SDL – Specifikation and Description Language

Überblick[1]

- 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,
  - Echtzeitanforderung
  - 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 (ITU-TS: ITU-Telecommunications Standardization) angewendet.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Überblick[1]

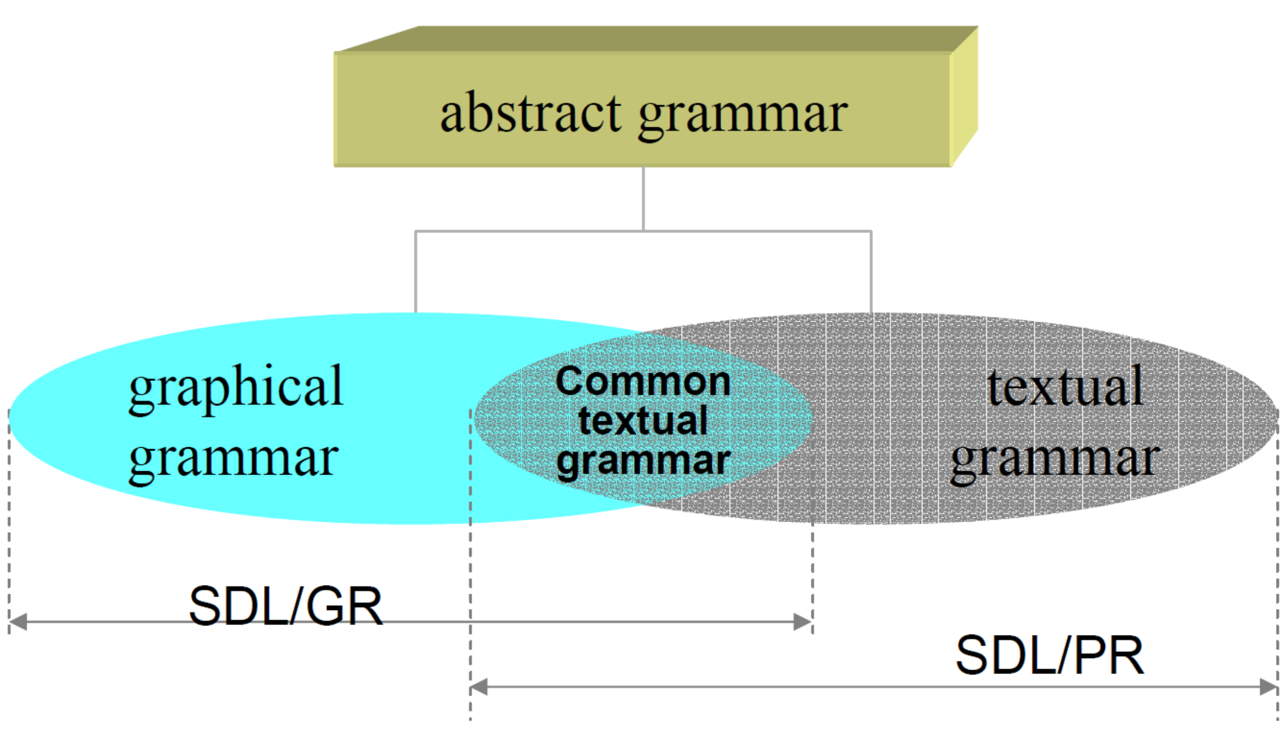
- 1968-1972: Erste Untersuchungen, Ziele für Entwicklung wurden formuliert
- 1972-1976: Grafische Darstellungsform SDL/GR (graphic representation) der Sprache wird im Ansatz entwickelt und im Orange Book als Z.101 bis Z.103 veröffentlicht.
- 1976-1980: Entwicklung der textualen Darstellungsform SDL/PR (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 abwärtskompatibel. Die Erweiterungen gehen in Richtung objektorientierter Spezifikation.
- 1996 Überarbeitung (Glätten) von SDL-92
- 2000 komplette Überarbeitung von SDL-96, vollständig Objektorientiert
- 2007: Korrekturen und kleine Änderungen zur Version von 2000
- 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 textualen Syntax
  - ANNEX D SDL-Nutzerhandbuch
  - ANNEX E Zustandsorientierte Darstellung und deren Bildelemente
  - ANNEX F Formale statische und dynamische Definition

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Ausprägungen von SDL[1]

- SDL/GR: Graphical Representation
- SDL/PR: Phrase (textuelle) Rrepresentation
- 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 Sprachelemente auf Symbole ab.
- Datenvereinbarungen werden in beiden Formen textual notiert.



