INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.121

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(02/2003)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Message Sequence Chart (MSC)

Specification and Description Language (SDL) data binding to Message Sequence Charts (MSC)

ITU-T Recommendation Z.121

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	**Z.120–Z.129**
Extended Object Definition Language (eODL)	Z.130–Z.139
Tree and Tabular Combined Notation (TTCN)	Z.140–Z.149
User Requirements Notation (URN)	Z.150–Z.159
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-computer interfaces for the management of telecommunications networks	Z.360–Z.369
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Distributed processing environment	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Z.121

Specification and Description Language (SDL) data binding to Message Sequence Charts (MSC)

Summary

This Recommendation provides a Specification and Description Language (Z.100) instantiation of both the syntactic and semantic elements of the Message Sequence Chart (Z.120) data interface, defines the default types, and also the syntax for allowable SDL data definitions that may be used in an MSC Document.

Source

ITU-T Recommendation Z.121 was prepared by ITU-T Study Group 17 (2001-2004) and approved under the WTSA Resolution 1 procedure on 13 February 2003.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Introduction

The data binding of SDL to MSC is provided in two parts. The first describes the syntactic part of the interface, which defines MSC document statements relating to the use of SDL as the data language, and the second describes the semantic part of the interface. The latter consists of the definition of a number of functions that are used to perform the syntactic, static semantic, and dynamic semantic evaluation of SDL data used in MSC.

ITU-T Recommendation Z.121

Specification and Description Language (SDL) data binding to Message Sequence Charts (MSC)

1 Scope

This Recommendation provides a Specification and Description Language (Z.100) instantiation of the data interface for Message Sequence Chart (Z.120) that is defined to be the default for Z.120.

The MSC Z.120 Recommendation has an open data interface that permits different users of the language to adopt different data languages for use within MSC diagrams and documents. ITU-T Rec. Z.121 defines a binding of the open interface to a fragment of SDL's data language, which is to be used as the default data language. That is, in the absence of an explicit instantiation of the data interface in an MSC document, the binding defined by this Recommendation is to be assumed, as is the language of allowable data definition strings.

This is the first version of ITU-T Rec. Z.121, based on the data interface definition that first appeared in the current ITU-T Rec. Z.120, published in 1999.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

– ITU-T Recommendation Z.100 (2002), *Specification and Description Language (SDL)*.

– ITU-T Recommendation Z.120 (1999), *Message Sequence Chart (MSC)*.

3 Syntactic interface

The MSC document provides a statement to declare the name of the data language used, so that tools supporting more than one language may distinguish them. It also provides statements to declare the parenthesis sequences used by the data language so that tools may correctly identify the extent of any data string embedded in an MSC. In this clause the default statements for the SDL data binding are given.

The language declaration is done within a data definition, which also contains the wanted data declarations: data types (with methods and operators), syntypes and synonyms.

3.1 Language declaration

The language declaration may be the given in the MSC document, and it identifies which data language is used throughout the MSCs in the scope of the enclosing MSC document. The following language declaration is defined for SDL, although being the default its use is optional:

language SDL**;**

If this language declaration is given, the parenthesis and escape declarations are implicitly declared and thus not needed to be explicitly given in each MSC document, as are the default types together with their corresponding wildcards.

3.2 Parenthesis and escape declarations

When SDL data is used in MSC according to the language declaration above, the following parenthesis and escape declarations are assumed:

parenthesis

nestable '(')', '[']', '(')', '{'}';

nonnestable '/*'*/', '/#'#/', '<<'>>';

equalpar /'/;

escape '?';

;

This declaration means that:

- the native Z.100 <comment>s and <note>s (/# a comment #/ and /* a note */) are accepted in data strings;
- Z.100 qualifiers (for example, <<package BasicTypes>>) may be used to resolve name conflicts;
- Z.100 character strings ('hello world!') are allowed in expression strings, using the native Z.100 form of a repeated apostrophe (') to allow a single apostrophe in a string;
- '?' is used as a general escape to allow the MSC stop tokens in the data strings.

If an MSC token that can denote the end of a data string, such as a semi-colon, is to be *used openly* within the data string, then it must be escaped with the '?' character. By *used openly* we mean that it is not 'hidden' within a pair of the SDL declared parenthesis. For example, a semi-colon does not need to be escaped when it occurs within an SDL string because it is hidden by the string delimiter characters.

The only SDL token that needs to escaped in an SDL context is a single quote within a string expression. SDL strings do not require any special treatment when used within an MSC data string.

An MSC parser looks for the longest match of parenthesis. An example: both '(.' and '(' are nestable parenthesis, so the parser would detect incorrect nesting of these parentheses.

