SDL - Specification and Description Language

Prof. Dr.-Ing. habil. Lutz Winkler, Hochschule Mittweida (FH) – University of Applied Sciences, FB Informationstechnik & Elektrotechnik
win@htwm.de http://telecom.htwm.de

- Ziel der Vorlesung: Kurze Einführung in die Sprache. Wesentliche Konzepte werden anhand eines Beispiels "SimplePABX" erläutert.

- Inhaltsübersicht :

Überblick

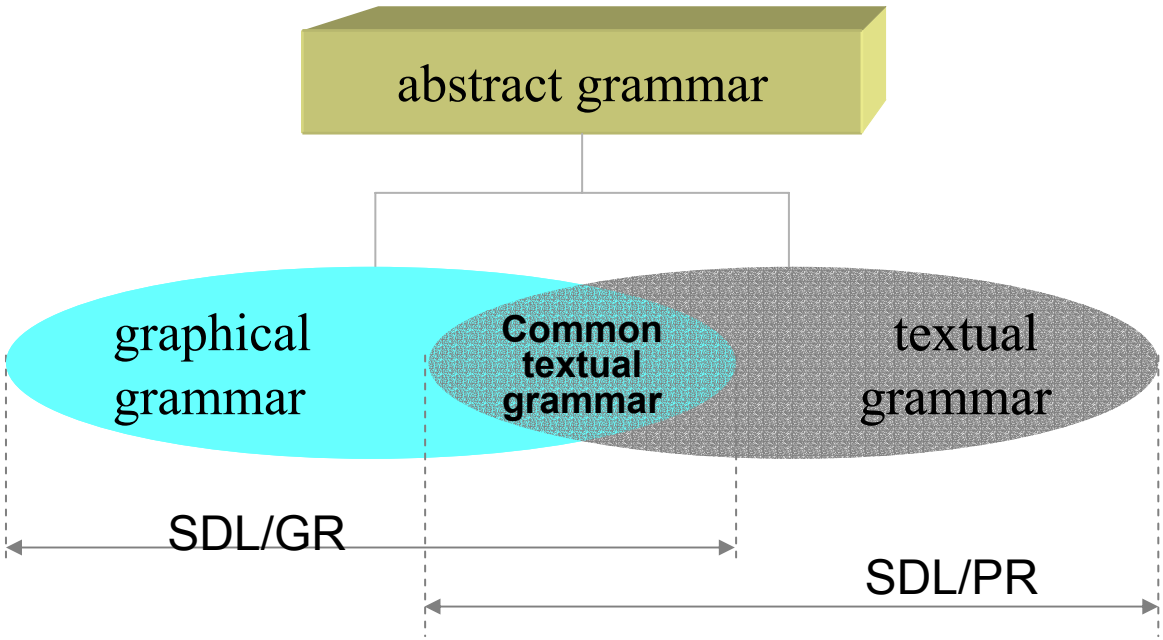
- **SDL** (Specification and Description Language) wurde entwickelt zur:
  - Specification oder WAS-soll-getan-werden-Beschreibung,
  - Description oder WIE-soll-es-getan-werden-Beschreibung.
- Gründe für die Entwicklung einer Spezialsprache waren, daß Funktional- und Protokollsoftware von Kommunikationssystemen eigene Spezifik haben:
  - große Parallelität gleicher und unterschiedlicher Prozesse,
  - Echtzeitanforderungen,
  - Viele Prozesse sind kooperierende Prozesse. Damit findet eine ausgeprägte Kommunikation zwischen ihnen statt.
  - Funktionen sind sehr komplex, müssen schrittweise verfeinert werden und von vielen gleichzeitig bearbeitbar sein
- **Ziele:**
  - Die Sprache soll einfach erlernbar und handhabbar sein, z.B. auch durch Nicht-Software-Fachleute.
  - Die Sprache soll rechnergestütze Softwareerstellung unterstützen.
- **Anwendung:** Beschreibung von Funktionen, Diensten, Protokollen usw., innerhalb und zwischen Netzknoten, zwischen Endeinrichtungen und Netzknoten aber auch für allgemeine Anwendungen.
  SDL wird in vielen ITU-TS-Empfehlungen angewendet.

Standardisierung

- 1968-1972: Erste Untersuchungen, Ziele für Entwicklung wurden formuliert
- 1972-1976: Grafische Darstellungsform SDL/GR (graphic representation) der Sprache wird im Ansatz entwickelt und im Orange Book als Z.101 bis Z.103 veröffentlicht.
- 1976-1980: Entwicklung der textualen Darstellungsform SDL/PR (phrase representation) und Veröffentlichung im Yellow Book als Z.101 bis Z.104
- 1980-1984:Vervollständigung und Harmonisierung beider Darstellungsformen und Herausgabe als Z.101 bis Z.104 im Red Book
- 1984-1988: Die Sprache wurde auf Basis einer exakten mathematischen Definition weiterentwickelt und im Blue Book unter Z.100 (SDL-88) veröffentlicht.
- 1992 wurde SDL-92 veröffentlicht. Sie beruht im wesentlichen auf SDL-88, ist aber nicht vollständig aufwärtskompatibel. Die Erweiterungen gehen in Richtung objektorientierter Spezifikation.
- In der SDL-Recommendation sind folgende Dokumente enthalten**:**
  - Z.100 SDL-Sprachbeschreibung
  - ANNEX A SDL-Schlagwortregister
  - ANNEX B Zusammenfassung der abstrakten Syntax
  - ANNEX C Zusammenfassung der konkreten grafischen und textualenSyntax
  - ANNEX D SDL-Nutzerhandbuch
  - ANNEX E Zustandsorientierte Darstellung und deren Bildelemente
  - ANNEX F Formale statische und dynamische Definition

Für SDL gibt es zwei syntaktische Ausprägungen

- **SDL/GR**: Graphical Representation,
- **SDL/PR**: Phrase Representation.
- GR und PR sind nur verschiedene Darstellungsformen der gemeinsamen Semantik.
- Die Sprachbeschreibung ist 3-geteilt:
  - abstract grammar,
  - concrete graphical grammar,
  - concrete textual grammer.



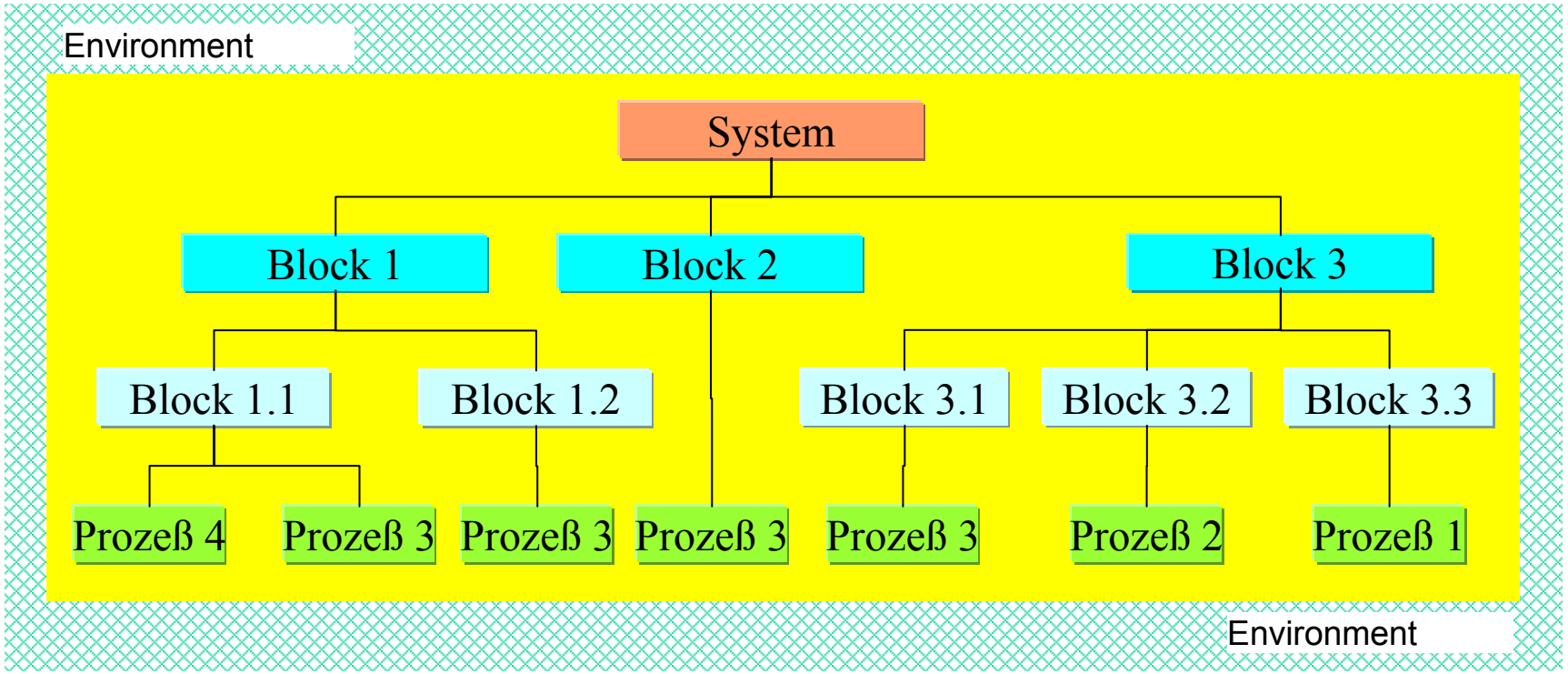
Merke: SDL/GR bildet nicht alle Sprachele-mente auf Symbole ab.