1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Hauptmerkmale von SDL[1]

- 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.
  - Dynamisches **Verhalten**, beschrieben durch PROCESSes (Prozessbeschreibung).
  - **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.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Architektur: Übersicht[1]

Environment

System

Block1

Block2

Subblock1

Subblock2

Process1

Process2

Process3

Process4

Procedure1

Procedure2

Procedure3

Procedure4

Procedure5

Dynamisches Verhalten

Signalaustausch über Channels

Signalaustausch über Signalrouten

- Bei der **Systemspezifikation** werden die Blöcke, die Kommunikationskanäle zwischen Blöcken mit den Signalen und die Kommunikationskanäle zwischen Blöcken und der Umgebung 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.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Architektur: SDL-Sprachelemente zur Architekturbeschreibung[1]

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.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Architektur: beispielhaftes System[1]

SYSTEM X

SIGNAL akt, pas, x(NATURAL), y(PId) ,i ,j ; SIGNALLIST L1= akt,pas;

C1 [(L1)]

B1

C2 [i, j]

B2

C3 [x,y]

BLOCK B2

CONNECT C2 AND R1;
CONNECT C3 AND R3;
SIGNAL k1(INTEGER);

R1 [i,j]

P1 (1,1)

R4 [k1,j]

R3 [x,y]

P8 (1,1)

PROCEDURE A1

DCL i NATURAL;

true

i>9

false

i:=0

i:=i+i

x(i)

x(i)

PROCESS P8

(1,1);

A1

Z0

/ Hier steht ein Kommentar */*
DCL z INTEGER;

k1(z)

j

A1

y(SELF)

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Verhalten: SDL-Sprachelemente zur Verhaltensbeschreibung[1]

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 PROCE-DURE

RETURN: Ende einer PROCEDURE und Rück-sprung 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

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Kommunikation[1]

- Kommunikation findet über SIGNALs statt. Signale können parametrisiert sein.
- Zwischen Blöcken werden Signale über CHANNELs, zwischen Prozessen über SIGNALROUTEs ausgetauscht
  - Channels und Signalroutes können uni- oder bidirektional sein

__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)

SYSTEM X

C1 [(L1)] B1 C2 [i, j] C3 [x,y] B2

SIGNAL akt, pas, x(NATURAL), y(PId) ,i ,j ; SIGNALLIST L1= akt,pas;

BLOCK B2

R1 [i,j] P1 (1,1) R4 [k1,j] P8 (1,1) R3 [x,y]

CONNECT C2 AND R1;
CONNECT C3 AND R3;
SIGNAL k1(INTEGER);

*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.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Kommunikation[1]

- 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.

Prozesse schicken per OUTPUT über SIGNALROUTEs SIGNALs an andere Prozesse

FIFO
PROCESS P1
i:=5
k1(i)

FIFO
PROCESS P8
Warten
k1(myInt)

Prozesse belassen SIGNALs in ihrem Briefkasten mit SAVE

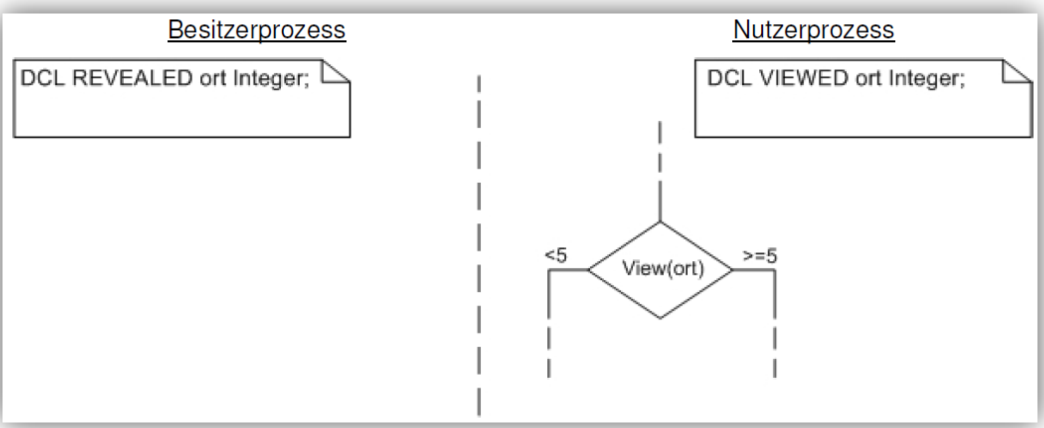
Prozesse konsu-mieren SIGNALs aus ihrem Brief-kasten mit INPUT

- 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.

