10 Formale Methoden zur Spezifikation von Protokollen

10.1 Einführung in formale Beschreibungstechniken

Ziele:

- Eindeutige, unmißverständliche Spezifikation von Kommunikationsdiensten und -protokollen
- Formale Basis ermöglicht Verifikation der Korrektheit
- Formale Basis ermöglicht maschinelle Verarbeitung, z.B. automatische Code-Erzeugung

Formale Beschreibungstechniken haben

- eine formale Syntax:
  Eine Menge von Regeln, die für die Sprache vorschreiben, welche Symbole benutzt und welche Ausdrücke konstruiert werden können.
- eine formale Semantik:
  Regeln für die eindeutige Interpretation der Sprachausdrücke auf der Basis eines mathematisch definierten semantischen Modells.

Kriterien für Formale Beschreibungstechniken (1)

- **Formale Definition**: verweist auf die Existenz einer formalen Syntax und einer formalen Semantik. Das unterliegende formale Modell einer FDT sollte sowohl die Entwicklung von analytischen Theorien zur Validation, Implementierung und zum Testen erleichtern als auch die Entwicklung von Werkzeugen vereinfachen.
- **Ausdruckskraft**: verlangt Sprachkonstrukte, die eine zweckmäßige Beschreibung eines weiten Bereichs von Eigenschaften und Funktionalitäten aus der Welt der Kommunikationssysteme erlauben. Die "Modellkraft" einer Technik ist wichtig, aber auch ihre praktische Anwendbarkeit zum Spezifizieren von komplexen Systemen.
- **Abstraktion**: beschreibt, in welchem Ausmaß implementationsunabhängige Spezifikationen vom unterliegenden Modell und von den Sprachkonstrukten unterstützt werden. Es sollten nur funktionale Anforderungen durch Spezifikationen definiert werden. Es ist zu vermeiden, daß formale Beschreibungen irrelevante Details enthalten, z.B. Implementierungsdetails

Kriterien für Formale Beschreibungstechniken (2)

- **Zusammensetzbarkeit**: erfordert Sprachfähigkeiten zum Zusammenfügen individueller Teile einer Spezifikation auf verschiedenen Wegen zu einem ganzen System. Komplexe Systeme sollten in kleinere Einheiten zerlegbar sein, die separat beschrieben werden können. Es muß aber immer noch möglich sein, das Gesamtverhalten des Systems aus dem Verhalten der Komponenten folgern zu können.
- **Struktur**: Spezifizierer sollen mit Mechanismen ausgerüstet sein, um formale Beschreibungen in einer angemessenen Art und Weise zu organisieren. Es sollte möglich sein, verwandte Aspekte zu aggregieren, nicht verwandte zu separieren, Phasen unterschiedlichen Verhaltens zu unterscheiden und so die Lesbarkeit durch die Definition einer geeigneten Struktur zu erhöhen.

Standardisierte FDTs

- ESTELLE (ISO)
- SDL (ITU-T)
- LOTOS (ISO)

Literatur

D. Hogrefe: Estelle, LOTOS und SDL. Springer Verlag, 1989

Computer Networks and ISDN Systems, Special Issue on Protocol Specification and Testing, Vol.14, No. 1, 1987

10.2 Endliche Automaten

Definition

Ein endlicher Automat (Mealy-Automat, finite state machine) ist ein Quintupel

$$< Z, E, A, \delta, \lambda >$$

mit

Z = Menge von Zuständen (endlich)

E = Menge von Eingabezeichen = Eingabealphabet

A = Menge von Ausgabezeichen = Ausgabealphabet

δ : Z x E -> Z = Zustandsübergangsfunktion

λ : Z x E -> A = Ausgabefunktion

Manchmal fügt man auch noch einen Startzustand

$$z_0 \in Z$$

hinzu.

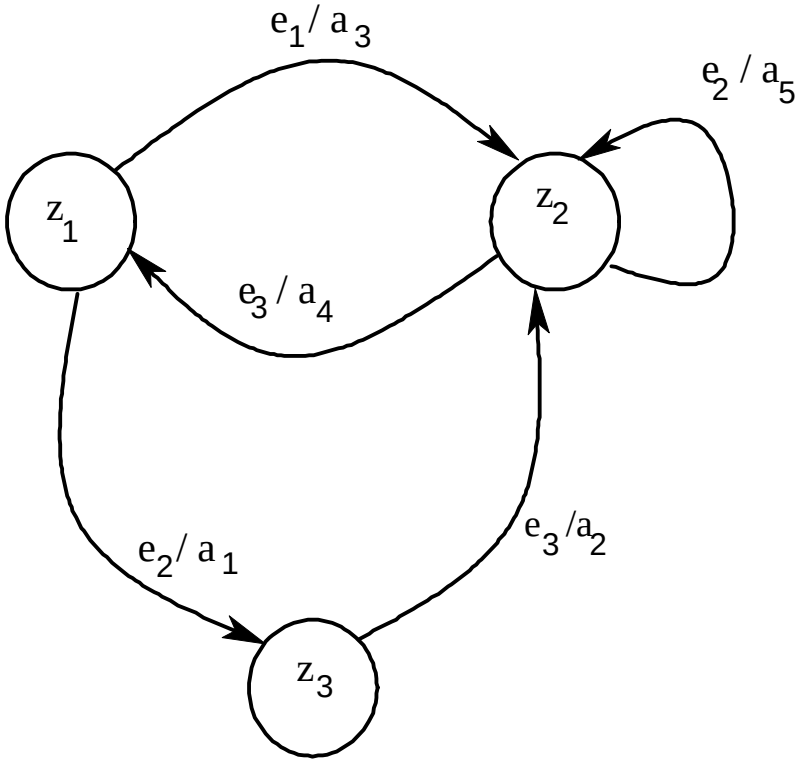
Literatur:

W. Brauer: Automatentheorie. Teubner-Verlag, Stuttgart, 1984

Darstellung als Diagramm

Ein endlicher Automat läßt sich anschaulich als Zustands-/Übergangs-diagramm darstellen. Ein Zustands-/Übergangsdiagramm ist ein Graph mit gerichteten Kanten. Die Zustände sind die Knoten, die Übergänge die Kanten. Jedes Element aus δ ergibt eine Kante: sie wird mit der auslösenden Eingabe $e_i \in E$ beschriftet. In gleicher Weise werden die Elemente aus λ an die Kanten geschrieben.

Beispiel:



Darstellung als Tabelle

Ein endlicher Automat läßt sich ebenso in Tabellenform darstellen:

E\Z	z_1	z_2	...	z_n
e_1	$\delta(e_1, z_1)$ $\lambda(e_1, z_1)$	$\delta(e_1, z_2)$ $\lambda(e_1, z_2)$	...	$\delta(e_1, z_n)$ $\lambda(e_1, z_n)$
e_2	$\delta(e_2, z_1)$ $\lambda(e_2, z_1)$	$\delta(e_2, z_2)$ $\lambda(e_2, z_2)$	...	$\delta(e_2, z_n)$ $\lambda(e_2, z_n)$
e_m	$\delta(e_m, z_1)$ $\lambda(e_m, z_1)$	$\delta(e_m, z_2)$ $\lambda(e_m, z_2)$	...	$\delta(e_m, z_n)$ $\lambda(e_m, z_n)$

Homomorphismus von Automaten M und M'

