# Joint ITU-T and SDL Forum Society workshop: ITU System Design Languages

## The SDL-2008 language revision

**Rick Reed**
**TSE Ltd**

**Rapporteur Q.11/17**



## Why SDL-2008?

- Living Language - evolution
  - SDL-76, SDL-84, SDL-88, SDL-92, SDL-2000
- Why not UML 2.x
  - Not adequate for executable models
  - Syntax and semantic variations
- Why not SDL-2000
  - Not fully implemented
  - Over complex
  - Outstanding feature requests
  - UML 2 support



## Evolution of SDL

- SDL-76 Basic process graphics
- SDL-84 Interaction and processes
- SDL-88 Formally defined
- SDL-92 Object orientation (types)
- SDL-2000 Type based + data extended
  - UML 1.4 support with Z.109 (11/99)
- SDL-2008 Simplified & improved (?)
  - Z.109 (06/07) rewritten for UML 2.x



## UML2 completeness

To use the UML2 Superstructure needs

- The notation to be fully defined;
- Binding of notation to the metamodel;
- Binding of semantic variation points.
  - A “Semantic Variation Point” section explicitly identifies the areas where the semantics are intentionally under specified to provide leeway for domain-specific refinements of the general UML semantics (e.g., by using stereotypes and profiles).

UML2 has various levels of compliance

## UML2 BNF (example)

<multiplicity-range> ::= [ <lower> '..' ] <upper>
<lower> ::= <integer> | <value-specification>
<upper> ::= '*' | <value-specification>

But <value-specification> is not defined.
Could be Expression | OpaqueExpression
Expression:
"special notations permitted, including infix"
OpaqueExpression:
"text strings in particular languages"



## Presentation Options

UML2: *Concrete syntax compliance does not require compliance to any presentation options*

- Tools may omit or use by default
- Portability assured by XMI support
- Tools often have other presentations
- Recognizably the same model?



## UML2 semantic variation

Some examples of the many semantic variations:

- Compatibility of Redefined & Redefining elements
- Determining method invoked by call operation
- Ordering of the events in the input pool

Many (not all) associated with action semantics
The variation points are resolved by:

- Using a particular tool (in a particular configuration)
- Applying a profile that binds the variations

Execution requires the variations to be bound



## UML action language

- Concrete syntax from outside UML
- Binding defines how objects behave
- Libraries from 'host' language
- Executable UML and SDL-2000*
- Z.109 = SDL-2000* action semantics
  - to be updated to SDL-2008

## Objectives for SDL-2008

- Better alignment with UML 2
- Clearly identified levels:
  basic, comprehensive, extra syntax + annotation
- Include missed requirements
- Flexible data notation (native, C, Java …)
- Simplify (where possible)
- Exclude unused features
- Keep 'backwards compatible'
  Tool/language reference alignment



## Alignment with UML

- Unicode names
- Lower bound on agent sets
- Signals for remotes on gates
- Input via a specific gate
- Time supervised states
- Synonym as read only variable
- Abstract grammar for loops



## Restructuring the Recs.

| SDL-2000 | SDL-2008 |
|---|---|
| Z.100 Main language | Z.100 Overview |
| *Z.101 not used* | Z.101 Basic |
| *Z.102 not used* | Z.102 Comprehensive |
| *Z.103 not used* | Z.103 Shorthand & ann. |
| Z.104 Data encoding | Z.104 Data |
| Z.105 … with ASN.1 | Z.105 … with ASN.1 |
| Z.106 CIF (incl. SDL/PR) | Z.106 (incl. SDL/PR) |
| Z.107 embedded ASN.1 | *Z.107 not used* |
| *Z.108 not used* | *Z.108 not used* |
| Z.109 … with UML | Z.109 … with UML |



## Z.101 Basic SDL-2008

- Core features
  - Lexical rules and framework
  - Type diagrams (block, process, state, procedure, operation)
  - Structure (typebased agents, gates, channels)
  - Behaviour (signals, variables, timers, start, states with inputs and saves, transitions, decision, task, output, create, procedure call, return)
  - Basic data (variables, assignments, expressions, operators, NOW, enumerated, structures, choices, pid & pid expressions, **syntype**s)
- Basis
  - Includes abstract syntax of most (not all) features
  - Canonical concrete syntax excluding shorthand/alternates

## Z.102 Comprehensive SDL-2008

- Everything from SDL-2000 Z.100(11/07):
  - Not deleted and not in Z.101 or Z.104
  - Not a shorthand or annotation (Z.103)
- Canonical syntax for additional features
  - Specialization (**inherits**), context parameters, remote procedures & variables, state aggregation, priority input, enabling condition, **none,** compound statements, synonyms and generic systems, macros.