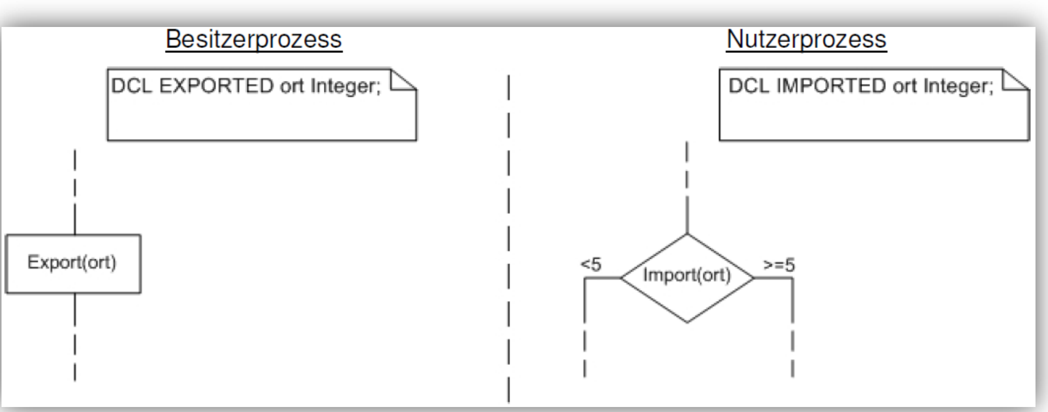
1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Kommunikation[1]

- Benötigt ein Prozess Zugriff auf eine Variable eines anderen Prozesses, muss ein Signalaustausch stattfinden.
- Wenn sich Prozesse im gleichen Block befinden nutzt man
  - Im Besitzerprozess:
    - REVEALED (offenbaren)
  - Im Nutzerprozess:
    - VIEWED (nachsehen)



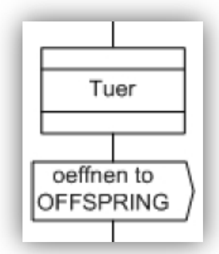
- Wenn sich Prozesse nicht im gleichen Block befinden nutzt man
  - Im Besitzerprozess:
    - EXPORTED (auslagern)
  - Im Nutzerprozess:
    - IMPORTED (einlagern)



1) Doreen Surek – Konzept und Implementation einer Testumgebung für SDL

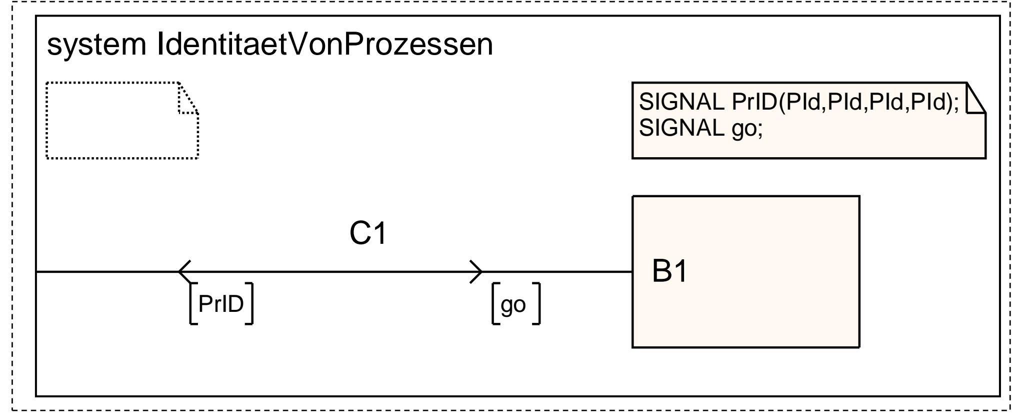
Dynamisches Verhalten[1]