3.3 Data declaration and use

The actual data type declarations in the native data language are done within the open string <data definition string> in an MSC document. The next clause describes the allowed SDL syntax within this string.

Since SDL <expression>s in general allow constructs that may introduce side-effects, the expression permitted in MSC will be constrained, in particular they shall not contain:

- <create expression>;
- <value returning procedure call>;
- <imperative expression>.

SDL comments (<note> and <comment body>) can be embedded in any of the MSC terminal strings. The static semantic functions defined in clause 4, define which SDL <expression>s are allowable in an MSC.

3.4 Default data types and wildcards

When MSC is used with SDL data according to the language declaration in 3.1, the SDL package Predefined is assumed, so the predefined data types can be referenced from <data definition string> or <type ref string> and predefined operators may be referenced inside an <expression string>.

MSC requires the data types Boolean, natural and time (see 5.11/Z.120). The corresponding data types in ITU-T Rec. Z.100 to be used as defaults are:

<<**package** Predefined>> **value type** Boolean;

<<**package** Predefined>> **syntype** Natural;

<<**package** Predefined>> **value type** Time.

For each of these default types, a default wildcard is implicitly declared as follows:

wildcards

anyB: Boolean;

anyN: Natural;

anyT: Time.

4 Semantic interface

The semantic interface is defined in three parts:

- well formedness, describing the grammar of allowable SDL data strings to be used in MSC;
- static semantics, identifying the type correctness requirements of well-formed SDL data strings;
- dynamic semantics, defining how to evaluate well-formed and type correct SDL expression strings used in MSC.

4.1 Well-formedness definitions

The data interface specifies four functions, Wf1, … , Wf4 that are required to define the syntactically valid data strings. Table 4-1 identifies the SDL grammar that corresponds to the valid data strings. For example, Wf1(*v*) is true if and only if *v* is a string that can be produced by the SDL grammar rule for <variable name>. The strings can contain SDL comments in the normal way so that the definitions apply to any string following the filtering of any comments as defined by <note> or <comment body>.

Table 4-1/Z.121 – Mapping of MSC data interface strings

Well-formedness function name	MSC terminal string	SDL grammar
Wf1	<variable string>	<variable name>
Wf2	<data definition string>	<left curly bracket> <data definition>* <right curly bracket>
Wf3	<type ref string>	<basic sort>
Wf4	<expression string>	<expression>

In the case of data definition strings, Wf2, the legal strings have to be defined by an auxiliary grammar rule in terms of the SDL production <data definition>, since there does not exist an SDL production that corresponds to the requirement. The rule permits any sequence of <data definitions> enclosed by a pair of braces (defined as <left curly bracket> and <right curly bracket>). The braces save having to escape any semicolon characters appearing in the data definitions, since a semicolon acts as the delimiting character for the data definition string itself. Thus, a valid <data definition string> forms a valid SDL package body – being a subset of the allowable items in a package.

4.2 Static semantics interface functions

The data interface specifies four predicates, Tc1, … , Tc4, one per data string class, that are required to assert that the static semantics of valid data strings are upheld. That is, the predicates will only be applied to strings that satisfy their corresponding well-formedness function. The static semantic rules that will determine the four predicates are described in 6.3/Z.100 and clause 12/Z.100. There is also EqVar, an auxiliary relation, required by the data interface used to identify equivalence between variable strings.

4.2.1 Tc1, Data definition strings

Tc1(*d*) is true of a syntactically valid data string *d* – i.e., Wf2(*d*) is true – if it satisfies the static semantics of SDL as defined in 6.3/Z.100. In particular, if a package body consisting of the string *d*, minus its enclosing braces, satisfies the syntax and static semantics of SDL packages, then Tc1(*d*) is true, and vice versa. This is illustrated in Figure 4-1 where the data string "{ *D* }" satisfies Tc1 only when the SDL package is statically correct.

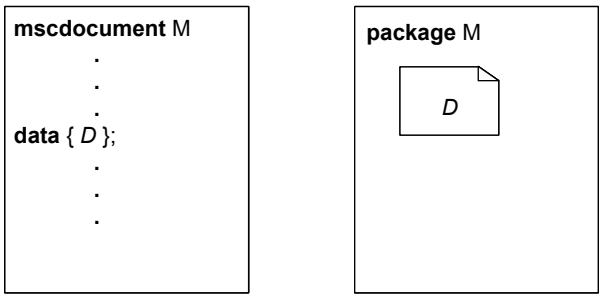


Figure 4-1/Z.121 – Equivalent SDL static requirements for data string

4.2.2 Tc2, Type reference strings

Tc2(*d*)(*t*) is true of a syntactically valid data string *d* and type reference string *t* – i.e., Wf2(*d*) and Wf3(*t*) are true – if and only if:

- Tc1(*d*), i.e., *d* conforms to the static semantics for SDL data;
- *t* is statically correct in the context of data string *d* according to the static rules of SDL.