Datenvereinbarungenwerden z.B.in beiden Formen textual notiert.

Hauptmerkmale

- Ein SDL-System besteht aus vier Komponenten:
  - **Architektur,** beschrieben durch die Konzepte:
    - SYSTEM,
    - BLOCK,
    - PROCESS,
    - PROCEDURE.
  - **Kommunikation**, realisiert durch SIGNALs, die über CHANNELs bzw. SIGNALROUTEs ausgetauscht werden.
  - **Verhalten**, beschrieben durch PROCESSes.
  - **Operationsdaten**, beschrieben durch *Abstract data types (ADTs)*.
- Die Prozeßbeschreibung basiert auf einer Extended Finite State Machine (EFSM).
- Prozesse sind unabhängig voneinander, können parallel laufen und kommunizieren über Signale.
- SDL-Systeme basieren auf einem virtuellen SDL-Betriebssystem. Dieses realisiert das Erzeugen und Terminieren von Prozessen und die damit verbundenen Allokation von Speicherplatz. Das Betriebssystem realisiert die Kommunikation innerhalb eines Systems und nach außen. Das Betriebssystem realisiert Timerfunktionen für die Prozesse. Es ist für den Benutzer von SDL-Entwicklungstools nicht sichtbar.

Architektur: Übersicht



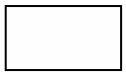
- Bei der **Systemspezifikation** werden die Blöcke, die Kommunikationskanäle zwischen Blöcken mit den Signalen und die Kommunikationskanäle zwischen Blöcken und der Umge-bung festgelegt. Letztere bilden das „Nutzerinterface“ oder das Interface zu anderen Systemen.
- Die **Blockspezifikation** umfaßt die Beschreibung der Blöcke. Deren Funktion kann in Subblöcke detailliert werden. Blöcke und Subblöcke werden durch Prozesse und Signalrouten mit ihren Signalen beschrieben.
- Die **Prozeßspezifikation** ist die Beschreibung des Verhaltens eines Systems. Diese erfolgt mit den Mitteln einer Extended Finite State Machine.

Architektur: SDL-Sprachelemente zur Architekturbeschreibung

ENV: Umgebung eines SDL-Systems

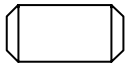
SYSTEM: Alle Anweisungen die zwischen SYSTEM <name> ... ENDSYSTEM stehen

BLOCK: Alle Anweisungen die zwischen BLOCK <namer> ... ENDBLOCK stehen. Soll ein Block später beschrieben werden, wird er referenziert durch BLOCK <name> REFERENCED.

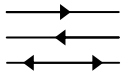


SUBSTRUCTURE: Dient der weiteren Partitionierung eines BLOCKs und umfaßt alle Anweisungen die zwischen SUBSTRUCTURE <name> ... ENDSUBSTRUCTURE stehen. Soll eine Substruktur später beschrieben werden, wird sie referenziert durch SUBSTRUCTURE <name> REFERENCED.

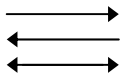
PROCESS: Beschreiben das Verhalten eines SYSTEM und umfaßt alle Anweisungen, die zwischen PROCESS <name> ... ENDPROCESS stehen. Er wird referenziert durch PROCESS <name >REFERENCED.



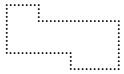
PROCEDURE: Beschreiben das Teilverhalten eines PROCESS und umfassen alle Anweisungen zwischen PROCEDURE <name> ... ENDPROCEDURE.



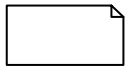
CHANNEL: Verbinden das ENV mit BLOCKs oder BLOCKs oder SUBSTRUCTUREs. CHANNELs können uni- und bidirektional sein. CHANNELs „transportieren“ SIGNALs. CHANNEL <name> FROM ... TO ... WITH ...



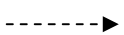
SIGNALROUTES: Verbinden das ENV mit PROCESSes oder PROCESSes. SIGNALROUTEs können uni- und bidirektional sein. SIGNALROUTEs „transportieren“ SIGNALs. SIGNALROUTE <name> FROM ... TO ... WITH .. SIGNALROUTEs ermöglichen die Dekomposition von CHANNELs.



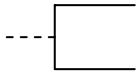
SELECT: Ist vergleichbar mit dem Konzept der bedingten Compilierung.
SELECT IF <boolean expression> ENDSELECT



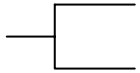
TEXT SYMBOL: enthält Kommentare, Deklarationen von DATATYPEs, SIGNALs, SIGNALLISTs, Variablen



Create line symbol: zeigt an, welcher PROCESS in einem anderen PROCESS die Erzeugung einer Instanz veranlassen kann.

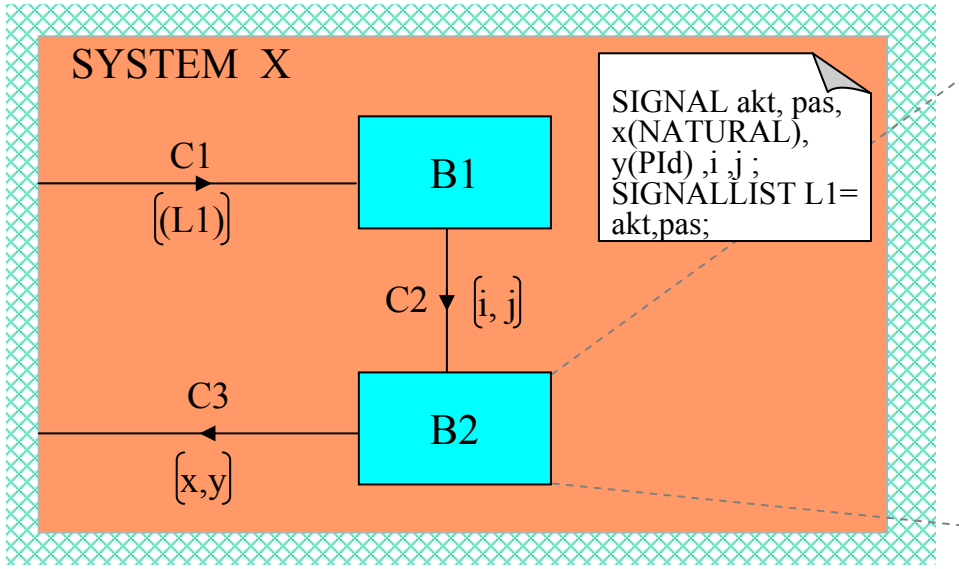


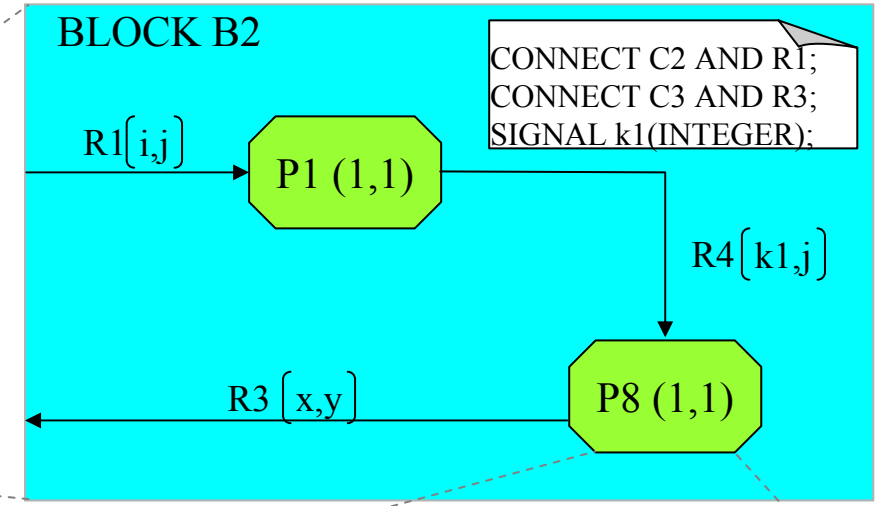
Comment area: die gestrichelte Linie zum COMMENT bedeutet, der Text ist ein Kommentar.