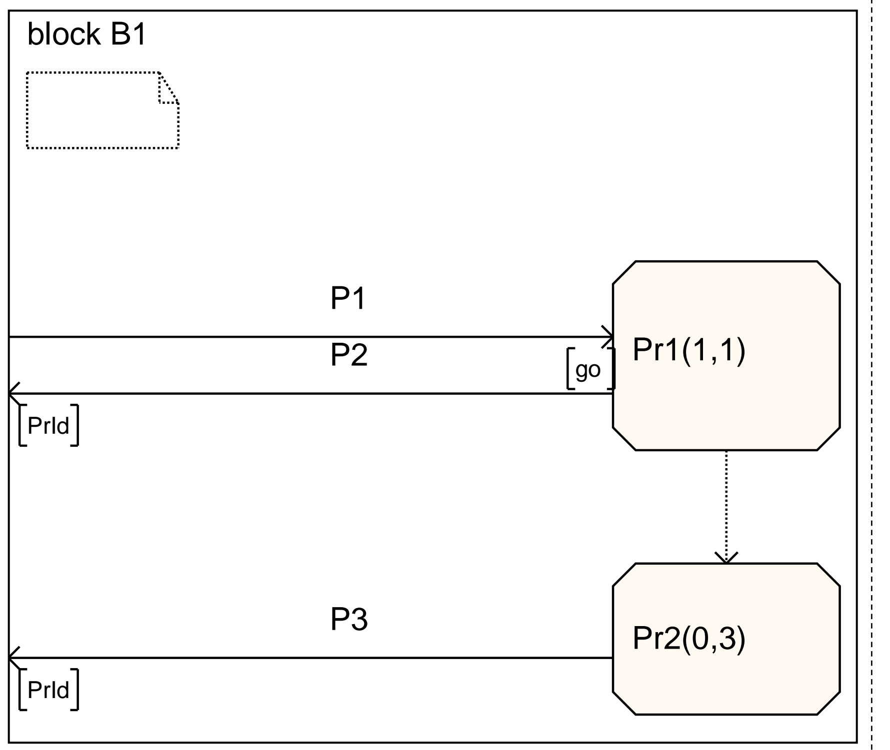
- Dynamisches Verhalten wird durch Prozesse beschrieben
- Prozesse können kreiert werden:
  - zum Zeitpunkt des Systemstarts,
  - während der Laufzeit (CREATE).
- Prozesse, egal wie sie erzeugt wurden, können zur Laufzeit terminiert werden (STOP)
- In einem SYSTEM können mehrere verschiedene Prozesse 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's, die aber nicht immer einen Wert besitzen:
  - SELF: Ist eine Referenz einer Instanz auf sich selber. Damit kann eine Instanz z.B. sich selber ein Signal schicken usw. (dieser Wert existiert immer)
  - PARENT: kennzeichnet den Prozess, durch den eine Instanz erzeugt wurde. Wurde der Prozess beim Start des Systems kreiert, ist der Wert NULL.
  - SENDER: PId des Prozesses, von dem zuletzt ein Signal konsumiert wurde.
  - OFFSPRING: In OFFSPRING steht der PId, der zuletzt kreierten Instanz (PId der zuletzt mit CREATE erzeugten Instanz)
- PId's sind ein gutes Mittel, um Signale gezielt zwischen inkarnierten Prozessen auszutauschen
  - Dazu im Output-Symbol das Signal angeben, gefolgt von „to“ und dem Namen, der die PId liefert

