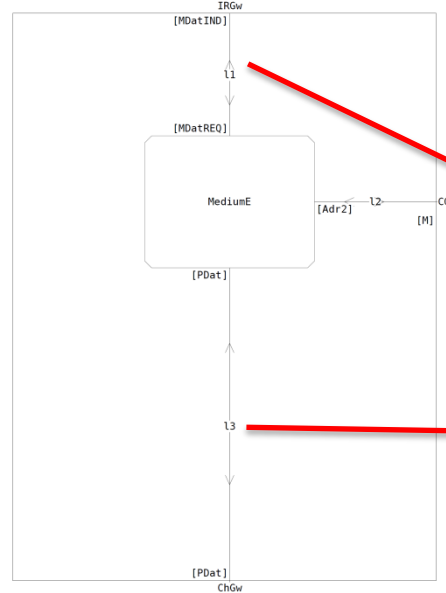
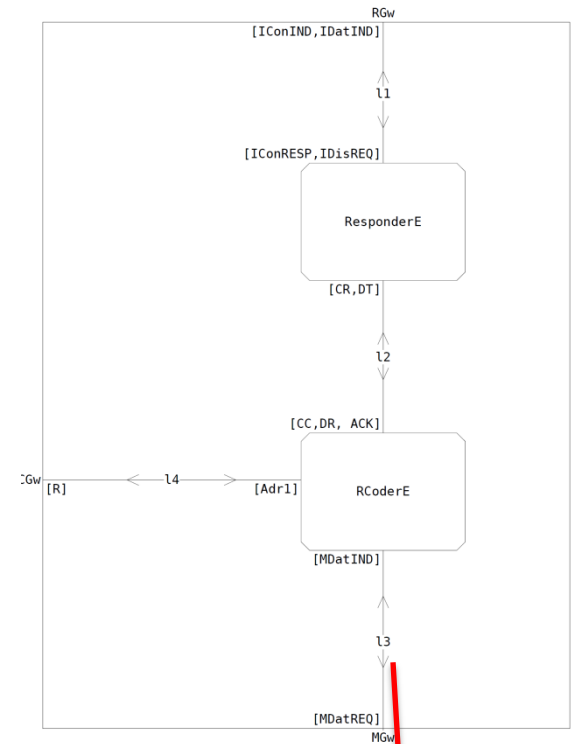
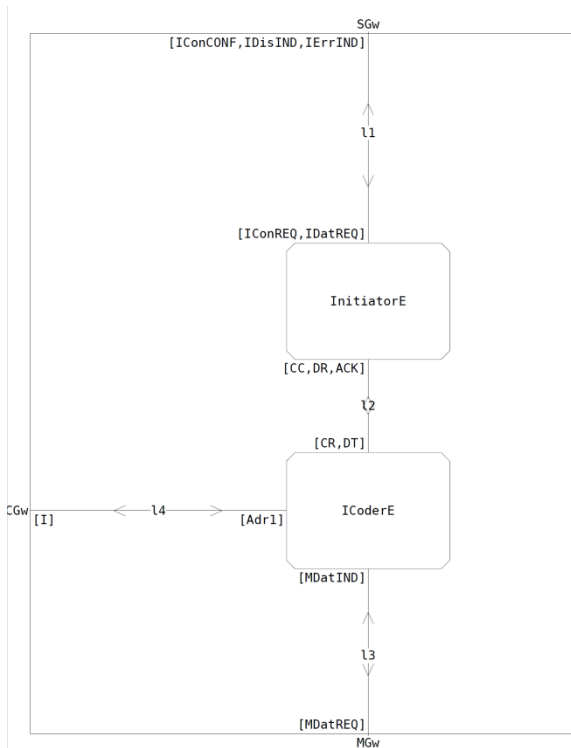
```
MESSAGE PDat(MDatType);
```

(Kurzschluss auf Anwendungsschicht-Ebene)
so kann Schicht für Schicht getestet werden

RTDS- InRes-Projektfile: Blocktypen



Protokolleinheiten der
- InRes-Schicht
InitiatorBT, ResponderBT
- Medium-Schicht
MediumBT



MESSAGE MdatREQ (MdatType),
MdatIND (MdatType);

MESSAGE Pdat (MdatType);

```

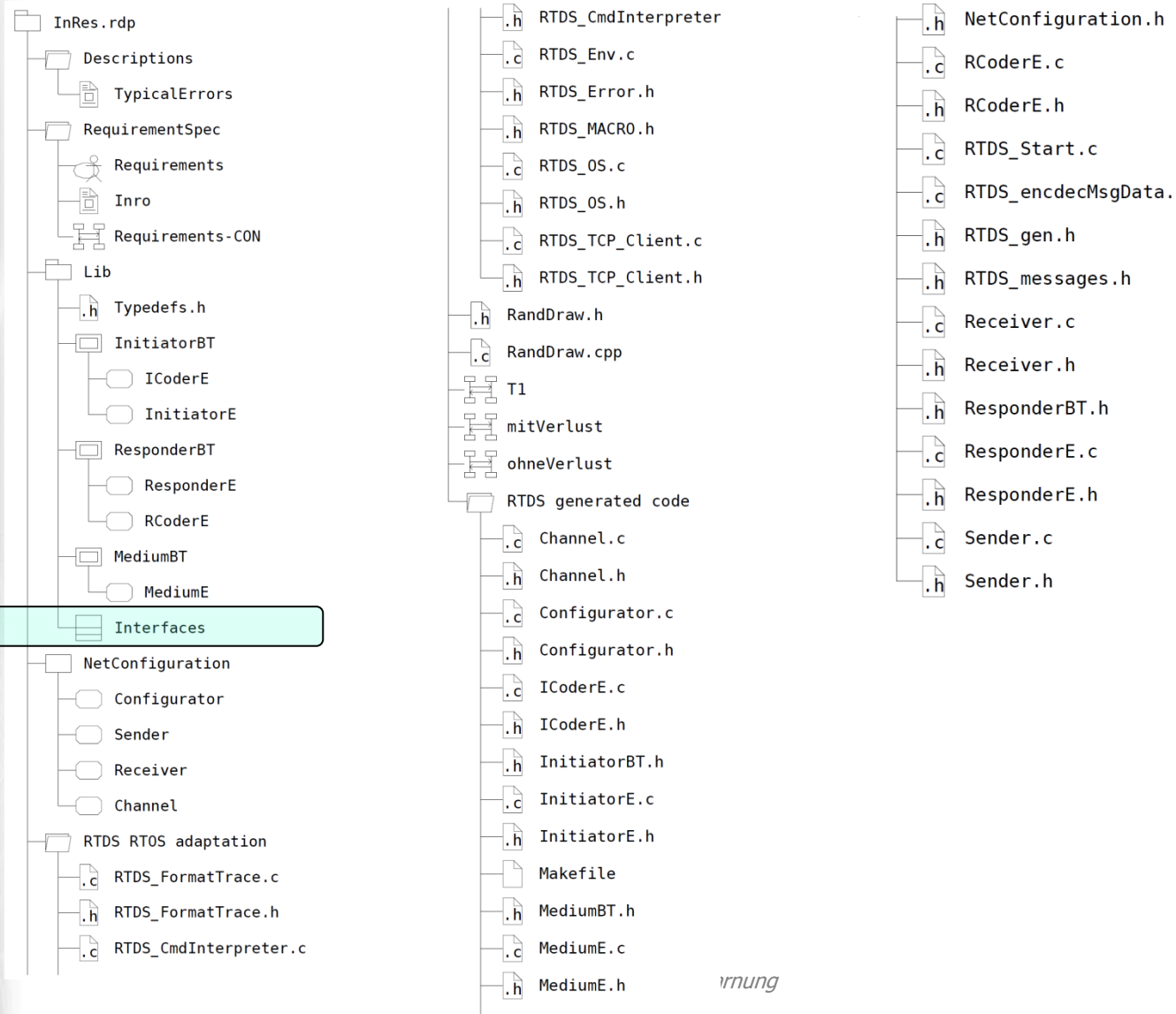
#ifndef TYPEDEFS_H_
#define TYPEDEFS_H_

typedef char ISduType;
typedef int SequenceNumber;

typedef enum
{eCR, eCC, eDR, eDT, eACK} IPduType;

typedef struct MdatType {
    IPduType id;
    SequenceNumber num;
    ISduType data ;
} MdatType;
#endif /*TYPEDEFS_H_*/
  
```

RTDS- InRes-Projektdatei: UML-Interface



Interfaces von Blocktypen

InitiatorBT
<pre>< I() {via:CGw} > Adr1(in p1:RTDS_QueueId) {via:CGw} > IConREQ() {via:SGw} > IDatREQ(in p1: char) {via:SGw} < IConCONF() {via:SGw} < IDisIND() {via:SGw} < IErrIND() {via:SGw} > MDatIND(in p1:MDatType) {via:MGw} < MDatREQ(in p1:MDatType) {via:MGw}</pre>

ResponderBT
<pre>< R() {via:CGw} > Adr1(in p1:RTDS_QueueId) {via:CGw} < IConIND() {via:RGw} < IDatIND(in p1:char) {via:RGw} > IConRESP() {via:RGw} > IDisREQ() {via:RGw} > MDatIND(in p1:MDatType) {via:MGw} < MDatREQ(in p1:MDatType) {via:MGw}</pre>

MediumBT
<pre>< M() {via:CGw} > Adr2(in p1:RTDS_QueueId, in p2:RTDS_QueueId) {via:CGw} < MDatIND(in p1:MDatType) {via:IRGw} > MDatREQ(in p1:MDatType) {via:IRGw} < PDat(in p1:MDatType) {via:ChGw} > PDat(in p1:MDatType) {via:ChGw}</pre>

Protokolleinheiten wurden nicht mit expliziter Typangabe definiert

deshalb:

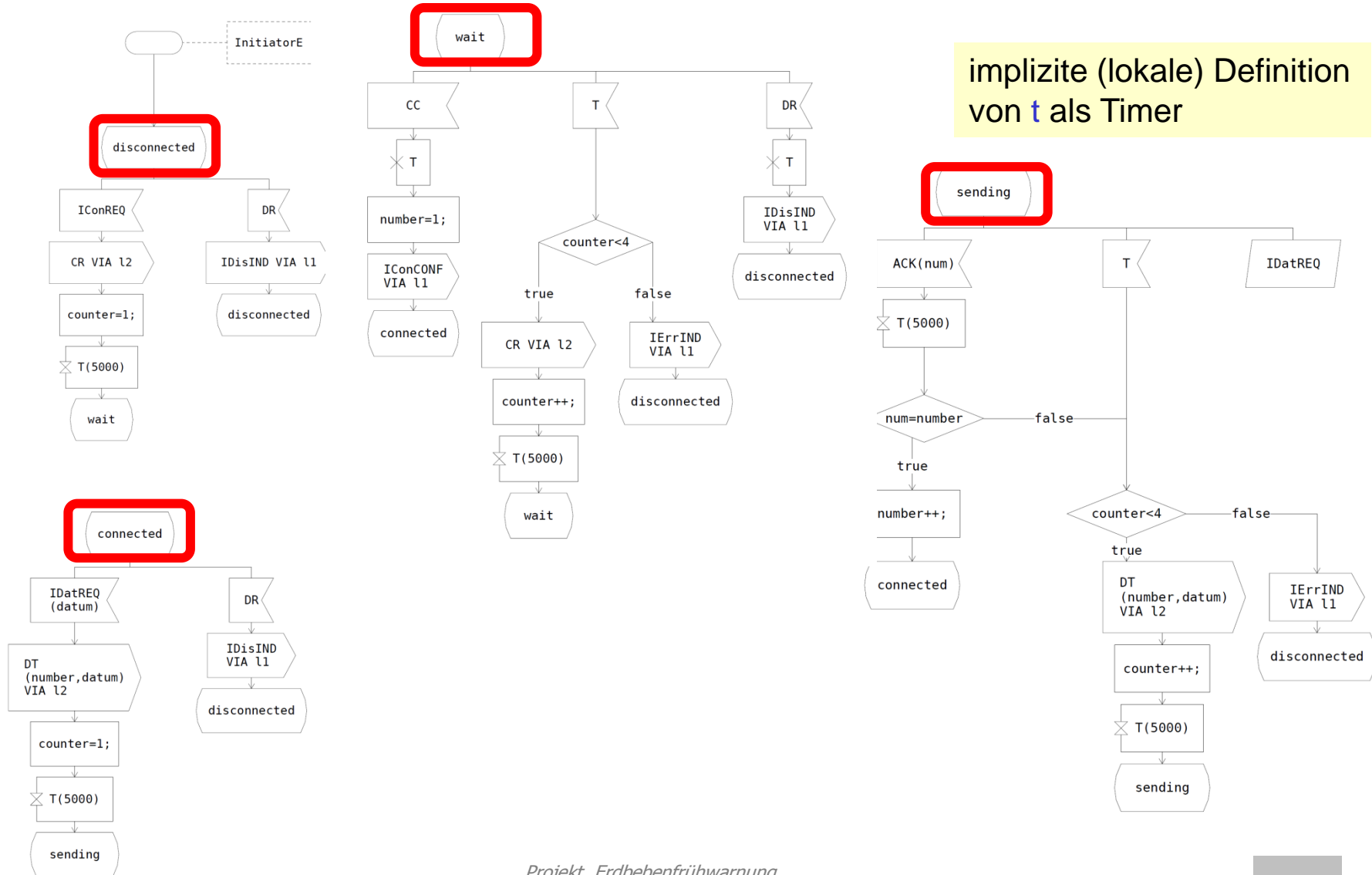
- entfällt deren Interface-Spec;
sie sind ja bereits in einem (und nur einem) Kontext eingebettet

```

int counter=0;
char datum;
SequenceNumber num, number;