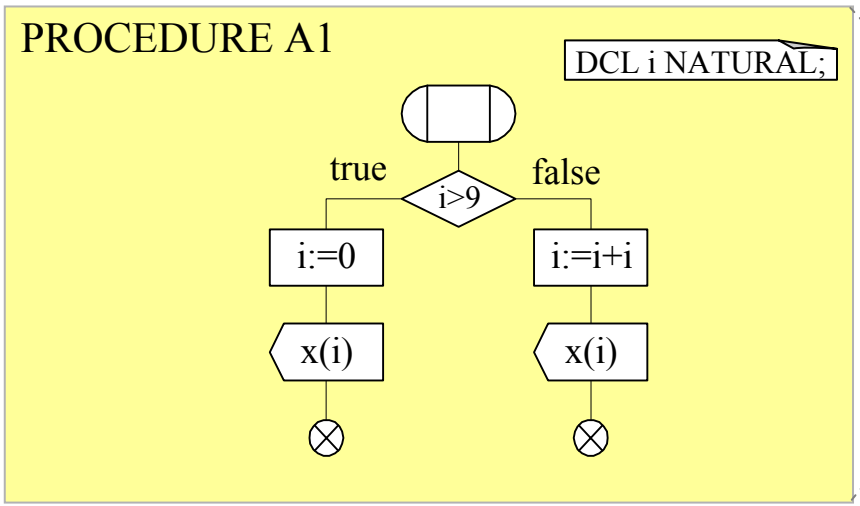


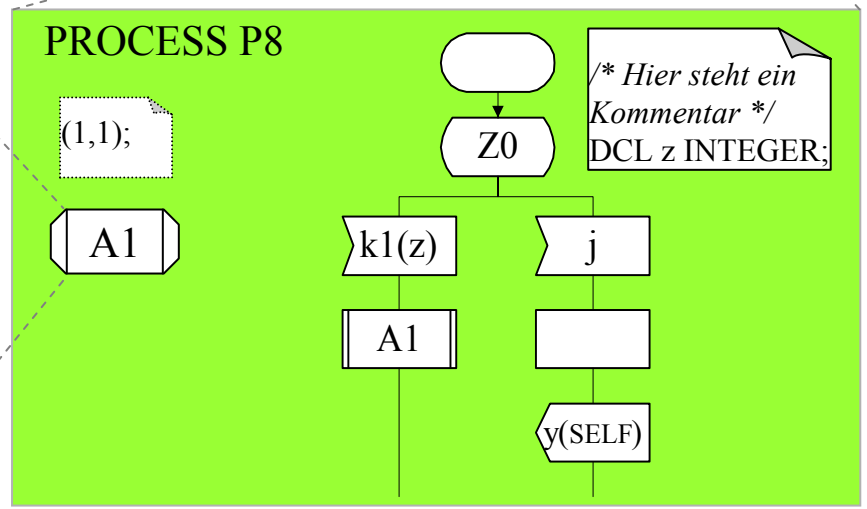
Text extension area: die durchgehende Linie zum COMMENT bedeutet, der Text ist ein Erweiterungstext.

Architektur: beispielhaftes System



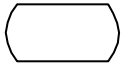






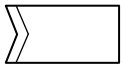
Verhalten: SDL-Sprachelemente zur Verhaltensbeschreibung

START: Eintrittspunkt eines Prozesses

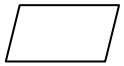


STATE: Prozeß in Erwartung eines INPUT

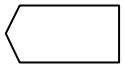
INPUT: Prozeß konsumiert SIGNAL und befindet sich im Zustandsübergang (transition)



PRIORITY INPUT: Prozeß konsumiert priorisiertes SIGNAL

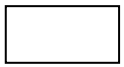


SAVE: Ein Signal wird noch nicht konsumiert bzw. verworfen, sondern im FIFO belassen

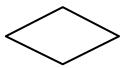


OUTPUT: Prozeß sendet an einen anderen Prozeß ein SIGNAL

PRIORITY OUTPUT: Prozeß sendet an einen anderen ein priorisiertes SIGNAL

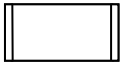


TASK: Operationsdaten werden bearbeitet oder TIMER gestartet bzw. gestoppt

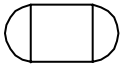


DECISION: Entscheidung, die zu einer Verzweigung der Transition führt.

JOIN: Flußlinie, die zu einem CONNECTOR oder einer Vereinigung führt.

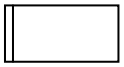


CALL: Eine PROCEDURE wird aufgerufen

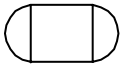


PROCEDURE: Eintrittspunkt einer PROCEDURE

RETURN: Ende einer PROCEDURE und Rücksprung zum aufrufenden Prozeß



MACRO: Aufruf einer MACRODEFINITION



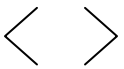
MACRODEFINITION:

ENDMACRO: Ende der MACRODEFINITION

CREATE: Dynamische PROCESS-Erzeugung

STOP: Beendigung eines PROCESS

ALTERNATIV: Beschreibung von Alternativen eines Prozeßverlaufs



PROVIDED: Freigabebedingung für eine Transition (BOOLEAN)

OUT- oder IN-CONNECTOR: Dienen zur Partitionierung einer Prozeßbeschreibung

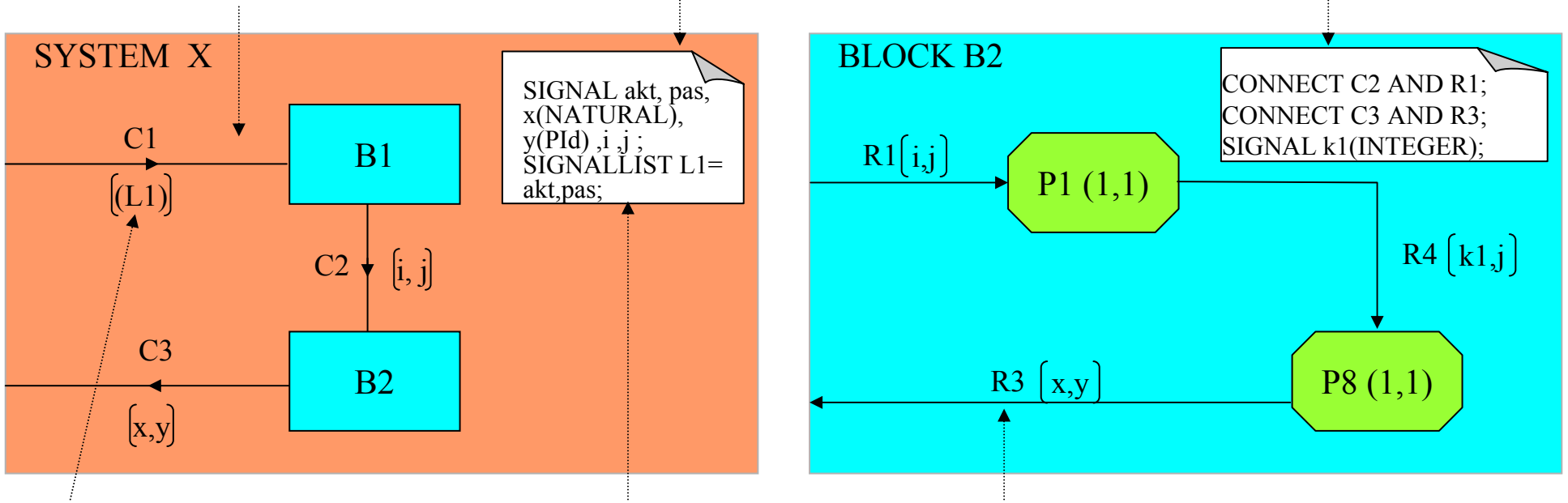
Kommunikation

- Kommunikation findet über SIGNALs statt. Signale können parametrisiert sein.
- Zwischen Blöcken werden Signale über CHANNELs, zwischen Prozessen über SIGNALROUTEs ausgetauscht.

