SDL-2010: Background, Rationale, and Survey

Rick Reed, TSE

SDL-2010: a revision of SDL-2000
ITU-T Specification and Description Language
Overview of session

1. Presentation of SDL-2010 cf. SDL-2000
   - Rationale for SDL-2010
   - Differences between SDL-2010 and SDL-2000

2. Discussion on SDL-2010 & further changes
Rationale for SDL-2010

- 11 years since SDL-2000
- SDL-2000 not fully implemented
- Impact of UML use
- Separation of concerns in standard documents
Reorganization

• Basic SDL-2010 - Z.101
  agent type diagrams containing agent instance structures with channels,
  state type diagrams and the associated semantics for these basic features.

• Comprehensive SDL-2010 - Z.102
  extends Basic SDL-2010 to full abstract grammar corresponding
  canonical concrete notation: includes features such as continuous
  signals, enabling conditions, type inheritance, and aggregate states.

• Shorthand notation & annotation in SDL-2010 - Z.103
  shorthands (agent diagrams, asterisk state) for easy and and more
  concise use; annotations (comments, create lines ...), aid
  understanding but do not add to the semantics.

• Data and action language in SDL-2010 - Z.104
  data types and expressions. SDL-2000 data notation or C with
  bindings to the abstract grammar and the Predefined data package.

• SDL-2010 combined with ASN.1 modules - Z.105
  As for SDL-2000 except Predefined extensions moved to Z.104.

• Common Interchange Format for SDL-2010 - Z.106
  Similar to SDL-2000 (updated for feature changes).
Basic SDL-2010

• Each diagram just one page!
• Lexical rules, frame use, page-numbers
• Framework, packages, referenced definitions
• Agent types, state types
• Typebased agents, procedures
• Channels, interfaces, signals
• State transition graphs:
  – start, state with input/save
• Transitions to:
  – nextstate, join, stop, return
• Actions:
  – tasks, create, proc. call, output, decision
• Timers, Data (overview)
Basic SDL-2010

USE Pkg

SYSTEM wom:Syst
/* system specification */

PACKAGE Pkg
/* package diagram */

SYSTEM Syst

SYSTEM TYPE Syst
/* system type diagram */

STATE TYPE Cst
/* state type diagram */
Comprehensive SDL-2010

• Completes the abstract grammar
  – Priority input, Enabling condition, Continuous signal
  – Spontaneous transitions
  – Composite state aggregation
  – Composite state named entry/exit, entry/exit procedures
  – Statement list in tasks/procedures/operations, loops

• Completes (not “syntactic sugar”) features
  – Inheritance (specialization)
    • Virtuality & Context parameters
  – Remote procedures/variables
  – Generic systems (select, optional transition)
  – Macros
  – Unicode handling
Shorthand SDL-2010 and Annotation

- Instance diagram
- Agent with state graph
- Asterisk input/save, implicit transition
- Signallist, interface as stimulus/on channel
- Asterisk state, multiple state appearance
- Multiple diagram pages
- Various syntax alternatives
- Create line (the only annotation left?)
System diagram (example)

```plaintext
SYSTEM wom                        1(1)

/* system diagram */

SIGNAL s;

s0
```

```plaintext
s0
```

```plaintext
s
```

```plaintext
s0
```
• Attempt to standardize a collective view
  – “… a graphical depiction of the relationships between <system specification> and <package diagram>s.”
Deleted: Nested diagrams
Deleted: Class symbols

- **st**
  - `i: Integer;
  - `b: Boolean;

- **PT**
  - `> s {via: gin}
  - `< t {via: gout}
Deleted: Associations

- Some concrete syntax rules to ensure UML-like
- But no meaning (in SDL-2000)
- And consistent with deleting class symbols
Deleted: Name class and spelling

- Removed as user feature
  - still used to explain Predefined data types

```plaintext
value type Digit = Character constants
'0','1','2','3','4','5','6','7','8','9' endsyntype;
value type Dstring
  inherits String < Digit > ( '' = emptystring )
adding
  operators ocs in nameclass
   ''' ( ('0':'9'))+ '''' ->this Dstring;
/*strings of digits '0' to '9', '00' to '99' ... */
operator ocs -> this Dstring
{result mkDs(substring(spelling(ocs),
  2,length(spelling(ocs)-2))};
operator private mkDs(d Charstring)->this Dstring
{result mkstring(d[1])/(if length(d)=1 then ''
  else mkDs(remove(d,1,1))};
endvalue type Digitstring;
```
Use with ASN.1 (Z.105)

```
USE MyASN1/INTERFACE MyMessages;

SYSTEM useASN1 1(5)
  /*implies the SDL-2010 interface*/

interface MyMessages
  {signal
    connectd ( Destination ),
    sendInfo ( Information ),
    disconnect ( ConnectRef );
  }
```

where

MyASN1
DEFINITIONS AUTOMATIC TAGS ::= BEGIN
  -- definitions including
  MyMessages ::= CHOICE {
    connectd Destination,
    sendInfo Information,
    disconnect ConnectRef}
  -- the data types mentioned
  -- above such as Destination
  -- and Information visible here.
END
Z.104 (10/2004) - implicit choice

