Tutorial: SDL-2010

Rick Reed

TSE

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

Rick Reed has been involved with ITU working on language standards since 1979. At that time he was working with and represented GEC Telecommunications (later GPT, then Marconi Communications, finally taken over by Ericsson), which he had joined as student apprentice before going to university in 1967. At first the ITU involvement was with the CHILL programming language related to his responsibility for software development tools and including his own work on a compiler, but from the mid-1980's his primary area of interest in ITU languages became the Z.100 Specification and Description Language. For this language Rick initially made a major contribution to the data model of SDL-88.

In 1986 Rick became the technical project coordinator from a large EU project SPECS involving various telecommunications companies including Alcatel, France Telecom, GPT, IBM and Philips concerned with the specification and programming environment needed for communication software. This also involved collaboration with other European projects in related software areas such as the required offline support platform and the online run-time environment. It is largely as a result of this EU work that SDL-92 was created, making the language more object oriented. The SPECS project continued into 1993. During this period Rick left GPT in 1991, and formed TSE to sell his services as an independent consultant using his experience from GPT. He continued in the key role in SPECS, including acting as the main editor for a book giving an overview of the SPEC project results published by North-Holland.

In the late 1980's, as well as representing his employer at ITU, Rick took on the role of being head of the UK representation in the ITU Software languages and applications Study Group 10. After forming TSE in 1991, he continued to represent the UK in this area. Subsequent to the merger of Study Group 10 and Study Group 17 in 2001, this role included the studies previously under Study Group 7: Data networks and open system communications (including the currently important topic of system security).

During 1996 to 2000 Rick managed and contributed to the development of the SDL-2000. From 2000 he contributed to and edited the Z.104, Encoding of SDL data, and Z.109, SDL-2000 combined with UML. He was ITU rapporteur for Q.11/17:Specification and Implementation Languages [2005-2008] and is now ITU rapporteur for Q.13/17:Formal languages and telecommunication software [2009-2012]. As part of the previous Q.11/17 and current Q.13/17 he contributes to and edits the standards for SDL-2010: a revision of SDL-2000.

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

The ITU Specification and Description Language was first standardized in 1976. Up to 1992 it was revised four-yearly, matching rapid advances in computer technology and the telecommunications industry. After SDL-92, the next major change was SDL-2000, which is still the current standard. A new version has been under development with an initial target of producing a version called SDL-2008. This is now renamed SDL-2010 and should be ready for the ITU approval process in December.

This session is in two parts:

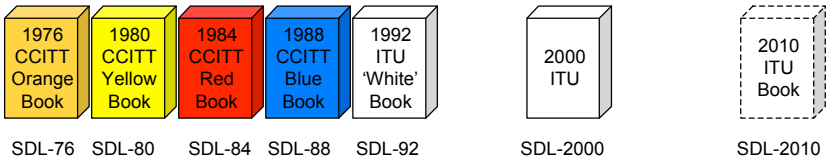
1. The differences between SDL-2000 and SDL-2010 will be presented.

   The changes range from extending the character set for names to full Unicode to the removal of features such as exception handling. Some new features are added such as input via, a specific gate enabling the signals received on different gates to be distinguished. The standards are also reorganized (and to some extent rewritten) to more clearly separate

   - the core language features in Basic SDL-2010,
   - additional concepts which with Basic SDL-2010 provide Comprehensive SDL-2010,
   - shorthand notations and annotations that make the language more practical to use,
   - data and the action language,
   - the use of SDL-2010 with ASN.1 modules.

2. After the presentation there will be a discussion on SDL-2010, and what the plan might be for further language changes.

Rationale for SDL-2010



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

The Specification and Description Language has been updated several times since it was first standardized in 1976. The differences between SDL-88 and SDL-92, and SDL-92 and SDL-2000 are document in the standard. Since the year 2000 there have been some maintenance updates but the standardized language has remained essentially unchanged.