***CHANNEL** C1 from ENV to BLOCK B1 überträgt SIGNALs der SIGNALLIST L1*

Das SIGNAL x trägt einen zusätzlichen Parameter vom Typ NATURAL, y einen vom Typ PId

Durch CONNECT-Deklarationen werden CHANNELS un SIGNALROUTEs logisch verbunden. (C2 AND R1, C3 AND R3)



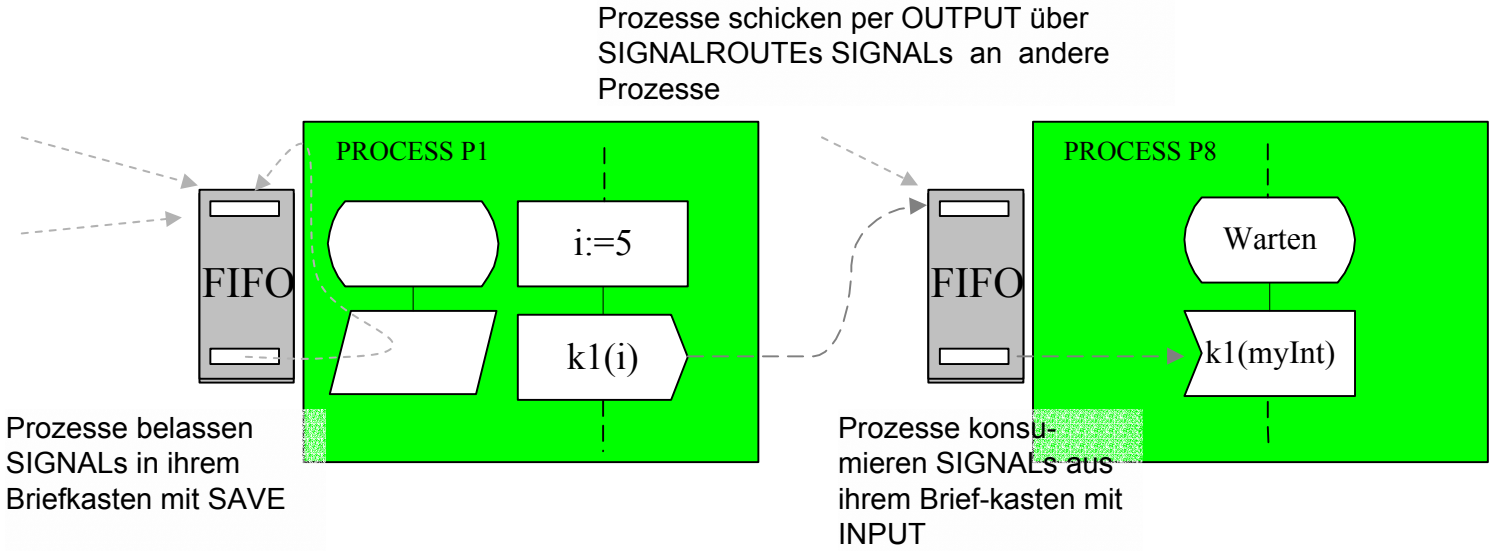
*SIGNALLISTs müssen durch **()** geklammert werden.!*

Aus den Signalen akt,pas wird eine SIGNALLIST L1 gebildet

***SIGNALROUTE** R3 from PROCESS P8 to ENV überträgt SIGNALs x und y.*

Kommunikation

- SIGNALs werden im Sendeprozeß mit OUTPUT <signal identifier> weggeschickt und in einem Empfangsprozeß mit INPUT <signal identifier> angenommen.
- Für jeden Prozeß existiert ein FIFO-organisierter Briefkasten. Alle Signale an einen Prozeß, weggeschickt mit OUTPUT, werden dort abgeliefert. Dies realisiert der unterlagerte SDL-Kern.

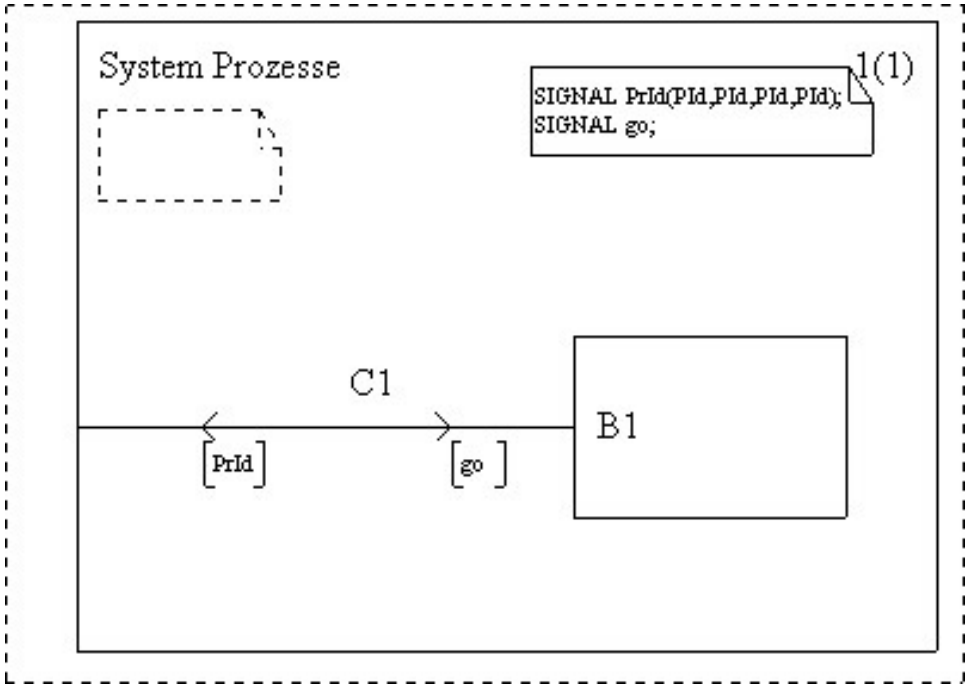


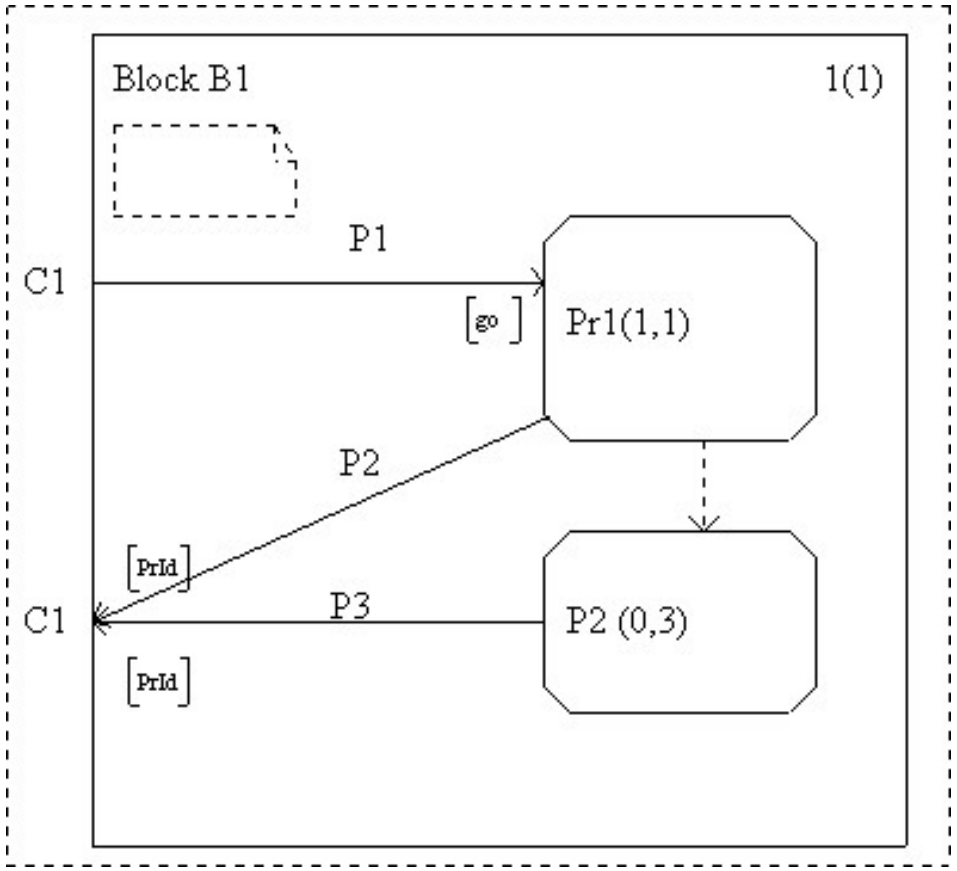
- Bekommt ein Prozeß Rechenzeit, entnimmt er das am längsten wartende SIGNAL aus dem Briefkasten und verarbeitet diesen INPUT.
- Die Kommunikation zwischen SDL-Prozessen erfolgt asynchron, d.h. der Sendeprozeß kann, wenn er Rechenzeit hat, Signale an andere Prozesse senden. Eine Quittung bekommt er aber nicht.