```

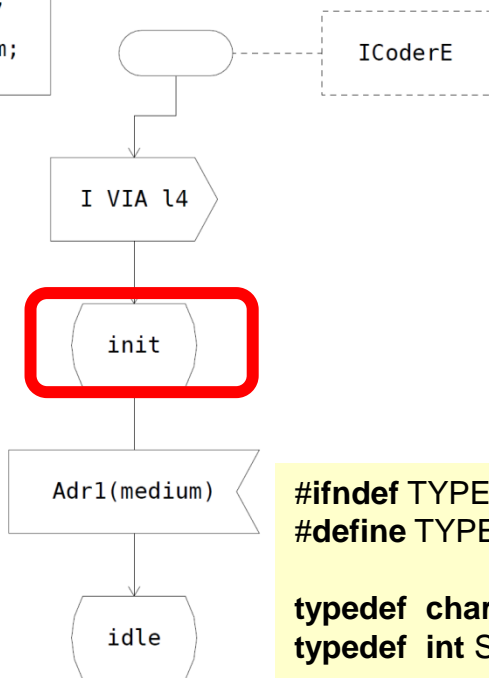
Initiator-Process



implizite (lokale) Definition von `t` als Timer

Codierer: (Initiator-seitig)

```
char datum;
SequenceNumber num;
MDataType mdatum;
RTDS_QueueId medium;
```

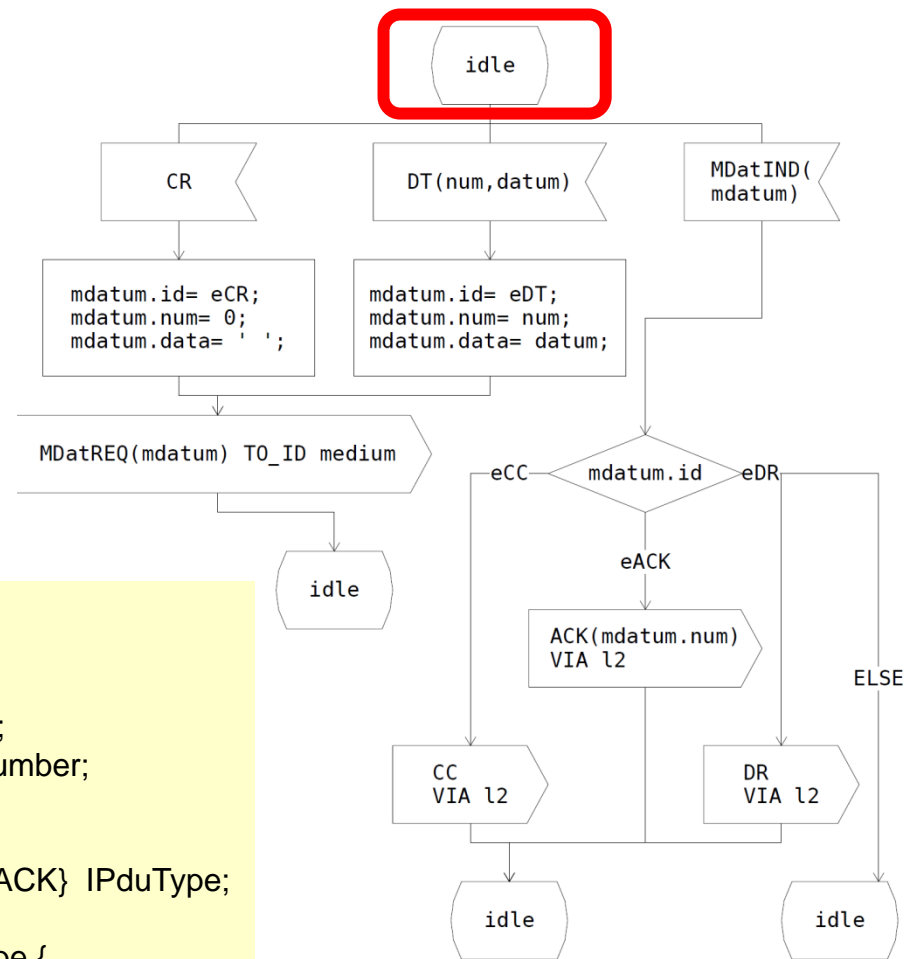


```
#ifndef TYPEDEFS_H_
#define TYPEDEFS_H_

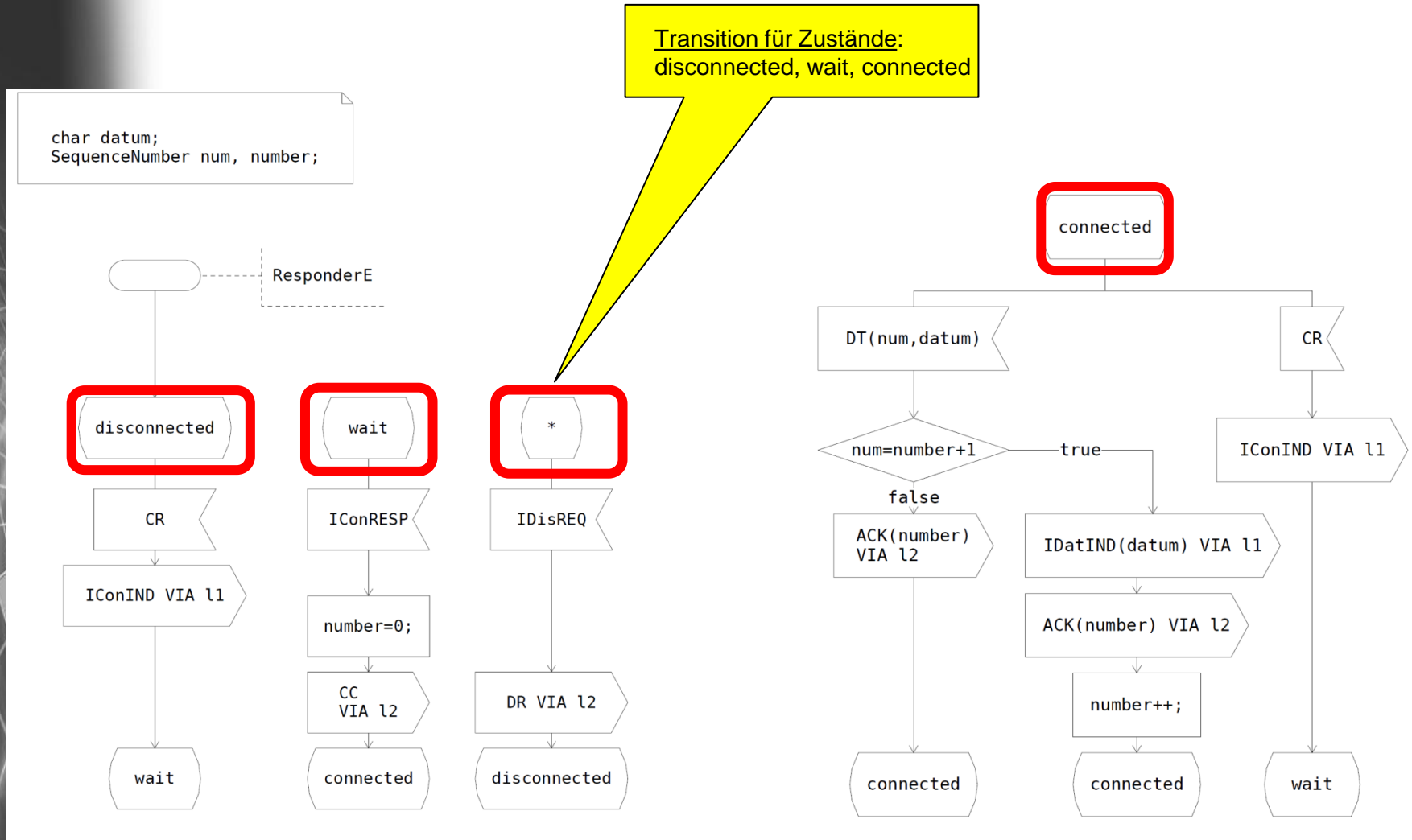
typedef char ISduType;
typedef int SequenceNumber;

typedef enum
{eCR, eCC, eDR, eDT, eACK} IPduType;

typedef struct MDataType {
    IPduType id;
    SequenceNumber num;
    ISduType data ;
} MDataType;
#endif /*TYPEDEFS_H_*/
```

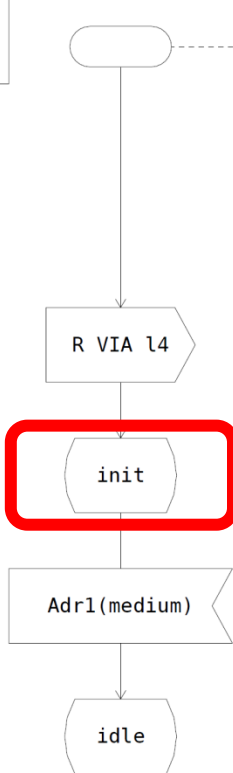


Responder-Process



Codierer: (Responder-seitig)

```
SequenceNumber num;
MDataType mdatum;
RTDS_QueueId medium;
```

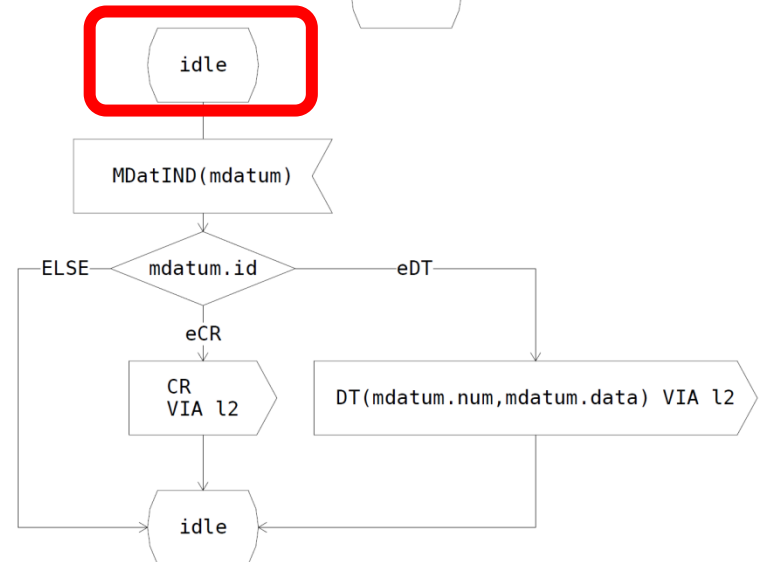
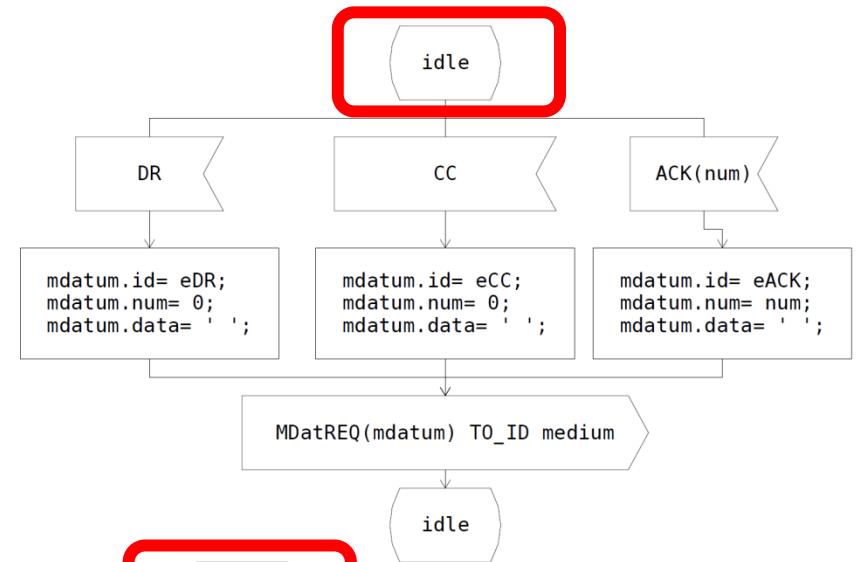


```
#ifndef TYPEDEFS_H_
#define TYPEDEFS_H_
```