When SDL-2000 was being developed, Telelogic and Verilog were promising to produce tools in 2000 or 2001. Telelogic acquired Verilog shortly afterwards removing competition, and a tool supporting SDL-2000 has never been produced. Support exists today in tools from PragmaDev, IBM (having acquired Telelogic) and Cinderella forr SDL-92 and some SDL-2000 features, and SDL-2000 support is no longer planned. One reason for SDL-2010 is to narrow the gap between tools and the standard.

The use of UML has changed since 2000, and there is no real benefit in UML class modeling features in the SDL-2000 language: these can be handled by UML and importing the UML 2.0 models according to the UML profile in Z.109. However to facilitate this a few language features should be refined. It is anticipated that tools will follow suit so that UML models can be integrated with SDL-2010 models.

The SDL-2000 language is considered rather large, but the essential language that captures “SDL-ness” is actually much smaller. In SDL-2010 there is a return to organizing the standard into a basic part with additional parts that build on the basic language to provide the additional, shorthand, annotation, detailed data model of the full language, support for ASN.1, and the common interchange format. In this way the different concerns are separated into different 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 - Z.101: contains the part of the Specification and Description Language Recommendations for SDL-2010, that covers core features such as agent (block, process) type diagrams containing agent instance structures with channels, diagrams for extended finite state types and the associated semantics for these basic features.

Comprehensive SDL-2010 - Z.102: contains a part of the Specification and Description Language Recommendations for SDL-2010 that extends the semantics and syntax of the Basic language in Rec. Z.101 to cover the full abstract grammar and the corresponding canonical concrete notation. This includes features such as continuous signals, enabling conditions, type inheritance, and composite states.

Shorthand notation and annotation in SDL-2010 - Z.103: contains the part of the Specification and Description Language Recommendations for SDL-2010 that adds shorthand notations (such as asterisk state) that make the language easier to use and more concise, and various annotations that make models easier to understand (such as comments or create lines), but do not add to the formal semantics of the models. Models transform shorthand notations from the concrete syntax of Rec. Z.103 into concrete syntax of Rec. Z.102 or Rec. Z.101.

Data and action language in SDL-2010 - Z.104: contains the part of the Specification and Description Language Recommendations for SDL-2010 that adds the data and action language used to define data types and expressions. In SDL-2010 it is allowed to use different concrete data notations, such as the {SDL-2000} data notation or C with bindings to the abstract grammar and the Predefined data package.

The underlying data model is fundamental to behaviour and provides sorts of data such as Boolean and Integer that are used in other language features. For that reason this underlying model and an overview of predefined data sorts and constructs is given in Z.100 annex D.

SDL-2010 combined with ASN.1 modules - Z.105: provides a mapping for ASN.1 modules to features defined in rest of the Specification and Description Language Recommendations for SDL-2010, so that the ASN.1 modules define data items that can be used with the rest of SDL-2010.

Common Interchange Format for SDL-2010 - Z.106: provides alternative textual syntax for the graphical syntax items defined in Z.101 to Z.105 that can be used as a Common Interchange Format (CIF) between SDL-2010 tools. The basic level of CIF provides only a textual equivalent of graphical items. The full CIF is intended for the interchange of graphical SDL-2010 specifications (SDL-GR) so that the drawings are recognizably the same.

Basic SDL-2010

- Each diagram is one page!
- No shorthand (incl. agent diagrams!)
- No inheritance or context parameters
- No remote procedures/variables
- No priority input or enabling condition
- No continuous signals
- No macros
- No synonyms
- No select or optional transitions