Abbildungen $\zeta : Z \rightarrow Z'$

$\eta : E \rightarrow E'$

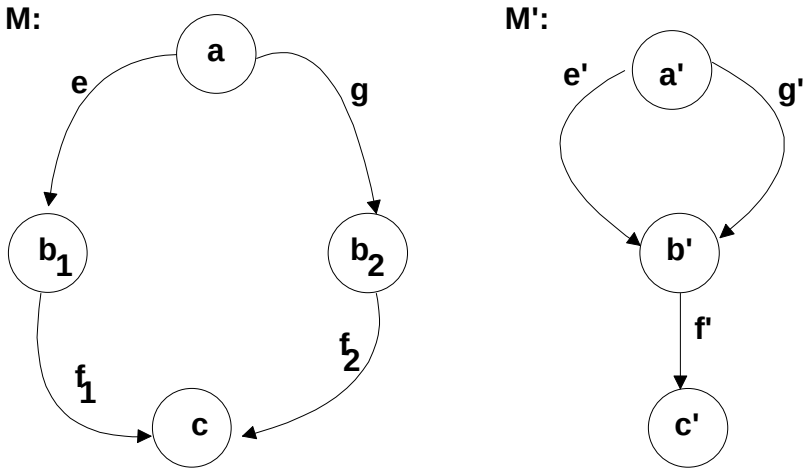
$\alpha : A \rightarrow A'$

wobei ζ, η, α surjektiv und

$\zeta(\delta(z,e)) = \delta'(\zeta(z), \eta(e))$ und

$\alpha(\lambda(z,e)) = \lambda'(\zeta(z), \eta(e))$

Beispiel:



$\zeta : a \rightarrow a'$

$b_1, b_2 \rightarrow b'$

$c \rightarrow c'$

$\eta : e \rightarrow e'$

$g \rightarrow g'$

$f_1, f_2 \rightarrow f'$

$\zeta(\delta(z,e))$: $a, e \rightarrow b_1 \rightarrow b'$

$a, g \rightarrow b_2 \rightarrow b'$

$b_1, f_1 \rightarrow c \rightarrow c'$

$b_2, f_2 \rightarrow c \rightarrow c'$

$\delta'(\zeta(z), \eta(e))$: $(a \rightarrow a', e \rightarrow e') \rightarrow b'$

$(a \rightarrow a', g \rightarrow g') \rightarrow b'$

$(b_1 \rightarrow b', f_1 \rightarrow f') \rightarrow c'$

$(b_2 \rightarrow b', f_2 \rightarrow f') \rightarrow c'$

Isomorphismus von Automaten M und M'

$\zeta : Z \rightarrow Z', \eta : E \rightarrow E', \alpha : A \rightarrow A'$ Homomorphismus von M in M'

ζ, η, α bijektiv (eineindeutig)

Erläuterung:

M und M' besitzen dann die gleiche Struktur, lediglich die Ein- und Ausgabezeichen und die Zustandsbenennungen können verschieden sein.

Äquivalenz von Automaten M und M' (1)

Definition

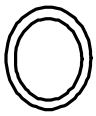
Zwei **Automaten** M und M' sind **äquivalent** genau dann, wenn

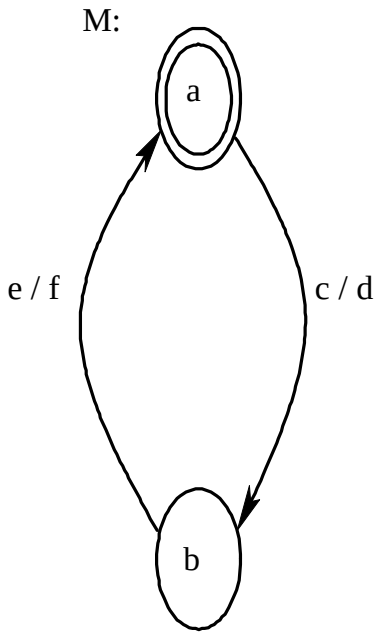
E = E'

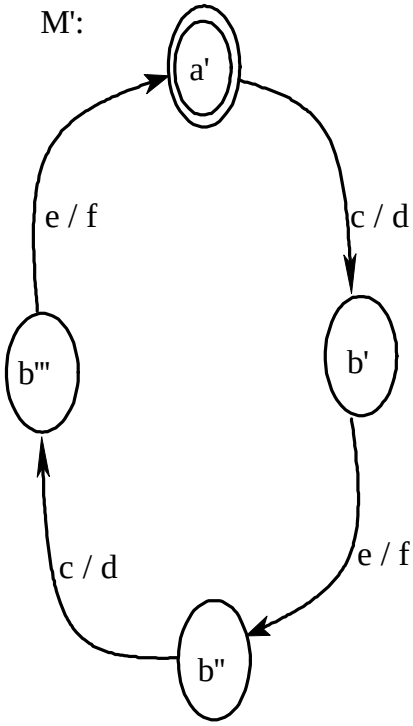
A = A'

und "beide Automaten erzeugen bei gleichen Eingabefolgen gleiche Ausgabefolgen".

Beispiel:







Äquivalenz von Automaten M und M' (2)

Zwei **Zustände** z, z' eines Automaten M heißen **äquivalent**, wenn gilt:

mit $M_1 = \triangleleft E, A, Z; \delta, \lambda; z \triangleright$ *und* $M_2 = \triangleleft E, A, Z; \delta, \lambda; z' \triangleright$ ist M_1 äquivalent M_2.

Also: z oder z' kann als Startzustand gewählt werden, ohne daß sich das E/A-Verhalten des Automaten ändert.

Am Beispiel oben: b''' ist **äquivalent** zu b'.

Weitere Eigenschaften von Endlichen Automaten

Zusammenhang

Ein Automat heißt **zusammenhängend**, wenn für jeden Zustand Z_i eine Eingabezeichenfolge existiert derart, daß der Zustand vom Startzustand aus erreicht wird.

Minimalität

Ein zusammenhängender Automat heißt **minimal**, wenn er keine äquivalenten Zustände enthält.

In einer Menge äquivalenter Automaten ist der minimale Automat derjenige, der die geringste Anzahl von Zuständen besitzt.

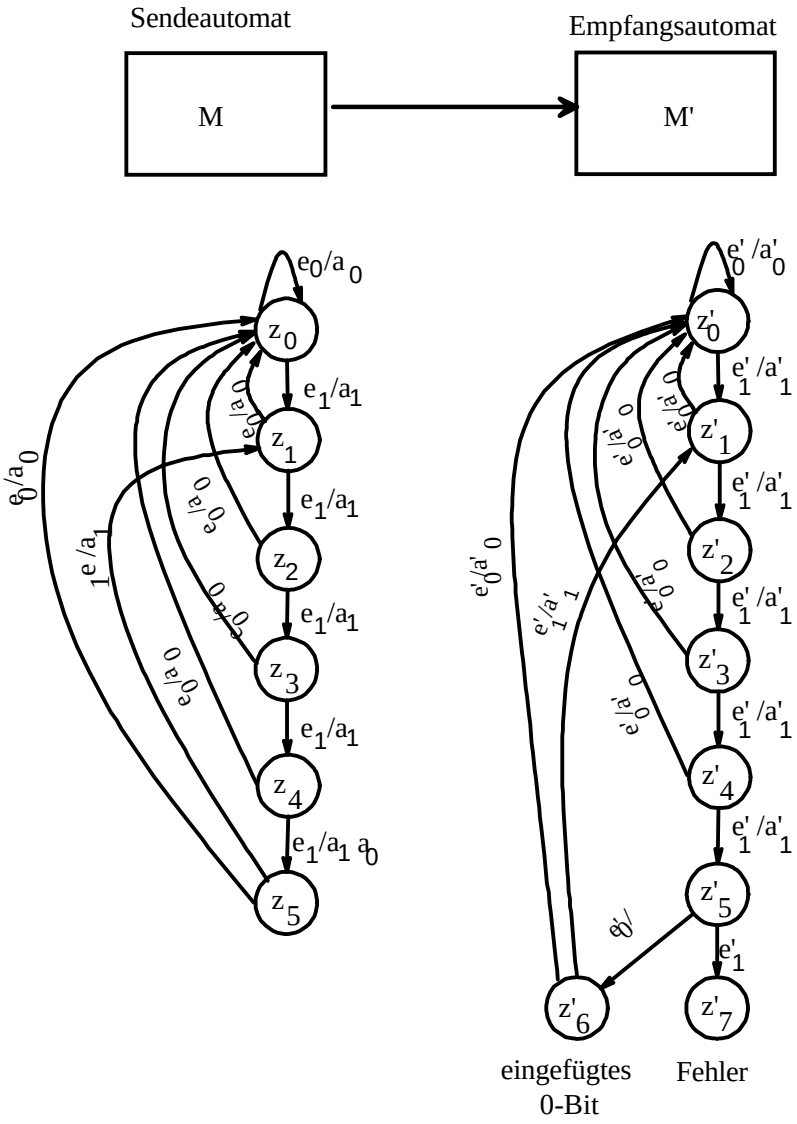
Äquivalente minimale Automaten sind isomorph.