Lebensdauer und Identität von Prozessen

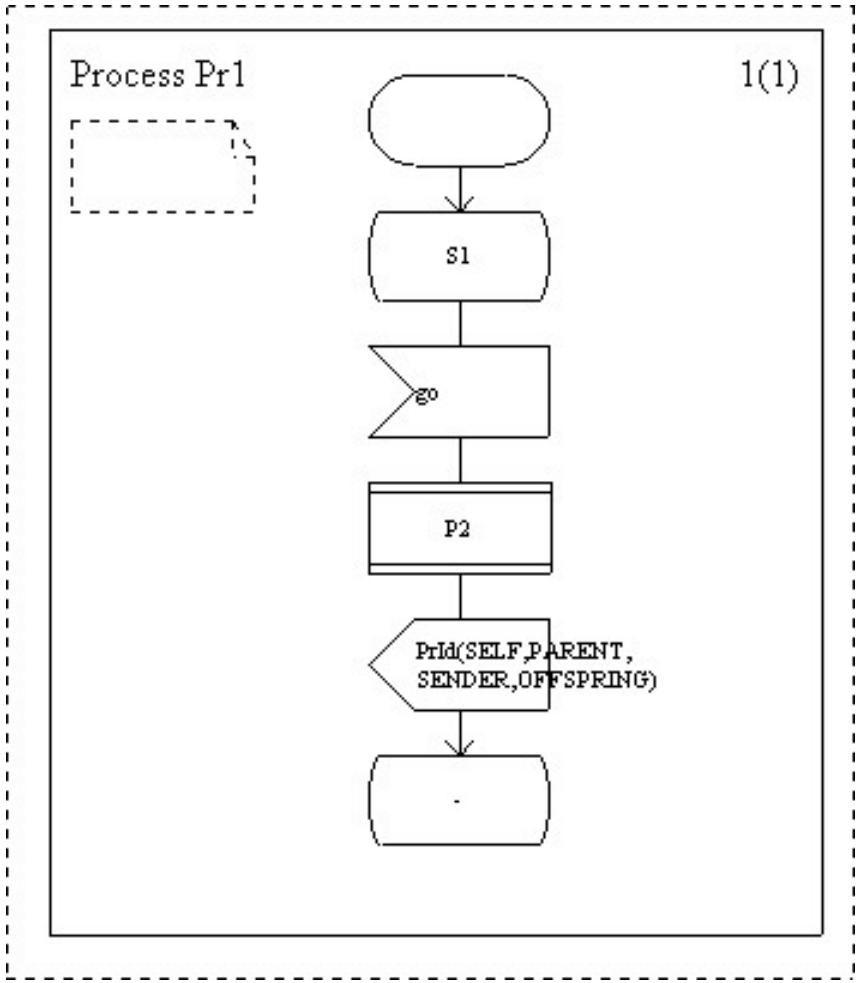
- Prozesse können kreiert werden:
  - zum Zeitpunkt des Systemstarts,
  - während der Laufzeit (CREATE).
- Prozesse, egal wie sie erzeugt wurden, können zur Laufzeit terminiert werden (STOP).
- In einem SYSTEM können mehrere verschiedene Prozesse existieren, aber auch mehrere Inkarnationen eines Prozesses. Damit Prozesse systemweit unverwechselbar sind, werden sie durch PId (Process Identifier) unterschieden. Für jeden Prozeß existieren automatisch 4 verschiedene PId:
  - SELF: Ist eine Referenz einer Instanz auf sich selber. Damit kann eine Instanz z.B. sich selber ein Signal schicken usw.
  - PARENT: Kennzeichnet den Prozeß, durch den eine Instanz erzeugt wurde. PARENT ist Null, wenn der Prozeß zum Systemstart erzeugt wurde.
  - SENDER: Ist eine Referenz auf den Prozeß, von dem das zuletzt konsumierte Signal kam.
  - OFFSPRING: In OFFSPRING steht der PId, der zuletzt kreierten Instanz (welchen Wert hat der PId die von mir zuletzt kreierte Instanz?)
- PIds sind ein gutes Mittel, um Signale gezielt zwischen inkarnierten Prozessen auszutauschen.

