Specification and Description Language (SDL)

Dr.-Ing. Andreas Könsgen
ajk@comnets.uni-bremen.de
Introduction

- Formal specification languages
- Performance evaluation of formally specified protocols by means of simulation and emulation
- Prediction of hardware requirements, e.g. used amount of memory and whether real-time conditions are met
- Emulation of SDL implementations for target processor systems
- In Comnets, only simulation is considered
SDL
Specification and Description Language

- Current version SDL-2000 standardized by ITU series Z.100
- SDL/GR: SDL Graphical Representation
- SDL/PR: SDL Phrase Representation
- System description in SDL:
  a set of communicating processes
- SDL process: Communicating Extended FSM: CEFSM (variables)
- History:
  - ITU (CCITT) identified need for formal specification techniques (approx. 1970)
  - First version of SDL in 1976
  - Established standard 1988
  - SDL-92: object oriented
  - SDL-96: update of SDL-92, corrections and small extensions
  - SDL 2000: current version
SDL: Hierarchy and Communication

- **Hierarchy:**
  - Environment (*Umgebung*)
  - System contains one or more blocks
  - Block can consist of blocks (recursive)
  - Block contains one or more processes
  - Process

- **Communication**
  - System communicates with the environment via channels (*Kanäle*)
  - Blocks communicate via channels
  - Processes communicate via signal routes (*Signalwege*)
  - Processes can create other processes and can terminate themselves
  - Communication is realized by messages that are called *signals*. Signals are generated and consumed by processes and forwarded by blocks.
SDL System: Static Structure
SDL Processes

- Behavior of a system described in SDL is defined by a set of processes.
- Process in SDL is an extended finite state machine.
- Processes exist in parallel and communicate via signals.
- Process is either in a state waiting for an input signal or in a state transition.

- During a state transition:
  - variables can be manipulated
  - decisions can be made
  - output can be generated
  - signals can be sent and
  - new processes can be created.

- An input signal to a process in a state transition is stored in an input queue. Each process has its own input queue.
SDL processes (cont.)

- 2 concurrent input signals are stored in random sequence.
- Process is identified by its process identifier (PID) (Prozesskennung).

```
Block bkAB

SIGNAL sInputA1, sInputA2, sInputB, sOutput;
SIGNALLIST s1InputA = sInputA1, sInputA2;
```

Diagram:

```
[(s1InputA)]                     [sInputB]                     [sOutput]
          |                          |                          |
          v                          v                          v
          -->                         -->                         -->
          Process prA                Process prB                Process prB
srRoute1  srRoute2               srRoute3
```
SDL example declaration

```plaintext
syntype Int_127 = Integer
constants 0:127
endsyntype;

dcl count Integer;
timer Signal Go;
```
SDL
Timer Example
(Taktgenerator)

Available Timer Actions:
set, reset, active
Newtype C1_TYPE
choice
  a Integer;
  b Charstring
  c Boolean
endnewtype C1_TYPE;

DCL
intvar Integer,
cstvar Charstring
boolvar Boolean;

DCL
InputData C1_TYPE;

wait_for_signal

InFromSender (InputData)

IntChosen TO SENDER
CharChosen TO SENDER
BoolChosen TO SENDER

wait_for_signal
### SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung Deutsch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Start</td>
<td>marks beginning of process, exists once for every process</td>
</tr>
<tr>
<td>S1</td>
<td>Zustand</td>
<td>symbolizes the state of a system, source or sink</td>
</tr>
<tr>
<td>input</td>
<td>Eingabe</td>
<td>always following a state symbol, if process in this state receives this input signal, it follows this resp. flow (transition)</td>
</tr>
</tbody>
</table>
## SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>save</td>
<td>Sichern</td>
<td>possibility to save an input signal for later processing</td>
</tr>
<tr>
<td>output</td>
<td>Ausgabe</td>
<td>sends a signal, usually at the end of a transition</td>
</tr>
<tr>
<td>task</td>
<td>Anweisung</td>
<td>general tasks, e.g. assignments or setting a timer</td>
</tr>
</tbody>
</table>

\[
a := 5
\]
### SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung Deutsch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &gt; 2</td>
<td>decision</td>
<td>boolean decision or case statement</td>
</tr>
<tr>
<td>true</td>
<td>Entscheidung</td>
<td>specification of labels for jumps</td>
</tr>
<tr>
<td>false</td>
<td>Verbindung</td>
<td>termination of a process</td>
</tr>
<tr>
<td>stop</td>
<td>Stop</td>
<td></td>
</tr>
</tbody>
</table>

