SDL-2010: Background, Rationale, and Survey

Rick Reed

TSE, The Laurels, Victoria Road, Windermere, LA23 2DL,United Kingdom.
rickreed@tseng.co.uk

Abstract. This invited paper concerns a revised version of the ITU-T Specification and Description Language standard, which is scheduled to be consented for approval by ITU-T during 2011. In this document and ongoing ITU-T work, the revised version is called SDL-2010. The current standardized (or in ITU-T terminology Recommended) version at the time of initially writing this paper (April 2011) was called SDL-2000. The paper gives some historical background on the development of the language. The paper includes rationale for the update of the language and the revised organization of the language standard. After the history, there is a description of the new organization followed by some details of the changed feature set of a revised version SDL-2010 compared with SDL-2000. The paper concludes with a snapshot of the status of the SDL-2010 standard.

The purpose of this paper is to provide an overview introduction to the revision of the ITU-T Specification and Description Language called SDL-2010.[1]

The Specification and Description Language has a long history as a published standard going back to 1976 [1], but other than corrections and some re-organization of material in 2002 [2], the language standard has remained stable since the publication of SDL-2000 in 1999 [3]. Considering that previously the language had been revised in 1980, 1984, 1988, 1992 and an addendum to the 1992 version in 1996 [4–8], this is the longest period of stability in the evolution of the language, as can be seen in Fig. 1. As explained in Sect. 1.1, this stability is to some extent due to changed market circumstances in the last decade. One incentive for a revised version of the language is to remove some differences between the language standard and implementations provided by tools: in particular some of the aspects of data introduced in SDL-2000 that have not been implemented (see Sect. 1.2), nesting of diagrams (see Sect. 1.3), and features without SDL-2000 semantics from the United Modeling Language (see Sect. 1.4). The resulting feature set is outlined in Sect. 1.5.

The organization of the standard documents is described in Sect. 2, followed by more detailed description in Sect. 3 of some features added or deleted. Finally the paper closes with the status at the time of writing in sect.4.

[1] The content of this paper is not entirely new because it reports work that has been in progress for some time in the domain of ITU-T standardization, but although some material has been made available as ITU-T temporary documents and some material presented at SAM-2010, it has not been widely published.

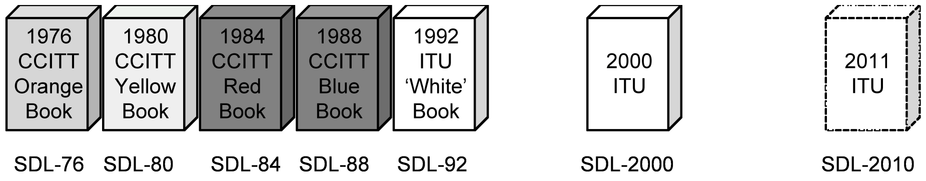

Fig. 1. Specification and Description Language publication dates

1 Background for the development of SDL-2010

SDL-2000 was completed in 1999 with versions of the ITU-T Recommendations Z.100, Z.105, Z.107 and Z.109. The main document was Z.100 [3] with Z.105 and Z.107 [9, 10] covering use of ASN.1 [11] with the SDL-2000, and Z.109 [12] covering use with the Unified Modeling Language (UML)[13]. A revised Common Interchange Format for SDL-2000 in ITU-T Recommendation Z.106 [14] supplemented these in the year 2000. Since then there have been some minor updates to these Recommendations. The text was reorganized in 2002 so that Z.100 [2] describes the graphical language, and the parts of the textual (SDL/PR phrase representation) that are alternatives to graphical representation (SDL/GR) were moved to the interchange format in Z.106[15]. At the same time a number of corrections and a few minor changes were made. In 2003 an Amendment [16] was issued to incorporate two new Annexes B and C that concern backwards compatibility and conformance to the standard. In 2007 Z.100 was republished incorporating all agreed amendments [17]. The changes have been minor: either re-organization or correction of flaws in the 1999 version, so that essentially SDL-2000 has not changed and has remained stable.

1.1 Status of SDL-2000

When SDL-2000 was being developed, right up until the ITU-T meeting at which it was approved there were two sizable software organizations that were promising to produce tools to support SDL-2000 in 2000 or 2001: Telelogic and Verilog. A merger of these two organizations in Telelogic was announced before the end of 1999, so that some competition was removed in the tool market. Although these commercial tools already supported some of the features of SDL-2000 by 2000, it is now unlikely that there will ever be a tool that approaches full support of SDL-2000 in its final form [15, 17–19]. Even the tool that best supported SDL-92, Cinderella, reached a position by 2007 when it would probably never offer full SDL-2000 support, because it has a smaller (at least in value terms) share of the ITU Specification and Description Language tool market and had to offer compatibility with Telelogic as the market leader at that time. Cinderella collaborated with Humboldt University that previously had not entered into the commercial tool market. The Humboldt SDL-tool implemented many of the features of SDL-2000 on a trial basis to test the feasibility of various ideas - indeed some features such as nested packages were implemented specifically to support feature requests promoted by Humboldt for OMG related work. In 2003 SOLINET announced the SAFIRE tool set, claiming that it is based on Z.100. In late 2004 PragmaDev, which previously supported a dialect called SDL-RT announced support also of Z.100. All four organizations (Cinderella, PragmaDev, SOLINET/SAFIRE, Telelogic) had commercial tools available in May 2006, though SOLINET/SAFIRE had ceased to be involved in ITU or SDL-Forum activities and the future of the language. By November 2008 all Telelogic products and services had become part of the IBM Rational Software portfolio, and the main tool vendors were IBM, PragmaDev and Cinderella (probably in order of market value at that time). At the time of writing all three of these vendors still offered Specification and Description Language tools.

