Date: Tue, 5 Mar 91 10:31:06 EST
From: edbark@radio.cme.nist.gov (Ed Barkmeyer)
Subject: Proposed IDN, as received from Chi Shue

-- General Issues:
-- We need to indicate how extensions are made to this syntax by other
-- standards.
-- We need to write down the scope rules.
-- We need to distinguish between the IDN proposal and example attributes.

IDN Grammar Syntax

1 Interface Structure

<interface> ::= interface <interface_identification>
begin <interface_body> end

/* possible syntaxes for <interface_identification>:

<interface_identification> ::= <interface_identifier>
[version <integer_literal>]

<interface_identification> ::= <interface_identifier>
[compatible with <Object_identifier> [, <Object_identifier>] ...

-- Each interface must have an associated (globally) unique
-- <Object_Identifier>. For convenience, one can also assign
-- a local "name" to an interface that one can locally refer to.
-- Furthermore, this local name can be used within an import list.
-- (See below).

-- NOTE: The usage of a local name to identify an interface is for local
-- usage only. When communicating the interface to a nonlocal entity, an
-- <Object_identifier> must be used.

<interface_identifier> ::= <Object_identifier>
| <local_name> <Object_identifier>

<local_name> ::= <Identifier>

*/

-- List of imported RPC interfaces

<interface_body> ::= [<imports>] <declaration>; [<declaration>;] ...

<imports> ::= import (<import_list>)

<import_list> ::= <import> [, <import>] ...

-- Each import is either identified by a unique Object id or by a local
-- name.

<import> ::= <Object_identifier> | <local_name>

<declaration> ::= <value_decl>
| <type_decl>

| <proc_decl>

2 Value Declarations

-- A value declaration introduces an abbreviation (an Identifier) for
-- a constant value.

<value_decl> ::= value <Identifier> : <type_spec> = <value_expr>

-- For CLIDT, one can construct a constant value for (almost) any type.
-- For RPC, the only value expressions needed are integers, and possibly
-- some others for use within attributes.
-- RPC will therefore restrict the productions in this section.

<value_expr> ::= <Identifier>
| <literal>
| <qualified_value>
| <composite_value>

-- An identifier is a reference to a value declaration (or to a parameter
-- if the value expression occurs on the RHS of a parameterized declaration).
-- A literal is a simple immediate value.

<literal> ::= <integer_literal>
| <real_literal>
| <character_literal>
| <boolean_literal>
| <enumerated_literal>
| <rational_literal>

<integer_literal> ::= [-]<Digit>...

<real_literal> ::= <integer_literal>.<Digit>...

<character_literal> ::= '<Character>' [(<char_set>)]

-- String literals are used in building other literals

<string_literal> ::= "<Character>..." [(<char_set>)]

<boolean_literal> ::= true | false

<enumerated_literal> ::= <Identifier>

<rational_literal> ::= <integer_literal>/<Digit>...

-- A qualified value T.V is the value V interpreted as a value of type T.
-- This is used only when there is a unique injection of V into T.
-- For example, Time."<iso8601-date>".
-- Not used in RPC, more on this later.

<qualified_value> ::= <type_spec> . <value_expr>

-- A composite is used for arrays, lists, tables, and so on. Each such type

-- of composite places additional restrictins on this basic syntax.
-- For example, a record looks like (id: value, id:value, ...)
-- Not used in RPC, more on this later.

<composite_value> ::= ([elt [, elt] ...])

<elt> ::= [<value_expr> :] <value_expr>

<integer_value> ::= <integer_literal> | <Identifier>

3 Datatype Declarations

<type_decl> :== <new_type_decl> | <type_macro>

-- A <new_type_decl> introduces a new type. This is the same as the current
-- CLIDT "new" keyword. This type is not the same as any other structurally
-- equivalent type. The left hand side can contain free variables that
-- are referenced on the right hand side.

<new_type_decl> ::= type <Identifier> [(<Identifier> [, <Identifier>]...)]
= <type_spec>

-- A <type_macro> introduces an abbreviation -- the left hand side is an
-- abbreviation for the right hand side. Unlike a <type_decl>, a
-- <type_macro> DOES NOT introduce a new type. The left hand side
-- can contain free variables that are referenced on the right hand side.