- **SDL/PR**: symbols used in SDL/PR
- **Bezeichnung Deutsch**: German terms for the symbols
- **Meaning**: English explanation of the symbols
## SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung Deutsch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure call</td>
<td>Prozedur-auf Ruf</td>
<td>calls a procedure</td>
</tr>
<tr>
<td>procedure</td>
<td>Deklaration einer</td>
<td>declaration of a procedure which is called in the</td>
</tr>
<tr>
<td>reference</td>
<td>Prozedur</td>
<td>current process</td>
</tr>
<tr>
<td>procedure start</td>
<td>Prozedur-anfang</td>
<td>start of a procedure definition</td>
</tr>
</tbody>
</table>
### SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung Deutsch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Textsymbol</td>
<td>Text symbols contain the declaration of data structures, variables, timers, and signals. System and block level: channels and signal routes</td>
</tr>
<tr>
<td>text extension</td>
<td>Text-erweiterung</td>
<td>comment</td>
</tr>
<tr>
<td>text extension</td>
<td>text</td>
<td>procedure return Rücksprungmarke end of a procedure, return to origin of call</td>
</tr>
<tr>
<td>task block</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## SDL Symbols

<table>
<thead>
<tr>
<th>SDL/PR</th>
<th>Bezeichnung Deutsch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>system type reference</td>
<td>Referenz auf Systemtyp</td>
<td>declaration of a system level type that can be inherited</td>
</tr>
<tr>
<td>block type reference</td>
<td>Referenz auf Blocktyp</td>
<td>declaration of a block level type that can be inherited</td>
</tr>
<tr>
<td>process type reference</td>
<td>Referenz auf Prozesstyp</td>
<td>declaration of a process level type that can be inherited</td>
</tr>
</tbody>
</table>
Example: Ticket Machine as an SDL System

See also
[Wa0102]
page 71-74
Block

block Hauptblock

C2

C1

Hauptprozess

(slToEnv) [ (slFromEnv) ]

process Hauptprozess

Fahrkartenautomat

DCL
Zielort, Ziele Charstring,
Fahrpreis, Bezahlt, Betrag Real;

Preis

process
Bezahlt := 0

Warte_auf_Fahrziel

Geldwurf
(Betrage)

Geldwurf
(Betrage)

Warte_auf_Geldwurf

Fahrpreis

Preisausgabe
(Fahrpreis)

Fahrpreis

>0

else

- Preis

- Preis

Abbruch

Bezahlt > 0

True

else

Warte_auf_Fahrziel

Geldwurf

(Betrage)

Bezahlt := 0

Geldwurf

(Betrage)
procedure Preis

'Zielort'

'Hamburg'

'Koeln'

'Muenchen'

Fahrpreis := 33.0000
Fahrpreis := 156.0000
Fahrpreis := 389.0000
Fahrpreis := -1
Signalling in SDL

- If more than one signal arrives at a process at a time, the randomly arriving signals are queued in FIFO order
  - Due to random arrival, order of processing the signals is unpredictable
- Only coarse prioritization possible for signal inputs (i.e. Two queues, standard and priority)
Multiple process instances

- In the block diagram, more than one instance of a particular process can be created by specifying a number after the process name.
- Example: a number of stations in a network.
- Each single instance must be individually addressable.
- Solution: each instance registers at an external process (simulation control). The external process responds by assigning a unique number (e.g., MAC-Id).

```
MACLayer(10)
```

```
SimulationControl
```

```
SIGNAL
sRegister_req,
sRegister_cnf(Natural)
```

```
MACLayer(10)
```
Saving signals

- In some cases, signals have to be processed in another order than FIFO
- Signals can be saved for later evaluation

The signal C is saved, so A and B are always processed before C, independent on the order of arrival
Runtime creation of processes (1)

- By default, an SDL process exists for the full simulation runtime.
- In some cases, it cannot be determined at compile-time how many instances of a process will be needed.
- Example: maintaining queues for different traffic categories; dependent on the simulator configuration.
- Dynamic processes, created at runtime.

```plaintext
newProcess
```
Runtime creation of processes (2)

- In the system view, the creating process points to the dynamic process by a dashed arrow, in addition to the signal arrow.
- The number in parentheses indicate the minimum and maximum allowed number of processes. No number means unlimited.
Exporting variables