Since 1999 the general market perception has developed. In 1999, part of the rationale for developing Z.109 as a UML profile for the ITU Specification and Description Language was because UML was perceived as a major competitor to the ITU language. Within the telecommunications industry some organizations were divided internally between those that favoured the ITU Specification and Description Language and fans of UML. A decade later the perspective is quite different, because the issue is not seen as whether to use UML or SDL-2000 (and other ITU languages), but how to use these together. In retrospect it would be easy to say this was always the way it was seen - but to be truthful this was not the case especially in 1997 and 1998 when SDL-2000 was being formulated. However, it is now clear that the state machine specification part of UML 2.1.2 is not really a complete language in itself, because of the semantic and syntactic variations that are allowed, and that to make its use practical an (implicit or explicit) profile for UML has to be used. The revised Z.109 profile of 2007 [20] is geared to the needs of the telecommunications industry by mapping UML 2.1.2 [13] onto the more precise (and therefore more practical) Z.100 semantics and (where UML 2.1.2 gives notation options or no specific notation or no notation) binding to the Z.100 syntax.

It is not by accident that the situation has been reached today where UML and ITU System Design Languages are seen as complementary rather than competing. Between 1999 and 2004 there was a significant involvement of ITU System Design Language experts in the ongoing development of UML, in particular for UML 2.0. The ITU languages have the good features of being well-defined and having action semantics that ensure specific behaviours. UML is good at object modelling and has proven to be a success at providing a framework for using different languages together - a feature that the ITU languages (for historical reasons) lack. Rather than defining new precise action languages for UML, or adding a framework scheme and object modelling to the ITU System Design Languages, the sensible way forward from a telecommunications system engineering point of view is to combine these features of both approaches.

It was therefore not a surprise to see the industry use tools that combine UML with the SDL-2000 semantic engine. This was the perspective of several major telecommunications manufacturers, and therefore the general direction of industry.

However, the situation with SDL-2000 after over a decade is unsatisfactory for all parties. Despite the 1996-1999 intention to ensure the language standard and tool support should be closely aligned (of course, ideally the same), this has not been the reality right through to the start of 2011. The language available to users is effectively SDL-92 with the 1996 addendum plus some of the features of SDL-2000 (which features depending on which tool is used) and often using legacy syntax for data. It was to ensure users are still able to produce SDL-models that are valid according to the standard that Annex B was added to Z.100 for SDL-2000, which allows the legacy syntax supported by tools.

Not only was SDL-2000 not fully supported, but also as UML and other languages such as the ITU-T User Requirements Notation [21], become more commonly used, there have been changing expectations of the facilities offered by the Specification and Description Language. Some, such as UML-like syntax, do not seem to be required. Others, that are not included in SDL-2000 such as timer supervised states, seem to be desirable.

1.2 The data issue

A data model that provides data with both sets of values (as in ASN.1) and operations is essential for any language that is to provide executable models or implementations. SDL-2000 made a major change to the way that data was defined: the algebraic axiom approach was removed from the user language, leaving just a constructive data approach (as in most programming languages). At the same time object data types were added. Leaving aside whether reference (object) data is actually needed and the "modernized" syntax in SDL-2010, whether SDL-2000 data is the best approach is reconsidered.

An SDL-2000 user is faced with the option of using either ASN.1 or SDL-2000 to define data types. If ASN.1 is used SDL-2000 provides a built-in set of operators. Similarly the built-in SDL-2000 data types provide a set of defined operators. The only real advantage of the SDL-2000 data types over ASN.1 is an arguably nicer syntax. The language could be made simpler by removing the SDL-2000 data types, but this would not be acceptable for legacy reasons.

Tools that produce target code for SDL-2000 are usually proprietary products of larger companies. Commercial tools usually implement ASN.1[2] and SDL-2000 data in one of two ways: providing translation to another programming language (usually C or C++), or producing code for a virtual machine and providing an emulator for that machine (written in some other language like Java or C). The advantage of either of these approaches is that they are target machine independent. However, there remains the issue of interfacing code from SDL-2000 with other code, especially device and message handlers and possibly the Real Time Operating System (RTOS).

[2] Whether ASN.1 compilation is done in a separate tool or not is a tool issue, not a language issue.

An alternative is to open up the language to external data types. In fact this was envisaged in SDL-92, with the **external** data syntax, but (as seen from MSC [22] and UML experience) it is difficult to define a language that can use the declarations and expression syntax in a plug-compatible way. Moreover, Z.121 [23] now provides an MSC to SDL-2000 data binding. Also from the user viewpoint the meaning of an expression in SDL-2000 would depend on the actual data language used, which may not be clear from context. As with MSC, there are requirements on any data language used, so that data is compatible with essential features such as timers. Despite these issues, from a user point of view using a data expression notation from another language can be a practical approach (as evidence see SDL-RT). The plan therefore for SDL-2010 was to first ensure SDL-2000 data is supported, and then define a way of providing a binding to other language syntaxes such as Java, C (or C++) or the data language of SDL-RT.

1.3 Diagram structure

It has been agreed for some time that although the Specification and Description Language has both a graphical (SDL/GR) and textual (SDL/PR) presentation form coupled by a common abstract grammar, the primary presentation form is graphical, and the textual form is used mainly as an interchange format. Specifications in the language therefore usually consist of a number of diagrams. While in SDL-2000 it is permitted that diagrams for inner components are drawn nested inside the diagrams for the enclosing component, in practice this is not done for two reasons: in general the resulting diagram would be too large, and full tool support is not provided. Even though tools support the printing of such nested diagrams to some extent, diagrams are usually generated separately, and for any reasonable size of system the nested diagrams become too extensive to handle, read or comprehend. Instead inner diagrams are referenced from enclosing diagrams, so that each diagram is a reasonable size. Tools support this approach.

On the other hand, the semantics of the language is defined in terms of a single hierarchical model in the abstract syntax, in which the referenced diagram replaces each reference (after eliminating any duplicates). In SDL-2000 the change from references to the hierarchy is theoretically done by transformations to a nested concrete syntax and then this concrete syntax is mapped to the abstract syntax. In SDL-2010 the change from references to the hierarchy is done by mapping referenced diagrams directly to the abstract grammar. In the concrete syntax the nested graphical form (which is not generally tool supported for diagrams in any case) is no longer part of the language. This is a worthwhile simplification, because the concrete grammar does not have to describe both forms and intermediate transformation. To some extent, moving SDL/PR to Z.106 in 2002 enabled this change.