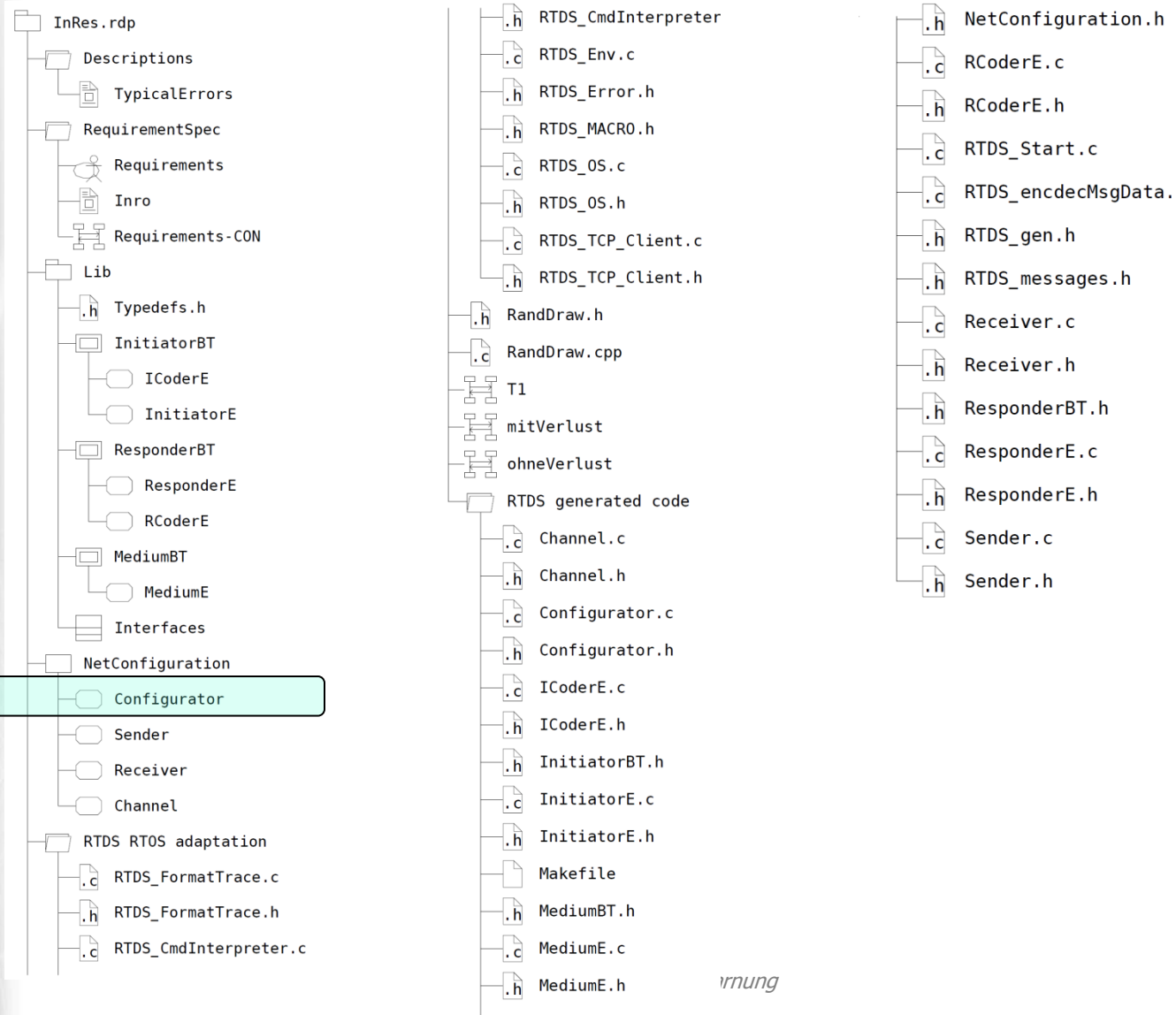
```
typedef char ISduType;
typedef int
SequenceNumber;
```

```
typedef enum
{eCR, eCC, eDR, eDT,
eACK} IPduType;
```

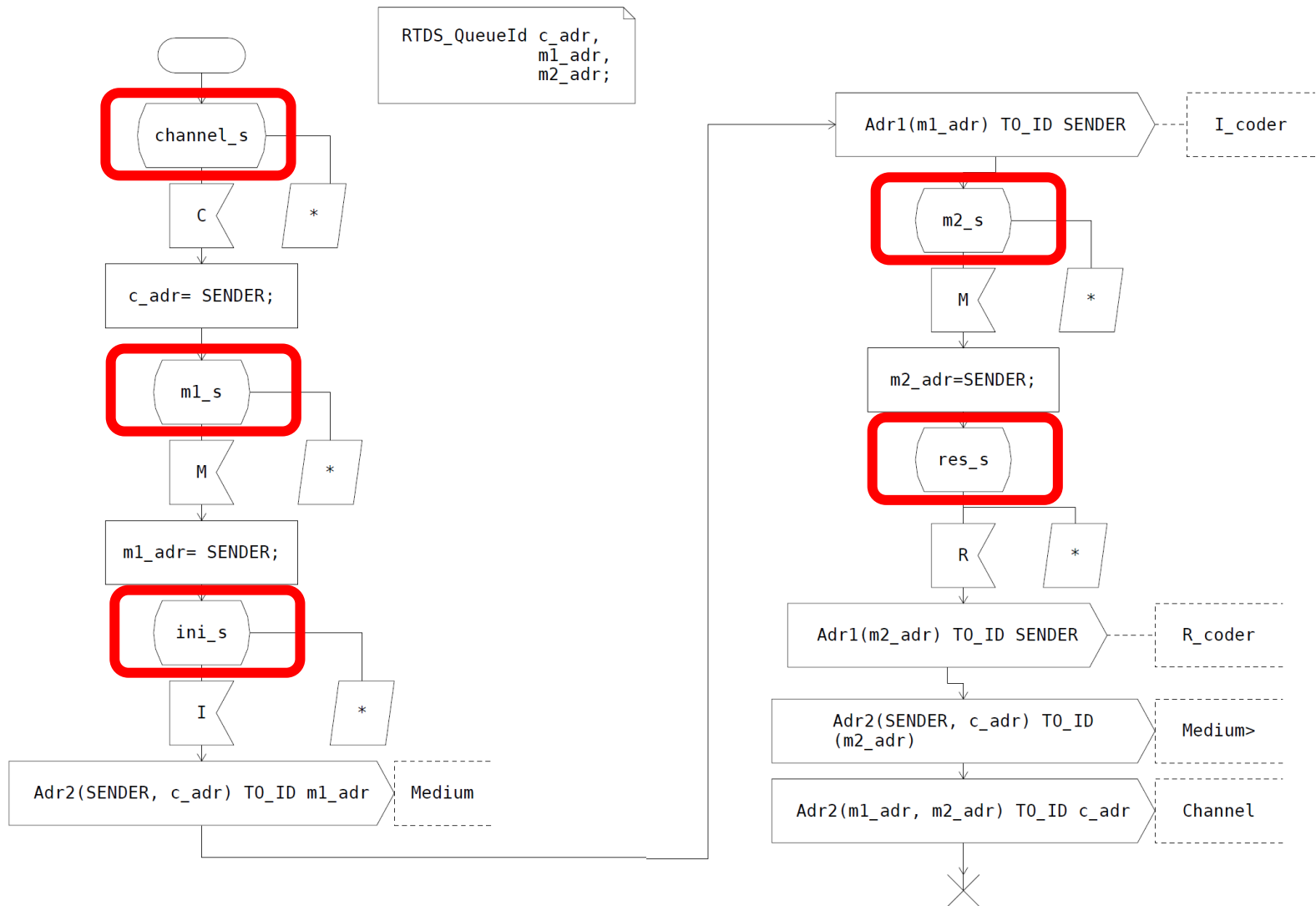
```
typedef struct MDataType {
int id;
SequenceNumber num;
ISduType data ;
} MDataType;
#endif /*TYPEDEFS_H_*/
```