## Z.103 Shorthand notation and annotation in SDL-2008

- Shorthands
  - Multiple page diagrams, Text extension
  - Multiple names in state/input, Multiple occurrences
  - Agent/state diagrams (*implicit agent/state type*)
  - Implicit transition, Asterisk state/input etc.
  - Block or system variables
  - Statement lists, textual procedure/operation definitions
  - Legacy data type syntax (**newtype**)
- Annotation
  - Comments, notes
  - Associations, Create symbol, Package dependency



## Z.104 Data and action language in SDL-2008

- Merger of Z.104 and Data clause of Z.100(11/07)
- Plus **package** `Predefined`
  - including parts from Z.104 and Z.105
- Excluding **nameclass**, **spelling**
  - Annex A defines constructs restricted to **package** `Predefined` **nameclass** and **spelling** -> annex A (used in `Predefined`)
- Z.104 for SDL-2008 concrete syntax
  - For declaration/assignment/expression needs refining to permit alternative data notations to be used -
  - So far only SDL-2000 syntax is allowed.
  - Abstract grammar and semantics defined by Z.104



## Status

- Draft of Z.100
- Reasonable draft of Z.101
- Z.102/Z.103 - work in progress
- Z.104 initial merge - needs more work
- Z.105 from old version - needs revision
- Z.106 no doc. - refs need updating
  - CIF for new/deleted features needs adding/deleting resp.
- Z.109 no doc. - min. update refs.
  - Z.109 can be improved to use SDL-2008

## What is deleted?

- Nested diagrams
  Only referenced diagrams (as in tools)
- Exception handling
  Defined exceptions -> behave in undefined way
- nameclass/spelling
- UML-like references (? use Z.109 ?)
- Associations (?)
- Visibility restriction (?)
- **object** data types (?)

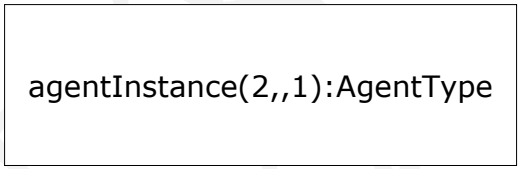


## What is (is to be) added

- UML alignment items
  Unicode, agent lower bound, remotes on gates, input **via**, time supervised states, **synonym** as read-only, loop abstract grammar.
- Alignment signals/structures
  signal definition as a structure type
  interface implies a choice type
- Access to last signal in transition



## Agent lower bound

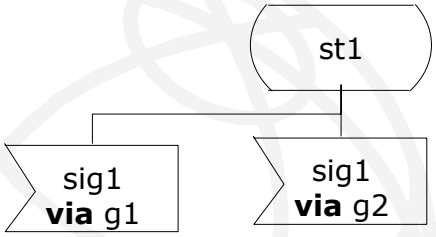



An agent instance with two initial instances, no maximum bound and a lower bound of 1.

If the maximum and lower bound are the same instances can neither be created or stopped.



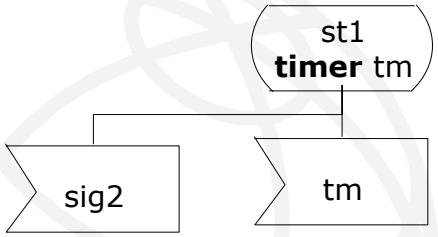
## Input via




Each signal/gate combination is treated as a separate stimulus.

If the **via** is omitted this means all gates.

# Timer supervised state




The timer tm is set on each entry to state st1 and reset on any exit from st1. The timer has to have a default value, or can be given one in the state. If there are multiple occurrences of st1 the timer can be omitted on some, but if given shall be the same.



# Signal as structure

**signal** s2 (Integer, Boolean);
implies
**structure** implied_name ( 1 Integer **optional**,
2 Boolean **optional**);

This can be used as a sort, for example
**dcl** vs **as signal** s2;
/***signal** optional if s2 suffices*/

Signal list on channel, gate, or defined by an interface implies a **choice** each elements of which has the name of one of the signals and has as the element sort the implied '**as signal**' sort.

This is consistent with the treatment of signal lists in the current Z.104.



# Access to the last signal

In an expression
**signal**
denotes a **choice** value for the signal from the last signal consumed. The sort of the value is the implied **choice** for the interface of the enclosing agent.

**signal**.sig1
therefore denotes the signal structure value if sig1 was received, and

**signal**.sig1.2
denotes the second parameter of the signal sig1.

The last signal is usually known from the input.

PresentExtract(**signal**)
otherwise gives the element (signal) name for use in a decision, or the Boolean expression

sig1Present(**signal**)
checks specifically for the choice element being a sig1.



# Discussion

- Is this going in the right direction?
- Are there other items to add/delete?
- Standard as a subset of all tools or include all features of all tools?
- Who is willing to complete the work?