Anwendung von Endlichen Automaten auf Kommunikationsprotokolle

Man betrachtet eine Instanz als "Schwarzen Kasten" und beschreibt das Verhalten als Zustands-/Übergangsverhalten. Die Elemente des Eingabe- und Ausgabealphabets bezeichnet man als **Ereignisse**. Ereignisse können an der Dienstschnittstelle und an der Protokollschnittstelle der Kommunikationsinstanz auftreten.

Ein verteiltes System besteht aus **kommunizierenden Endlichen Automaten**. Auf dem Nachrichtenpfad (Kanal) werden Ereignisse übertragen, die von einem Automaten gemäß seiner Ausgabefunktion λ generiert und vom anderen gemäß seiner Zustands-/Übergangsfunktion δ akzeptiert werden.

Beispiel: "Bit Stuffing"

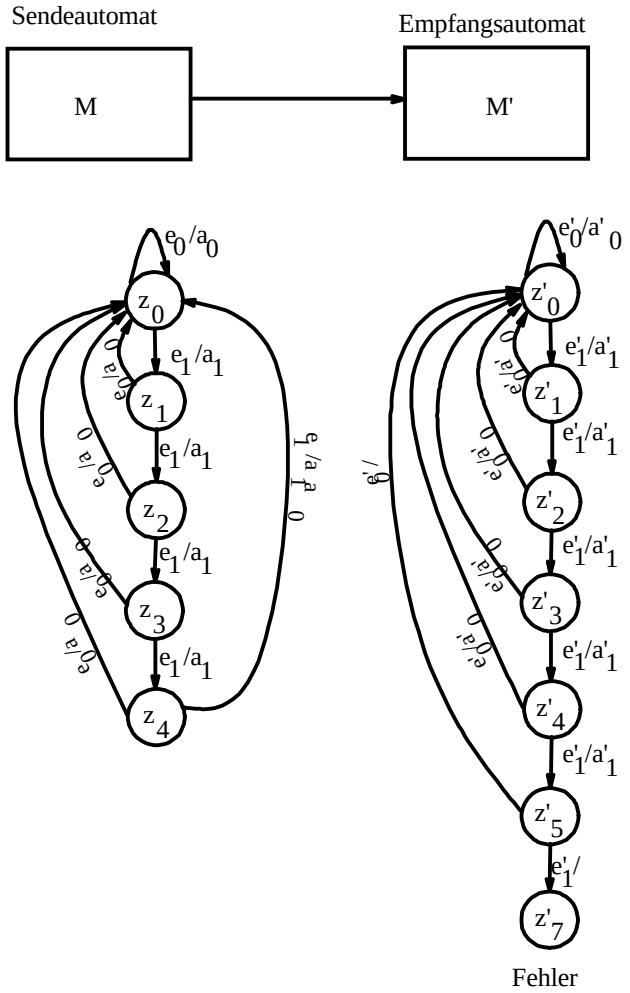


M:
Ereignis e_0 = Empfangen eines 0-Bits von höheren Schicht
Ereignis e_1 = Empfangen eines 1-Bits von höheren Schicht
Ausgabe a_0 = Erzeugen eines ausgehenden 0-Bits auf der Leitung
Ausgabe a_1 = Erzeugen eines ausgehenden 1-Bits auf der Leitung

M':
Ereignis e'_0 = Empfangen eines 0-Bits von höheren Schicht
Ereignis e'_1 = Empfangen eines 1-Bits von höheren Schicht
Ausgabe a'_0 = Erzeugen eines ausgehenden 0-Bits auf der Leitung
Ausgabe a'_1 = Erzeugen eines ausgehenden 1-Bits auf der Leitung

Vereinfachung der Automaten für "Bit Stuffing"

Unter Ausnutzung der Äquivalenz der Zustände z_5 und z_0 läßt sich der Sendeautomat wie folgt vereinfachen:



M:
Ereignis e_0 = Empfangen eines 0-Bits von höheren Schicht
Ereignis e_1 = Empfangen eines 1-Bits von höheren Schicht
Ausgabe a_0 = Erzeugen eines ausgehenden 0-Bits auf der Leitung
Ausgabe a_1 = Erzeugen eines ausgehenden 1-Bits auf der Leitung

M':
Ereignis e'_0 = Empfangen eines eintreffenden 0-Bits auf der Leitung
Ereignis e'_1 = Empfangen eines eintreffenden 1-Bits auf der Leitung
Ausgabe a'_0 = Weitergeben eines 0-Bits an die höhere Schicht
Ausgabe a'_1 = Weitergeben eines 1-Bits an die höhere Schicht

Analoges gilt für den Empfangsautomaten.

Erweiterte Endliche Automaten (1)

Reine Endliche Automaten können bei praktischen Problemen aus Lesbarkeitsgründen i.a. nur bei Zugrundelegung von Vereinfachungen zur Spezifikation verwendet werden. Um vollständige Spezifikationen zu erzielen, wird das Konzept endlicher Automaten (und ihre Beschreibung mittels Übergangsdiagrammen) erweitert.

Die Erweiterungen betreffen:
a) den Automaten selbst
b) die Kopplung mehrerer Automaten
c) die Beschreibung des Automaten

Zu a):
Variablen/Parameter, Zuweisungen, Bedingungen
Zur Behandlung von Zustandsinformation, die nicht unmittelbar den dynamischen Ablauf eines Protokolls/Dienstes beeinflußt (z.B. Nutzdaten, Adressen) werden **Variablen** und ggf. Parameter eingeführt:
Zustandsraum des erweiterten Automaten
$Z' = Z \times V_1 \times V_2 \times ... \times P_1 \times P_2 \times ...$
mit

- Z: Für den grundlegenden Ablauf relevante Zustände
- V_i Wertebereich der Variablen 'i'
- P_i Wertebereich des Parameters "i"

Erweiterte Endliche Automaten (2)

Im reinen Automaten werden Zustandsübergänge in Abhängigkeit vom aktuellen Eingabezeichen ausgeführt. Im erweiterten Automaten muß ebenfalls die aktuelle Variablenbelegung berücksichtigt werden. Hierzu werden **Bedingungen** (logische Formeln) über Variablen, Parametern und aktuellen Eingabezeichen verwendet. Eine Bedingung wird einem Übergang zugeordnet; dieser findet statt, wenn die Bedingung wahr wird.

Im reinen Automaten wird bei einem Zustandsübergang ein Ausgabezeichen erzeugt. Im erweiterten Automaten muß zusätzlich die Variablenbelegung veränderbar sein. Hierzu können **Zuweisungen** einem Übergang zugeordnet werden.

Zeitgeber

Da Zeitschranken (englisch: Timer) des öfteren benötigt werden, ist es sinnvoll, diese sozusagen als Variablen mit Eigenleben vorzusehen.