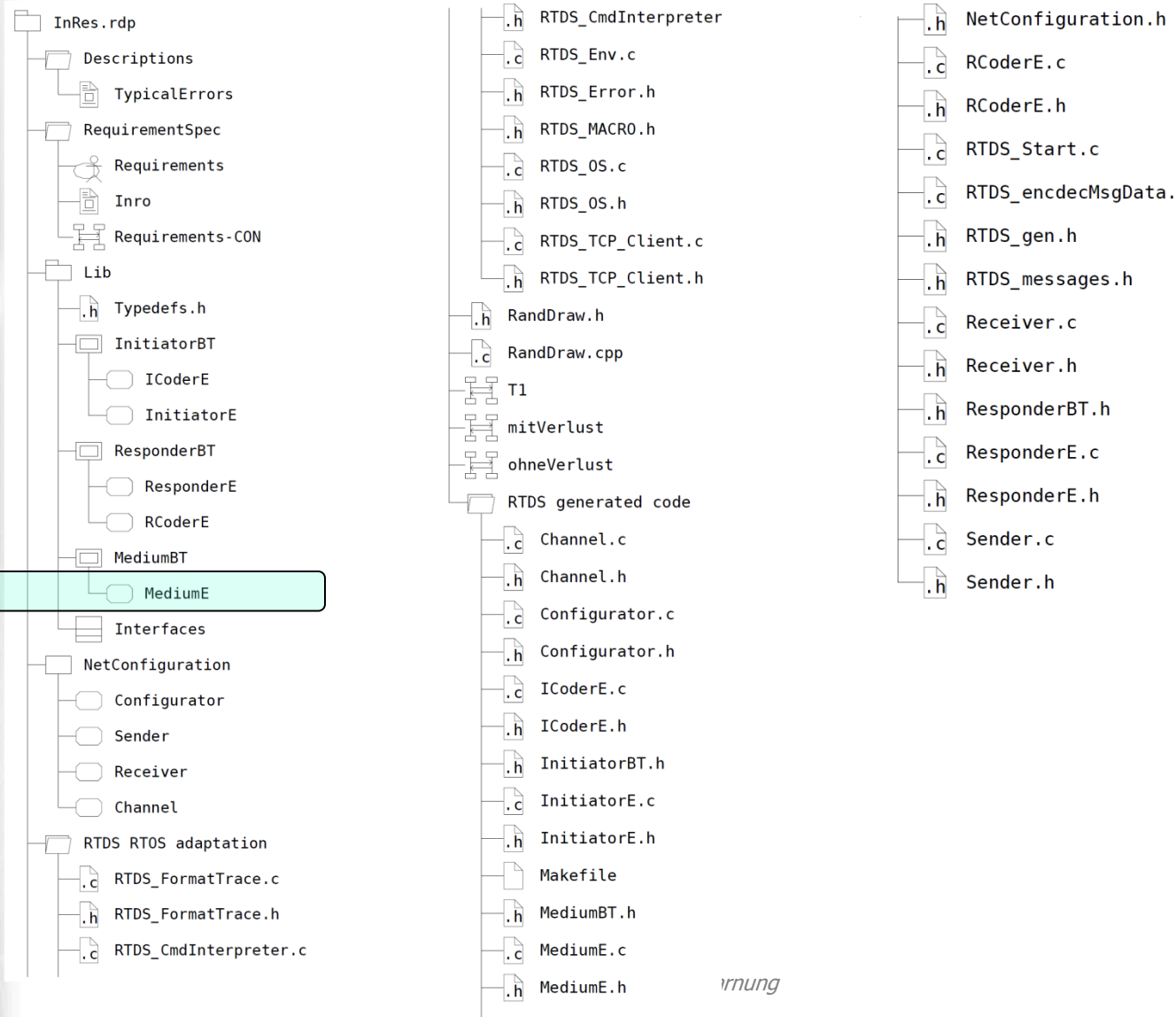
RTDS- InRes-Projektdatei: Configurator



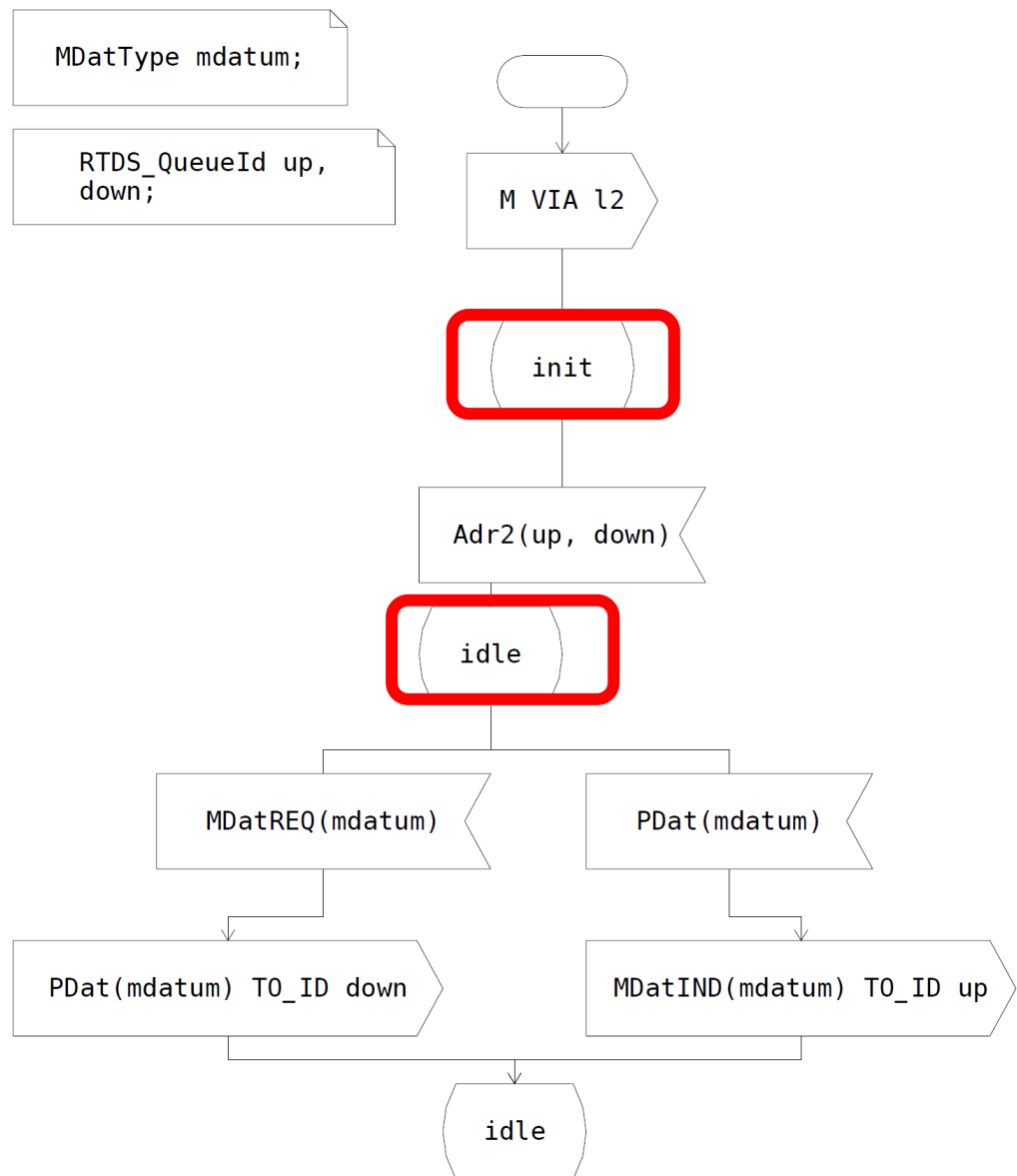
Systemkonfigurator



RTDS- InRes-Projektdatei: Medium-Protokolleinheit



Medium-Entity: (Initiator- und Responder-seitig)



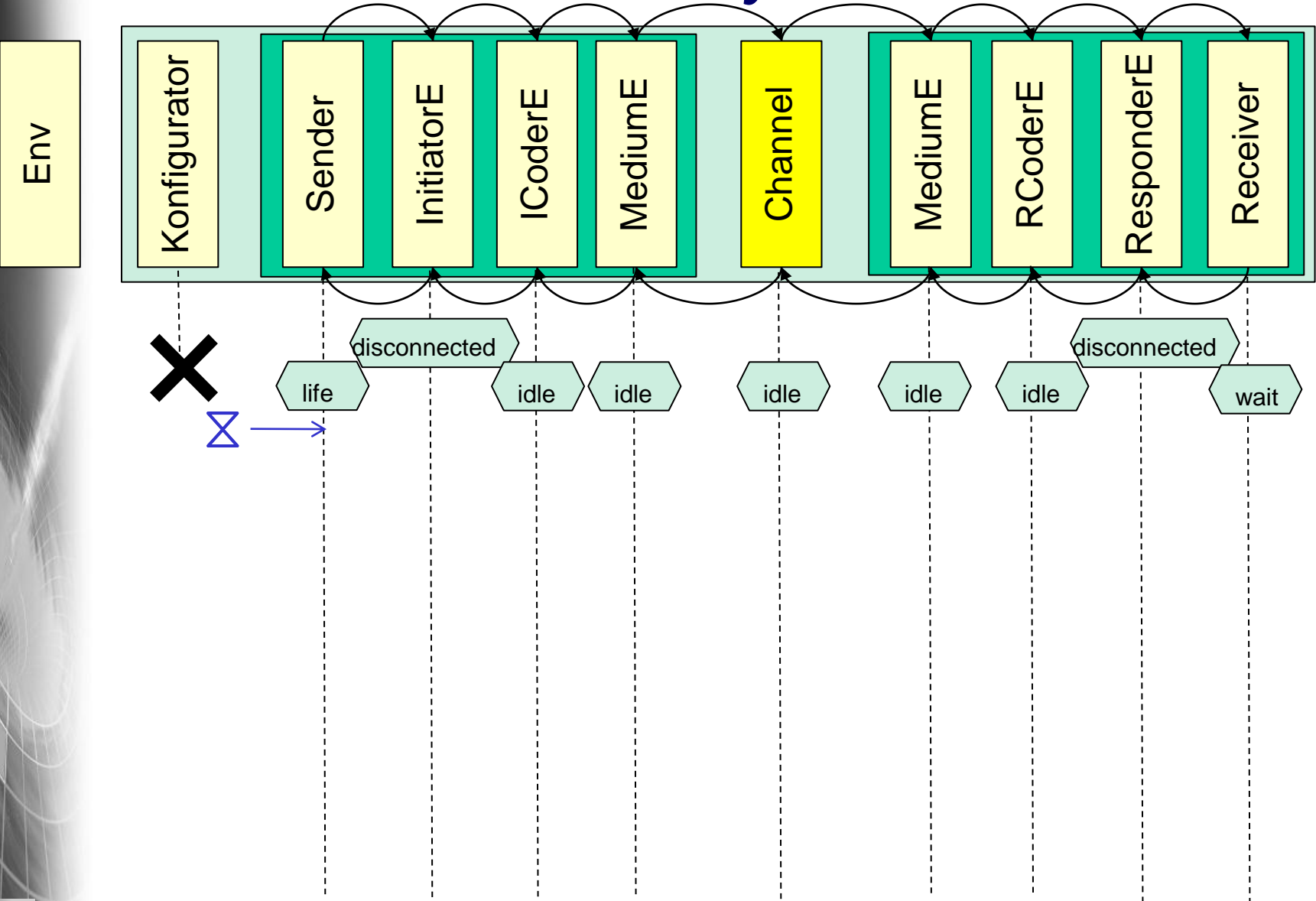
```
#ifndef TYPEDEFS_H_
#define TYPEDEFS_H_

typedef char ISduType;
typedef int
SequenceNumber;

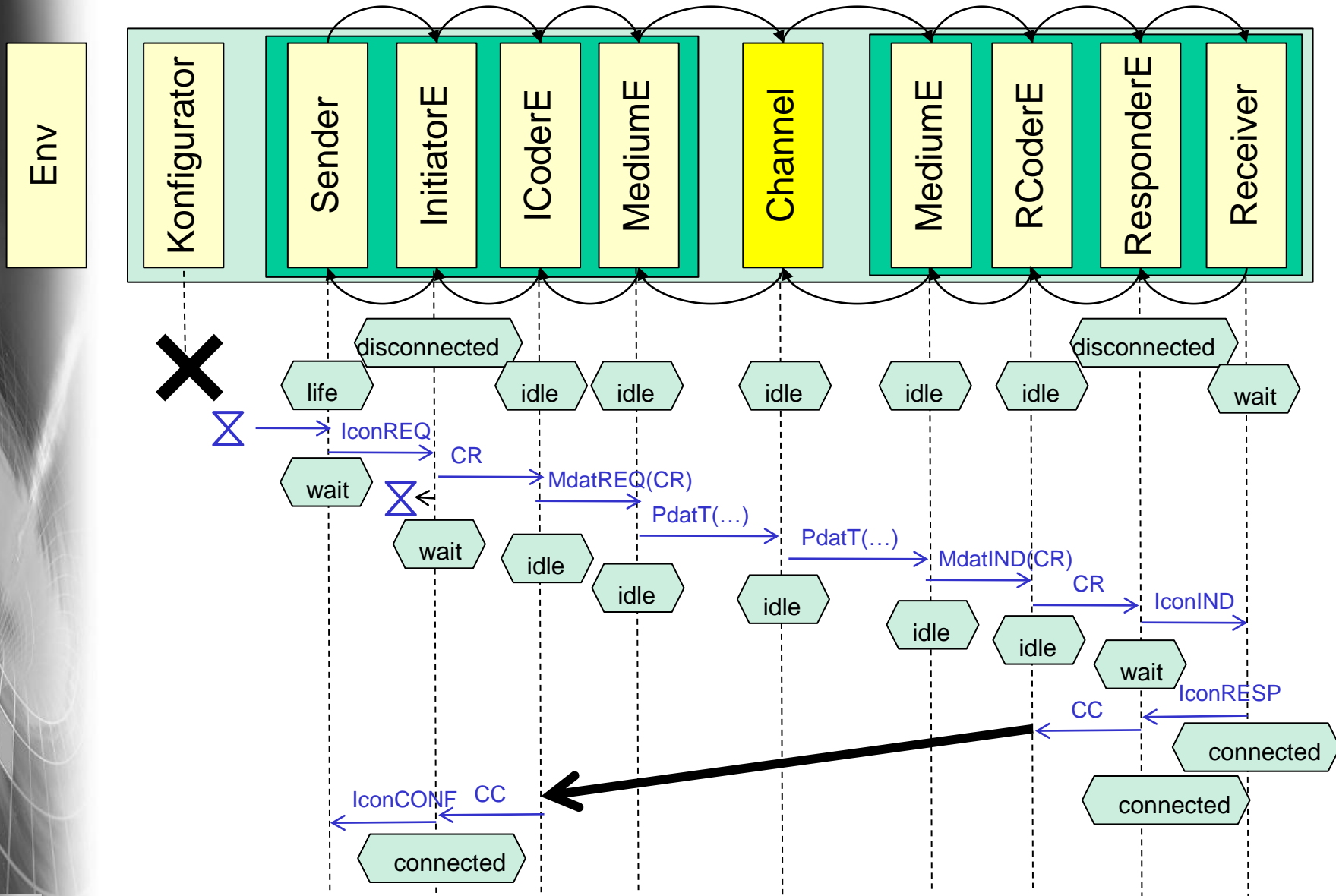
typedef enum
{eCR, eCC, eDR, eDT,
eACK} IPduType;

typedef struct MDatType {
    int id;
    SequenceNumber num;
    ISduType data ;
} MDatType;
#endif /*TYPEDEFS_H_*/
```

