Specification and Description Language (SDL)

Dr.-Ing. A. Diab
Outline

• Introduction
• System Development with SDL
• Conclusions
• References
Introduction
Motivation

• Performance Issues play a major role in the systems engineering process
  – Performance-related problems are
    – Major reasons for project failures
    – Mostly due to poor design rather than to implementation details

• To avoid performance pitfalls
  – Integrate performance-related activities in early phases of the systems engineering process
  
  ➔ Performance engineering
Formal Descriptions Techniques (FDT)

- Allow the lifting of functional assertions from testing phases to more abstract phases of the system engineering process
- Extend system engineering concepts
  - Associate a formal definition to the language constructs
    - Enable defining the semantics of a given system specification
    - Allow to formally reason about the system during early design
  \[\Rightarrow\text{Early detection of errors, thus, reducing the cost of corrections}\]
- The widely used FDT is the Specification and Description Language (SDL)
System Development with SDL
Overview

• Specification and description of systems
  – Specification of a system
    – The description of the required behavior of the system
  – Description of a system
    – The description of the actual behavior of the system

• Easy to understand for both specifications creators and viewers

• Independent of design paradigms

• Independent of implementations details

• Object oriented language

• Include formal semantics

• Best adequate for communication systems
  – Message passing instead of shared memory
  – Operation on data
History

- Standardized by the International Telecommunication Union (ITU)

- SDL-76 (First version)
  - Recommendations on how to draw process graph symbols (Behavior)

- SDL-80
  - Blocks are introduced, textual phrase representation becomes a part of the language

- SDL-84
  - Abstract data type concept is introduced

- SDL-88
  - Minor changes
History

• SDL-92
  – Object-oriented extensions

• SDL-96
  – Minor changes

• SDL-2000 (widely deployed version)
  – Architecture and data models have changed

• SDL-2010 (expected coming version)
  – The major influence is likely the UML2.0

• Notes
  – SDL-88, SDL-92, SDL-2000 and SDL-2010 \(\rightarrow\) versions of the standard
  – SDL'99 and SDL'01 \(\rightarrow\) the 1999 and 2001 SDL Forum
  – SDL2003, SDL2005 and SDL2007 \(\rightarrow\) the 11th, 12th and 13th SDL Forum
Basic Concept of Design Flow

System architecture (SDL editor)

Use cases (MSC editor)

SDL coding

Code analysis

Code generation

Simulation

SDL code generation

C/C++ source code

Compile and link

Execution file

OK

No

Yes
Agents

• Express system, block and process concepts

• Agent declaration
  – Attributes
    • Parameters, variables, procedures, etc.
  – Behavior
    • Implicit/explicit state machine
  – Internal structure
    • Contained agents, channels, etc.

• Agent instance is an Extended Finite State Machine (EFSM)
  – Identity
  – Signal input queue
  – Life cycle
  – Reactive behavior specification
Agents

Block Scheme 1

- dcl i Natural,
  c Character;

- Contained agent

- Declaration of communication channels

- Component 1

- Agent Scheme 1

- Declaration of local variables

- Component 2
Agents hierarchy

System Scheme1

Scheme1

Scheme2

Block Scheme1

dcl i Natural,
c Character;

Process_1

Process_2

Process Process_1

Initiating

Disconnected

L2_LU(CH)

set(now+p,T1);

layer2connector

EFSM
System Development with SDL: System Specification
System Specification

- Topmost level of abstraction of SDL specification
  - Very abstract view of the system without going into details

- The system is a set of blocks that communicate with each other and
  with the environment via channels
  - The mode of communication on the channels is via signals

- The system’s environment behaves in SDL-like manner

- System specification contains four parts
  - System package reference part
  - System declaration part
  - System types specification part
  - Block interaction part
System Specification

Graphical representation

Textual representation

```plaintext
<system package references>

system <system name>

<system declaration part>

<system types specification part>

<block interaction part>

[<system package references>]
System <system name>;
<system declarations>
<types>
<block interaction>
endsystem [<system name>];
```
System Declarations

- System declarations are contained in one or more text symbols

```
<declarations>
```

- In the system level, you can define signals only (variables are defined inside processes)

```
Signal
<signal name>,<signal name>, <signal name>(type of content)
```

```
Signal
Advertisement,Solicitation,
BU(MIP_Message),
BA(Boolean);
```
System Types Specification

- System types specification part contains the types defined for the system
  - Block type references
  - Process type references
  - New data types

- Block type references

**Graphical representation**

```
<block type name>
```

**Textual representation**

```
Block type <block type name> referenced;
```
System Types Specification

- Process type references

**Graphical representation**

```
<process type name> referenced;
```

**Textual representation**

- New data types

```
Newtype <new type name>
<declaration of variables>
endnewtype <new type name>;
```
Block Interaction

• Block interaction area
  – The blocks and the channels connecting the blocks with each other and with the environment

  – Block

    Graphical representation

    <block name>

    Textual representation

    Block <block name>
    <processes and channels declaration>
    endblock <block name>;