1.4 Features without formal semantics

Some features (such as comments, paging, create lines, multiple type references) do not add to the semantics of an application model, but are provided to allow annotation to be presented for the benefit of engineers. While these features should be checked for consistency, tools otherwise ignore them. In SDL-2010 these features are separated from the extended finite state machine parts of the language.

In particular, the Association feature was added in SDL-2000, to allow UML-like associations to be shown between types (or “classes” in UML terminology). This had no semantic meaning in SDL-2000, and is arguably now better covered by using UML tools and applying the UML profile in Z.109.

1.5 Feature deletion, retention and extension

So where does this leave SDL-2000?

As for previous versions (SDL-88, SDL-92) the language definition has had several years of stability, and it was appropriate to consider what change should be made for the new version. First scheduled for consent in 2008, the revision was initially named SDL-2008. This date was not achieved. At the September 2009 ITU-T meeting it was renamed SDL-2010, the expected year for consent at that time. Though still not completed in December 2010, it was then decided (reaffirmed April 2011) not to change the name again: work had progressed sufficiently for completion in 2011 to seem certain.

As for previous revisions, one objective was to simplify the language. A new objective was to have a clearly defined basic SDL: the SDL-Task Force (a small consortium outside ITU-T) was ostensibly set up with this objective, but that organization had by October 2005 effectively ceased to exist, and the target of this group was not an SDL-2000 subset. In addition extension proposals for the language had come from many sources such as the SDL-RT, the SDL-Task Force, and industry users. There were also ideas considered previously but not incorporated into SDL-2000 and a few ideas to support UML profiling. There were some additions made in SDL-2000 compared to the previous SDL-92 version that had **not** been widely implemented, such as object data, the UML class symbol for types and UML-like associations. Some features, such as exception handing had been implemented in just one tool. These largely unimplemented features were considered for deletion.

Because SDL-2000 is a richer language than SDL-92 it was agreed there was no point in retaining features that have not been widely supported or used. On the other hand, features that are widely used should not be deleted, even if a better alternative exists or is proposed, because it has been found this kind of change leads to significant legacy problems.

A pragmatic approach has been taken: some features are being deleted and some potentially useful ones (based on the participating expert contributions tempered by user and tool vendor feedback) were added. The way Recommendations are approved at ITU-T has changed since 1999, so that significant comments can now be made and handled in a more open way during the approval stage. One criterion for feature retention or addition is compatibility with UML. There are two reasons for this: UML is a coherent framework for binding ITU-T languages together so SDL-2010 needs to be consistent with the UML model, and secondly the language provides the needed precise action semantics to UML. The creation of a UML profile for the telecommunications action language in Z.109 (06/07) [20] for SDL-2000 was obviously a key determinant for this compatibility, and generated a few necessary or highly desirable changes to Z.100.

The current draft has been prepared on the assumption that exception handling[3] is deleted (while keeping the timers on remote procedures). Object data is for the time being removed, but further study is in progress. Esoteric features (such as name class) are removed. In addition the Association feature and the use of UML class symbols for types are removed. An overview of the set of features of SDL-2010 is given in Sect. 3.

The net result is (as for previous versions) SDL-2010 contains both existing language features implemented in tools, and additional language features that (if and when supported) will enhance the language. The composite state feature does not exactly fit into either of these categories. In SDL-2000 composite states replaced the **service** concept of SDL-92, but is both a more powerful concept than **service** and more compatible with UML, though currently lacking tool support. For SDL-2010, returning to the SDL-92 **service** was not an option, because UML compatibility would be diminished, and deleting the feature was not an option either, because the SDL-92 **service** maps to an aggregate composite state. Moreover, the way an SDL-2000 state machine behaves is defined in terms of the way an instance of a composite state type behaves, and in SDL-2010 this underlying model is clarified and strengthened in the revised description of state machines.

2 Organization of the SDL-2010 language standard

This section outlines the organization of the SDL-2010 standard.

In SDL-2000 there was already a successful clear separation of abstract grammar, concrete grammar and shorthands (further considered in Sect. 2.1). SDL-2010 takes a further step by separating concerns as described in Sect. 2.2 into documents for the core language (see Sect. 2.3), full coverage of the abstract grammar (see Sect. 2.4), shorthand notation (see Sect. 2.5) and data (see Sect. 2.6) into separate documents, while retaining the separate document for ASN.1 use (see Sect. 2.7) and the common interchange format (see Sect. 2.8).

2.1 Shorthand transformation models

Since SDL-88 and in SDL-92 and SDL-2000, transformation models have been used to define a number of language features, where a given concrete syntax is

[3] Exceptions are still raised by certain constructs (such as indexing OutOfRange), but cause the further behaviour of the system to be undefined because they are not handled.

transformed into another concrete syntax. These features are called “shorthand” productions. While these features are often so useful and practical that they are essential, they are not essential in a theoretical sense, as the transformed concrete syntax can (usually) be used instead of the shorthand version. In fact the abstract syntax and language semantics are (or at least should be) defined only for concrete syntax that cannot be transformed. It is this canonical syntax that is mapped to abstract ayntax. The semantics and (as far as possible) constraints are expressed in terms of the abstract syntax.

An objective in SDL-2010 is to keep the core of the language as small as possible (and therefore easier to understand), and as far as possible separate the description of the transformation models from the core parts of the language. This is reflected in a reorganization of the SDL-2010 standard compared with SDL-2000 (see below).

2.2 Reorganization of the documents for the language standard

SDL-2010 is reorganized so that core features are defined within the Z.101 part of the language definition, with the remaining (retained) more complex language features described in subsequent parts (Z.102, Z.103, Z.104, Z.105 and Z106). In the new organization Z.100 is re-utilized to provide an overview of the set of Recommendations.