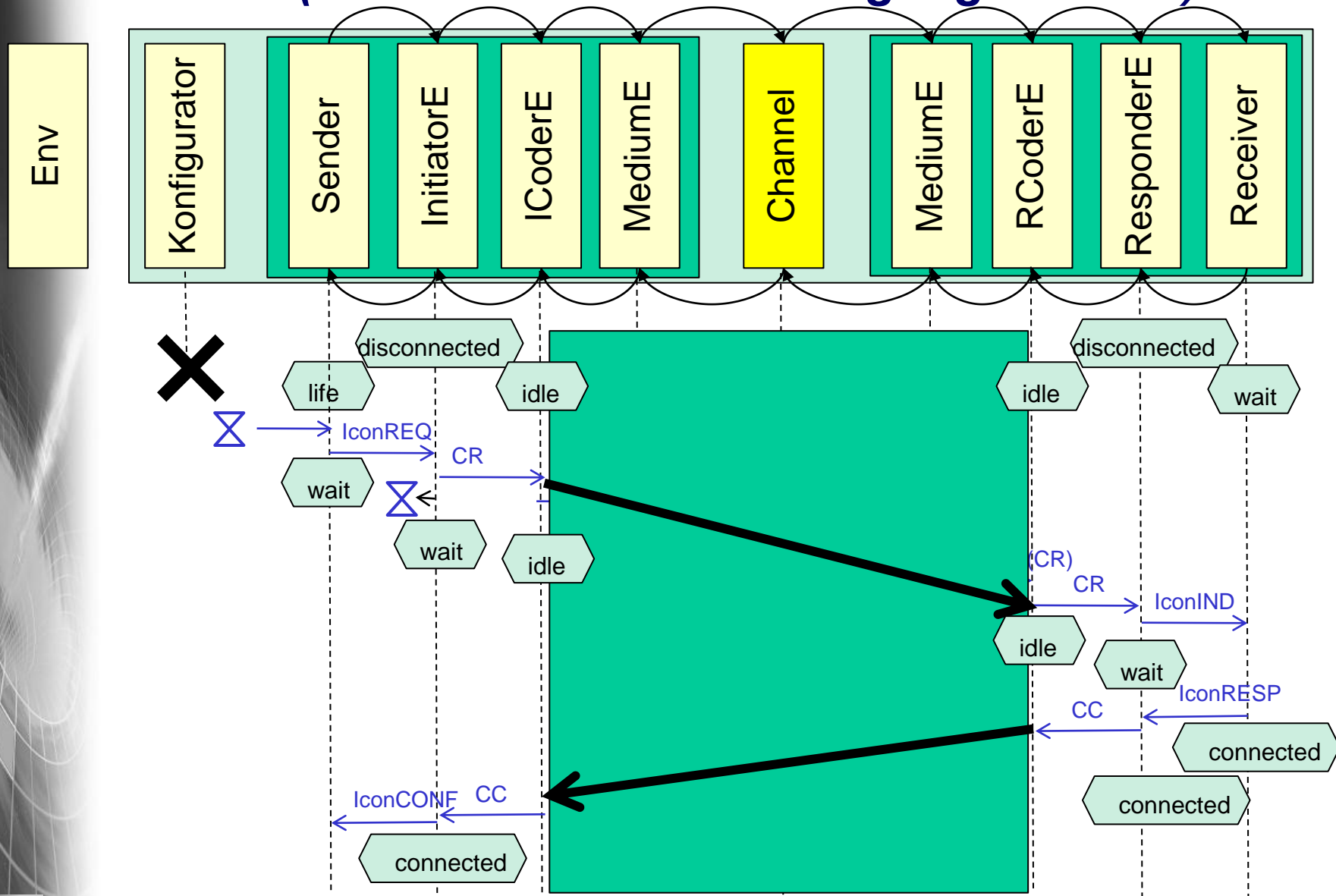
Aufbau eines initialen Systemzustandes



Ablauf



Ablauf (Ausblenden der Übertragungsschicht)



6. SDL-Konzepte (Präzisierung)

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

Kap. 7 INRES-Protokoll

Fortsetzung jetzt

Prozeduren in SDL

... in zwei Ausprägungen, die zweckmäßiger Weise unterschiedliche Codegenerierungsvarianten nach sich ziehen

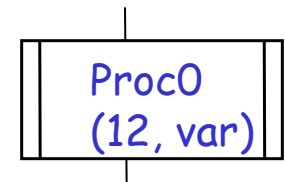
- a) **zustandsbehaftet**
 - Urform: ohne Rückgabewert
 - erweiterte Form: mit Rückgabewert (Spezialfall der Urform)
- b) **zustandslos** (Spezialfall von a)
- c) Funktionen/Memberfunktionen gab es vor SDL-2000 nicht (nur Operatoren)

Diagrammarten und Aufruf in Standard-SDL:



*Referenzsymbol
ohne Signaturangabe!*

Prozedur-Diagramm

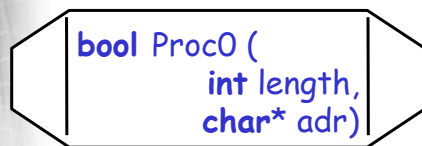


Prozedur-Aufruf

Prozeduren und Funktionen in SDL/RT

- SDL-Prozeduren
 - a) zustandslos / zustandsbehaftet: ohne Rückgabewert
 - b) zustandslos / zustandsbehaftet: mit Rückgabewert
- C-/C++ Funktionen, Member-Funktionen

Diagrammarten und Aufruf in SDL/RT:

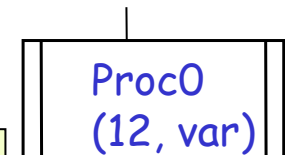


*Referenzsymbol
mit Signaturangabe!*

Prozedur-Diagramm

Procedure Proc0

Zustandsgraph



Aufruf

PROCEDURE-Call in SDL/RT

Aufruf **nicht** innerhalb von Tasks erlaubt

```
...;  
x= myProc(p);  
...
```

...es sei denn,
dass **myProc** eine C-Funktion ist

Aufruf einer
zustandslosen SDL-Prozedur
entspricht einer C-Funktion
(empfohlen in SDL/RT)

Achtung: Rekursivität ist erlaubt

```
r= oneProc(p)
```

```
anotherProc(p)
```

Aufruf einer
echten SDL-Prozedur
(zustandsbehaftet,
ohne Rückkehrwert)

Aufruf an allen Stellen erlaubt, wo Tasks
im Zustandsdiagramm möglich sind


Zustandsbehaftete Prozeduren in SDL

... sind parametrisierbare Zustandsübergangs-Teilgraphen eines Prozesses mit:

- **dynamischen** Kontextinformationen des aufrufenden Prozesses (Timer, Inputpuffer, Empfangsnachrichten)
- **statischen** Kontextinformationen der Deklarationsumgebung: Variablen, Funktionen, Typen
- **eigenem** Namensraum für *Zustände, Marken, Variablen, Datentypen*

Achtung: lokale Nachrichten und Timer sind dabei **nicht** zugelassen
Warum nicht?

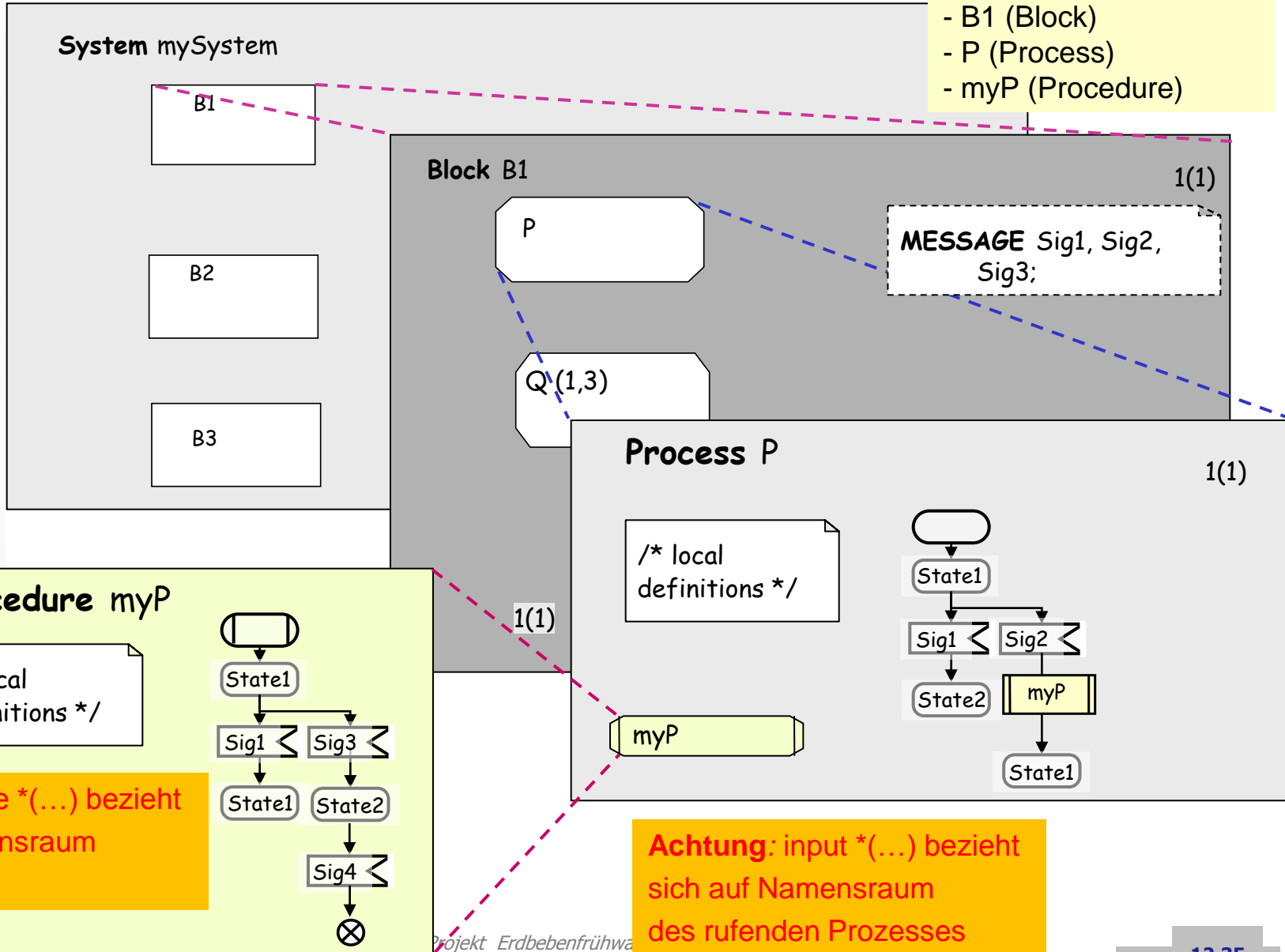
- eigenem *Start-Symbol* 

- eigenem *Return-Symbol* und evtl. Rückgabewert  *<expression>*

Geschachtelte Namensräume

separate Namensräume

- mysystem (System)
- B1 (Block)
- P (Process)
- myP (Procedure)



Erlaubte Deklarationsniveaus

- Package
- System (Systemtyp): system-global
- Block (Blocktyp): block-global
- Process (Processtyp): prozess-global
- ~~Service (Servicetyp): service-global~~
- Procedure: prozedur-global

Erlaubte Aufrufkontexte

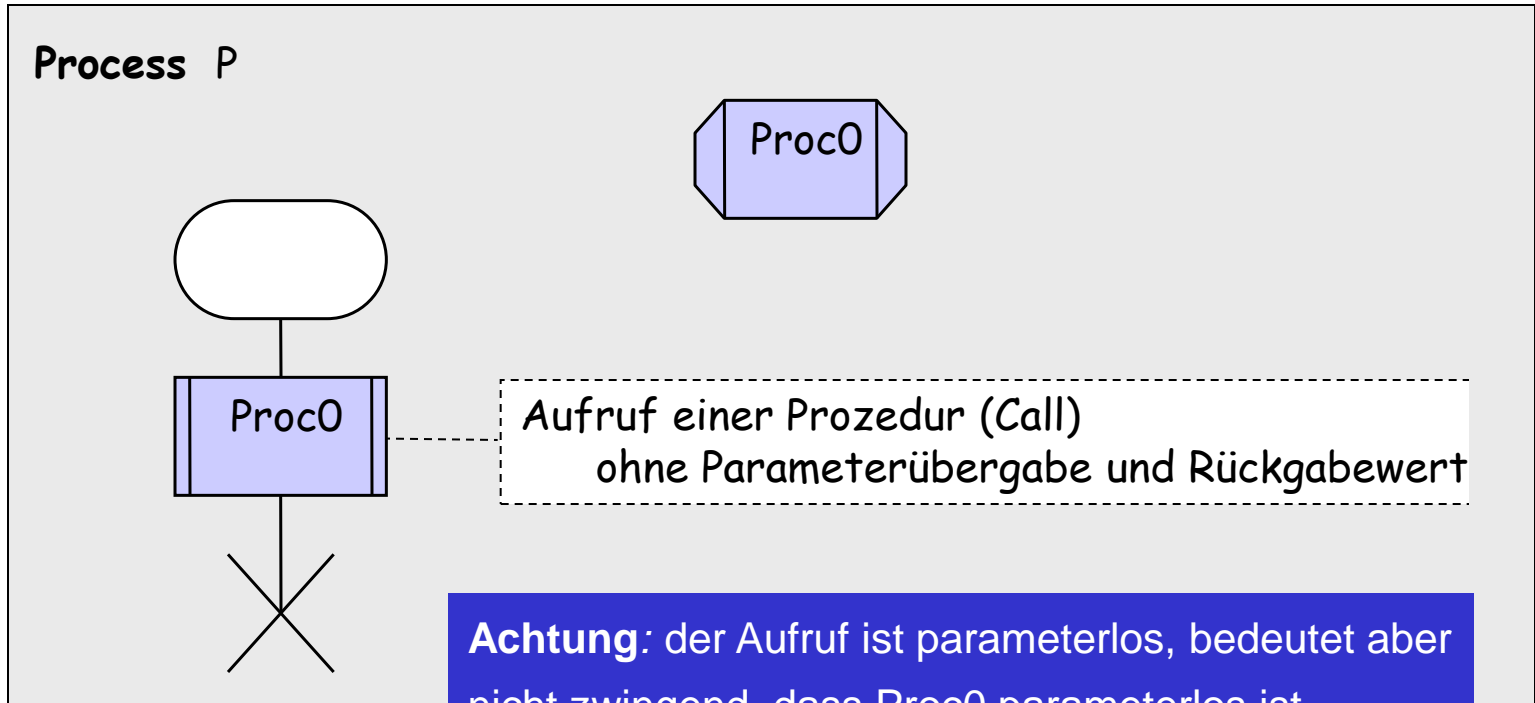
Verhaltensgraph von einem

- Process
- ~~Service~~
- Procedure

Prozedur-Verhaltensgraph
operiert über Eingangspuffer
des Prozesses, zu dem der
Aufrufkontext gehört