- Problem: a value has to be delivered among a large number of processes inside a block
- Example: MAC address inside the processes of a protocol stack
- Declaration in the block:
  \[
  \text{REMOTE MACId Natural;}
  \]
- Declaration in the exporting process:
  \[
  \text{DCL EXPORTED MACId Natural;}
  \]
- Usage in the exporting process:
  \[
  \text{MACId := ...; /* assign a value */ export(MACId);}
  \]
- Declaration in the importing process:
  \[
  \text{DCL IMPORTED MACId Natural;}
  \]
- Usage in the importing process:
  \[
  \text{thisMACId := import(MACId);}
  \]
- The value cannot be stored in the imported variable, hence the extra variable thisMACId is needed
Continuous signals

- Sometimes it is needed to monitor an (imported) variable for a change of its value
- Example: at the simulation start, all processes of the MAC layer wait before starting execution until the MACId is available
- Inside the symbol, a boolean condition for continuation is specified

```
WaitForMACId
Import(MACId) != invalid
ready
```
Catching invalid signals

- To keep the overview on the simulator's behaviour, it should be made sure that all signals are caught in any state.
- To cause some printout for unexpected/forgotten signals:

```
  * (any state)
  * (any signal)
  printError (Print error message)
  - (return to previous state)
```
Abstract Data Types (ADT) [Ellsberger 1997]

- ITU-T Recommendation Z.105
- Definition of data in an implementation-independent way
  - Define the result of operations on data objects without constraining how the result is obtained
- For pre-defined and user-defined data
- Specified by
  - one or more literals, e.g. names of possible values
  - operators for that sort
  - A formula or a C language function describing each operator
simple Example

NEWTYPE tBoolean

LITERALS
  true, false;
OPERATORS
  not: tBoolean -> tBoolean;
AXIOMS
  not(true) == false;
  not(false) == true;
ENDNEWTYPE;

Usage of new type in SDL:

DCL var1 tBoolean, var2 tBoolean;
...
var1 := false;
var2 := not(var1);
Example with CNCL

NEWTYPE tBGenFibo /*#NAME ´CNFiboGPtr´ */
LITERALS
    newBGenFibo /*#NAME ´new_BGenFibo´*/
OPERATORS
    newBGenFibo /*#NAME ´fibo_constructor´ */
        : integer -> tBGenFibo
    /*#ADT
    #TYPE
    #include <CNCL/FiboG.h>
    typedef CNFiboG* CNFiboPtr;
    #HEADING
    #define new_BGenFibo() fibo_constructor(123)
    extern CNFiboGPtr fibo_constructor(int seed);
    #BODY
    extern CNFiboGPtr (int seed = 54217317) {...}
    */
ENDNEWTYPE;
Usage in SDL

DCL

    fibo_generator tBGenFibo; /* base generator */
    negExpVal := tDistNegExp; /* negative exp. Distribution */
    meanValue Real := 1.0,        /* mean value for distribution */
    interarrivalTime Real;       /* current random value */

    fibo_generator := newBGenFibo;

    negExpVal := newDistNegExp(meanValue, fibo_generator);

    interarrivalTime := GetNegexpReal(negExpVal);
Utilisation of ADTs

- **Input/Output:**
  - Printing console messages
  - File reading (configuration) and writing (analysis)

- **Communication with the CNCL**
  - Random base generator
  - Distribution functions
  - Statistical analysis (LRE)

- **General calculations**
  - Absolute power to dBm

- **Parallelisation of simulators**
  - Interface to the High Level Architecture
Software development cycle

1. Real scenario
2. SDL specification
3. C implementation
4. Compilation
5. Simulation run
6. Statistical evaluation
Software development cycle

- Real scenario
- SDL specification
- C implementation
- Compilation
- Simulation run
- Statistical evaluation
Building a simulation

- Tool to develop SDL programs: Telelogic Tau
  - Graphical editor with online syntax check for system, block and process level
  - Syntactic and semantic analysis
  - SDL to C translation
- Optionally: Editing C files (text) for ADTs
- Compilation of all C files into object files
- Compilation of special C files shipped with the Telelogic Tau tool which contain the code for process control, signal control and scheduling
- Linking the files to executable simulator
  - Object files generated from SDL
  - Object files shipped with the Telelogic tool
  - Object files and libraries (e.g. CNCL) containing ADT code