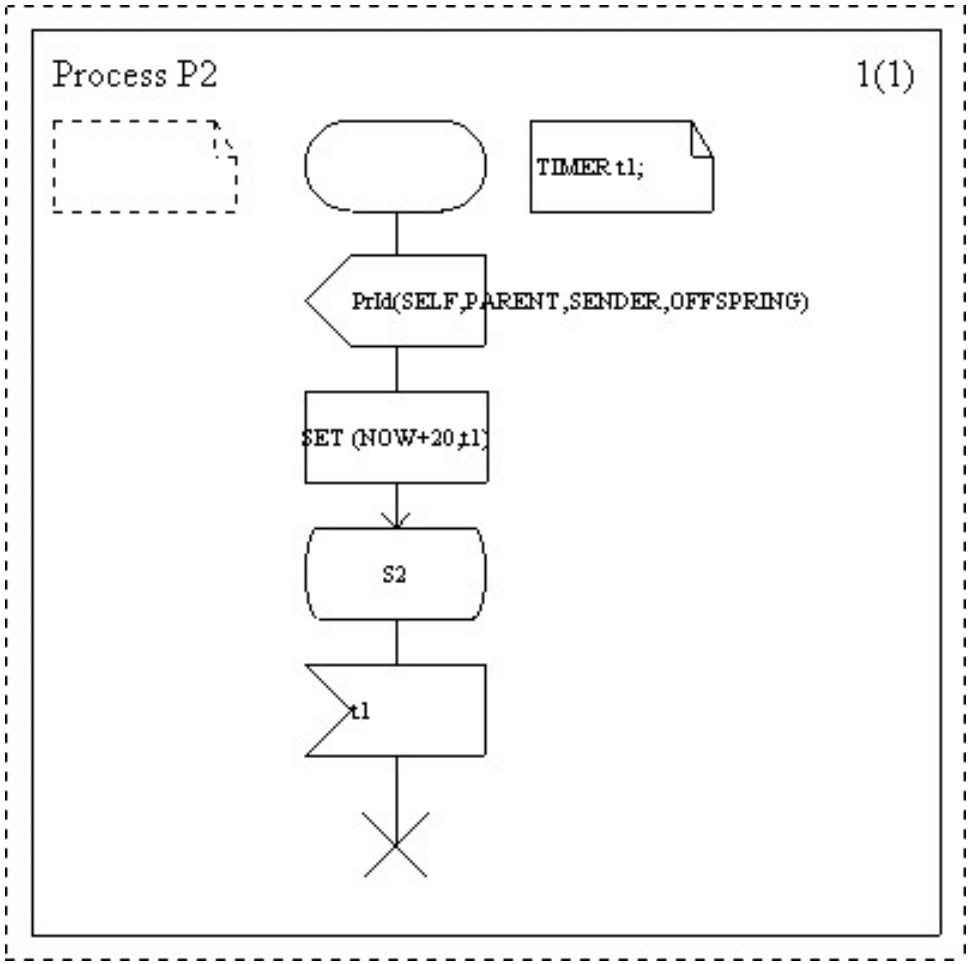
Beispiel: Identität von Prozessen



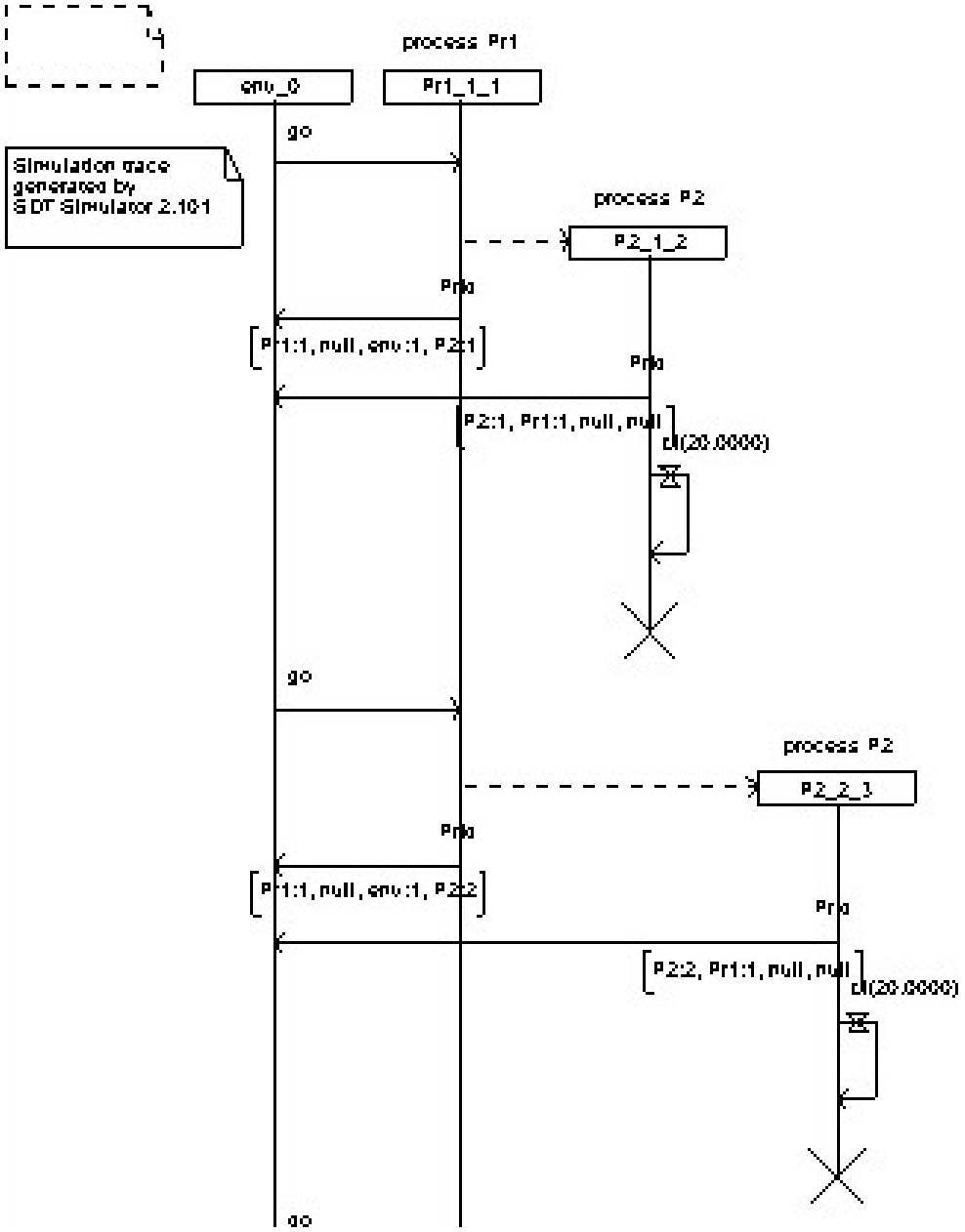


Beispiel: Identität von Prozessen





Beispiel: Identität von Prozessen



Abstract Data Types

- SDL ist eine Spezifikations- und Beschreibungssprache die einen rechnergestützten Softwareentwurf unterstützen soll. Die Zielsprache einer Implementation kann CHILL **CCITT High Level Language**, C oder eine andere Programmiersprache sein.
- In SDL wird deshalb eine eigene Syntax zur Definition von Daten verwendet. Alle Datentypen sind Abstract Data Types **ADS**. Ein ADS besteht aus:
  - **Literals**: Deklaration möglicher Werte (0,1,2, | true, false | up, down, left, right usw.)
  - **Operators**: Die Deklaration möglicher Operationen (plus, minus | NOT | >, <, <=, >= usw.)
  - **Axioms**: Der Definition der algebraischen Regeln.
- Wichtige Datentypen sind aber schon vordeklariert. Dies sind die SORTs:
  - Boolean:
  - Character: *alle IA5-Zeichen*
  - Charstring: *String aus IA5-Zeichen*
  - Integer: *negative, positive Ganzzahlen*
  - Natural: *positive Ganzzahlen*
  - Real:
  - PId: *Prozeßidentifier*
  - Duration: *Intervall zwischen Zeitpunkten (Dauer)*
  - Time: *Zeitpunkt (das NOW-Konzept liefert die momentane Systemzeit)*
- Des weiteren gibt es zur Erzeugung komplexerer Strukturen Generatoren für:
  - String,
  - Array.

Abstract Data Types: Deklaration der Sort Boolean aus Z.100


```
NEWTYPE Boolean
   LITERALS True, False;
   /*Werte die dieser Typ haben kann*/
   OPERATORS
        "NOT"  :Boolean              ->Boolean;
        "="    :Boolean, Boolean     ->Boolean;
        "/="   :Boolean, Boolean     ->Boolean;
        "AND"  :Boolean, Boolean     ->Boolean;
        "OR"   :Boolean, Boolean     ->Boolean;
        "XOR"  :Boolean, Boolean     ->Boolean;
        "=>"   :Boolean, Boolean     ->Boolean;
   AXIOMS
        "NOT"  (True)                ==(False);
        "NOT"  (False)               ==(True);

        "="    (True, True)          ==(True);
        "="    (True, False)         ==(False);
        "="    (False, True)         ==(False);
        "="    (False, False)        ==(True);

        "/="   (True, True)          ==(False);
        "/="   (True, False)         ==(True);
        "/="   (False, True)         ==(True);
        "/="   (False, False)        ==(False);

        "AND"  (True, True)          ==(True);
        "AND"  (True, False)         ==(False);
        "AND"  (False, True)         ==(False);
        "AND"  (False, False)        ==(False);

        "OR"   (True, True)          ==(True);
        "OR"   (True, False)         ==(True);
        "OR"   (False, True)         ==(True);
        "OR"   (False, False)        ==(False);

        "XOR"  (True, True)          ==(False);
        "XOR"  (True, False)         ==(True);
        "XOR"  (False, True)         ==(True);
        "XOR"  (False, False)        ==(False);

        "=>"   (True, True)          ==(True);
        "=>"   (True, False)         ==(False);
        "=>"   (False, True)         ==(True);
        "=>"   (False, False)        ==(True);
ENDNEWTYPE Boolean;
```

Abstract Data Types: Was man damit machen kann

•Was auf den ersten Blick wie eine Strafe aussieht, muß keine sein. Für eine Fahrstuhlsteuerung wurde in Z.100-Appendix I folgende Deklaration vorgestellt:

```
NEWTYPE Direction
        LITERALS up,down;
        OPERATORS
            change_dir: Direction  ->Direction;
        AXIOMS
            change_dir(up)     ==down;
            change_dir(down)==up;
    ENDNEWTYPE;
```

Man kann also eigene zugeschnittene Sorten erzeugen. In diesem Beispiel wurde die Sorte "Direction" deklariert. Eine Variable dieser Sorte kann die Werte "up" oder "down" zugewiesen bekommen. Auf Variable dieser Sorte kann man die Operation "change_dir" anwenden.