Impliziter Parameter
Bezug zum Kontext vom
Aufrufer-Prozess

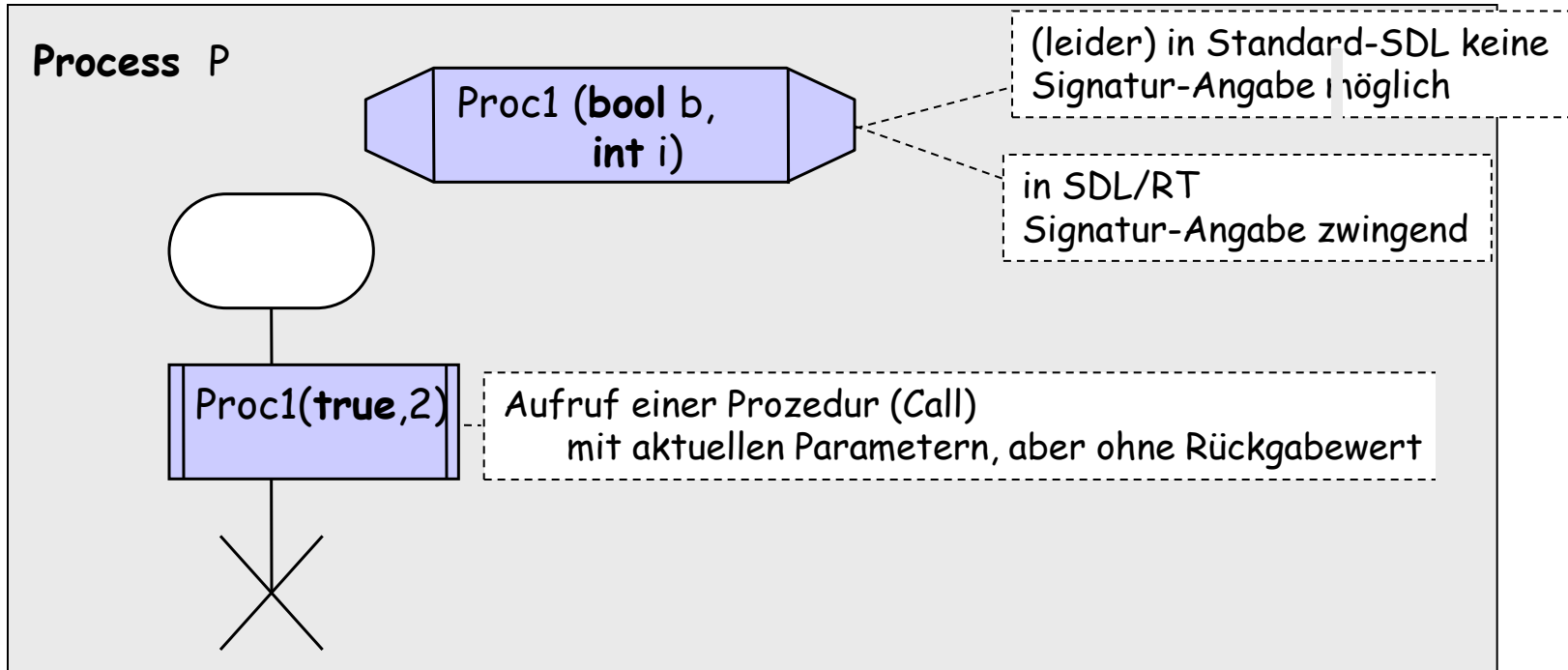
Deklaration und Aufruf einer Prozedur ohne Rückgabewert (1)



Achtung: der Aufruf ist parameterlos, bedeutet aber nicht zwingend, dass Proc0 parameterlos ist (dieselbe Konvention wie bei Signalparametern)

- Aufruf einer Prozedur **ohne** Rückgabewert darf an allen Stellen erfolgen, wo TASK-Aktionen erlaubt sind
- In SDL/RT: für Aufruf einer Prozedur **mit** Rückgabewert gilt gleiche Regel !!! (Abweichung vom Standard)

Deklaration und Aufruf einer Prozedur ohne Rückgabewert (2)



Parameterübergabearten

für Prozeduren in Standard-SDL

- **in** (default)
call-by-value: Ausdruckswerte der aktuellen Parameter werden in Kopie den formalen Parametern zugewiesen
- **in/out**
call-by-reference: formale Parameter agieren als Synonyme der aktuellen Parameter während der Prozedur-Ausführung

für Prozesse und Nachrichten

- nur **in** möglich

für Prozeduren in SDL/RT

- **C-Konventionen**

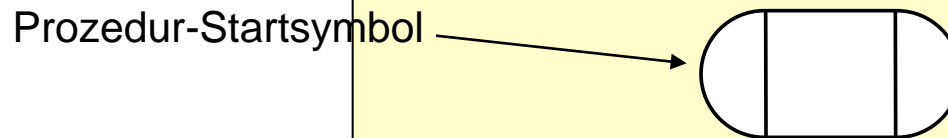
Beispiel: Definition einer zustandslosen Prozedur

SDL/PR

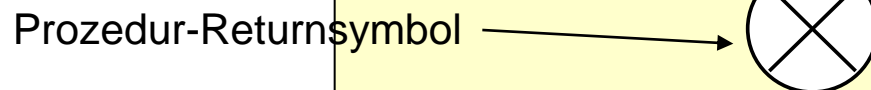
```
int procedure P (n int, b bool, r &double);  
  start;  
  return if b then n else n + int(r);  
endprocedure P;
```

SDL/GR

```
int procedure P (n int, b bool, r &double);
```



Task, Create, Call, Decision, Label, Join möglich →



if b then n else n + int(r)

Zustandsbehaftete Prozedur

Zustandsgraph ähnelt dem eines Prozesses

Besonderheiten



- eigenes start- Symbol
 - Graph kann **nicht** mit Input beginnen
- eigene Zustandsmenge
- eigenes stop-Symbol (return)
- Marken und Zustände des Rufers können **nicht** aktiviert werden
 - `state *(...)` bezieht sich **nur** auf Namensraum der Prozedur
- Zuordnung des Signalpuffers: rufender Prozess
- gültige Signal-Inputmenge stammt vom rufenden Prozess
 - `input *(...), save *(...)`

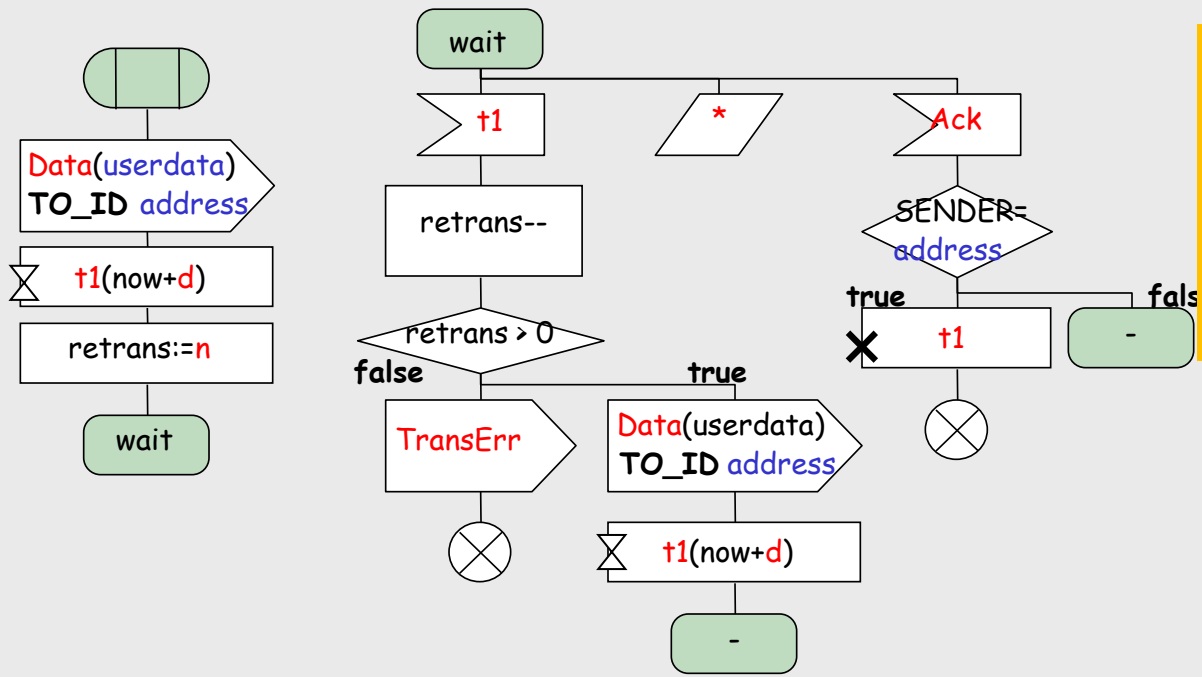
Procedure sendData (userdata Userdata; address RTDS_QueueId)

```

retrans int; /* Anzahl möglicher Übertragungswiederholungen */
/* Kontextinformationen
d Duration;
n int;
struct UserDataType ...;
MESSAGE Data(UserdataType), Ack, TransErr;
timer ...;
signalset ...;
*/

```

Bereitstellungs-
möglichkeiten



SDL/RT
staischer
globaler
Namensraum
(System,
Block,
Prozess)

Standard-SDL
als
expliziter
dynamischer
Kontext-
Parameter

Allgemeines Konzept: Remote-Prozeduren (in Standard-SDL)

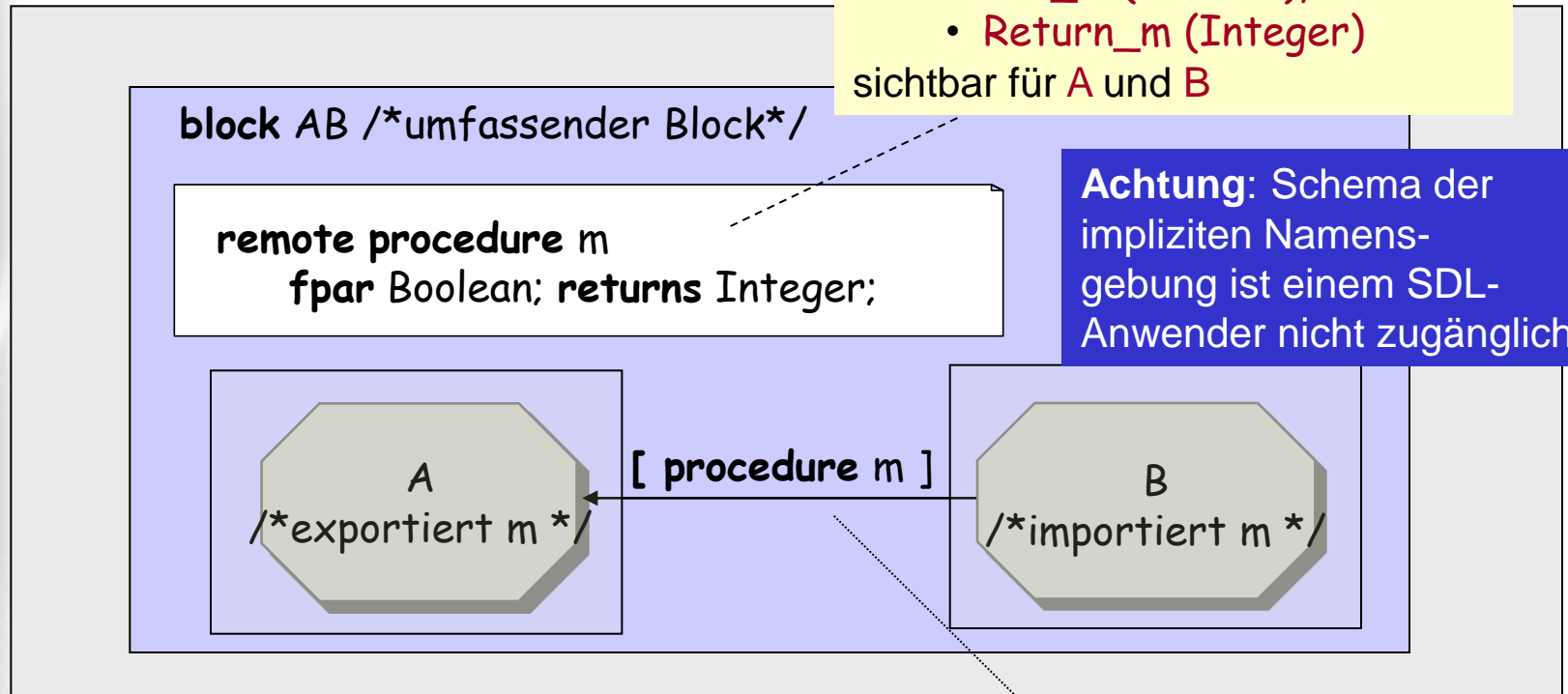
- **Entfernter Methoden-Ruf**
 - Ruf einer Methode eines anderen Objektes
- **Rollen** der beteiligten Prozess-Instanzen
 - **Exporteur** bietet Dienst als Prozedur an
 - **Importeur** verwendet die Prozedur als Dienst
- **Lokalisierung** der Partner
 - Prozess-Instanzen (Exporteur und Importeur-Instanzmengen) können verschiedenen Blöcken angehören
 - es muss aber immer einen umfassenden Block geben, der sowohl Importeur als auch Exporteur enthält (spätestens: System)
 - dieser umfassende Block hat die jeweilige Remote-Prozedur (Name und Signatur) zu deklarieren