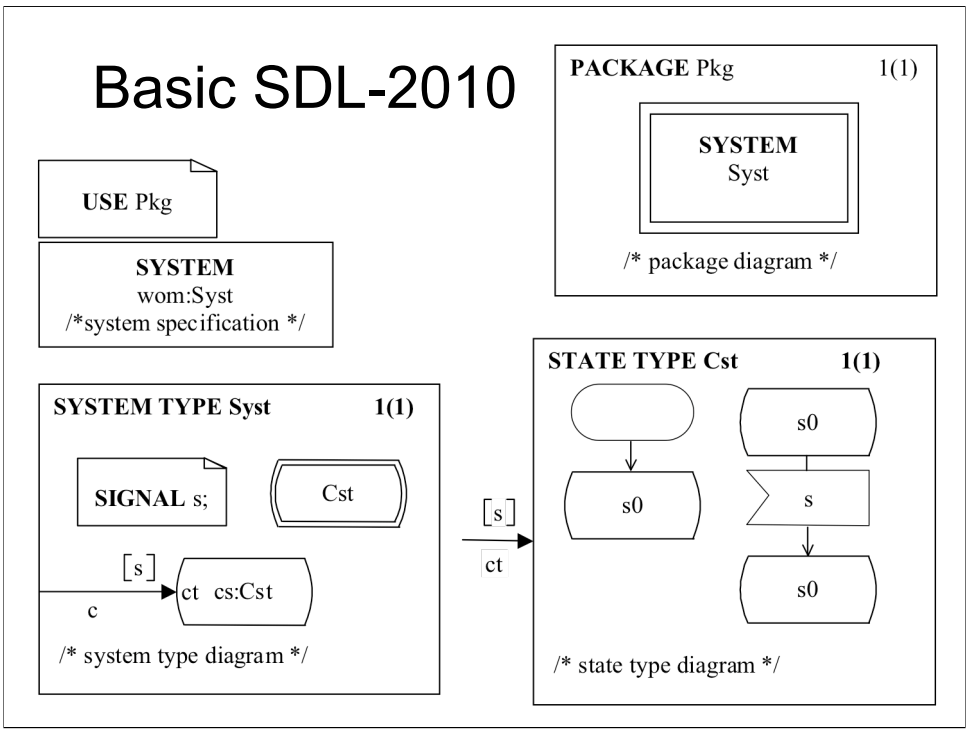
Basic SDL-2010 allows modeling of systems based on type definitions. Agent instances have to be type based instances, and the state machine of an agent has to be a type based state. Agent diagrams are a shorthand for an agent based on an anonymous agent type and are therefore defined in Z.103. Similarly a state graph in an agent type is a shorthand defined in Z.103 for a state machine that is a type based instance of a state type. This is not changed from SDL-2000, except the transformations are moved to Z.103.

The page structure is made part of the concrete syntax of diagrams with the number of pages being restricted to one per diagram in Basic SDL-2010, and additional pages being a shorthand (for one large diagram) in Z.103.

Most of the abstract grammar of SDL-2010 is covered by Basic SDL-2010. The abstract grammar that is not covered is for specialization (also known as inheritance) of types, the additional types of stimulus for an input (priority input, enabling condition, continuous signal, remote procedures and remote variables), synonyms and the generic definition features. These and macros are covered in Z.102.

Basic SDL-2010 includes the abstract grammar and basic syntax for data, so that expressions and assignments can be used, but Z.104 contains the full definition of predefined data, detailed semantics and alternative concrete syntax for data features.

Many of the frequently used language features (such as asterisk input) are shorthand notations, that are transformed from concrete syntax to concrete syntax. These models are given in Z.103 together with additional annotation features.



In SDL-2010 references to diagrams are the canonical concrete form. In the concrete syntax nested diagrams are not allowed (and were not supported by tools in any case).

In the example an <sdl specification> is a <system specification> is a <typebased agent definition> is a <typebased system definition> associated with a <package use area>. The system symbol references the system type Syst defined in package Pkg.

In package diagram Pkg, the system type symbol references the system type Syst.

In the system type diagram Syst, the state symbol represents the state machine of the based on the (composite) state type Cst. In Basic SDL-2010 an agent type (system, block or process) never contains a state graph, but only contains other (typebased) agents and (if the agent type has a state machine) a typebased state for the state machine. In this case the (composite) state type of the state machine is referenced locally by the state type symbol Cst. The referenced state type Cst gives the state graph.

This is not really changed from SDL-2000, but the transformation from agent diagrams to agent references using implicit agent types, and a state graph at the agent level to references to an implicit state type for the state machine of the agent containing for the graph was hidden in the general description of agents. In SDL-2010 the usual more concise forms are not defined in Basic SDL-2010, and the transformations given in Z.103 result in Basic SDL-2010.