Beispiel:

```
DCL Richtung Direction;
...
    TASK Richtung:=up
    TASK change_dir(Richtung) /*Richtung=down*/
```

•Mittels SYNTYPE kann man eine vorhandene Sorte auf eigene Bedürfnisse eingrenzen.

```
SYNTYPE myInteger=INTEGER
CONSTANTS -3,0:5,8,10
ENDSYNTYPE
```

In diesem Fall wurde eine Sorte"myInteger", basierend auf der Sorte "Integer" gebildet. Variablen der Sorte "myInteger" können nur folgende Werte annehmen:

-3, 0, 1, 2, 3, 4, 5, 8, 10.

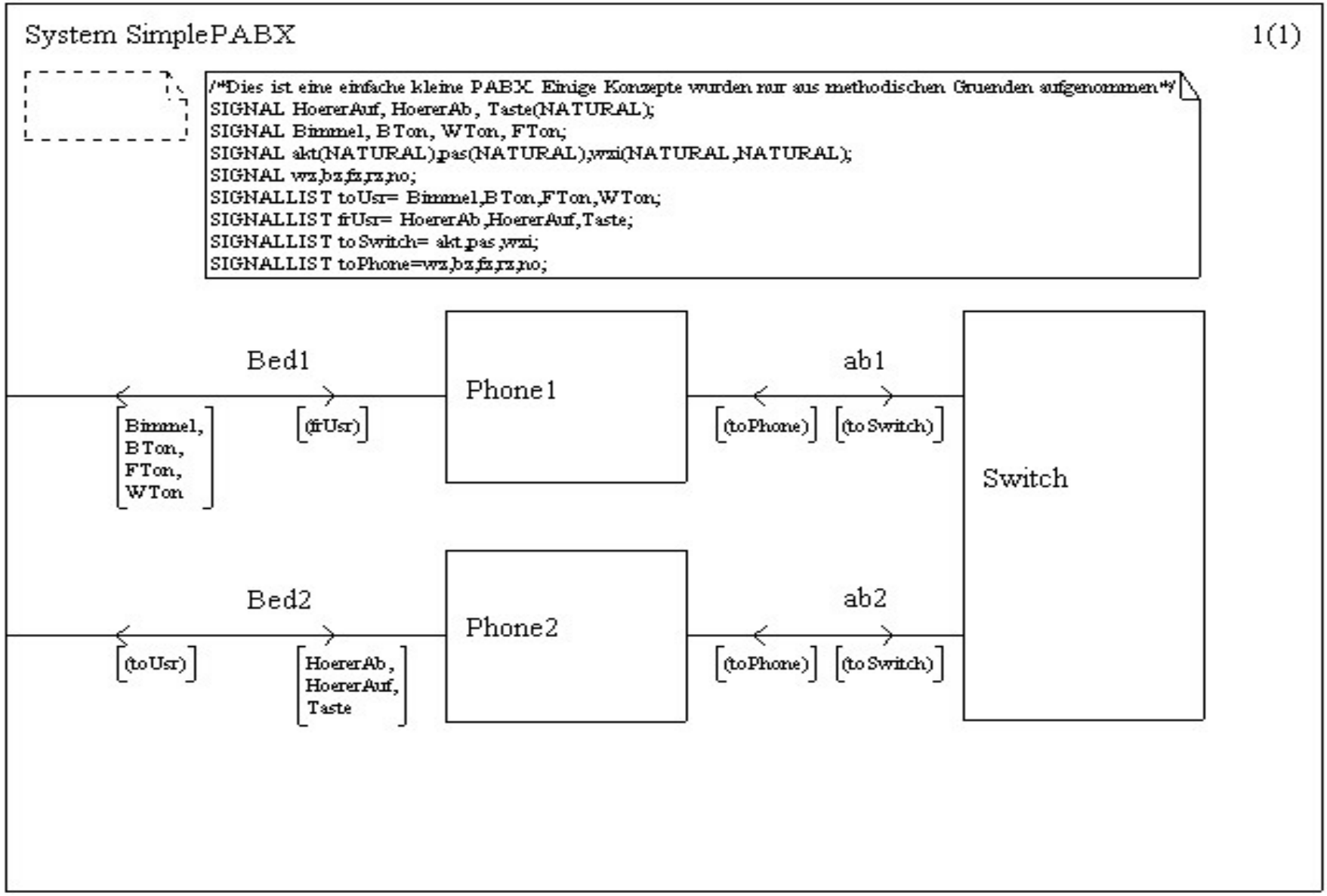
•Ein Beispiel für die Erzeugung von **STRUCT**s und **ARRAY**s ist der Blockdefinition "Switch" enthalten.

•Daten in SDL gehören grundsätzlich nur einem Prozeß. Nur er darf diese Daten manipulieren und sichtbar machen.

•Mit **REVALED** beim Besitzer und **VIEWED** beim Mitnutzer kann man Daten für andere Prozesse im gleichen Block sichtbar machen (siehe Process "Manager").

•Mit **EXPORTED** und **IMPORTED** kann man Daten über Blockgrenzen sichtbar machen.