Anyone who has been tracking the language for a number of years will be aware that this structure for the language definition is not new: the 1988 version (SDL-88) defined “Basic SDL” and then a number of additional features that extended “Basic SDL”. This structure does not invalidate tools and applications that use the “full” language, while providing an identifiable subset. Some of the proposed benefits of having a clearly defined subset were:

- It makes it easier to teach and learn the basics of the language;
- It makes it easier to produce and maintain tools that can handle such a subset;
- If all tools that claim to support the standard have to support this subset, it gives a level of guaranteed portability.
- Such a subset would characterize essential “SDLness”.

Getting agreement on what should and should not be in such a subset is not an easy task. There will be many different opinions backed up by different experiences and value judgements. Work already existed, such as studies at ETSI, which could lead a consensus result, so it was argued the potential benefits (some of which are outlined above) would justify the effort. Eventually it was agreed there were insufficient participants really interested in formally defining a subset, so that explicit definition of such a subset was not a specific objective for SDL-2010.

The objective of the reorganization has been separation of concerns. The essential behaviour of a system defined using SDL-2010 depends on the extended finite state machine model of Rec. Z.101 (coupled with the behaviour of expressions of Rec. Z.104). The other Recommendations Z.102, Z.103, Z.104, Z.105

and Z.106 provide language features that (respectively): make the language more comprehensive, make the language easier and more practical to use, provide the full data model and action language, enable ASN.1 to be used, and define the interchange format. A summary of the distribution of features over the main Recommendations Z.101 to Z.104 is given in Table. 1.

2.3 Basic SDL-2010 - Z.101

Recommendation 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 machines and the associated semantics for these basic features. A state machine is a composition of states of an agent: that is, a composite state, which is an instance of a state type. For that reason, state types and composite states are part of the basic language.

Most of the abstract grammar of SDL-2000 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.

The example given in Fig. 2 illustrates Basic SDL-2010 use, but is not intended to be useful as it specifies a kind of write only memory,

In Basic SDL-2010 the system model is based on type definitions: agent instances are type based instances and the state machine of an agent is a type based state. The page structure is part of the concrete syntax, but in Basic SDL-2010 each diagram contains only one page. Multiple pages and agent diagrams (instead of type based agent instances) are shorthand notations. The key contents of Basic SDL-2010 are:

– Lexical rules including revised frame use and page numbers;
– Revised framework, package use and referenced definitions;
– Agent (system, block , process) types and state types;
– Typebased agents, procedures, channels and signals;
– State transition graphs with start and state nodes with inputs and saves;
– Transitions: each to a nextstate, join;stop or return node;
– Task (assignment), create (agent), procedure call, output (signal), and decision actions;
– Timers and key data features.

Each diagram is restricted to one page, and there are no agent diagrams (only agent types and typebased agents).

In the example an <sdl specification> is a <system specification>, which is a <typebased agent definition> that in turn 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

Feature	Z.101 Basic	Z.102 Comprehen.	Z.103 Shorthand	Z.104 Data
Lexical rule, frame use, page numbers	X	UCS Chars.		
Framework, packages, referenced definitions	X			
Agent types, state types	X			
Typebased agents, procedures	X			
Channels, interfaces, signals	X			
State transition graphs: start, state with input/save	X			
Transitions to; nextstate, join, stop, return	X			
Actions; tasks, create, proc. call, output, decision	X			
Timers, Data (overview)	X			
Macros		X		
Substate entry/exit points		X		
Aggregate states (fork/join)		X		
Type context parameters		X		
Specialization/inheritance		X		
Virtuality		X		
Remote procedures/variables		X		
Spontaneous transitions, continuous signals, priority inputs		X		
Statement lists in tasks		X		
Generic systems: select, transition option			X	
Various syntax alternatives			X	
Comment symbol, create line			X	
Text extension symbol			X	
Multiple page diagrams			X	
Statement lists in tasks			X	
Agent diagrams			X	
State graph in agent (type)			X	
Implicit channels, optional names			X	
* state/input/save, implicit input			X	
Predefined data package				X
Textual procedure/operator body				X
Data encoding (see text)				X

Table 1. Features in Z.101, Z.103, Z.103 and Z.104

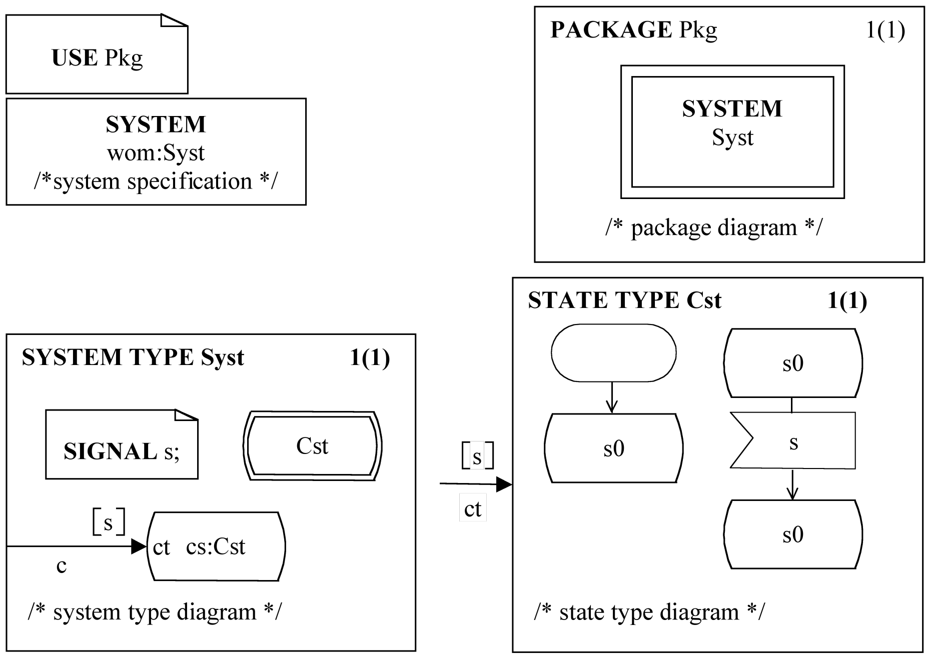

Fig. 2. System wom in Basic SDL-2010

symbol references the system type `Syst` . In the system type diagram `Syst`, the state symbol represents the state machine 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 machine. In this case the (composite) state type of the state machine is referenced locally by the state type symbol containing `Cst`. The referenced state type `Cst` gives the state graph.