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

Most of the abstract grammar is in Basic SDL-2010. For example, the abstract syntax includes parent identifiers for specialization, but the concrete syntax, rules and constraints are omitted. The abstract grammar introduced in Comprehensive SDL-2010 is for a small number of features not essential for most models. In the case of composite states, it was considered not to include these for Basic SDL-2010, but state machines are defined by (composite) state types so state types are essential. On the other hand the more complex scenario of a state aggregation type that contains instances of other state types is excluded in Basic SDL-2010 so is included in Comprehensive SDL-2010.

The most important addition in Comprehensive SDL-2010 is specialization with the linked features of virtuality and context parameters (without and with constraints) . These features do not (currently) have abstract syntax, but the rules for specialization are described under the heading *Semantics*. The concrete syntax for specialization, virtuality and context parameters is added in Comprehensive SDL-2010.

Remote procedures and remote variables are currently explained by transformation models, and therefore arguably could be considered as “shorthand” features. Though the description is given this way, it is expected that the remote features are treated as (and perhaps implemented as) remote calls of the underlying system. It has been discussed to replace the transformation model with abstract grammar concepts, but there has been a reluctance to replace the well-defined models with a natural language description on abstract syntax. It should be noted that variables and remote procedures are the last transformations to be done according to Annex F.

Generic system description and macros are similar in that they both carried out on the system model before the fully validity and meaning of the resulting system is considered. They are therefore not the same as “shorthand” notations.

The Unicode extension to SDL-2010 is placed here as it is not essential.

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?)

The transformation of agent diagrams and agents with a state graph is best illustrated by the example.

The concept is that annotation is removed and the remaining shorthand notations are transformed into canonical concrete syntax of Basic SDL-2010 or Comprehensive SDL-2010. If the Comprehensive SDL-2010 is not itself canonical, it will be further transformed into canonical concrete syntax. The abstract grammar is then derived.

The asterisk input/save and implicit transition avoid the user having to list every signal for every state.

Signallist or an interface identifier can also be used as a shorthand for the signals identified.

The asterisk state saves writing a transition for all states. Very useful when combined with asterisk save.

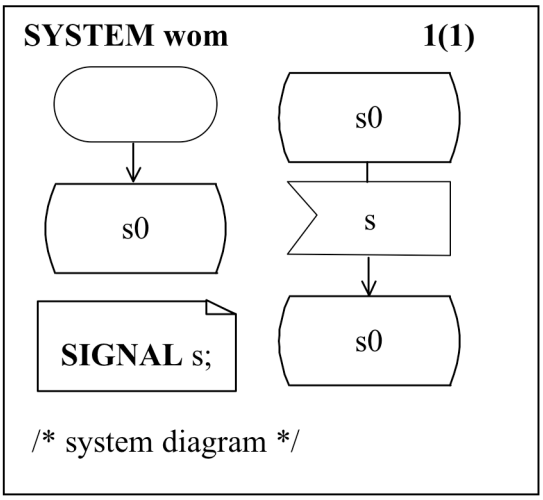
Multiple state “shorthand” allows the same state name to appear in several state symbols, and therefore allows the user to organize a model by signals and/or transition actions rather than states.

There are various other syntax alternatives (such as the internal input, set/reset timer symbols, interface gate …). These are added as Shorthand SDL-2010.

The concrete syntax for multiple diagram pages is given, which transforms to a single page.

Create line appears to be the only annotation remaining if associations and package dependencies are removed from the standard.

System diagram (example)



This is the system diagram for the previous example in Basic SDL-2010.

The state graph is transformed into a state type, that is instantiated as a typebased state machine with a gate added for the incoming signal.

The diagram is then transformed into a (system) agent type in an implied package, and a typebased system instance that uses the implied package.

It is obvious for that in this case it is easier for the user to specify it as above. However, the semantics really are as given previously.

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.

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.