For a path that has encoding, a data type is implicitly defined that corresponds to the SDL:

```plaintext
value type Implicitname /* an implicit and unique name */
{
  choice

  signal1 value
  {
    struct
    1 Sort11 optional;
    2 Sort12 optional;
    3 Sort13 optional;
    /* ... and so on for each parameter of signal1 */
  };

  signal2 value
  {
    struct
    1 Sort21 optional;
    2 Sort22 optional;
    3 Sort23 optional;
    /* ... and so on for each parameter of the signal2 */
  };

  signal3 NULL; /* no parameters */
  /* ... and so on for each signal */
}
```
Signal as Structure as Signal

A signal definition with parameters defines a structure data type that is a new <basic sort> alternative <as signal>

<as signal> ::= as signal <signal identifier>

signal memo ( Cs, Receipt); /* defines value type anon{struct anon1 Cs optional; anon2 Receipt optional};*/
dcl my_memo as signal memo /* anon */ , r Receipt;
   r := my_memo.2; /* .anon2 */
/* access by <field number> alternative to <field name> */
   <field number> ::= <integer name> */

signal memo2 ( c1 Cs, r2 Receipt); /* names fields for value type anon3 { struct c1 Cs optional; r2 Receipt optional};*/
value type Memo_struct { struct cs1 Cs; r3 Receipt};
signal memo3 ( struct Memo_struct); /* names fields for value type anon4 { struct cs1 Cs optional; r3 Receipt optional};*/
dcl my_memo2 as signal memo2, my_memo3 as signal memo3;
   my_memo2.c1 := my_memo3.cs1 /* field names */
Choice as interface, gate or channel

An interface, gate or channel definition defines a choice data type that is a new `<basic sort> alternative `<as interface>``, `<as gate>` or `<as channel>`:

- `<as interface>` ::= `as interface <signal identifier>`
- `<as gate>` ::= `as gate <gate identifier>`
- `<as channel>` ::= `as channel <channel identifier>`

```plaintext
interface pad{signal memo(Cs,Receipt),
  memo2(c1 Cs,r2 Receipt),
  memo3(struct Memo_struct);} /* defines

value type anon5 {choice
  memo as signal memo;
  memo2 as signal memo2;
  memo3 as signal memo3}; */

dcl m as interface pad /* anon5 */;
  r := if memoPresent(m) then m.memo.2 /* anon2 */
  else if memo2Present(m) then m.memo2.r2
  else r fi fi;
```

Similarly, for a gate or channel a choice data type for the signals carried.

Because an `<interface use list>` includes signals defined elsewhere, the name of signal is not necessarily unique, so it is also allowed to denote the field name of the choice by `<as signal>`, and to use Present(n) to test the presence if field n.

```plaintext
r := if Present(2) then m.as signal memo2.r2 else r fi;
```
Choice as gate or channel

A gate or channel definition defines a choice data type for all the signals carried by the channel in either direction.

The channel above defines

```c
/* value type anon7 {choice
    s1 as signal s1;
    s2 as signal s2;
    s3 as signal s3;
    s4 as signal s4};*/
```

dcl c1_choice as channel c1 /* anon7 */;

The choice types are useful in combination with input and output.

```
* in choice
  c1_choice
```

**Issue not resolved:** Variable is NOT transition local so still need to copy signal from input port to choice variable
Defining synonyms as variables

In SDL-2010 the concept of a synonym is redefined as a read-only variable. This was not possible before SDL-2000 because variables could not be defined anywhere.

Nothing is changed in the concrete grammar, but a synonym in SDL-2010 then has a Variable-definition in the abstract grammar that enables mapping of UML read-only attributes to synonyms when using Z.109.
Lower bound on number of instances

<number of instances> ::= ( [<initial number>] [, [<maximum number>] [, <lower bound> ] ] )

This makes it possible to specify the number of instances is static.

myprocess(2,2,2) /*always 2 */

Or for the minimum number to be stated.

myprocess(2,,1) /*never < 1*/

Attempting to stop an instance in a set at the lower bound raises OutOfRange, so to avoid this occurring a new active expression

active (myprocess)

gives the number of active instances in an agent set.
Only one input or save can contain $s$ via $g_1$ for the same state.
Only one input or save can contain $s$ (without a gate).
Assume signal $s$ is the next signal in the input queue.
In state $s_0$, if $s$ arrived via $g_1$ the next state is $s_1$. If $s$ did not arrive via $g_1$ next state is $s_2$.
In state $s_1$, if $s$ arrived via $g_2$ the next state is $s_2$.
If $s$ did not arrive via $g_2$, the signal remains in the input queue
(if this is the only transition from $s_1$, until a signal $s$ arrives via $g_2$).
In state $s_2$, if $s$ arrived via $g_3$ the signal remains in the input queue.
If $s$ did not arrive via $g_3$, the next state is $s_3$.
If there is no explicit input/save for $s$ without a gate, there is an implicit input for $s$ without a gate.
In a process (rather than a process type), the name of a channel attached to a gate can be used
for the via. In the implicit process type this is transformed to the gate connected to the channel.
Priority input value

<priority input list> ::=  
   <priority stimulus> {, <priority stimulus>}*

<priority stimulus> ::= 
   <stimulus> [ priority [ <priority name> ] ]

The change is to allow multiple levels of priority specified by a <priority name> (a natural number literal).
The higher the number the lower the priority, with zero the highest priority (consistent with Continuous signal priority).