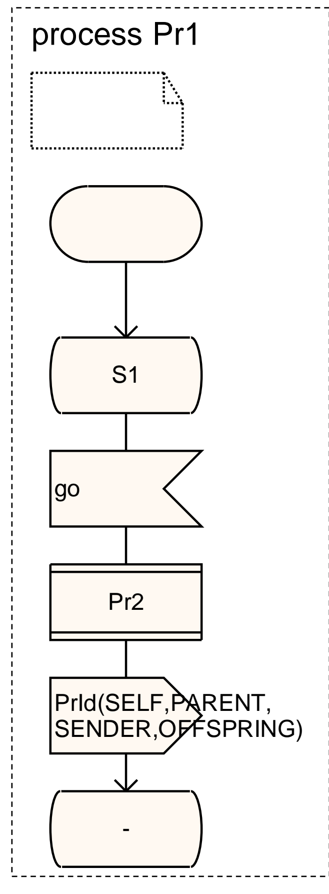


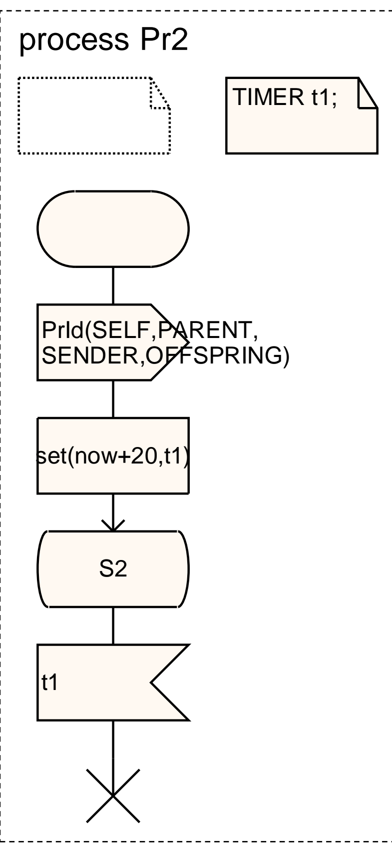
1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Dynamisches Verhalten: Beispiel: Identität von Prozessen[1]



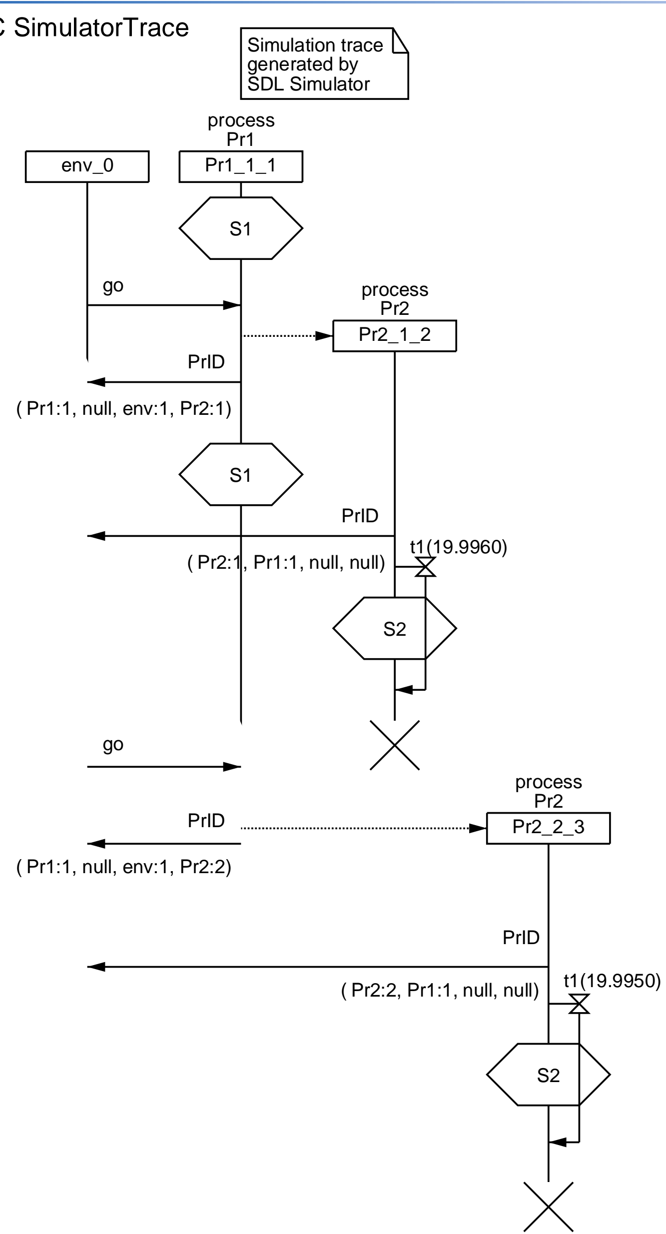






1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Dynamisches Verhalten: Beispiel: Identität von Prozessen[1]