The latter requirement is equivalent to saying that a declaration of a variable of having type *t* in the context of a package containing *d* (minus enclosing braces) is statically correct. This is illustrated in Figure 4-2 where the example type reference string *t* of an MSC variable declaration satisfies Tc2(*d*) only when the SDL process is statically correct. The package referenced by the SDL process is constructed as per Figure 4-1.

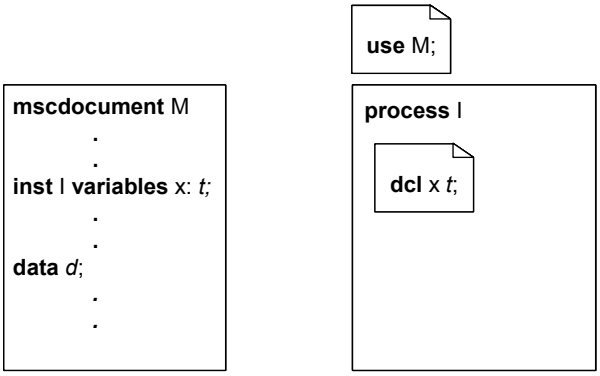


Figure 4-2/Z.121 – Equivalent SDL static requirements for type reference string

4.2.3 Tc3, Expression Strings

Tc3(*d*)(*S*)(*e*) is true of a syntactically valid data string *d* and expression string *e*, and set of typed variable assignments *S*, if and only if:

- Tc1(*d*), i.e., *d* is a statically valid data definition string;
- for each valid variable string type string pair (*v*, *t*) in the set *S*:
  - Tc2(*d*)(*t*), i.e., *t* is a statically valid type reference in context of data string *d*;
- *e* is a statically correct SDL expression:
  - in the context of data string *d*;
  - given the assignment of types to variables defined by *S*.

The definition is equivalent to an SDL expression *e* satisfying the static type rules of SDL in the context of a package containing the data definition string (minus enclosing braces) *d*, and for each paring of variable *v* with type string *t* in *S,* there exists an SDL variable declaration of *v* having type *t*. This is illustrated in Figure 4-3 where the expression string *e* of an MSC message parameter satisfies Tc3(*d*)(*S*), in which *S* is the set of typed variables { (*x1*, *t1*), (*x2*, *t2*), … , (*xn*, *tn*) }, only when the SDL process shown is statically correct. The package referenced by the SDL process is constructed as per Figure 4-1.

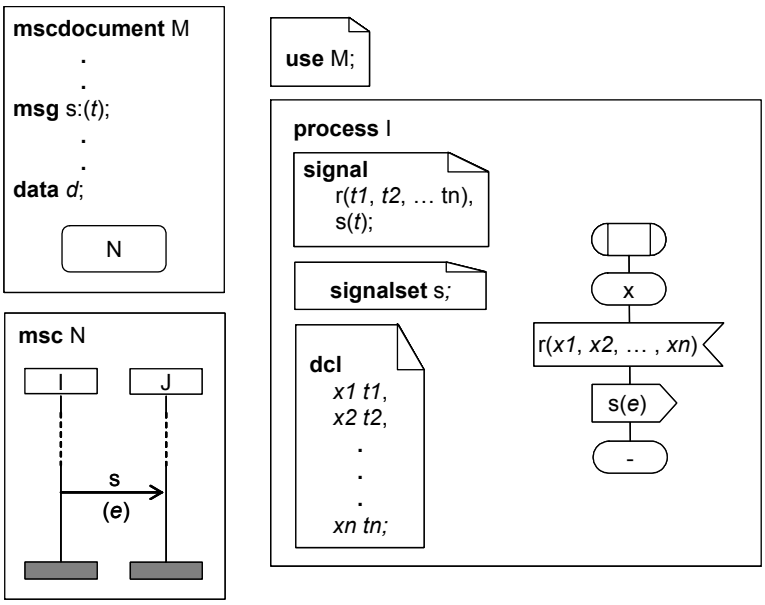


Figure 4-3/Z.121 – Equivalent SDL static requirements for expression string

For example, if an expression contains a variable undeclared in the set *S*, then it will fail the SDL static rules, and hence the Tc3 predicate will also be untrue.

The variable strings appearing in argument set *S* must denote unique variables, and therefore we define a *canonical form* to be the variable string is stripped of any surrounding white space or comments.