Diese könnten z.B. durch Deklarationen vereinbart werden.

```
timer T1 (200 msec);
```

Spezielle Operationen:

START (Timer)	Timer starten
RESET (Timer)	Timer rücksetzen, anhalten

und die Abfragefunktion

TIMEOUT (Timer): bool	Zeit abgelaufen

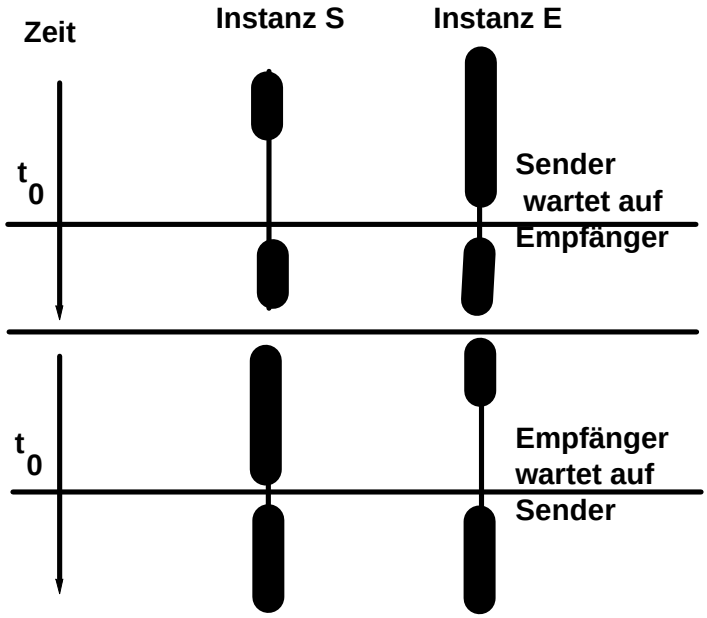
ermöglichen es, Timer anzusprechen.

START und RESET können als Zuweisungen aufgefaßt werden. TIMEOUT erlaubt es, den Timer-Zustand in Bedingungen zu testen.

Kanäle

Um der Bedeutung des Orts (Dienstzugangspunkts) gerecht zu werden, an dem ein Ein- oder Ausgabeereignis auftritt, können **Kanäle** eingeführt werden. Ein Kanal verbindet i.a. genau zwei Automaten und transferiert Zeichen in einer Richtung.

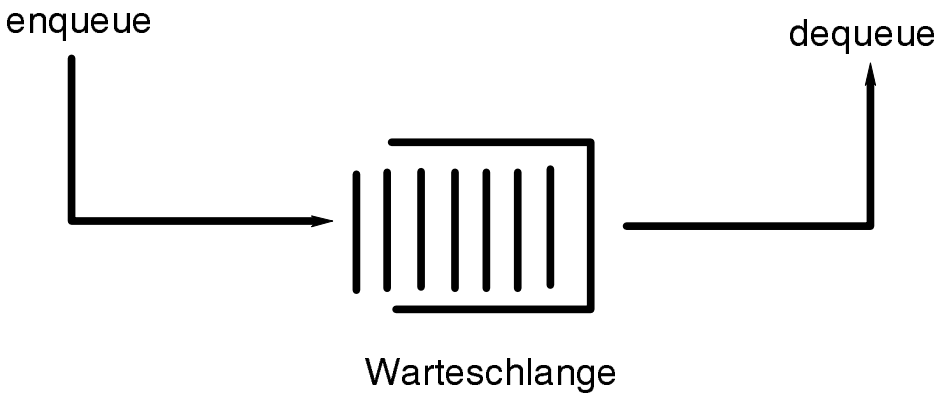
Im einfachsten Fall sind Kanäle nicht-speichernd, d.h. ein gesendetes Zeichen muß unmittelbar zum Sendezeitpunkt auch empfangen werden (synchrone Kommunikation, auch "Rendezvous-Konzept" genannt).



Warteschlangen (puffernde Kanäle)

Kanäle können zwischenspeichernde (puffernde) Eigenschaft besitzen.

Die Kapazität kann (zu Spezifikationszwecken) unbegrenzt sein oder mit einem Limit (maxímale Anzahl von Nachrichten) angegeben werden. Zwischenspeichernde Kanäle werden auch Warteschlangen genannt.



Ein Sender kann mittels 'enqueue' ein Zeichen (eine Nachricht) im Kanal hinterlegen. Er wird nur blockiert, wenn die Speicherkapazität überschritten wird.

Zeitlich unabhängig davon kann ein Empfänger mittels 'dequeue' ein Zeichen aus dem Kanal lesen und entfernen. Er wird nur blockiert, wenn kein Zeichen gespeichert ist. Als Spezialfall (Kapazität = 0) ergibt sich die Rendezvous-Kopplung.

Sprachen zur Spezifikation von Endlichen Automaten

Zur programmiersprachlichen Beschreibung eines Automaten wird i.a. der Zustandsraum Z als Wertemenge eines Aufzählungsdatentyps vereinbart. Ferner werden Variablen, Kanäle und Timer sowie benötigte Datentypen und Konstanten vereinbart.

Der "ausführbare" Teil des "Programms" besteht nicht aus einer Folge von Anweisungen, sondern aus Abschnitten. Jeder Abschnitt ist einem Wert aus Z zugeordnet. Innerhalb eines solchen Abschnitts werden die Übergänge, die aus dem entsprechenden Zustand möglich sind, durch Angabe der Bedingung, des Zielzustands und der Aktionen, die mit dem Übergang auszuführen sind, beschrieben.

Beispiel:
Die von der ISO standardisierte Sprache ESTELLE

Literatur:
D. Hogrefe: ESTELLE, LOTOS und SDL. Springer-Verlag, Heidelberg, 1989

10.3 Estelle

Eine von der ISO standardisierte Sprache auf der Basis der Erweiterten Endlichen Automaten (EFSM: Extended Finite State Machine).

Eine ESTELLE-Spezifikation beschreibt eine hierarchisch geordnete Menge von EFSMs, genannt **Modulinstanzen** (oder **Tasks**), die parallel ablaufen und durch Nachrichtenübermittlung miteinander kommunizieren.

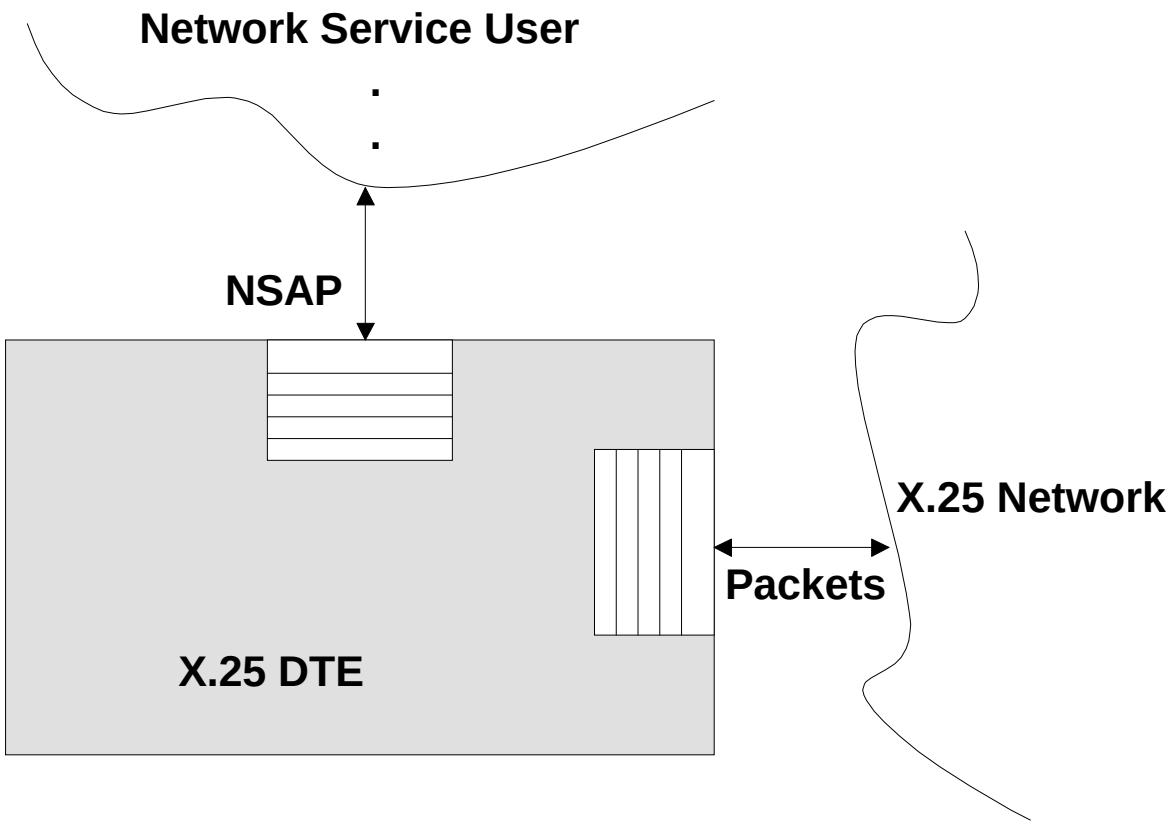
Nebenzustandvariablen, Prädikate und Aktionen von EFSMs werden als Variablen, Funktionen, Anweisungen und Prozeduren in PASCAL-Notation dargestellt. Die Programmiersprache PASCAL ist vollständig in Estelle einbezogen und wird auch für Typdefinitionen verwendet. Kommunikation zwischen Tasks wird durch bidirektionale **Kanäle** erreicht, die je zwei Interaktionspunkte verschiedener Modulinstanzen verbinden. Jedem Interaktionspunkt ist eine unendlich große FIFO-Warteschlange zugeordnet, die Nachrichten (genannt **Interaktionen**) auf der Empfängerseite puffert.