In SDL-2000, the transformations (from agent diagrams to agent references using implicit agent types, and from a state graph at the agent level to reference an implicit state type for the state machine of the agent containing for the graph) were hidden within the general description of agents. In SDL-2010 the more concise, usual forms are not defined in Basic SDL-2010. Instead the transformations are given in Z.103 and result in Basic SDL-2010.

2.4 Comprehensive SDL-2010 - Z.102

Recommendation 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 type inheritance, continuous signals, enabling conditions, and aggregate states. Other features of Comprehensive SDL-2000 are important to complete the language: virtual types, parameterized types with context parameters, remote procedures/variables, generic systems (select, optional transition), macros and Unicode handling. Features without full abstract grammar are:

Inheritance (specialization) which allows one type to be a specialized subtype of another type, so that it inherits properties, structure and the way instances behave from the parent super-type while additional properties, structure and actions can also be added. There is an optional parent identifier defined in the abstract syntax to link the specialized type to the parent type. However, the abstract grammar part given by the *Semantics* description of a specialized type definition is still largely in terms of combining the content of the super-type with the content of the specialized definition described by reference to concrete syntax items and is mostly unrevised from SDL-2000 (and probably SDL-92).

Virtuality where a type marked as **virtual** defined within an enclosing type is inherited in any sub-type of the enclosing type, and as a virtual type is allowed to be redefined in the sub-type. The redefined type can be marked either **redefined** in which case it is virtual and is allowed to be redefined in a sub-type of the sub-type, or marked **finalized** in which case redefinition is not allowed. Virtuality is currently handled at the concrete syntax level: there (currently) is no associated abstract syntax.

Context parameters are another feature handled at the concrete syntax level. In this case a copy is made of the body of the base type with context parameters and the actual parameters are substituted for the formal parameters rather like macro parameter substitution, but with type checking on the actual parameters. With all the parameters bound, the resulting body is used to define a type.

Remote procedures/variables are defined in terms of an implicit signal exchange between the caller and the owner of the procedure/variable with the caller waiting in an implicit state for a response. For simplicity, these are described in terms of models that use Basic SDL-2010 concrete grammar, but in principle they could (with some effort) be redefined in terms of abstract grammar.

Select and optional transition provide concrete syntax for including/excluding parts of diagrams based on the value of a <simple expression>: `Boolean` in the case of **select** used in agent structure diagrams, and typically `Boolean` or `Integer` for a <transition option area> of a state graph. It is possible to statically evaluate a <simple expression> during the <sdl specification> analysis and the resulting *SDL-specification* is determined from the selected parts. Consequently there is no abstract grammar for this feature.

Macros and Unicode handling are both carried out on the system model before the full validity and meaning of the resulting system is considered. They are therefore not the same as shorthand notations. The handling of Unicode is a lexical issue resulting in unique tokens for each name. Macros can be defined anywhere, have global scope and are expanded before other analysis results: the division of the <sdl specification> into diagrams and and/or files is (in theory) ignored, so there is no abstract grammar.

2.5 Shorthand notation and annotation in SDL-2010 - Z.103

Recommendation 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. The key additional features are:

- Agent diagram (has an implied agent type)
- Agent with state graph (has an implied state type)
- Asterisk input/save, implicit transition
- Signallist, interface as stimulus/on channel
- Asterisk state, multiple state appearance
- Multiple diagram pages
- Various syntax alternatives
- Create lines, comment area, text extension

The example given in Fig. 3 is a redefinition of the `wom` system using the features of Shorthand SDL-2010. Compared with Fig. 2 only one diagram is now required, because the types are implied.

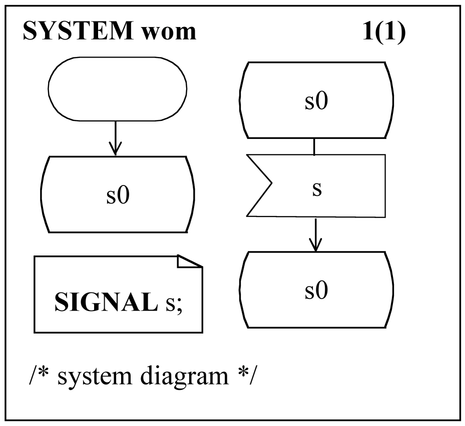

Fig. 3. System wom in Shorthand SDL-2010

2.6 Data and action language in SDL-2010 - Z.104

Recommendation 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.

The SDL-2000 Z.104 [18], which concerns the encoding of data, was initially incorporated into Z.104 for SDL-2010. A key feature is applying encoding rules for ASN.1 and it has therefore been decided to move this material to Z.105.

2.7 SDL-2010 combined with ASN.1 modules - Z.105

Recommendation 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. This is unchanged from SDL-2000 except the data type definitions are moved to Z.104 for SDL-2010. However, since the publication of Z.104 [18] in 2004, there has been a standardized why of invoking encoding/decoding rules, which is described in paper [24]. An ASN.1 CHOICE in an ASN.1 module can be imported as an interface. For example, if an ASN.1 module named `MyASN1` contains a CHOICE named `MyMessages`, this can be imported using the package use area:

USE MyASN1/**INTERFACE** MyMessages;

attached to the system diagram. If `MyMessages` is defined in `MyASN1` as:

```
MyASN1
      DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
-- definitions including
      MyMessages ::= CHOICE {
              connect Destination,
              sendInfo Information,
              disconnect ConnectionRef}
-- the data types mentioned above such as Destination
-- and Information visible here.
END
```

this implies the SDL-2010 **interface**:

```
interface MyMessages {
      signal  connect ( Destination ) ,
              sendInfo ( Information ) ,
              disconnect ( ConnectionRef ); }
```

The user therefore has a simple and direct way of defining in ASN.1 the signals to be used in the SDL-2010 model. The channel that carries the signals of the interface can be defined with **interface** `MyMessages` as a <signal list> associated with the channel. The set of encoding rules used on the channel are specified after the channel name with (for example, for Packed Encoding Rules) **encoding** `PER`.