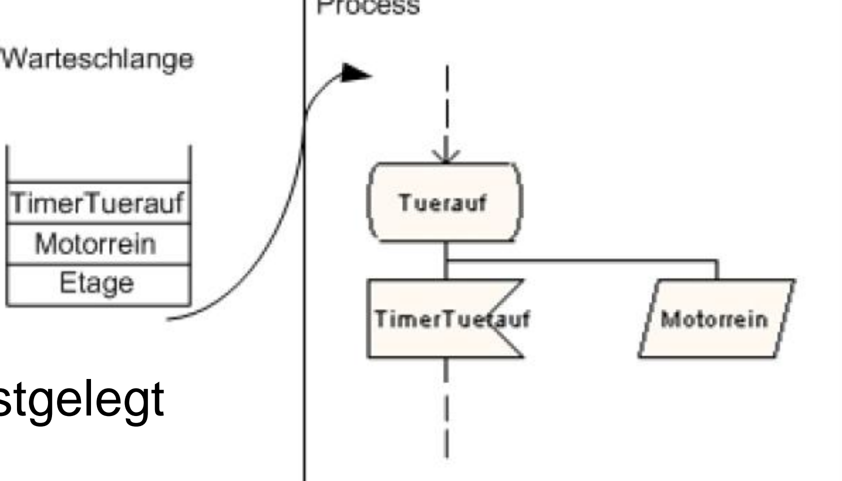
1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Dynamisches Verhalten[1]

- Ist die Adresse des Zielprozesses nicht bekannt und das Signal kann von mehreren Prozessen empfangen werden, kann ein Signal auch über einen definierten Channel oder eine Signalroute an den Zielprozess geschickt werden
  - Dazu wird im Output-Symbol das Signal angegeben, gefolgt von „via“ und dem Namen des Channels bzw. der Signalroute

trenne
via C1

- Empfangene Signale, die zum gegenwärtigen Zeitpunkt nicht für einen Zustandsübergang genutzt werden, werden verworfen.
  - Wenn Signal zu einem späteren Zeitpunkt benötigt wird, kann es mittels des Save-Symbols gespeichert werden.
  - Das Signal wird erst dann verarbeitet, wenn der entsprechende Zustand im Prozess erreicht wurde.



- Datentyp und Signaldefinitionen werden textuell festgelegt
  - Signal-Definition mit Präfix SIGNAL
  - Signallisten-Definition mit Präfix SIGNALLIST
  - Datentypdefinitionen
    - mit Präfix NEWTYPE und Suffix ENDNEWTYPE
    - Oder mit Präfix SYNTYPE und Suffix ENDSYNTYPE
- Definition von Daten (Variablen) erfolgt in Prozessebene mittels „DCL“

1) Doreen Surek – Konzept und Implementation einer Testumgebung für SDL

Abstract Data Types[1]

- SDL ist eine Spezifikations- und Beschreibungssprache die einen rechnergestützten Softwareentwurf unterstützen soll. Die Zielsprache einer Implementation kann z.B. Pascal, 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	- Real
- Character: *alle IA5-Zeichen*	- PId: *Prozessidentifier*
- Charstring: *alle IA5-Zeichen*	- Duration: *Intervall zwischen Zeitpunkten (Dauer)*
- Integer: *negative, positive Ganzzahlen*	-Time: *Zeitpunkt (das NOW-Konzept liefert die momentane Systemzeit)*
- Natural: *positive Ganzzahl*	

- Des weiteren gibt es zur Erzeugung komplexerer Strukturen Generatoren für:
  - String,
  - Array.

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Abstract Data Types: Deklaration der Sort Boolean aus Z.100[1]

```
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;
```

1) Prof. Dr. Ing.-habil. L. Winkler – SDL

Abstract Data Types: Was man damit machen kann[1]

- 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 die 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.

- Mittels STRUCT kann man eine Struktur anlegen, die aus mehreren verschiedenen Variablen besteht

```
NEWTYPE Daten STRUCT
Name Charstring;
Telefonnummer Integer;
ENDNEWTYPE;
```

  - *In diesem Fall besteht die Struktur „Daten“ aus den Variablen „Name“ (Typ: Charstring) und „Telefonnummer“ (Typ: Integer). Wenn nun eine Variable „Telefonbuch“ vom Typ „Daten“ angelegt wird, kann mittels „Telefonbuch!Name“ auf die Variable zugegriffen werden.*

- Ein Array kann mittels ARRAY angelegt werden

```
NEWTYPE Zimmer ARRAY (Natural,Boolean)
ENDNEWTYPE;
DCL Hotel Zimmer;
```

  - *In diesem Fall wird ein neuer Typ „Zimmer“ als Array definiert, welches die Typen Natural und Boolean enthält. Für die jeweilige Zimmernummer kann true(Zimmer frei) oder false(Zimmer belegt) eingetragen werden.*

1) Prof. Dr. Ing.-habil. L. Winkler – SDL