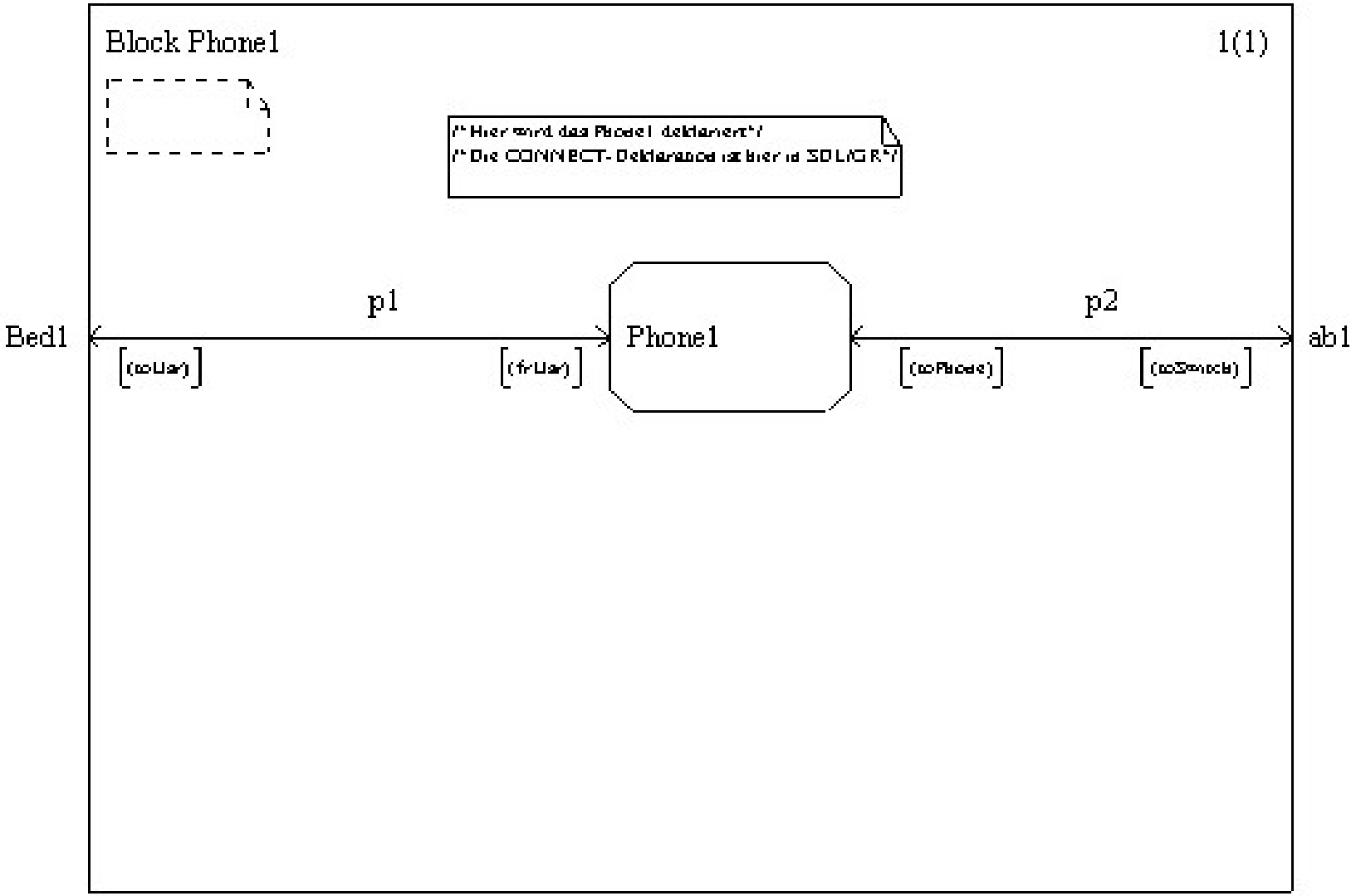
System SimplePABX in SDL/GR



System SimplePABX in SDL/PR


```
SYSTEM SimplePABX;
/*Dies ist eine einfache kleine PABX. Einige Konzepte wurden nur aus methodischen
   Gruenden aufgenommen*/
SIGNAL HoererAuf, HoererAb, Taste(NATURAL);
SIGNAL Bimmel, BTon, WTon, FTon;
SIGNAL akt(NATURAL),pas(NATURAL),wzi(NATURAL,NATURAL);
SIGNAL wz,bz,fz,rz,no;
SIGNALLIST toUsr= Bimmel,BTon,FTon,WTon;
SIGNALLIST frUsr= HoererAb,HoererAuf,Taste;
SIGNALLIST toSwitch= akt,pas,wzi;
SIGNALLIST toPhone=wz,bz,fz,rz,no;
CHANNEL Bed1 FROM Phone1 TO env WITH Bimmel,BTon,FTon,WTon;
            FROM env TO Phone1 WITH frUsr);ENDCHANNEL Bed1;
CHANNEL ab1  FROM Phone1 TO Switch WITH(toSwitch);
            FROM Switch TO Phone1 WITH(toPhone);ENDCHANNEL ab1;
CHANNEL Bed2 FROM Phone2 TO env WITH(toUsr);
            FROM env TO Phone2 WITH HoererAb,HoererAuf,Taste;ENDCHANNEL Bed2;
CHANNEL ab2  FROM Phone2 TO Switch with(toSwitch);
            FROM Switch TO Phone2 WITH(toPhone);ENDCHANNEL ab2;
BLOCK Phone1 REFERENCED;
BLOCK Switch REFERENCED;
BLOCK Phone2 REFERENCED;
ENDSYSTEM SimplePABX;
```

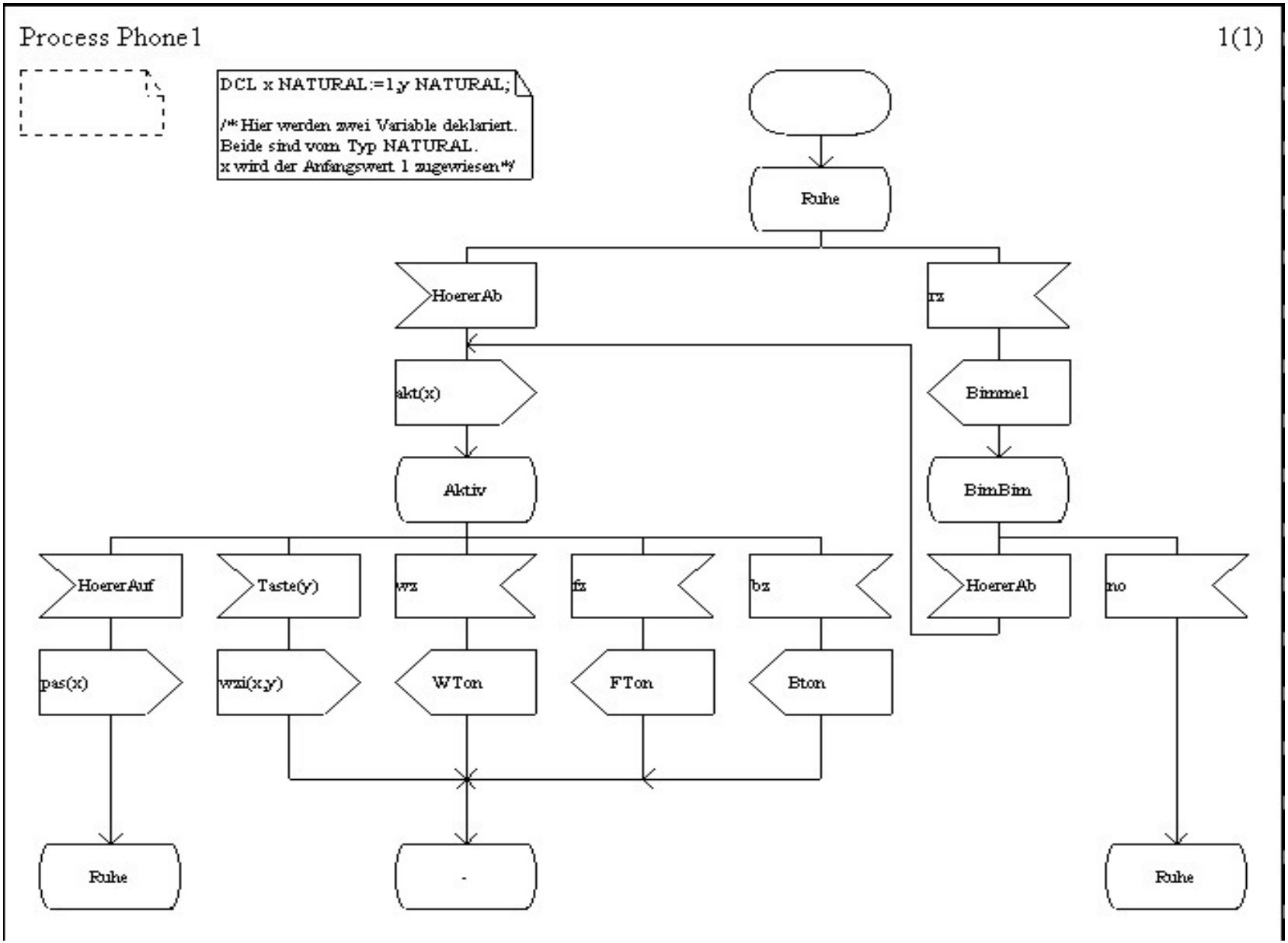
Block Phone1 in SDL/GR



Block Phone1 in SDL/GR


```
BLOCK Phone1;
   /*Hier wird das Phone1 deklariert*//*Die CONNECT-Deklaration ist hier in SDL/GR*/
SIGNALROUTE p1    FROM Phone1 TO env WITH (toUsr);
                  FROM env TO Phone1 WITH (frUsr);
SIGNALROUTE p2    FROM Phone1 TO env WITH (toSwitch);
                  FROM env TO Phone1 WITH (toPhone);
PROCESS Phone1 REFERENCED;
CONNECT Bed1 AND p1;
CONNECT ab1 AND p2;
ENDBLOCK Phone1;
```

Process Phone1 in SDL/GR



Process Phone1 in SDL/PR


```
PROCESS Phone1;
DCL x NATURAL:=1,y NATURAL;
/*  Hier werden zwei Variable deklariert.
    Beide sind vom Typ NATURAL. x wird der
    Anfangswert 1 zugewiesen*/
    START;
    NEXTSTATE Ruhe;
    STATE Ruhe;
          INPUT HoererAb;
    grst56:
                OUTPUT akt(x);
          NEXTSTATE Aktiv;
          INPUT rz;
                OUTPUT Bimmel;
          NEXTSTATE BimBim;
    ENDSTATE;
    STATE Aktiv;
          INPUT HoererAuf;
                OUTPUT pas(x);
          NEXTSTATE Ruhe;
          INPUT Taste(y);
                OUTPUT wzi(x,y);
    grst57:
          NEXTSTATE -;
          INPUT wz;
                OUTPUT WTon;
                JOIN grst57;
          INPUT fz;
                OUTPUT FTon;
          JOIN grst57;
          INPUT bz;
                OUTPUT Bton;
          JOIN grst57;
    ENDSTATE;
    STATE BimBim;
          INPUT HoererAb;
          JOIN grst56;
          INPUT no;
          NEXTSTATE Ruhe;
    ENDSTATE;
ENDPROCESS Phone1;
```

Literatur

/Z.100/ **CCITT Specification and Description Language (SDL)**
ITU-T Recommendation Z.100, 03/93

/Z.100_AppI&II/ **SDL Methodology Guidelines,** 03/93
SDL Bibliography, 03/93

/Z100_ANNEX_D/ **Anex D to Recommendation Z.100: SDL User Guidelines,** 1988