<type_macro> ::= macro <Identifier> [(<Identifier> [, <Identifier>]...)]
= <type_spec>

<type_spec> ::= [<type_attributes>] <primitive_type_spec>
| [<type_attributes>] <generated_type_spec>
| [<type_attributes>] <type_decl_ref>
| [<type_attributes>] <type_spec> <subtype_qualifier>

3.1 Primitive Datatypes

<primitive_type_spec> ::= <integer_type>
| <real_type>
| <char_type>
| <boolean_type>
| <enumerated_type>
| <procedure_type>
| <octet>
| <state>
| <ordinal>
| <time>
| <bit>
| <rational>
| <scaled>
| <complex>

-- RPC restrictions: RPC does not support the types <state>, <ordinal>,

-- <time>, <bit>, <rational>, <scaled>, and <complex>.

-- NOTE: the <octet> type given above is not in CLIDT. It can be viewed
-- as the CLIDT type: array [1 .. 8] of bit.
-- For RPC usages, <bit> is almost never useful (whereas <octet> is).
-- Therefore, RPC uses <octet> as the primitive type in place of <bit>.

<integer_type> ::= integer

<real_type> ::= real (<relative_error>)

<relative_error> ::= <integer_value>

<char_type> ::= character [(<char_set>)]

-- The standard will indicate a default character set to be used if
-- one is not specified.

<char_set> ::= <Identifier>

<boolean_type> ::= boolean

<enumerated_type> ::= enumerated (<Identifier> [, <Identifier>] ...)

<procedure_type> ::= proc (<parameter_dcls>)
[returns (<return_arg>)]
[<exception_list>]

<octet> ::= octet

-- Note: the IDN does not define context_handles or binding_handles
-- as primitive datatypes. The RPC standard, however, defines them as
-- generated types defined in an standard interface that all interfaces
-- implicitly import:
-- type context_handle = array [0 .. c_handleSize] of octet
-- type binding_handle = array [0 .. b_handleSize] of octet

<state> ::= state (<Identifier> [, <Identifier>] ...)

<ordinal> ::= ordinal

<time> ::= time (<relative_precision> [, <radix>, <factor>])

<bit> ::= bit

<rational> ::= rational

<scaled> ::= scaled (<radix> , <factor>)

<complex> ::= complex (<relative_error>)

3.2 Generated Datatypes

<generated_type_spec> ::= <record_type>

| <choice_type>
| <array_type>
| <ptr_type>
| <list>
| <set>
| <bag>
| <table>

-- RPC restrictions: RPC does not support the types <list>, <set>,
-- <bag>, and <table>.

-- NOTE: The array type given here differs from the array type in CLIDT
-- in 3 ways: the index set is always integer, the size of the array
-- can vary (like CLIDT lists), and arrays can be multi-dimensional.
-- Except for the last issue, arrays presented here could be viewed as
-- CLIDT lists. However, the last issue is more intricate. As
-- multi-dimensional arrays are viewed as important by RPC, this document
-- presents the more general array concept. Since there is disagreement
-- on this issue, however, the array concept given here is only
-- preliminary. A joint discussion between CLIDT and RPC committees is
-- needed to iron out these differences.

<record_type> ::= record (<member_list>)

<member_list> ::= <member> [, <member>] ...

<member> ::= [<component_attributes>] <Identifier> : <type_spec>

<choice_type> ::= choice (<member_list>)

-- The index of each array dimension is implicitly always integer.
-- The array_bounds_list is a sequence of subtypes, one for each
-- array dimension. Each subtype is a range of integer values,
-- where either (or both) sides of the bound are allowed to be "*",
-- indicating an indeterminate bound. An empty range signifies [0 .. *].

<array_type> ::= array <array_bounds_list> of <type_spec>

<array_bounds_list> ::= <array_bounds_declarator>
[<array_bounds_declarator>] ...

<array_bounds_declarator> ::= <[> [<array_range>] <]>

<array_range> ::= <array_bound> .. <array_bound>

<array_bound> ::= <integer_value> | *

<ptr_type> ::= pointer_to <type_spec>

<list> ::= list_of <type_spec>

<set> ::= set_of <type_spec>

<bag> ::= bag_of <type_spec>

<table> ::= table_of (<element> , <key>)

<element> ::= <type_spec>

<key> ::= <type_spec>

3.3 Type Declaration References

-- A <type_decl_ref> is reference to a new type declaration or a type macro.
-- If the reference is only a single identifier, and it occurs on the RHS of
-- a parameterized declaration, it also may be a reference to a parameter.