Estelle-Grundlagen

Um eine Modulinstanz erzeugen zu können, wird ein Modulkopf mit einem Modulrumpf in einer **init**-Anweisung verbunden. Die Interaktionspunkte der Tasks werden mit den Interaktionspunkten anderer Tasks durch die Benutzung der **connect**- oder **attach**-Anweisung verbunden.

Alle Modulinstanzen laufen in Übereinstimmung mit ihren Attributen parallel. Während der dynamischen Ausführung des Systems können mittels **release**-, **disconnect**- und **detach**-Anweisungen Modulinstanzen gelöscht oder Kommunikationsverbindungen umgeleitet werden.

Estelle - Beispiel: X.25 DTE (1)



Estelle - Beispiel: X.25 DTE (2)

Dienstprimitive der Schicht 3 nach ISO

connection establishment	N-CONNECT-req N-CONNECT-ind N-CONNECT-rsp N-CONNECT-cnf
data transfer	N-DATA-req N-DATA-ind N-DATA-ACKNOWLEDGEMENT-req N-DATA-ACKNOWLEDGEMENT-ind N-EXPECTED-DATA-req N-EXPECTED-DATA-ind N-RESET-req N-RESET-ind N-RESET-rsp N-RESET-cnf
connection release	N-DISCONNECT-req N-DISCONNECT-ind

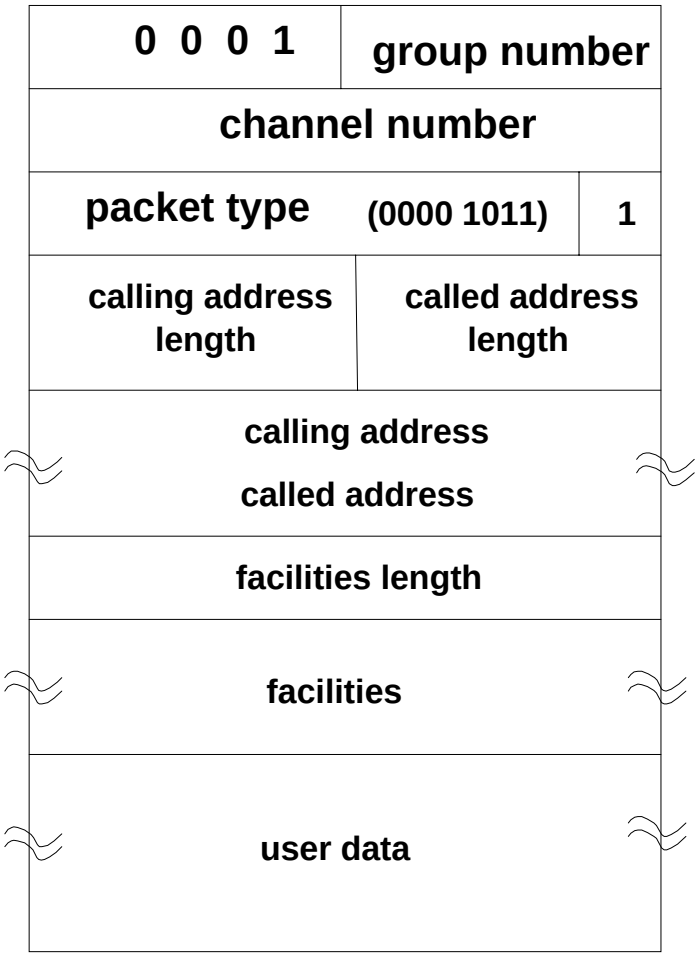
Estelle - Beispiel: X.25 DTE (3)

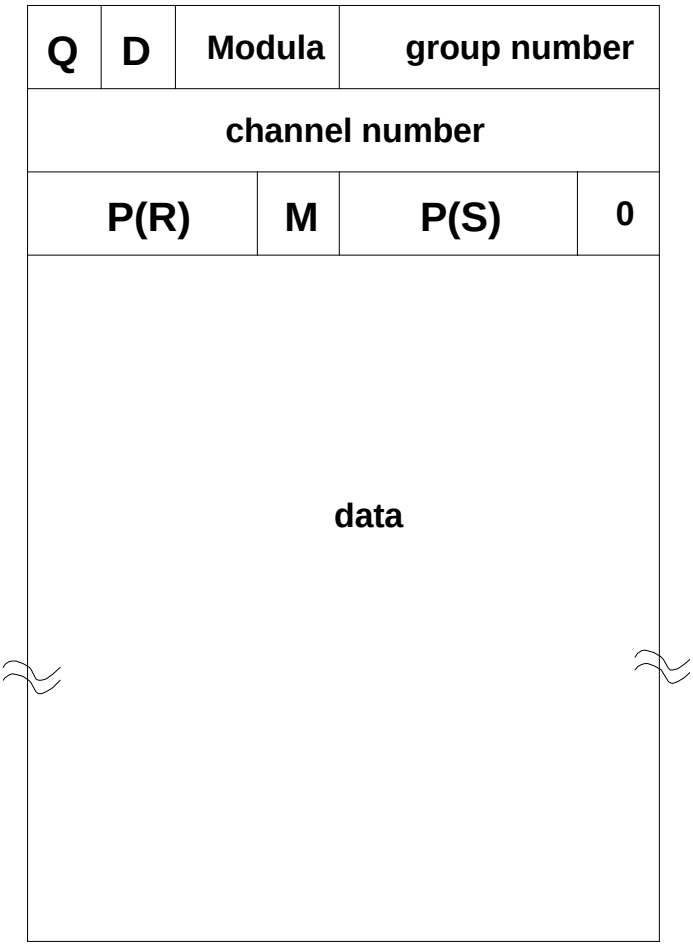
X.25 - Pakete nach CCITT

connection establishment (call setup)	CALL REQUEST INCOMING CALL CALL ACCEPTED CALL CONNECTED
data transfer	DATA INTERRUPT INTERRUPT CONFIRMATION RECEIVE READY RECEIVE NOT READY REJECT RESET TEQUEST RESET CONFIRMATION
connection release (call clearing)	CLEAR REQUEST CLEAR INDICATION CLEAR CONFIRMATION
restart	RESTART REQUEST RESTART CONFIRMATION DIAGNOSTIC REGISTRATION REQUEST REGISTRATION CONFIRMATION

Estelle - Beispiel: X.25 DTE (4)

PDUs für PLP CALL-REQUEST und DATA





Estelle - Beispiel: X.25 DTE (5)