Deklaration: Remote-Prozeduren

Transformation in implizite Signale:

- **Call_m** (Boolean),
- **Return_m** (Integer)

sichtbar für A und B



Namen von Remote-Prozeduren können in

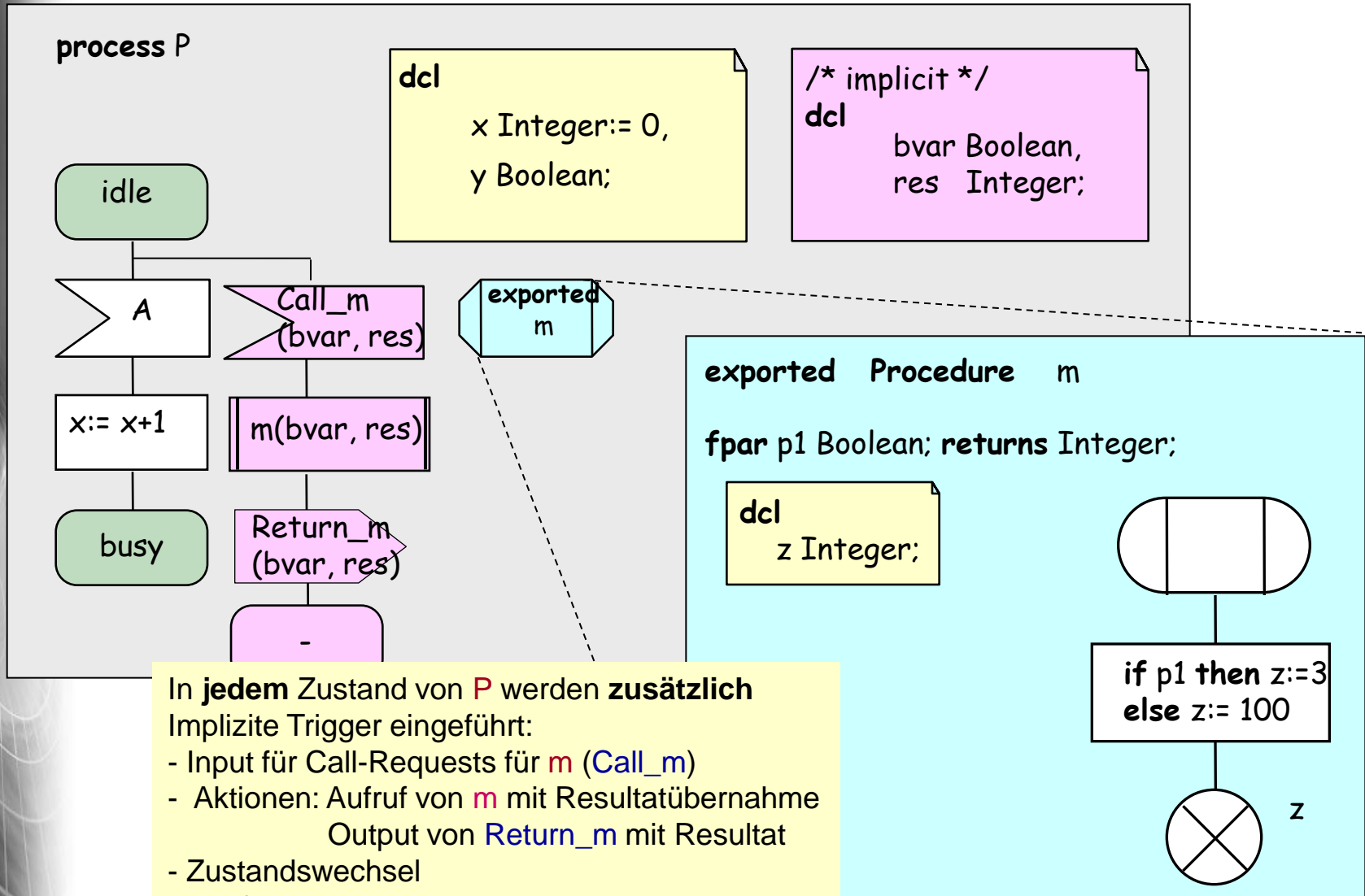
- Signallisten,
- Kanälen, Routen, Gates und
- Signalsets

erscheinen (wenn nicht, dann implizit)

Angabe aus Richtung des Rufers/Importeurs (obwohl bi-direktional)

Exporteur einer Methode

```
/* implicit */
signal
  Call_m (Boolean, Integer),
  Return_m (Boolean, Integer)
```



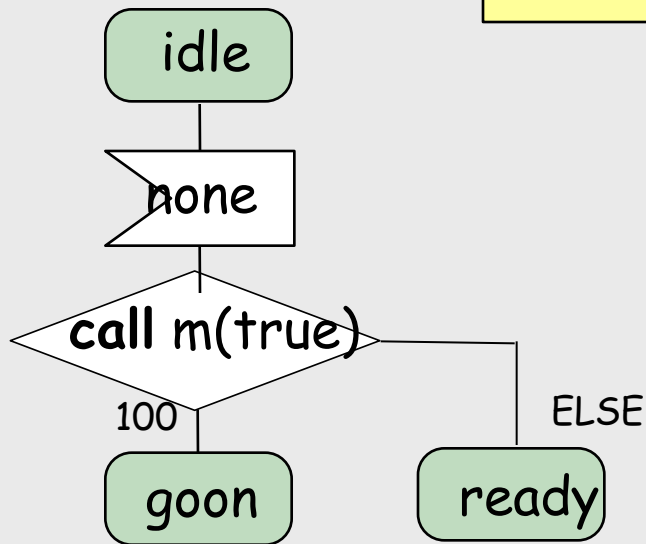
In **jedem** Zustand von **P** werden **zusätzlich** Implizite Trigger eingeführt:

- Input für Call-Requests für **m** (**Call_m**)
- Aktionen: Aufruf von **m** mit Resultatübernahme
- Output von **Return_m** mit Resultat
- Zustandswechsel

Rangfolge:
FCFS, wie normale Input-Signal-Trigger

Importeur einer Methode

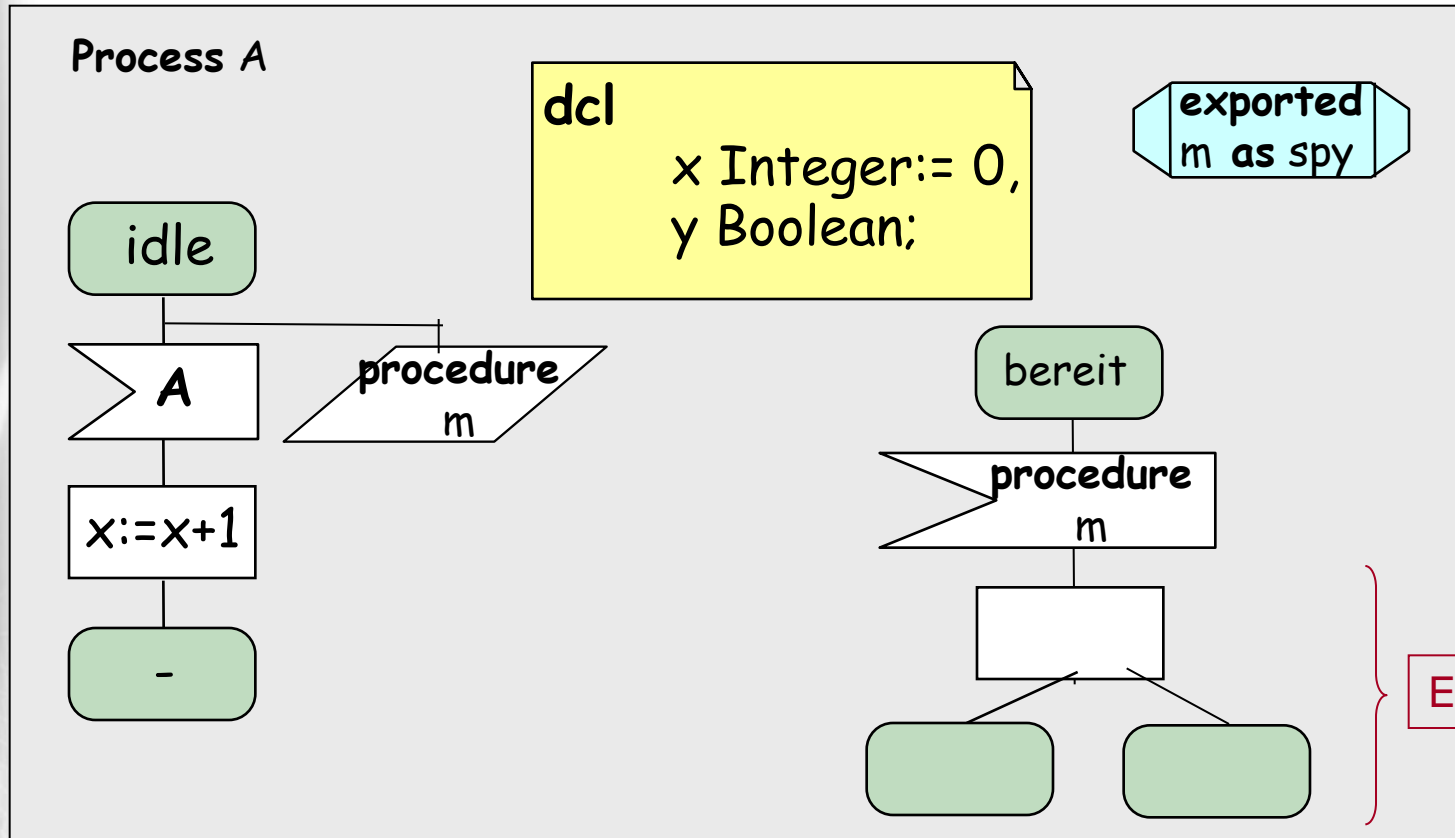
process B



imported procedure m
fpar p1 Boolean; returns Integer;

- B bleibt nach Aufruf von *m* solange blockiert, bis Return-Wert von *m* vorliegt und in der Decision ausgewertet werden kann
- *call m(...)* mit *to*, *via* kann benutzt werden, um aus Menge potentieller Exporteure genau einen Exporteur oder eine Teilmenge (Prozessinstanz-Menge) auszuwählen, aus der der Exporteur **nichtdeterministisch** bestimmt wird