<macro_instance> ::= <Identifier> [(<expr_list>)]

<expr_list> ::= <expr> [, <expr>] ...

<expr> ::= <type_spec> | <value_expr>

3.4 Subtypes

<subtype_spec> ::= <range> | <max> | <min> | <plus> | <restrict> | <view>

-- The range subtype can be used on any ordered type. A precise bound must
-- be a value of that ordered type. The infinities are only usable in
-- subtyping integer and real.

<range> ::= range (<lower_bound> , <upper_bound>)

<lower_bound> ::= <precise_bound> | neg_infinity

<upper_bound> ::= <precise_bound> | pos_infinity

<precise_bound> ::= <value_expr>

-- The max and min subtypes can be used to form subtypes of the list, set,
-- bag, and table types. They bound the size of the aggregate.

<max> ::= max (<integer_value>)

<min> ::= min (<integer_value>)

-- Plus is CLID's extended. Restrict is CLID's selected.
-- View is CLID's explicit subtype. These 3 are NOT used in RPC.

<plus> ::= plus (<expr_list>)

<restrict> ::= restricted_to (<expr_list>)

<view> ::= viewed_as (<type_spec>)

4 Procedure Declarations

<proc_dcl> ::= [<proc_attributes>]
                proc <Identifier> ([<parameter_dcls>])

[returns (<return_arg>)]
[<exception_list>]

<parameter_dcls> ::= <param_dcl> [, <param_dcl>] ...

<param_dcl> ::= <direction> <parameter>

<parameter> ::= [<param_attributes>] <Identifier> : <type_spec>

<direction> ::= in | out | inout

<return_arg> ::= [<param_attributes>] [<Identifier> :] <type_spec>

<exception_list> ::= raises (<exception_dcl> [, <exception_dcl> ...])

<exception_dcl> ::= <Identifier> ([<parameter> [, <parameter>] ...])

5 Attributes

<type_attributes> ::= <[> <type_attribute> [, <type_attribute>] ... <]>

<proc_attributes> ::= <[> <proc_attribute> [, <proc_attribute>] ... <]>

<param_attributes> ::= <[> <param_attribute> [, <param_attribute>] ... <]>

<component_attributes> ::= <[> <comp_attribute> [, <comp_attribute>] ... <]>

-- The rest of this section contains proposed attributes for RPC.

5.1 Type Attributes

-- There are two kinds of <type_attributes>.
-- (I) endpoint-specific attributes specify how to map the
-- interface type to a particular implementation on either
-- the caller/callee side (or both). An endpoint specific attribute
-- contains only local mapping information; therefore, it does not
-- effect any protocol. Examples include: how to map the choice type
-- to discriminated unions.
-- (II) Contractual attributes (also called type_restriction_attributes)
-- contain information that both sides must
-- agree to. These attributes may affect protocol. Examples include:
-- type restrictions that indicate a more efficient wire encoding of
-- the datatype (e.g., sparse), a restricted use of pointers (e.g.,
-- unaliased) that allows a more efficient marshalling routine to be
-- used, etc.

<type_attribute> ::= <type_restriction_attribute>
| <endpoint_specific_attribute>

-- The sparse attribute is used in conjunction with the array, list,
-- set, bag, and table types. It indicates that most values of the
-- aggregate are expected to have a default value.
-- The unaliased attribute is used to indicate that a pointer structure
-- contains no aliasing. It can be used only in conjunction with a

-- pointer type.
-- The nonnull attribute is used to indicate that a pointer cannot have
-- a null value. It can be used only in conjunction with a pointer
-- type. This attribute only makes sense if pointers are allowed
-- to have null values, a still unresolved issue.

<type_restriction_attribute> ::= sparse (<value_expr> | <Identifier>)
                                | unaliased
                                | nonnull
                                | <lifetime>
                                | <procedure_attribute>

-- The <lifetime> attribute indicates that a procedure parameter (i.e., a
-- closure) may only have a restricted lifetime. In particular, the
-- environment in which the procedure parameter lives may only exist
-- during the current call (i.e., the procedure parameter is a "callback")
-- or during the duration of the current caller/callee binding.