2.8 Common Interchange Format for SDL-2010 - Z.106

Recommendation 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.

2.9 Formal definition (Annex F to Z.100)

The initial plan is that no formal definition be 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 this is out of date, but in combination with the obsolete 2007 version of Z.100 (for SDL-2000) [17] it 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 [17] provides more detail than Z.100 to Z.106 for most SDL-2010 features. 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 metamodelling) might be considered.

3 Changed, new and deleted features

A number of changes in SDL-2010 compared to SDL-2000 are the result of reviewing and rewriting to divide the text into several Recommendations. Some changes, such as removing nested diagrams, resulted in both deletion and a rewrite of retained text. Changes, such as making pages part of the diagram syntax, required additional syntax rules as well as modifying existing rules. The items described here only cover changes that are significant new or deleted items. For example, although the text on statement lists has had a major revision, there was no intention to change the language – only to improve Recommendation text, therefore this change is not described here.

3.1 Synonym as a "read only variable"

A <synonym definition> represents a *Variable-definition* in the context in which the synonym definition appears with special property that the variable is read-only. The <synonym name> represents the *Variable-name*. Writing to the variable is not possible, because <synonym> is not allowed where assignments could take place. The concrete syntax is not changed and synonyms can be used as in SDL-2000. The revised semantics better matches implementation by means of a variable containing the synonym value, which can easily be changed if the value is changed.

3.2 Lower bound of agent instance sets

It is allowed to specify a *Lower-bound* on agent instance sets as in Fig. 4, which by default is zero matching SDL-2000. If the same `Natural` is used for the initial, maximum and lower bound, the set is fixed, which was not possible to specify in SDL-2000. A *Stop-node* in an instance set that is already at the *Lower-bound* causes the exception `OutOfRange` to be raised, but the number of active instances can be found from the integer built-in expression **active**(**this**) or **active**(`ais`) where `ais` is an agent instance set. The new syntax is:
<number of instances>::=([<initial number>][,[<maximum number>][,<lower bound>]])

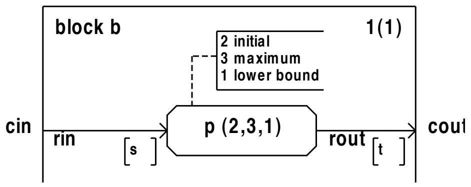

Fig. 4. Process instance set with a lower bound

3.3 Input and Save via gate

In SDL-2010 when a signal is placed in an input port, it is stored with the identity of the gate on which it arrived at the destination agent. This allows the transition taken to be determined by the gate as well as the signal identity. The example in Fig. 5 is used to illustrate the feature. Assume an instance of 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).

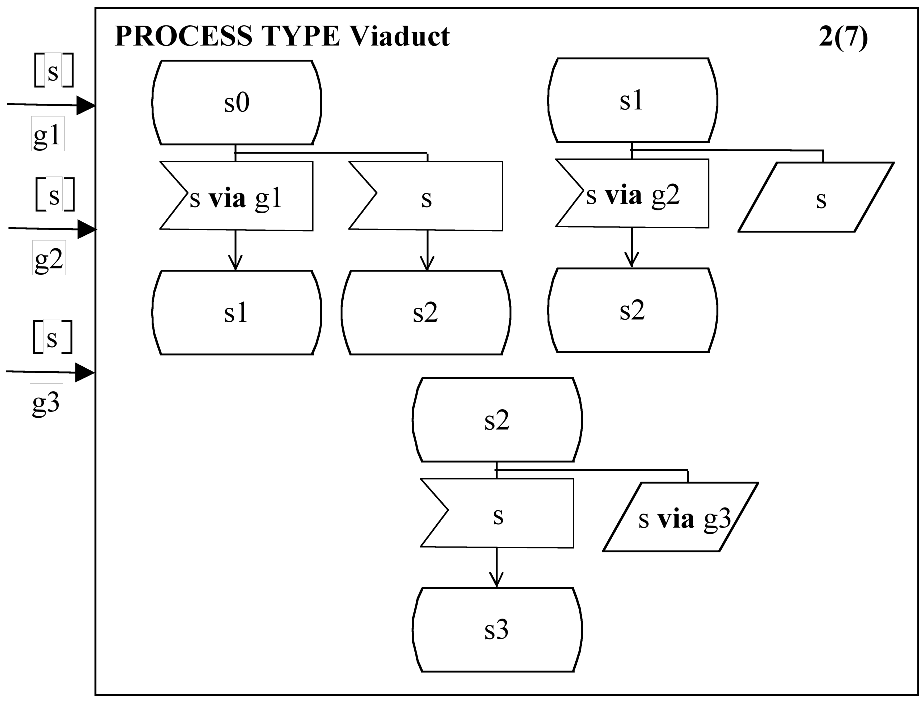

Fig. 5. PROCESS TYPE `Viaduct` using the input via gate feature

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 `s` 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 no explicit 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) diagram, the name of a channel to the process is used for the via. In the implicit process type for the process, this is transformed to the implicit gate connected to the channel.