  – Block instantiating

    Graphical representation

    <block name>:
    <block type name>

    Textual representation

    Block <block name>: <block type name>;
Block Interaction

• Channels connect block instances with each other and with the environment

• Transport signals

• Two types of channels
  – Delaying channels
    – signals transported experience delays on the channel
  – Non-delaying channels
    – Delay is zero
Block Interaction

- Delaying channels

**Graphical representation**

![Graphical representation of delaying channels](image)

**Textual representation**

```plaintext
channel <channel name>
  from <block name | env>
  via <gate>
  to <block name | env>
  via <gate>
  with <signal list>;

from <block name | env>
  via <gate>
  to <block name | env>
  via <gate>
  with <signal list>;
endchannel <channel name>;
```
Block Interaction

- Non-delaying channels

**Graphical representation**

```
[<signal list>] [<signal list>]
[<gate>] ← [<gate>]
    [<channel name>]

[<signal list>]
[<gate>] → [<gate>]
    [<channel name>]
```

**Textual representation**

```
channel <channel name> nodelay
from <block name | env>
via <gate>
to <block name | env>
via <gate>
with <signal list>;

from <block name | env>
via <gate>
to <block name | env>
via <gate>
with <signal list>;
endchannel <channel name>;
```
Example

```
use PackageSDL / process type Viewer;

system ToffeeVendor

signal
Coin10, Coin50, Coin100, CoinX, Button, Undo, Disp1,
  Overpay, Empty, Status, Complete, Maint, Exists, Paid,
  Toffee, Chocolate, Gum;

Dialogue

Flush

Pay

WMgr: WareType1

outgate

sync

Exists, Paid, CoinErr

Status, Complete, Maint

Coin10, Coin50, Coin100, CoinX

WareType1

[Coin10, Coin50, Coin100, CoinX]

[Disp1, Disp2, Overpay, Empty]

[Button, Undo]

InpC

Specification of toffee vendor system [2]
System Development with SDL: Block Specification
Block Specification

• A block specification describes a set of processes that communicate with each other and with the environment of the block
  – Communication with channels in the environment
  – Processes communicate with each other via signal routes

• Block specification contains four parts
  – Block package reference part
  – Block declaration part
    – Declarations of the signals transported
  – Block types specification part
    – Declarations of block/process types used
  – Process interaction part
    – Declarations of the processes contained in the block
Block Specification

Graphical representation

Textual representation

```
[block package references>

block <block name>

<block declaration part>

<block types specification part>

<process interaction part>

[endblock [<block name>];
```

```
[block package references>]]
block <block name>;
    <block declarations>
    <types>
    <process interaction>
    <channels to route connections>
endblock [<block name>];
```
Signal Routes

- Signal route are only non-delaying

**Graphical representation**

```
[<signal list>]  [<signal list>]
[<gate>]  -------->  [<gate>]
[<signal route name>]
```

**Textual representation**

```
signalroute <signalroute name>
from <process name | env>
via <gate>
to <block name | env>
via <gate>
with <signal list>;
```

```
from <process name | env>
via <gate>
to <block name | env>
via <gate>
with <signal list>;
```
Signal Routes

- Signal route should be connected to a channel locating in the environment of the block

**Graphical representation**

```
<channel name> [ <signal route name> ]
```

**Textual representation**

```plaintext
connect <channel name> and <signal route name>;
```

- Signal lists on the signal route should match that on the channel
Example

Specification of the block **Dialogue** (toffee vendor system) [2]
System Development with SDL: Process Specification
Process Specification

- Processes determine the behavior of the blocks they are included inside.
- Interconnected with each other via signal routes.
- Can be created while designing the system or even at run-time.
- Different instances of a process may be active at the same time and may have different states.
- Realizes an EFSM that is either in
  - a stable state or
  - A transition between states.
Process Specification

Graphical representation

<process package references>

process <process name>

<process declarations part>

<process types specification part>

<process graph part>

Textual representation

[<process package references>]

process <process name>

<process declarations>

<types>

<process body>

endprocess [<process name>];
Process Body

- Process body as an EFSM

**Graphical representation**

```
[<state name>]
```

**Textual representation**

```
state <state name>;
input <signal name>;
<transitions>
nextstate <state name>;
endstate [<state name>];
```
Process Body

- Process body as a FSM

Graphical representation

Textual representation

```plaintext
state <state name>;
input <signal name>;
nextstate <state name>;
endstate [<state name>];
```

- `state` <state name>
- `input` <signal name>
- `nextstate` <state name>
- `endstate` <state name>
Definition of Data Types

• Processes are capable of working with data/variables of certain types (referred to as sorts in the semantic of SDL)

• Specification of a sort
  – Value, operator and literal

• Main sorts defined for SDL
  – Pre-defined sorts
  – Structure sorts
  – Array sorts
  – Set sorts
  – Enumeration sorts
Definition of Data Types

• Pre-defined sorts
  – Integer
  – Boolean
  – Real
  – Character
  – Charstring

• Structure sorts
  ```
  newtype MessageContent struct
  type charstring;
  sequence integer;
  auth boolean;
  endnewtype MessageContent;
  ```
Definition of Data Types

- A value of a variable (message) from the type `MessageContent` can be
  (`. `Reg_Rqst`, 1200, true .)
  - `message!type := `Reg_Rqst``;
  - `message!sequence := 1200`;
  - `Message!auth := true`;

- **Array sorts**
  
  ```
  newtype MessageLists
  Array (integer, MessageContent)
  endnewtype MessageLists;
  ```

- MessageLists is an array that
  - has an index of the sort integer and
  - Components of the sort MessageContent
Definition of Data Types

• Set sorts

  \textbf{newtype} Integerset
  Powerset(integer)

  \textbf{adding operators}
  \hspace{1em} \text{take}: \text{Integerset} \rightarrow \text{integer};

  \textbf{endnewtype} Integerset;

• Example

  \text{myset} \text{Integerset};
  \text{a} \text{integer};
  \text{a} := \text{take(myset)};
Definition of Data Types

• Enumeration sorts