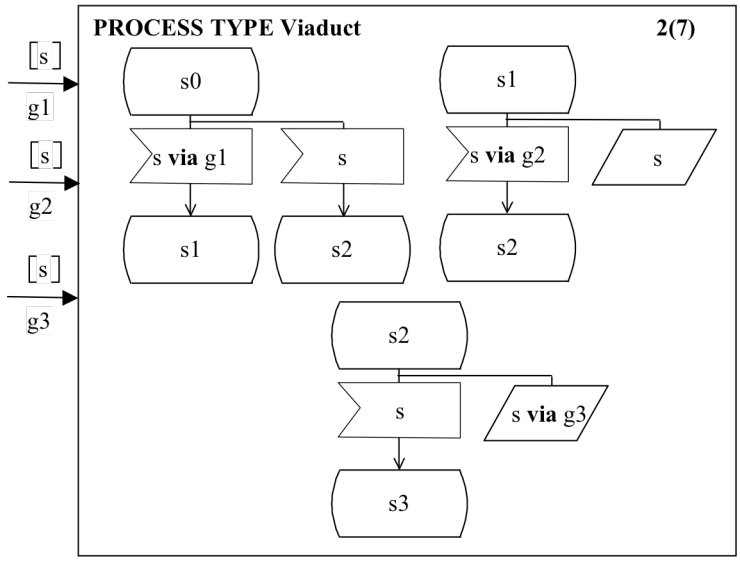
```
myprocess(2,,1) /*never < 1*/
```

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.

Input via



Assume a signal s is the next signal in the input queue.

In state s0, if s arrived via g1 the next state is s1. If s did not arrive via g1 the next state is s2. Only one input or save can contain s via g1 for the same state. Only one input or save can contain s (without a gate).

In state s1, if s arrived via g2 the next state is s2. If s did not arrive via g2, the signal remains in the input queue (if these are the only transitions from s1, until a signal s arrives via g2).

In state s2, if a arrived via g3 the signal remains in the input queue. If s did not arrive via g3, the next state is s3.

If there is explicit no input or save for s without a gate, there is still an implicit input for s without a gate back to the same state.

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

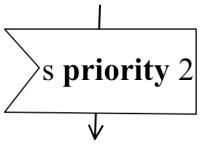
<priority input list> ::=

<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 higher the priority. Omitting the <priority name> is the same as 0.

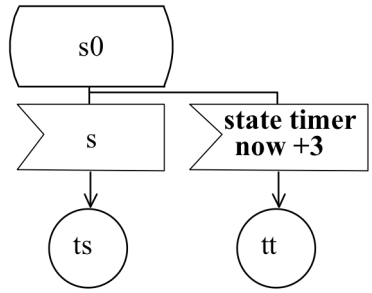


Currently only one level of priority. The change allows multiple levels of priority.

If the input port contains signals matching any of the inputs with the highest level of priority for the current state, the first such signal is consumed. Otherwise, this is repeated for each of the next higher levels of priority in turn until either a signal is consumed for a priority input or there no signal for a priority input.

A priority input without a gate, takes precedence over a lower priority input with a gate or an input with a gate without priority.

Timer supervised states



<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.

State-node :: *State-name Save-signalset Input-node-set*

Spontaneous-transition-set Continuous-signal-set

Connect-node-set [*Composite-state-type-identifier*]

[*State-timer*]

State-timer ::

Time-expression Timer-identifier Expression Transition*

If a state has a *State-timer*, the timer is set when the state is entered and reset when a *Transition* is entered from the state, except for an empty *Transition* that leads back to the same state (no *Graph-node* and terminates in a *Nextstate-node* for the same state). If the timer expires while in the state, the timer signal is consumed and the *Transition* of the *State-timer* is taken. The *Expression* list is a for a timer with sorts.

<state timer association area> ::=

<solid association symbol> **is connected to** <state timer area>

<state time area> ::=

<plain input symbol> **contains** { <virtuality> <state timer> }

is followed by <transition area>

<state timer> ::=

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

<Time expression> uses an implicit timer for the state, whereas **set** clause uses a defined timer which can have a default duration.

If the state is a composite state, which the timer expiration is treated in the same way as signal causing an exit from the composite state.