An omitted <priority name> is one greater than the highest explicit <priority name>.
Assume signals X, Y, Z, N each with one parameter saved in x, y, z, n resp.. Assume Tx, Ty, Tz and Tn connect to transitions that terminate in state st1. Assume st1 reached & input port contains in arrival order signals N, Y, Z, X. Z input for has an implied Priority-name of 3.

The signal Y is enabled and consumed - has highest priority and Ty is taken. If no Y signals have arrived, at st1 again signal X is enabled and consumed. If no X or Y signals have arrived, at st1 again signal Z is enabled and consumed. If no X, Y or Z in the input port in state st1, Signal N is enabled and consumed. If the inputs for two or more signals have the same input priority, the signal that arrived first is consumed.

Priority input value example

Assume signals X, Y, Z, N each with one parameter saved in x, y, z, n resp.. Assume Tx, Ty, Tz and Tn connect to transitions that terminate in state st1. Assume st1 reached & input port contains in arrival order signals N, Y, Z, X. Z input for has an implied Priority-name of 3.

The signal Y is enabled and consumed - has highest priority and Ty is taken. If no Y signals have arrived, at st1 again signal X is enabled and consumed. If no X or Y signals have arrived, at st1 again signal Z is enabled and consumed. If no X, Y or Z in the input port in state st1, Signal N is enabled and consumed. If the inputs for two or more signals have the same input priority, the signal that arrived first is consumed.
Using signallist as an interface

• In SDL-2010

Every **signallist** definition is a shorthand for an interface definition.

    signallist sl := s, t, u, v;

Defines an interface definition sl containing signals s, t, u and v.

The notation “(sl)” has the same meaning as “interface sl”

When used on a channel or gate or in another signallist, the signallist name stands for the list of signals of the interface (as in SDL-2000).
<state timer> ::= 

    state timer <Time expression> | set <set clause>

For an optional State-timer, the timer is set entering the state and reset entering a Transition, except for an empty Transition to the state. Timer expiry: the timer signal is consumed + Transition of the State-timer taken.
Pending: Specified delay

Output-node ::= Signal-identifier Actual-parameters
[Signal-destination] Direct-via
 Activation-delay Signal-priority

<output body item> ::= <signal identifier> [<actual parameters>]
[ <activation delay> [ <signal priority> ]]

<activation delay> ::= active <Duration expression>
<signal priority> ::= priority <Natural expression>

If <activation delay> is omitted, the Activation-delay is zero: that is, there is no delay in activating the signal at the destination.

If <signal priority> is omitted, the Signal-priority is zero: that is, the signal has the highest signal priority.

Conveyed "availability time" = now + Activation-delay - sent only if Activation-delay +ve

Agent

If a signal conveys an availability time, the signal is not delivered to the input port until this time has been reached.

The set of retained signals is ordered in the input port according to their availability time, which for signals that do not convey an availability time is the same as their arrival time.
Issues: pids and instance sets

Issue: How to handle the pid values of agent instances.
Option: if an agent instance set is named ais, treat the expression
\[ ais[k] \]
as indexing a string of pid values for the ais instances.
But if \( k \) is an integer index, may be too simplistic as the number of instances is potentially variable, so ais[2] may not be constant.
Could have \( k \) as a “key value”, but then need some way of defining keys.

Issue: Initializing static agent instance sets.
Not currently allowed to give any agent parameters in the set definition. In any case, each instance probably needs different parameters. Before system execution so dynamic evaluation is not possible. There may be a relationship between keys (see above) and parameters.
Deleted: **object** data types

- **Complex**
  - Not implemented (as in standard)
  - Implicit conversions value/object
  - Dynamic data types
  - Not supported by ASF 2.1 or encoding rules

- **Strategy**
  - Remove from initial SDL-2010, but further study on extending Z.104 is in progress.
ITU maintenance guidelines (extract)

Study group experts should determine the level of support and opposition for each change and evaluate reactions from users. A change will only be put on the accepted list of changes if there is substantial user support and no serious objections to the proposal from more than just a few users. Finally, all accepted changes will be incorporated into a revised ITU-T Rec. Z.100. Users should be aware that until changes have been incorporated and approved by the Study Group responsible for ITU-T Rec. Z.100 they are not recommended by ITU-T.
Status

Z.100 (Introduction) needs finishing
Z.101 to Z.105 ‘almost complete’ need review.
Z.106 needs to be revised
Z.1xx Object data - in progress - Feb 2012
Z.109 Work in progress for Feb 201