```haskell
newtype Item
  literals toffee, chocolate, tee;
  operators
    ordering;
    item_value: Item -> Integer;
endnewtype Item;
```

• All operators other than equality and inequality should be defined for the enumeration sort

• The term ordering defines the operators <, <= and >
Declaration of Variables

**Graphical representation**

```
dcl x,y,z integer;
dcl auth boolean := true;
```

**Textual representation**

```
dcl x,y,z integer;
dcl auth boolean := true;
```
Example

Specification of the process **control** (toffee vendor system) [Ref]
Processes Communication

- Process communication

![Diagram showing process communication with signal routes and input queues (FIFO)]
Processes Communication

Input queue (FIFO)

State **state1**: signal (a) initiates a transition to state2

State **state2**: signal (b) does not initiate any transition
System Development with SDL: Example
Mobile IP Fast Authentication Protocol

Layer3-Frequent Handoff Region (L3-FHR)

MN’s specific data (soft states)

IP-based network

New AR

Old AR

Trust

MAC

L2-HO

Net

L3-HO

Phy

Transport of bits
System View

Signals to control the simulation of MIFAService
Block MIFAService

Each important network element is specified as a block.

Wireless/wired channels are specified as blocks.
Block WirelessChannel1

The channel that connects the block to Mobile_Node block.

The channel that connects the block to FA1 block.
Process WCH1_Medium_Manager1

receives the data the MN sends:

- MDAT_out(d)
- d!payload!CH: WirelessCH1;
- MDAT_(d) via Internal

sends the data the medium carry:

- MDAT_in(d)
- Idle

Idle, d WirelessMedium_SDUType, Idle
Process WCH1_Medium_Manager1

newtype WirelessMedium_SDUType struct
payload WCH_MIFASDU;
endnewtype WirelessMedium_SDUType;

newtype WCH_MIFASDU struct
Mtype WCH_PDUType;
CH CR;
ADVStruct Advertisement;
REGStruct RegistrationRequest;
endnewtype WCH_MIFASDU;

Type of the message sent
Block Mobile_Node

The node is built according to ISO/OSI model

Physical layer Service Access Point (SAP)

Name of the channel that connects the MN with the wireless channel
Block MN_Physical_Layer

Converts the data received from the wireless channel to data the link layer can understand and vice versa.
Block MN_DataLink_Layer

- Converts data between the network and physical layer
- Filters the data oriented at the L2_Handoff process

Process to realize a layer 2 handoff procedure
Block MN_Network_Layer

- Converts data between the network and data link layer
Process NetLayer_Coder

```plaintext
process NetLayer_Coder

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCONind_</td>
<td>SduMtype:=MCONind; SduCH:=CH;</td>
</tr>
<tr>
<td>MDISind_</td>
<td>SduMtype:=MDISind; SduCH:=CH;</td>
</tr>
<tr>
<td>ASol</td>
<td>SduMtype:=ASol; SduCH:=CH;</td>
</tr>
<tr>
<td>RegRqst(REG)</td>
<td>SduMtype:=RegRqst; SduCH:=CH;</td>
</tr>
<tr>
<td>PrRtSol</td>
<td>SduMtype:=PrRtSol; SduCH:=CH;</td>
</tr>
<tr>
<td>Reset_ADV</td>
<td></td>
</tr>
<tr>
<td>Reset_REG</td>
<td></td>
</tr>
<tr>
<td>L3Dataout(Sdu)</td>
<td>Idlem</td>
</tr>
</tbody>
</table>
```

- `MCONind_`: MCONind event is triggered.
- `MDISind_`: MDISind event is triggered.
- `ASol`: ASol event is triggered.
- `RegRqst(REG)`: RegRqst(REG) event is triggered.
- `PrRtSol`: PrRtSol event is triggered.
- `Reset_ADV`: Reset_ADV event is triggered.
- `Reset_REG`: Reset_REG event is triggered.
- `L3Dataout(Sdu)`: L3Dataout(Sdu) event is triggered.
- `Idlem`: Transition to the Idle state.
Conclusions
Conclusions

• SDL
  – Specification and description of systems
  – Easy to understand
  – Independent of design paradigms and implementation details
  – Object oriented language
  – ...

• SDL helps in detecting design errors before real implementation

• Note
  – SDL provides if the system works as designed, not how good/bad the system is
References
References