4.2.4 Tc4, Typed expression strings

Tc4(*d*)(*S*)(*t*, *e*) is true of a syntactically valid data string *d* and expression string *e*, and set of typed variable assignments *S*, if and only if:

- Tc2(*d*)(*t*), i.e. *t* is a statically valid type reference string in context of data string *d*;
- Tc3(*d*)(*S*)(*e*), i.e. *e* is syntactically and statically valid expression string in context of *d* and variable type assignment *S*;
- *e* can have the type *t* according to the static rules of SDL.

The last clause is the equivalent to saying that in addition to the rules given for a statically valid expression *e* as defined by predicate Tc3, the expression legally can appear as argument to an SDL output signal having the declared parameter type *t*. This is also illustrated by Figure 4-3.

4.2.5 EqVar, Equal variable strings

EqVar(*v1*, *v2*) is true of two syntactically valid variable strings *v1* and *v2* if and only if:

- the strings *v1'* and *v2'* remaining after removal of comments and white space from *v1* and *v2* respectively are identical.

This definition reflects the naming rules for SDL, in which variables differing in any character, including differences in case, represent different variables. The strings *v1*' and *v2*' are said to be in canonical form, as defined in 4.2.3.

4.3 Dynamic semantics interface functions

The data interface requires four functions to be defined – three are auxiliary, and the final *Eval* function is used here to compute the value of SDL expression identified with the Wf2 predicate. The auxiliary functions are straightforward to understand and not necessarily unique – that is, different interpretations may be made without affecting the *Eval* function. The relevant clauses of

the SDL Recommendations are given as part of each function's description where required. The functions refer to the static checking predicates defined above, which in turn rely on the well-formedness predicates initially defined.

4.3.1 Vars, Extract variables

Vars is a subsidiary function that is required in the computation of the dynamic traces of an MSC. Given a context, the result of applying *Vars* to an expression *e* is the set of variables the expression contains. Each variable is paired with the number of occurrences of the variable in expression *e*. The full definition of *Vars* can be given by structural induction on the SDL grammar of expressions, but only an informal definition is given here; first the domain of *Vars* is defined.

Vars(*d*)(*S*)(*e*) is defined if:

- *d* is a syntactically valid data string, *e* is a syntactically valid expression string, and *S* is a set of variable strings each paired to a syntactically valid type string;
- Tc3(*d*)(*S*)(*e*) is true, i.e., expression string *e* conforms to the static requirements of SDL.

Notice that the static checking requirements defined by Tc3 ensure that *Vars* can only be defined if all the variables contained in the expression are 'declared' in set *S*.

Provided Vars(*d*)(*S*)(*e*) is defined, its value is given by:

- if *e* is a constant, then Vars(*d*)(*S*)(*e*) is the empty set.
- if *e* is a variable string, then Vars(*d*)(*S*)(*e*) = { (*v*, 1) }, wherein *v* is the canonical form of the variable string defined in 4.2.3; n.b. *v* must appear in *S*.
- if *e* is a compound expression, then Vars(*d*)(*S*)(*e*) is calculated by summing the number of occurrences of a variable across its constituent expressions.

4.3.2 Replace, Variable replacement

Replace is a subsidiary function that is required to compute the dynamic traces of an MSC that employs wildcards. As such, its definition does not need to be uniquely defined for the SDL language, since its effect would be internal to any tool supporting MSC with SDL data. Therefore, this Recommendation does not need to fix it definition completely.

Replace(*d*)(*v1*, *n*, *v2*)(*e*) is defined if:

- *d* is a syntactically valid data string, *v1* and *v2* are syntactically valid variable strings, and *e* is a syntactically valid expression string;
- Tc3(*d*)(*S*)(*e*) is true, i.e., expression string *e* conforms to the static requirements of SDL.

The result of Replace(*d*)(*v1*, *n*, *v2*)(*e*) is an expression string *e′* in which the *n*th occurrence of the variable *v1* has been replaced by variable *v2*. There are a number of ways to define 'the *n*th occurrence', and this is left open in this Recommendation.

4.3.3 NewVar, New variable

Like *Replace*, *NewVar* is a subsidiary function that is required to compute the dynamic traces of an MSC that employs wildcards. As such, its definition does not need to be uniquely defined for the SDL language, since its effect would be internal to any tool supporting MSC with SDL data. Therefore this Recommendation does not require any particular way to be used.

NewVar(*d*)(*S*) is defined if:

- *d* is a syntactically valid data string, and *S* is a set of variable strings each paired to a syntactically valid type string;
- Tc1(*d*) is true, i.e., data string *d* minus its enclosing braces conforms to the static requirements of an SDL package body.