3.4 Multiple priority levels of input

In SDL-2000 inputs can be with or without priority, but there are no levels of input priority. Priority inputs of a state are considered for enabling signals in the input port for consumption before other inputs without priority. In SDL-2010 there are multiple levels of priority and inputs with highest priority are considered first. If there are no signals for a priority level, inputs with the next

priority are considered until either a signal is enabled or all priority inputs have been considered, after which inputs without priority are considered. Priority is specified by a *Priority-name* given as a `Natural` value, but note that zero is the *highest* priority, one is lower and the *highest* number given has the *lowest* priority. The reason is to be consistent with the existing SDL-2000 *Priority-name* for *Continuous-signal* where zero is taken first. However, if the <priority name> is omitted this implies a number one greater than the highest number explicitly given: that is, the lowest priority. A priority input without a gate, takes precedence over a lower priority input with a gate or an input with a gate without priority. The new syntax is:
<priority input list>::=<priority stimulus>**{**,<priority stimulus>**}***
<priority stimulus>::=<stimulus>[<priority clause>]
<priority clause>::= **priority** [<priority name>]

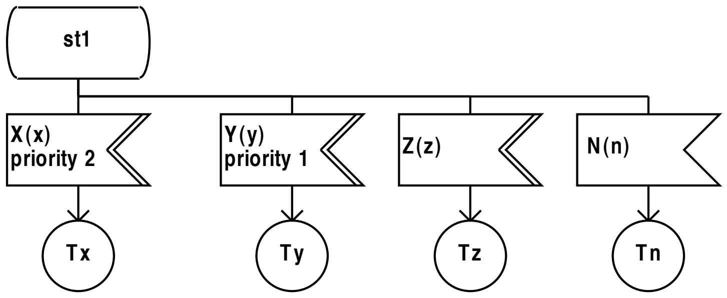

Fig. 6. Multiple levels of input priority

For the example in Fig. 6 assume the signals `X`, `Y`, `Z` and `N` each with one parameter saved in `x`, `y`, `z` and `n` respectively. Assume state `st1` is reached with the input port containing in order of arrival signals `N`, `Y`, `Z` and `X`. Assume the connectors `Tx`, `Ty`, `Tz` and `Tn` connect to transitions that terminate in state `st1`. The input for `Z` has an implied *Priority-name* of 3. The signal `Y` is enabled and consumed because this input has the highest priority and `Ty` is taken. When `st1` is reached again signal `X` is enabled if no `Y` signals have arrived in the meantime, and on next occasion signal `Z` (assuming no `X` or `Y` signals have arrived). Signal `N` is only enabled if there are no signals `X`, `Y` or `Z` in the input port when in state `st1`. If the inputs for two or more signals have the same input priority, the signal that arrived first is consumed.

3.5 Timer supervised states

SDL-2010 is extended to cover timer supervised states. If state has a <state timer area> (see syntax below), the *State-node* has a *State-timer*. The timer is set entering the state and reset entering a *Transition*, except for an empty

Transition to the state (for example from an implicit transition). If the timer expires while in the state, the timer signal is immediately consumed: it is a higher priority input than any other input. The *Transition* for the <transition area> of the <state timer area> is taken. The syntax is:

<state timer area>**::=**<plain input symbol> **contains** <state timer>
is followed by <transition area>

<state timer> **::=** **state timer** <time expression> | **set** <set clause>

A <state timer> with **state timer** <time expression> as in Fig. 7 uses an implicit timer for the state, whereas **set** <set clause> uses a defined timer which can have a default duration. If the state is a composite state, the timer expiration is treated in the same way as a signal causing an exit from the composite state.

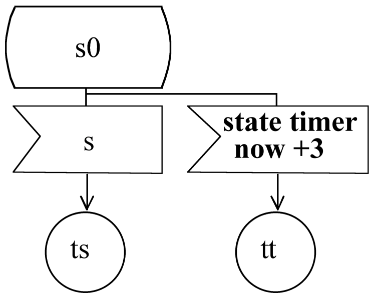

Fig. 7. Timer supervised state with an implicit timer

3.6 Deleted features

As mentioned above, nested diagrams are no longer part of the standardized concrete syntax. Instead each diagram is referenced from the parent. Associated with this change, <specification area> that was an optional graphical depiction of the relationships between <system specification> and <package>s is deleted. These features were not supported by tools in a consistent way, or as defined in the Recommendations.

SDL-2000 included UML-like concrete syntax for type references and also UML-like associations. These were not well supported and added nothing to the semantics of the language: they just made the language Recommendation more complex and difficult to understand.

Since SDL-92 there has been a mechanism called *nameclass* for defining a generator for a set of names of literals or operators of a data type. This is used to define the literals for the Integer data type, character strings for the Charstring data type, and similarly for the literals of Bitstring, Real, Duration and Time. The nameclass feature is used with **spelling**, which provides the character string that corresponds to the spelling of the nameclass literal. While these are essential for the `Predefined` data types, there is no need to provide them as a feature of the user language. These have therefore been moved to Annex A of Z.104 to be used only (like axioms) for the definition of the `Predefined` data types. It is therefore no longer expected that tools support these features.

For the time being object data types are no longer part of the language. However, similar data types are widely used in other languages and the purpose of including them in SDL-2000 was to provide the advantages of reference computing. The further study plan for SDL-2010 is to reconsider how object data types are incorporated into the language, leading to a revision of Z.104 and/or the drafting of an additional Recommendation. One key issue is to minimize (preferably eliminate) dynamic binding of data types.

4 Conclusion

The status of SDL-2010 as this paper was initially being written in April 2011 was that a reasonable draft existed for Z.100 to Z.105, so that it was not unrealistic to expect a final draft to be consented for initiation of the ITU-T approval process at the start of September 2011. By the time this paper was presented in July 2011, improved drafts were available, to be circulated for review and comment up to and including the ITU-T meeting starting in the last week of August 2011.

On the other hand, as this paper was being revised for publication there were still issues in the existing drafts to be corrected and further discussed. The concept of support for specifying channel delay values was accepted, but the text needed to be finalized. Work has started on a further Recommendation, probably to be approved in 2012, to include in SDL-2010 **ref** and **own** variables and parameters that are associated with a reference to a value. Also, work has not really started on revising Z.106. So there was still work to do, and the reader is advised to check if the September 2011 date was met, or whether there is a further delay until 2012.