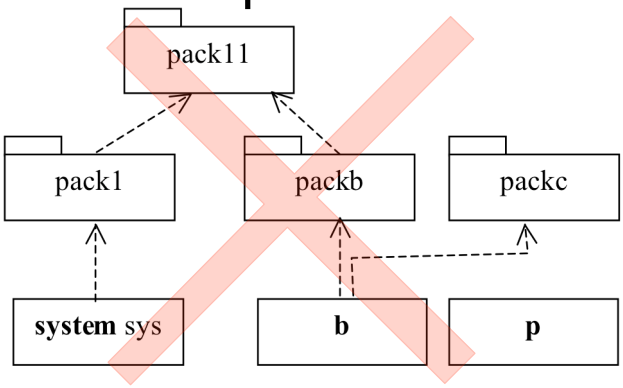
Defining synonyms as variables

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.

Deleted: Specification area



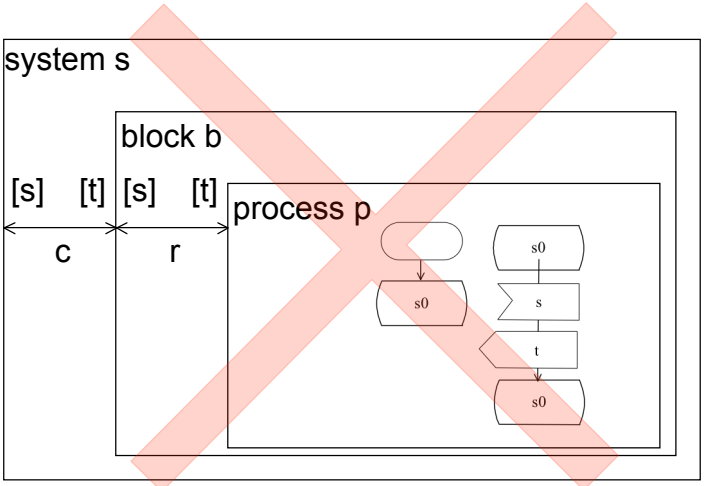
- Attempt to standardize a collective view
  - *"… a graphical depiction of the relationships between <system specification> and <package diagram>s."*

Every tool supports something like this - but not in a standard way. Cinderella does something like the standard.

There is no need for this to be standardized.

As a corollary package dependency (which in any case had to be consistent with package use) is deleted.

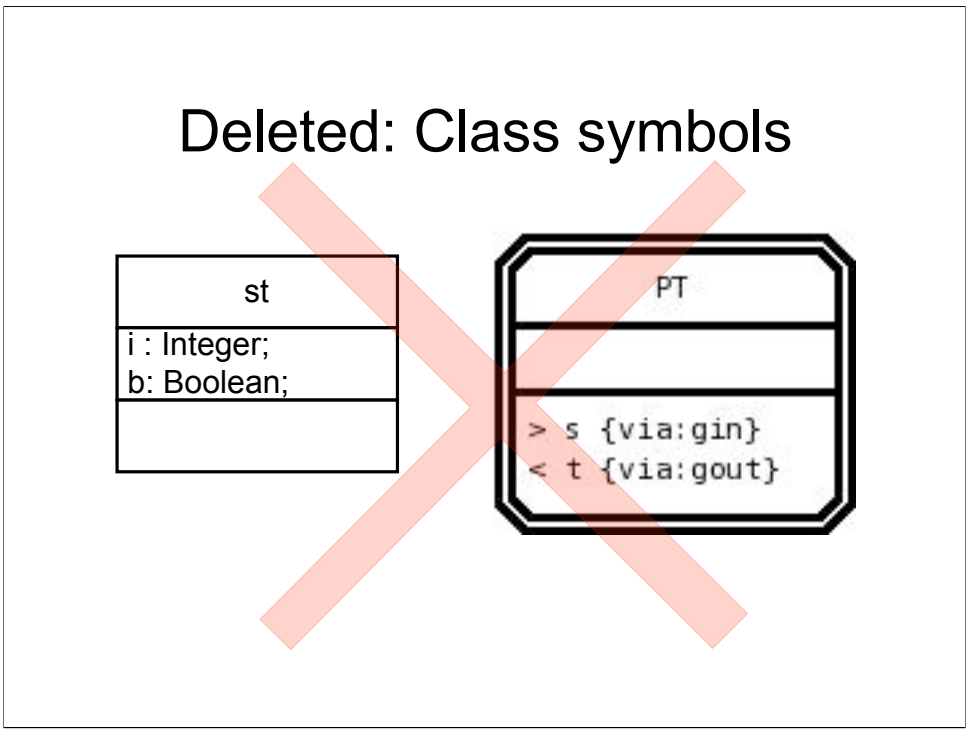
Deleted: Nested diagrams



Not practical. Not supported. Just complicated the language definition.