The result of NewVar(*d*)(*S*) is a syntactically valid SDL variable string that is different to any contained in the set *S*, as defined by the EqVar function. There are many ways of defining how the new variable string is determined and this Recommendation does not require any particular way to be used.

4.3.4 Eval, Evaluation of expressions

Eval is a function that is required in the computation of the dynamic traces of an MSC. Given a data context and an assignment of values to variables – the current state – the result of applying eval to an expression *e* is its value. Its definition is formally linked to:

a) the *compute* function as defined in 2.1.3.1/Z.100 Annex F3;

b) the evaluation of a function on its operands as defined in 2.1.3.1/Z.100 Annex F3, and

c) the *Eval* function in 3.5/Z.100 Annex F3. The semantics of expressions in SDL is described in 12.2/Z.100.

Eval(*d*)(*e*)(*A*) is defined only if:

- *d* is a syntactically valid data string, *e* is a syntactically valid expression string, and *A* is a set of variable strings each paired to a data value;
- Tc3(*d*)(*S*)(*e*) is true, i.e., expression string *e* conforms to the static requirements of SDL in the context of *d* where its variables have type assignments defined by set *S*.

Note that the above conditions may be satisfied, but the *Eval* function not be defined if the expression is computed outside of its domain; for example, if *e* contains a division by zero. The second clause contains a type tagged set of variables *S* that must correspond to the domain value tagged set of variables *A*; it is assumed that the types can be inferred from the domain values to form the required set *S* from *A*.

The value of Eval(*d*)(*e*)(*A*) is the value that the SDL expression *e* takes in the context of a package having a body corresponding to data string *d*, whenever its variables have the values defined by the state *A*. More formally, the function Eval(*d*)(*e*) is defined to be the same as the function defined by a simple SDL system in which:

- there is an input signal that takes a list of variables as parameters, each variable matching a variable in the state *A*;
- there is an output signal whose sole parameter is defined by expression *e*;
- there is a package whose body consists of the data string *d* minus its enclosing braces.

An execution of such an SDL system will compute and output the value of expression *e*, given an input signal that contain values that are used to assign values to each of the expression's variables. This is illustrated by Figure 4-3 where the value of Eval(*d*)(*e*)(*A*), in which *A* is the set of variable assignments { (*x1*, *a1*), (*x2*, *a2*), … , (*xn*, *an*) }, is equal to the output value of signal s, whenever the input signal r is given the value (*a1*, *a2*, … , *an*). Since *A* is a set but the input r takes a list of parameters, a unique correspondence can be formed by lexicographically ordering the variable strings, so that *x1* will be the variable string that lexicographically comes first, and *a1* will be the first parameter value, etc.

5 Example

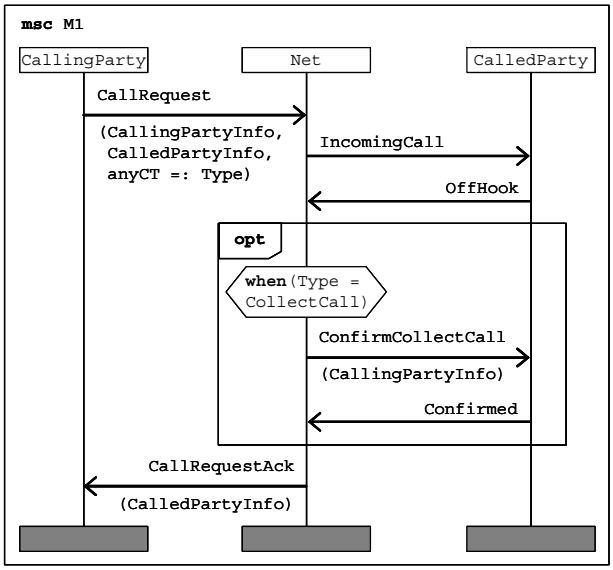
5.1 Use of default SDL interface

```
mscdocument M
inst Net
  variables Type: CallType;
inst CallingParty;
inst CalledParty;
msg  CallRequest: (SubscriberInfoType,
                   SubscriberInfoType,
                   CallType);
msg  ConfirmCollectCall,
     CallRequestAck: (SubscriberInfoType);

language  SDL;
wildcards anyCT: CallType;

data {
  value type SubscriberInfoType {
    literals CallingPartyInfo, CalledPartyInfo;
  }
  value type CallType {
    literals CollectCall, OrdinaryCall;
  }
};
```

M1



SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	**Languages and general software aspects for telecommunication systems**

Geneva, 2003