Another concern is tool support. There has been little innovation in tools for the language in the last few years. This is probably because the major source of income for tool providers has been the telecommunications industry, and despite the rapid growth of the Internet, companies such a Nortel, Nokia, Sony Ericsson and Motorola have had trading difficulties. No matter how good SDL-2010 is, if there is a lack of funds for tool development SDL-2010 is likely to remain theoretically interesting, but only implemented to the same subset as SDL-2000. The good news is that this subset is probably a bigger percentage of SDL-2010 than it was of SDL-2000, as I estimate more has been deleted than added in the revision.

Finally, if you read this paper, and would like to participate in the language review process, contact the author. This applies even if September 2012 has already passed, because there is a review period during the ITU-T approval process, and there is a defined maintenance procedure subsequent to publication by ITU-T. It is expected that the language will evolve further to meet user needs.

References

1. CCITT Orange Book, Volume VI.4, Programming Languages for Stored-Programme Control Exchanges, ISBN 92-61-00421-0, pp. 3–23, ITU, Geneva, 1977.
2. ITU-T – International Telecommunication Union: Recommendation Z.100 (08/02): Specification and Description Language (SDL) , Geneva, Switzerland (August 2002); http://www.itu.int/rec/T-REC-Z.100-200208-S/en
3. ITU-T – International Telecommunication Union: Recommendation Z.100 (11/99): Specification and Description Language (SDL) , Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-Z.100-199911-S/en
4. CCITT Yellow Book, Volume VI Fascicle VI.7, Functional Specification and Description Language (SDL). Man Machine Language (MML), Recommendations Z.101 – Z.104 and Z.311 – Z.341, ITU, Geneva, 1981. *I don't have a copy of this – a more accurate reference is needed*
5. CCITT Red Book, Recommendations Z. 100 to Z. 104: Specification and Description Language, ITU, Geneva, 1985. *I don't have a copy of this – a more accurate reference is needed*
6. ITU-T – International Telecommunication Union: Recommendation Z.100 (11/88): Specification and Description Language (SDL) , Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-Z.100-198811-S/en
7. ITU-T – International Telecommunication Union: Recommendation Z.100 (03/93): CCITT Specification and Description Language (SDL), Geneva, Switzerland (March 1993); http://www.itu.int/rec/T-REC-Z.100-199303-S/en
8. ITU-T – International Telecommunication Union: Recommendation Z.100 Addendum 1 (10/96): Corrections to Recommendation Z.100, CCITT Specification and Description Language (SDL) , Geneva, Switzerland (March 1993); http://www.itu.int/rec/T-REC-Z.100-199610-S!Add1/en
9. ITU-T – International Telecommunication Union: Recommendation Z.105 (11/99): SDL combined with ASN.1 modules (SDL/ASN.1), Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-Z.105-199911-S/en
10. ITU-T – International Telecommunication Union: Recommendation Z.107 (11/99): SDL with embedded ASN.1, Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-Z.107-199911-W/en
11. ITU-T – International Telecommunication Union: Recommendation X.680-X.695 (11/08): Information Technology - Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules, Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-X.680-X.693-200811-P/en
12. ITU-T – International Telecommunication Union: Recommendation Z.109 (11/99): SDL combined with UML, Geneva, Switzerland (November 1999); http://www.itu.int/rec/T-REC-Z.107-199911-S/en
13. Object Management Group: Unified Modeling Language, http://www.omg.org/spec/UML/ and http://www.omg.org/spec/UML/2.1.2
14. ITU-T – International Telecommunication Union: Recommendation Z.106 (11/00): Common interchange format for SDL, Geneva, Switzerland (November 2000); http://www.itu.int/rec/T-REC-Z.106-200011-S/en
15. ITU-T – International Telecommunication Union: Recommendation Z.106 (08/02): Common interchange format for SDL, Geneva, Switzerland (August 2002); http://www.itu.int/rec/T-REC-Z.106-200208-I/en
16. ITU-T – International Telecommunication Union: Z.100 (2002) Amendment 1 (10/03): Backwards compatibility and compliance, Geneva, Switzerland (October 2003); http://www.itu.int/rec/T-REC-Z.100-200310-S!Amd1/en

17. ITU-T – International Telecommunication Union: Recommendation Z.100 (11/07): Specification and Description Language (SDL) , Geneva, Switzerland (November 2007); http://www.itu.int/rec/T-REC-Z.100-200711-I/en
18. ITU-T – International Telecommunication Union: Recommendation Z.104 (10/04): Encoding of SDL data, Geneva, Switzerland (October 2004); http://www.itu.int/rec/T-REC-Z.104-200410-I/en
19. ITU-T – International Telecommunication Union: Recommendation Z.105 (07/03): SDL combined with ASN.1 modules (SDL/ASN.1), Geneva, Switzerland (July 2003); http://www.itu.int/rec/T-REC-Z.105-200307-I/en
20. ITU-T – International Telecommunication Union: Recommendation Z.109 (06/07): SDL-2000 combined with UML, Geneva, Switzerland (June 2007); http://www.itu.int/rec/T-REC-Z.107-200706-I/en
21. ITU-T – International Telecommunication Union: Recommendation Z.151 (11/08), User Requirements Notation (URN) – Language definition, Geneva, Switzerland (November 2008); http://www.itu.int/rec/T-REC-Z.151/en
22. ITU-T – International Telecommunication Union: Recommendation Z.120 (02/11), Message Sequence Chart (MSC), Geneva, Switzerland (February 2011); http://www.itu.int/rec/T-REC-Z.120/en
23. ITU-T – International Telecommunication Union: Recommendation Z.121 (02/03), Specification and Description Language (SDL) data binding to Message Sequence Charts (MSC), Geneva, Switzerland (February 2011); http://www.itu.int/rec/T-REC-Z.121/en
24. Reed, R.: Data encoding for SDL in ITU-T Rec. Z.104, In Systems Analysis and Modeling - SAM2004, pp.80-95, LNCS 3319, Springer