No longer needed for language definition.

The transformation from referenced items to the hierarchical abstract model is handled when going from the concrete to the abstract syntax (rather than concrete to concrete syntax).



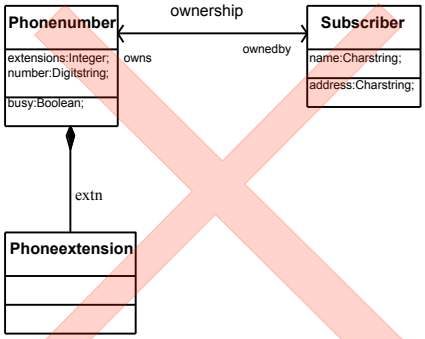
These have had limited support in tools. Each tool uses them differently.

In SDL Suite they can be used for modeling data types invoking non-standard reference computing.

In RTDS they are used to define gates (rather than put them on diagrams).

They are not supported in Cinderella 1.3 - the latest version (which in fact supports very little of SDL-2000).

Deleted: Associations



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

These were in any case just annotation as far as SDL-2000 was concerned, though there are some concrete syntax rules to restrict associations to the equivalent UML associations.

It was required that deleing associations did not change the meaning of the SDL-2000 (as far as defined by the standard).

Associations were only allowed between type references for agent types, data types and interfaces

Deleted: Name class and spelling

```
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;
```

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

Deleted: **object** data types

- Complex
  - Not implemented (as in standard)
  - Implicit conversions value/object
  - Dynamic data types
  - Not supported by ASN.1 or encoding rules
- Strategy
  - Remove, but consider adding
    Ref and/or Own + ORef as provided by Telelogic
    (but without implicit conversions)

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.

Discussion issues

- Are deleted features wanted?
- What (else) can be deleted?
- What is (still) missing?
- What is expected from tools?
- Formal definition
- Meta-language

Deletion: Features that have been deleted are either not implemented in tools (and therefore cannot really be used), or are implemented differently from SDL-2000, or are not (though to be) used. Have the right choices been made? Are there more features that can be deleted, as it is not be too late to delete features – it is much easier to remove rather than add features.

Missing features: New features can be considered, but should have contributions to ITU-T. SDL Forum Society members can participate in the ITU-T work.

Tool support: There are a set of compliance rules for tools. Every feature does not have to be supported by every tool, but obviously that is desirable. Also there is no benefit in a feature that is unlikely ever to be supported by a tool.

Formal definition: The current plan is that no formal definition is provided for SDL-2010, due to a lack of resources to modify the existing model or generate a new one. Instead the published Z.100 Annex F for SDL-2000 is referenced. It is therefore noted that the formal definition is out of date, but in combination with the obsolete 2007 version of Z.100 (for SDL-2000) provides a more formal definition for SDL-2000 than currently available for SDL-2010. Most of SDL-2010 is intended to be unchanged from SDL–2000, therefore Annex F to Z.100 with the obsolete 2007 version of Z.100 provides more detail than Z.100 to Z.106 for SDL-2010. If there is an inconsistency between Annex F to Z.100 for SDL-2000 and other parts of Z.100 to Z.106 for SDL-2010, it is either because there is an error in Z.100 to Z.106 or because there is a specific change to SDL-2010 compared with SDL-2000. If a change from SDL-2000 is not documented in Z.100 to Z.106, further study is needed to determine if the inconsistency is an error or intended. If work is done to replace the formal definition, alternatives are to update the existing model or completely replace it with a new one, in which case a different approach (such as metamodeling) might be considered.

Meta-languages: For further work on SDL-2010, extensions to the meta-grammar used from Z.111 may be beneficial, and an update Z.111 considered.