<lifetime> ::= callback | extended_callback

-- An <endpoint_specific_attribute> may be qualified by the keyword
-- "caller"/"callee" to indicate that the attribute applies only to
-- one endpoint.
-- The local_representation(X) attribute is used to indicate that a given
-- type is represented by a local type X. The encode/decode
-- attributes provide the names of local routines to encode and decode
-- the interface type to/from a local representation.
-- The <switch_type> attribute is used to indicate a "switch" type
-- that is used locally to discern what case a choice is in. (An
-- example is given below).

<endpoint_specific_attribute> ::= caller (<endpoint_specific_attribute>)
                                | callee (<endpoint_specific_attribute>)
                                | local_representation(<Identifier>)
                                | encode(<Identifier>)
                                | decode(<Identifier>)
                                | switch_type (<s_type>)

<s_type> ::= <integer_type>
        | <char_type>
        | <boolean_type>
        | <enumerated_type>
                | <Identifier>

5.2 Procedure Attributes

<procedure_attribute> ::= at_most_once | idempotent

5.3 Parameter Attributes

-- NOTE: the following param attributes are open for discussion.
-- Param attributes indicate a restricted relationship that
-- must exist between the in and out values of an inout parameter.
-- The same_structure attribute is used in conjunction with an inout

-- pointer parameter to indicate that the "topology" of the structure
-- pointed to by the param does not change during the call.
-- The same_size attribute is used in conjunction with an inout open array,
-- list, set, bag, and table parameter. It indicates that the size of the
-- aggregate does not change during the procedure call. The same_case
-- attribute is used in conjunction with an inout choice parameter to
-- indicate that the case of the choice does not change during the
-- procedure call.
-- The <dynamic_information> attribute is used to inform the marshalling
-- routine where certain runtime information can be found to help in
-- marshalling (see below).

<param_attribute> ::= same_structure
| same_size
| same_case
| <dynamic_information>

5.4 Component Attributes

-- The <discriminant_value> attribute is used in each arm of a choice
-- in order to identify the particular arm of the choice with a
-- set of values from the <switch_type> domain. It must be used in
-- conjunction with the <switch_type> attribute.
-- The <dynamic_information> attribute is used to inform the marshalling
-- routine where certain runtime information can be found to help in
-- marshalling (see below).

<comp_attribute> ::= <discriminant_value>
| <dynamic_information>

<discriminant_value> ::= case (<d_value> [, <d_value>] ...)

<d_value> ::= <value_expr> | default

-- The <discriminant_is> attribute is used to tell the marshalling routine
-- where the discriminating value for a choice is. (If the choice is
-- embedded within a record, then the discriminating value must be
-- another component of the record. If the choice is a parameter of a
-- procedure, then the discriminating value must be another parameter
-- of the procedure.) Similarly,
-- the <bounds_is> attribute is used to tell the marshalling routine where
-- the bounds of a conformant array or a varying array can be found.
-- The caller attribute is used to indicate that
-- a component of a record or a parameter of a procedure call is provided
-- by the caller only for use by the runtime. It is not part of the
-- type and is not to be transferred on the wire. A typical usage of
-- this attribute is in conjunction with the <discriminant_is>
-- and <bounds_is> attributes. Upon return from the call, this component
-- or parameter is reconstructed by the runtime.
-- Similarly, the callee attribute indicates that a
-- component or parameter is not part of the type and is not transferred on
-- the wire. Instead, it is constructed by the runtime and provided to the
-- callee. A typical usage of this attribute is in conjunction with
-- the <discriminant_is> and <bounds_is> attributes.

```
<dynamic_information> ::= <discriminant_is>
                        | <bounds_is>
                        | caller
                        | callee

<discriminant_is> ::= discriminant_is ( <Identifier> )

-- One can now write the following in an interface:
--
-- declare T = [switch_type(integer)]
--          choice([case(5)] a: int,
--                 [case(10)] b: real,
--                 [case(default)] c: boolean);
--
-- proc foo([caller] x: integer, [discriminant_is(x)] y: T)

<bounds_is> ::= first_is ( <attr_var_list> )
              | length_is ( <attr_var_list> )
              | min_is ( <attr_var_list> )
              | size_is ( <attr_var_list> )

<attr_var_list> ::= <attr_var_dcl> [ , <attr_var_dcl> ] ...

<attr_var_dcl> ::= [ <attr_var> ]

<attr_var> ::= <Identifier>
```