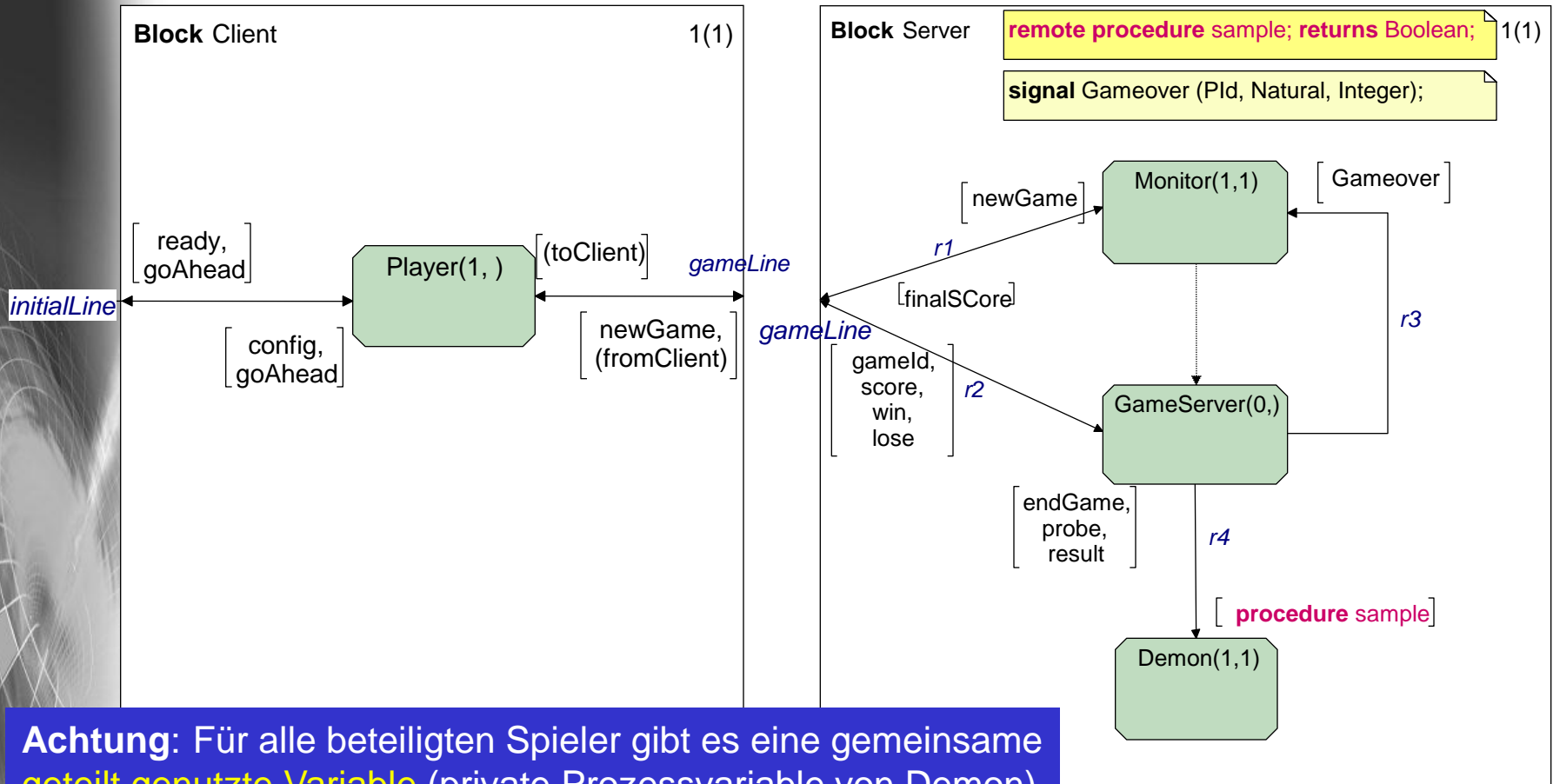
Verhaltensflexibilität des Exporteurs



Erweiterung der Standardsemantik:

- Prozedur kann unter anderem Namen exportiert werden (hier: **spy**)
- explizite Angabe von Zuständen, die Call-Requests akzeptieren (hier: **bereit** bei zusätzlicher Zustandsgraph-Erweiterung)
- explizite Angabe von Zuständen, die Call-Requests zurückstellen (hier: **idle**)

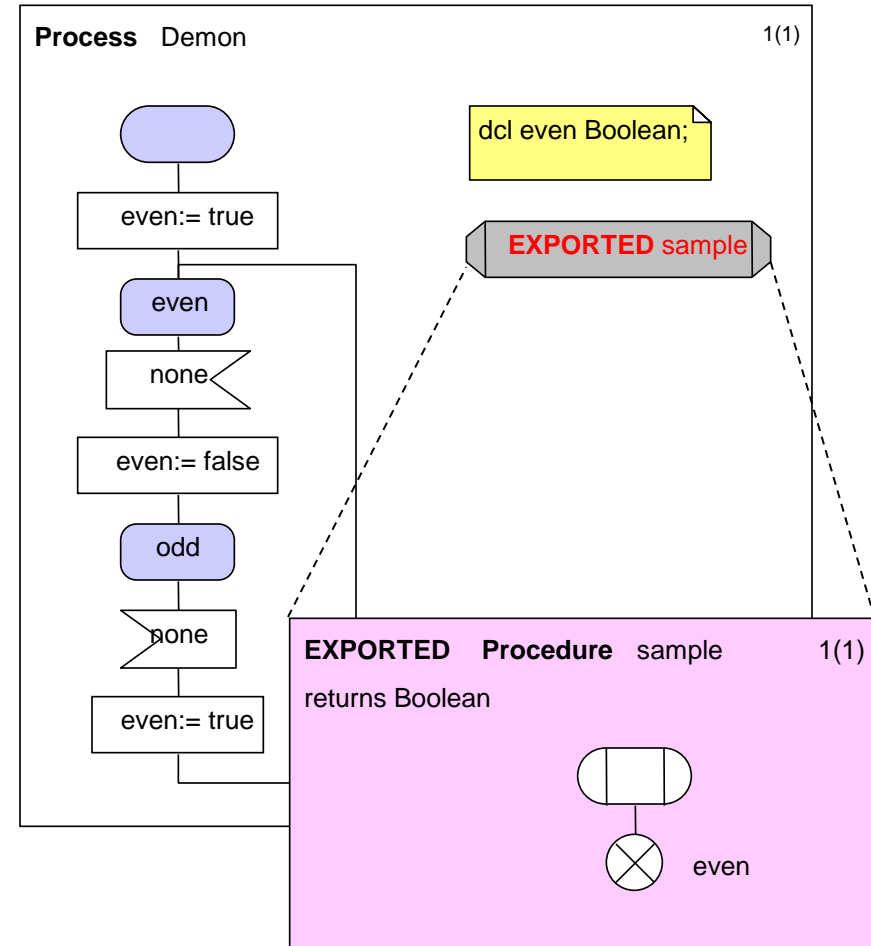
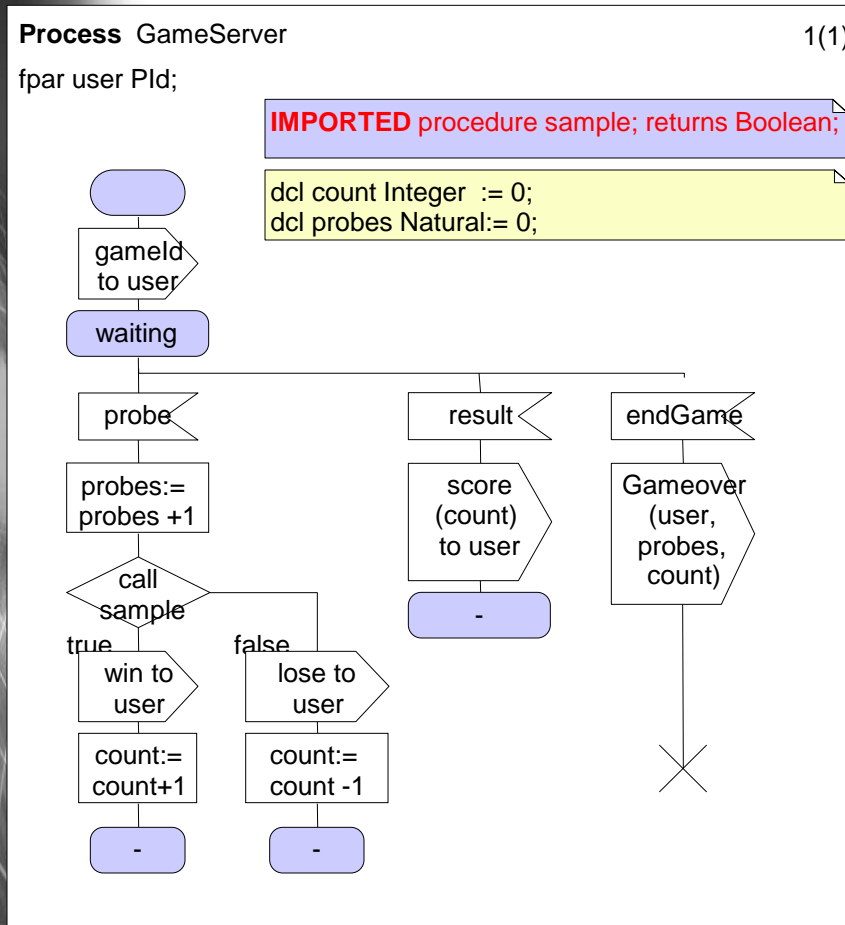
RPC-Beispiel: Demon-Game



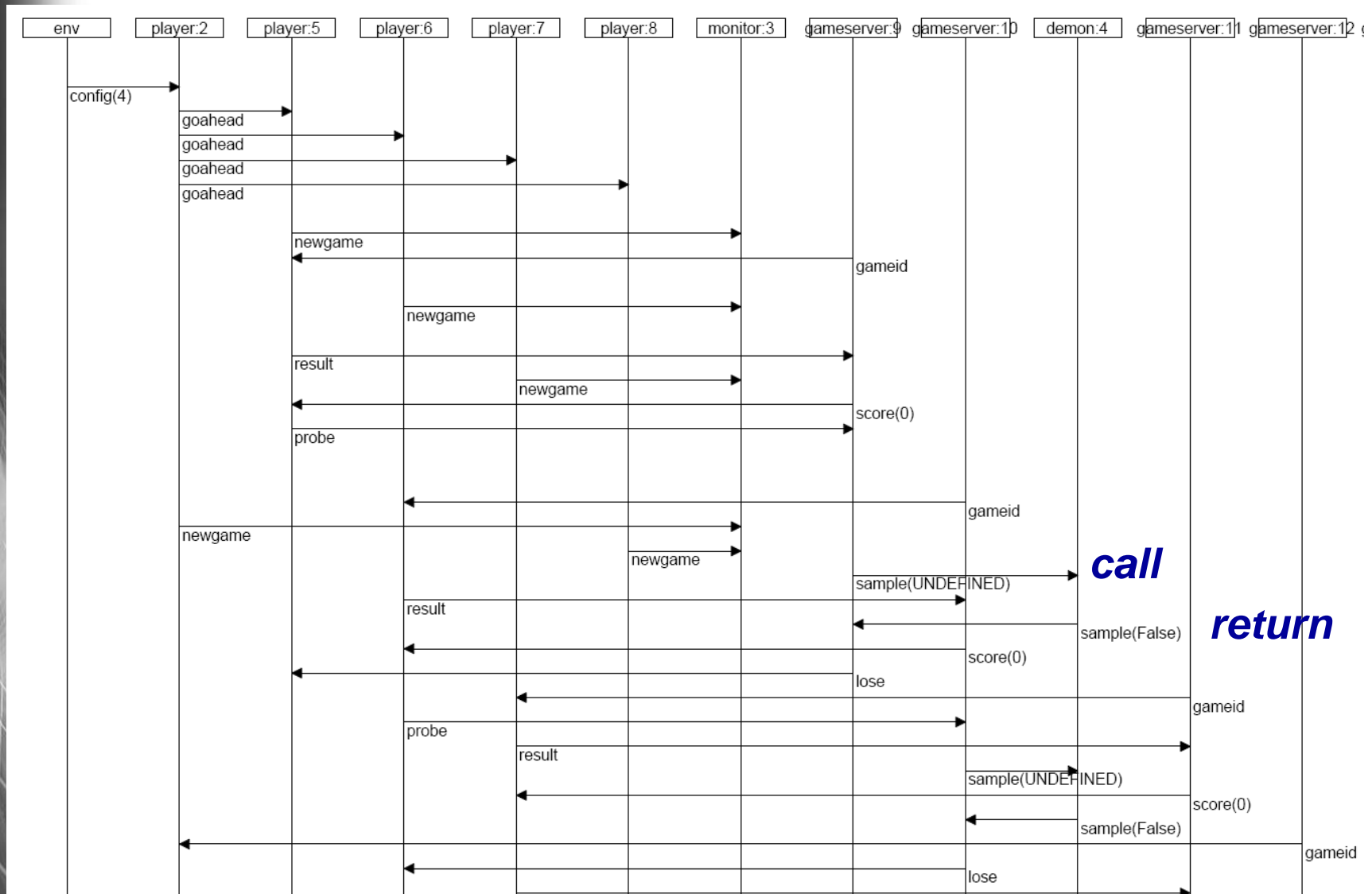
Achtung: Für alle beteiligten Spieler gibt es eine gemeinsame **geteilt genutzte Variable** (private Prozessvariable von Demon)
 Ein gleichzeitiger Zugriff per **Remote Prozedur** muss zwangssequentialisiert werden

Dreiklang: Remote – Imported - Exported

REMOTE procedure sample; returns Boolean;



Ablauf mit Ruf und Return einer Remote-Prozedur



RPC-Nachbildung in SDL/RT