Estelle - Spezifikation (Ausschnitt)

```
modul X_25_DTE activity (LC: Logical_channel_number);
   ip NSAP    :Network_service_access_point (PROVIDER);
      Packets :X_25_network_access_point    (DTE);
end;
channel Network_service_access_point  (USER, PROVIDER);
   by USER: N_CONNECT-req  (N_CR_SP : N_CONNECT_request);
     ...
   by PROVIDER:N_CONNECT_ind (N_CI_SP:N_CONNECT_
   indication);
    ...
   ;
channel X_25_network_access_point (DTE, DCE);
   by DTE: Call_request    (CR_PDU: CALL_REQUEST);
     ...
   by DCE: Incoming_call   (IC_PDU: INCOMING_CALL);
    ...
   ;
```

Estelle - Beispiel: X.25 DTE (6)

```
TYPE
N_CONNECT_request   =    RECORD
         called_address: ARRAY [1..MAX_ADD] OF CHAR;
         calling_address: ARRAY [1..MAX_ADD] OF CHAR;
         receipt_confirmation_selection   : BOOLEAN;
         expedited_data_selection         : BOOLEAN;
         quality_of_service  : Network_quality_of_service;
         user_data    : ARRAY [1..MAX_UDAT] OF OCTET
                  END;
...
CALL_REQUEST        =     RECORD
         header: Packet_header;
         calling_dte_address_length: INTEGER;
         called_dte_address_length: INTEGER;
         called_dte_address: BCD_STRING;
         calling_dte_address:BCD_STRING;
         facility_length: INTEGER;
         facility_field: Facilities;
         user_data: ARRAY [1..16] OF OCTET
                  END;
```

Estelle - Beispiel: X.25 DTE (7)

```
Packet_header       =      RECORD
q_bit          : BOOLEAN;
d_bit          : BOOLEAN;
modulo           :(modulo_8, modulo_128);
lc_number        :INTEGER;
packet_type_id : OCTET
                            END;
...
LC_status_type     =      (free, pending, busy);
```

```
modvar dte_1: X_25_DTE (1c);
 ...
 initialize

      begin
        init dte_1 with X_25_DTE_body;
        ...
        connect dte_1.NSAP      to ...;
        connect dte_1.Packets   to ...;
      end;
```

Estelle - Beispiel: X.25 DTE (8)

```
body X_25_DTE_body for X_25_DTE;
     state p1, p2, p3, p4, p5, p6, p7;
     VAR LC_status: LC_status_type;
     ...
     initialize to p1;
         begin
            LC_status: = free;
            ...
         end;
     trans
         from          p1
         to         p2
         when       NSAP.N_CONNECT_req
         provided      LC_status = free
         begin
            CR_PDU.header.q_bit    : = FALSE;
            CR_PDU.header.d_bit    : =
                    N_CR_SP.receipt_confirmation_selection;
            CR_PDU.header.1c_number : = LC;
            ...
            output Packets.Call_request (CR_PDU);
            LC_status                : = pending;
          end;
     ...
end;
```

10.4 SDL (Standard Description Language)

Eine von der ITU-T standardisierte Sprache auf der Basis der Erweiterten Endlichen Automaten (EFSM = Extended Finite State Machine).

Die wichtigsten Sprachkonstrukte von SDL (1)

Die syntaktischen Formen von SDL

- SDL/PR (SDL Phrase Representation)
- SDL/GR (SDL Graphical Representation)

SDL/PR und SDL/GR sind sematisch äquivalent.

SDL/GR	SDL/PR
Empfänger (1,1)	**PROCESS Empfänger (1,1)** **REFERENCED;**

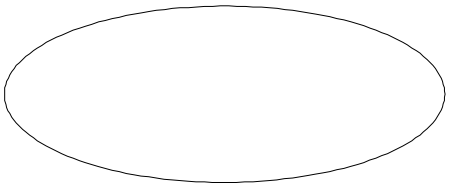
Die wichtigsten Sprachkonstrukte von SDL (1)

Zustände

In SDL wird das Verhalten eines Prozesses mit Zuständen und Zustandsübergängen beschrieben.

Zustandssymbol:

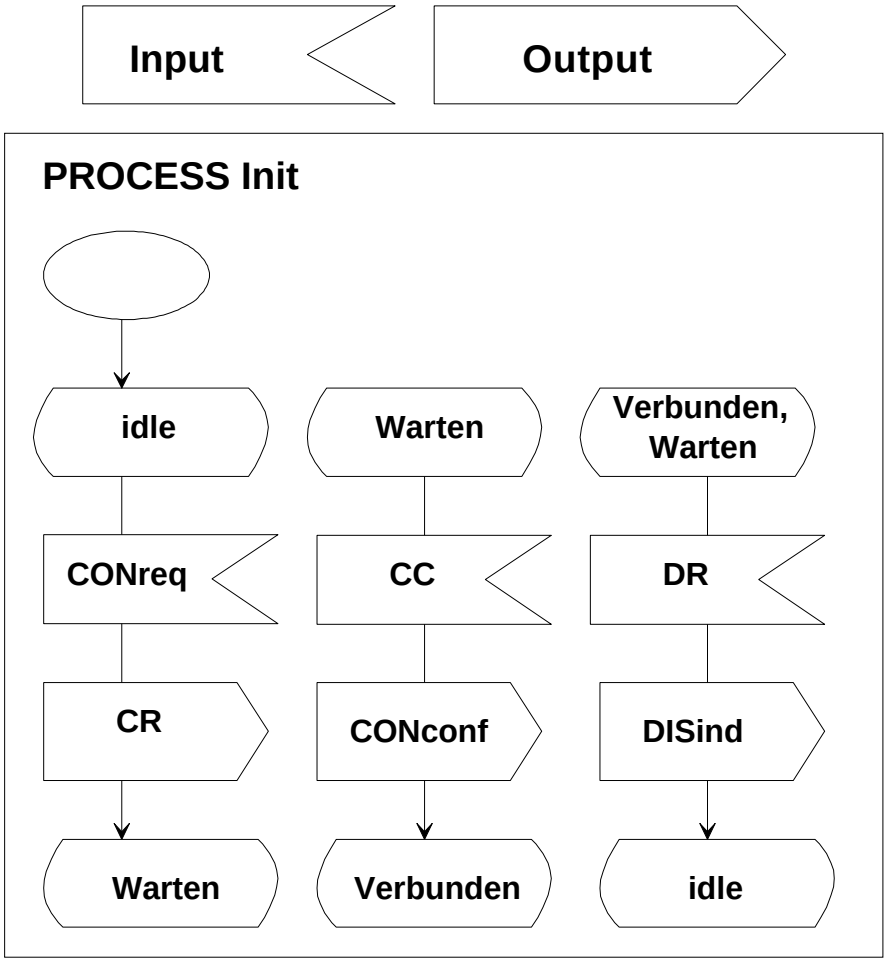
Startsymbol:



Die wichtigsten Sprachkonstrukte von SDL (2)

Signale

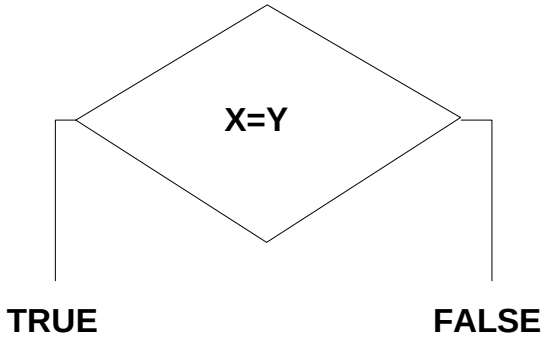
- Datenaustausch
- Anstoß einer Transition



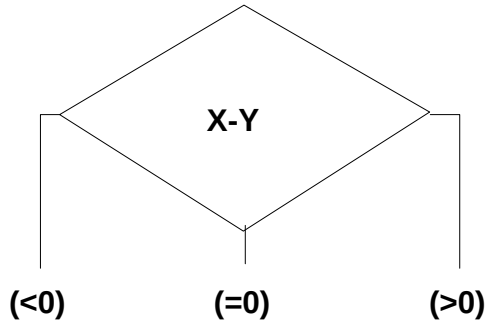
Die wichtigsten Sprachkonstrukte von SDL (3)

Alternativen

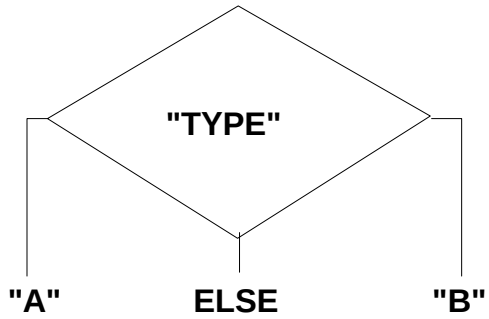
- einfache Alternative



- mehrfache Alternative



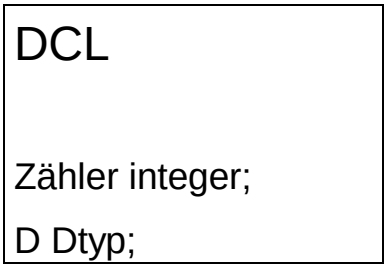
- Alternative mit ELSE



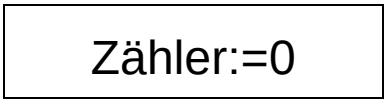
Die wichtigsten Sprachkonstrukte von SDL (5)

Daten in SDL

- Definition von Daten im **Textsymbol**



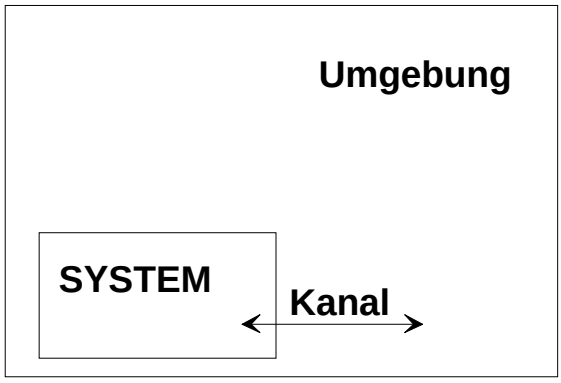
- Manipulation von Daten im ***TASK***-Symbol



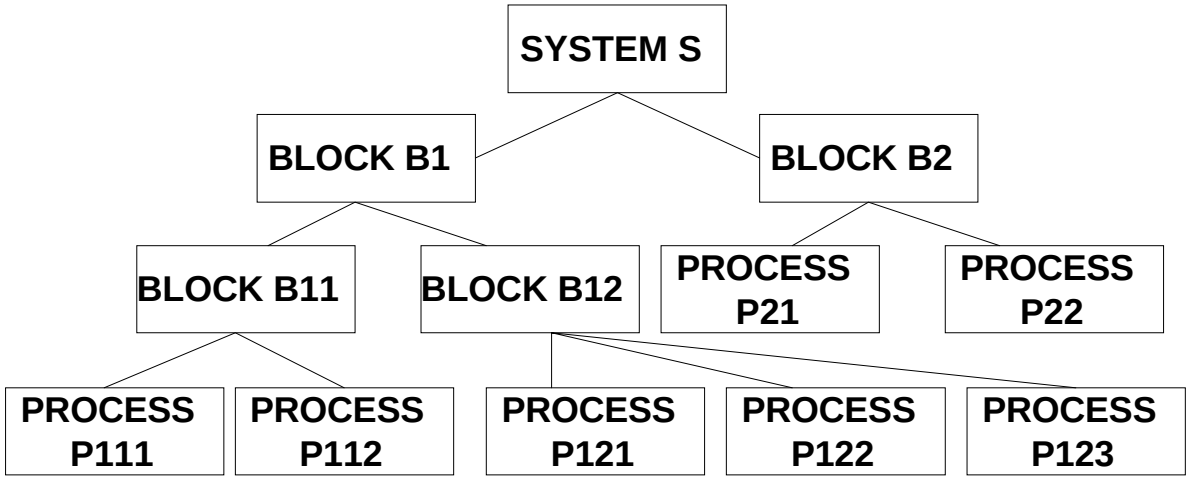
Strukturierung und Prozeßkommunikation

System-, Block- und Prozeßstrukturen

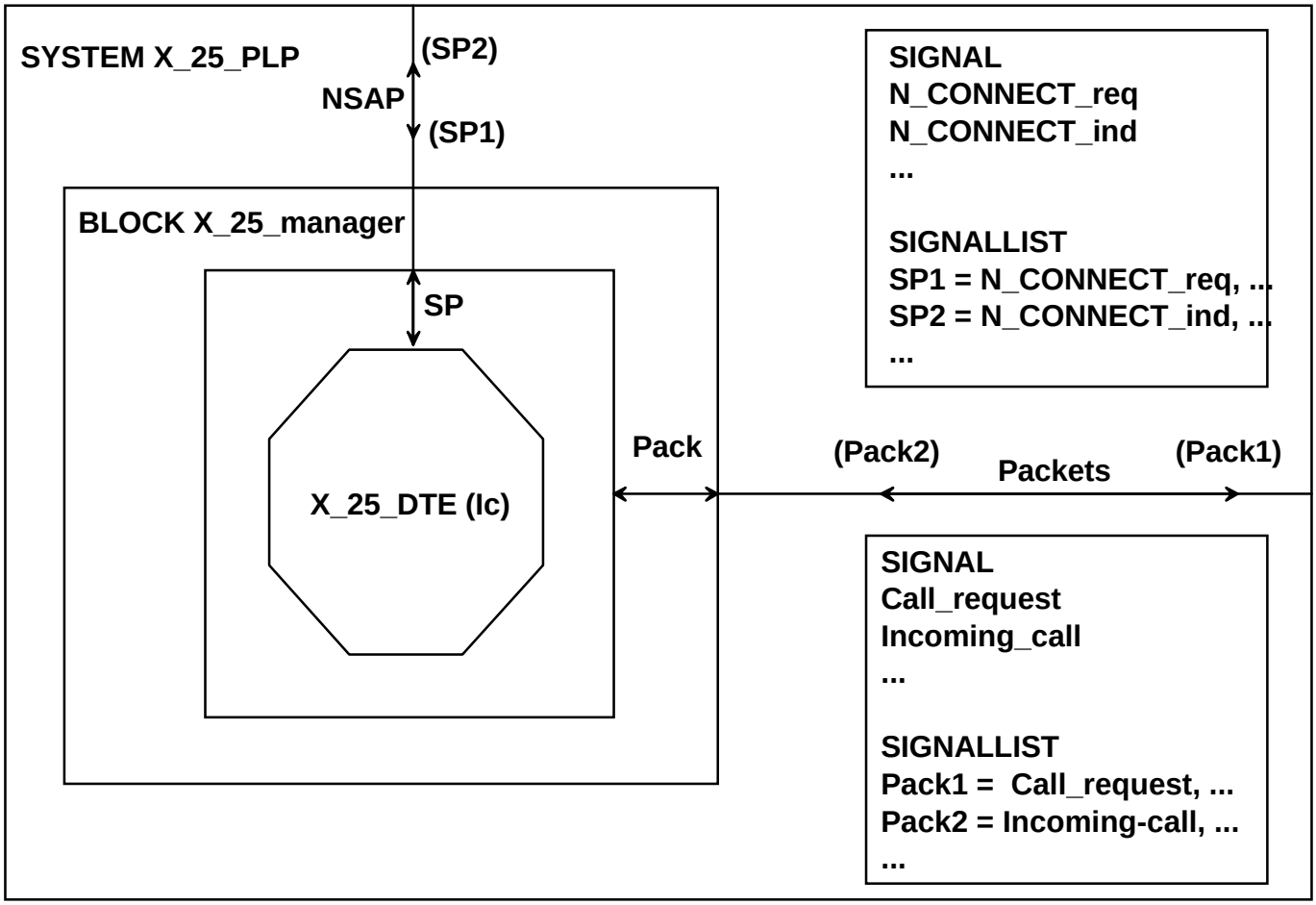
- System
- Umgebung
- Kanäle



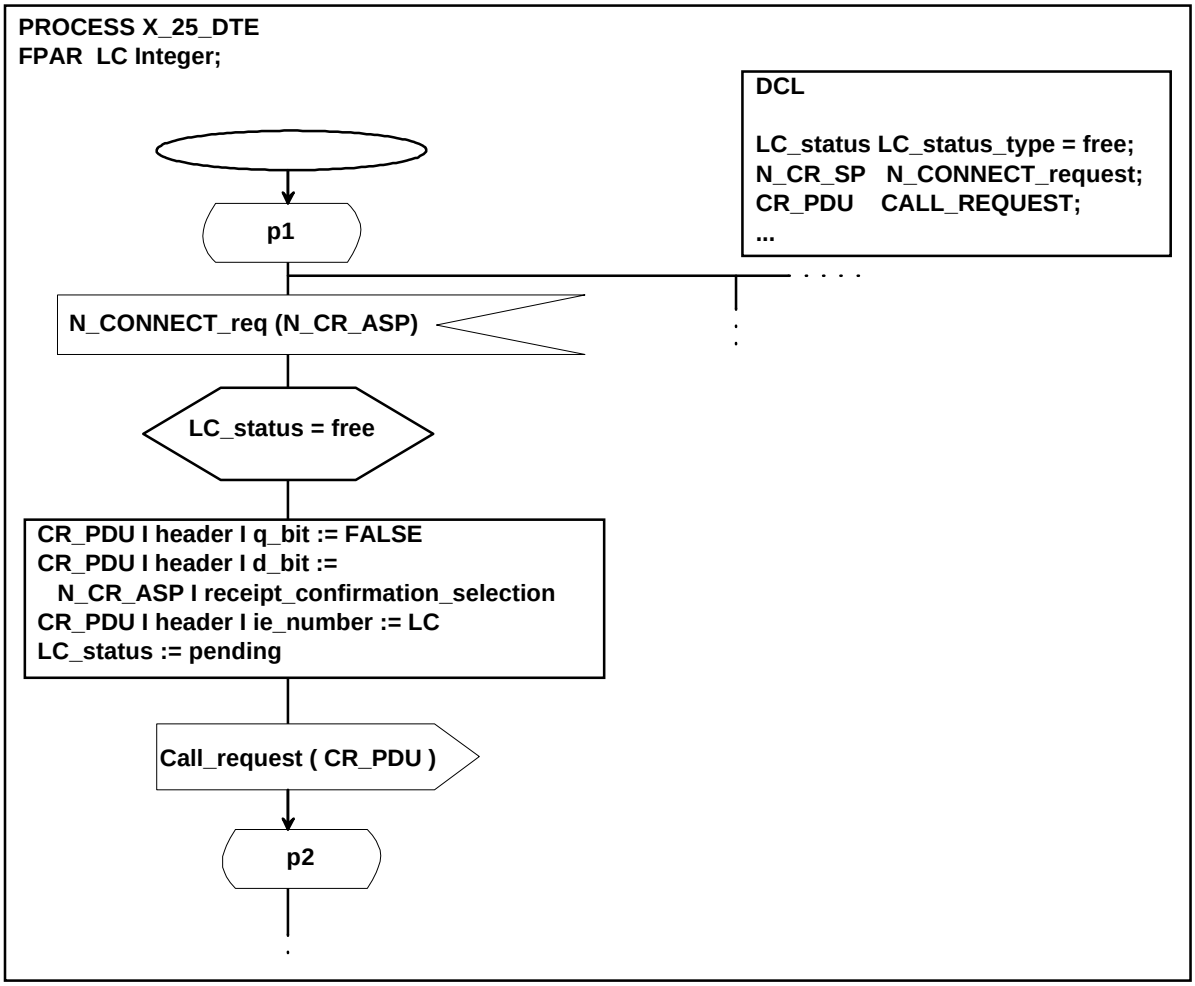
- System
- Blöcke
- Prozesse



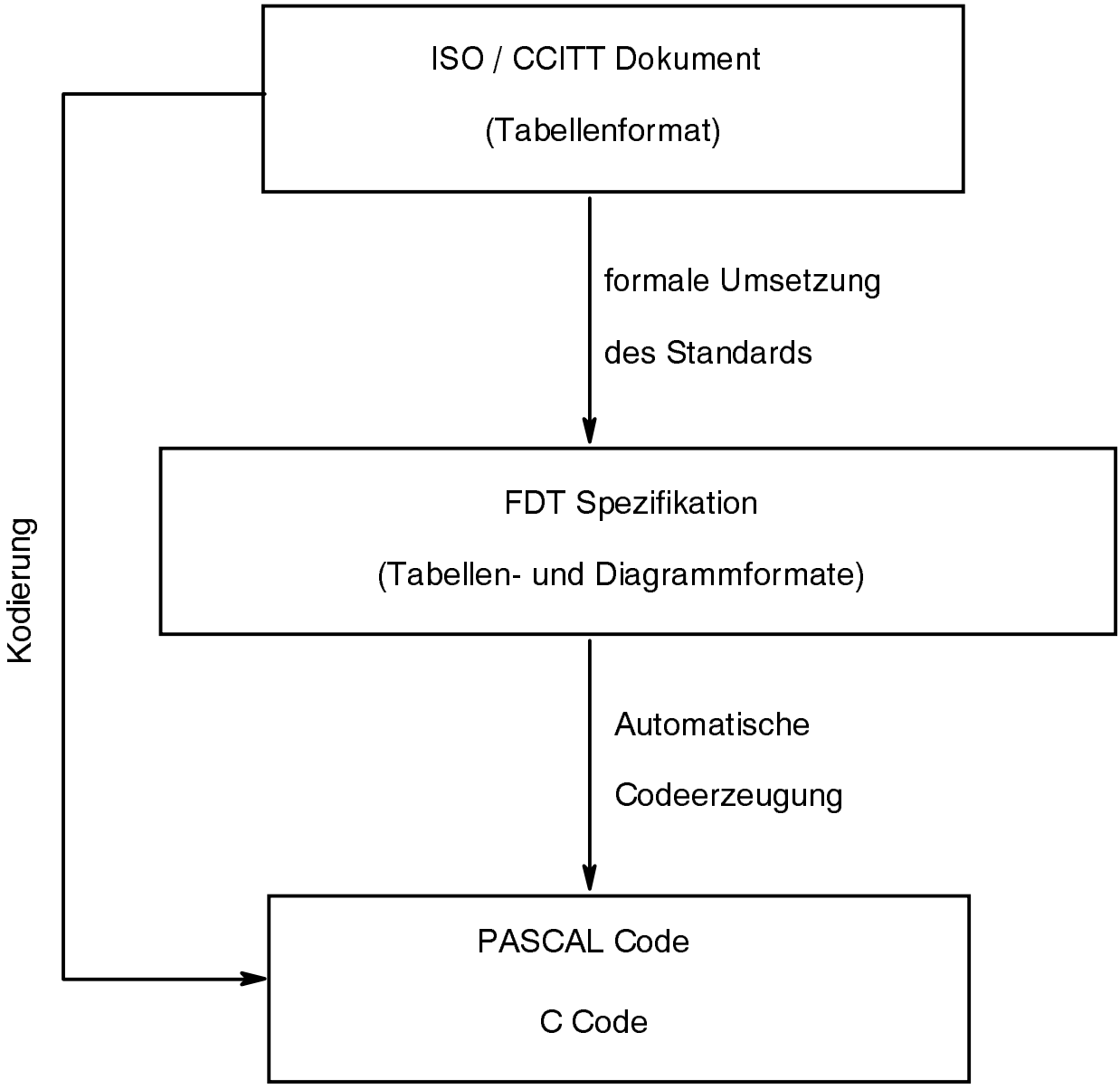
SDL- Beispiel: X.25 DTE



SDL-Prozeß für X.25 DTE



10.5 Unterstützung der Protokollentwicklung durch Werkzeuge



Beispiel für eine Entwicklungsumgebung