From: Ed Greengrass

To: WG11

Subject: Response to Turba's Comments on CLIDT WD 5 (WG11/N262), part 1 of 3

Date: Fri, 12 Jul 91 16:35:55 -0400

I have recently received Tom Turba's comments (SC22/WG11 document # 168) on Common Language-Independent Datatypes Working Draft #5, X3T2/91-109 JTC1/SC22/WG11 N233. I disagree with his views on Null and Undefined. Indeed, I have written several papers on these subjects: WG11/N211, WG11/N224, and WG11/N258 (the latter discusses a number of outstanding CLIDT issues). I answer Turba's comments with excerpts from the latter:

****

(1) Turba: A null record or null variant is not a null datatype. It is simply a record or variant that has no content. Its datatype is that of a record, array, etc.

****

Answer:

Choice of a Null type as an alternative to type T should not be confused conceptually with an "empty" variant though it might be REPRESENTED that way. For example, it has been argued that:

```
record of (
    a: integer
    b: choice of (
      integer,
      real,
      null
    )
)
```

is conceptually equivalent to :

```
record of (          record of (          record of (
    a: integer,          a: integer,          a: integer,
    b: integer           b: real          )
)                    )
```

No! This view blurs important distinctions. There are at least two important distinct cases:

Case 1 (the variant record case):

We are defining variations on the SAME record. For example, below I discuss the case of a Personnel record containing a Name of Spouse field. In that case, choosing type Null for the Name of Spouse field means that the person is unmarried and hence doesn't have a spouse. This is an important piece of semantic information. Whether it is represented by a null value or by absence of the field altogether is purely a representation issue. Conceptually, the field has type Null.

Case 2 (the multiple record type case):

We are defining two or more DIFFERENT types of record which may appear in the same "position". For example, in the above example, the integer "a" might be a record type field so that each value of "a" corresponds to a different type of record. In that case, it might be quite reasonable to say that for one record type, the field "b" is absent because "b" might have no meaning for that type of record. But of course, in that case, it would be more correct conceptually to write:

```
choice of (
record of (
    a: integer,
    b: integer
),
record of (
    a: integer,
    b: real
),
record of (
    a: integer,
)
```

to express the idea that three DIFFERENT record types are involved.
Alternatively, one might write:

```
record of (
    a: integer,    /* record type */
    choice of (
    body_of_record_type_1,
    body_of_record_type_2,
    body_of_record_type_3
    )
```

where, in the present example, the body is trivial: b: integer, b: real, or b: null.

****

(2) Turba: Representation of the absence of a value in a position where something may or not be found in the search of a database does not require the invention of a null datatype. What is normally returned in such positions is a set of objects, where that set may be the null set of those objects.

****

Answer:

Consider a personnel record with a field Character String for Name of Spouse. What should that field be for an unmarried person? Answer: an instance of type Null, meaning that there is no spouse. Note that this is NOT a null character string but an instance of the distinct type Null (which has only one value, "null", in its domain). We do not want to return a null set of objects but a value asserting that the given person does not have a spouse.

It should be stressed that many types have values that can be confused with Null or even used to REPRESENT Null. Lists, Tables, Sets, etc, can be empty, Integers can have the value zero, Characters can have the value blank, and so on. However, none of these is conceptually equivalent to Choice (T, Null) for type T. Barkmeyer has an excellent discussion of these various cases in SC22/WG11 N216.

****

(3) Turba: [The datatype of a null record, null array, etc] is that of a record, array, etc.

****

Answer:

One COULD attach Null as a value to every CLIDT type, i.e., instead of Choice (T, Null), one could specify that every type T has a special Null value. This would be very messy because the normal characterizing operations of type T would not apply to its Null value (if Integer has a null value, what is 3 + null?) and one would have to add a new Null() predicate operation to every type T. It is surely much cleaner to define Null as a separate type.

****

(4) Turba: Undefined is not a datatype!!! It does not have any values; it does not have any properties on those values (because there are no values); and it does not have any operations on those values.

****

Answer:

Suppose that the spouse in the above personnel record DOES exist but we don't know his/her name because the employee specified his marital status as "married" but accidentally left the "name of spouse" field blank on his entry form. In that case, we choose to represent "Name of Spouse" as a value of type Undefined. This means that we know the spouse exists but we don't know his/her name. The spouse's name "value" is undefined in the sense that it hasn't been specified to us. A better term for this datatype is "Unknown". Hence, Null is a datatype whose value EXPLICITLY denotes non-existence. Undefined (or as I prefer, Unknown) EXPLICITLY denotes that the value DOES exist but is not currently known. Its semantics is, "This variable has a value but the value is currently unknown". In different application domains, this value would have more specific meanings, such as, "X refuses to supply the value," "X failed to supply the value," "The value has not yet been obtained but will be," "The value is unobtainable," etc. Following Barkmeyer, one might define Unknown as State (refuses_to_supply, failed_to_supply, not_yet_obtained, unobtainable, ...).

A comparison between two objects of type Unknown IS defined. For instance, in my example of a record with spouse's name unknown, equality of records would mean, "In both these cases, the individual in question has not supplied the name of his spouse." If the records represented successive points in time, the meaning would be, "The individual has STILL not supplied the name." If records of two different individuals were compared, a comparison of the "name of spouse" fields would mean, "These individuals have both not supplied the names of their spouses." It could be quite reasonable to ask for the names of all employees who have not supplied the names of their spouses. Needless to say, equality of unknown spouse names does NOT mean that the names are equal, but rather that they are equally unknown.

****

(5) Turba: The correct classification for undefined is that it is a state of a variable or field that can contain a value of a datatype. It is the state when the variable or field has not been attributed a value of its datatype.

****

Answer:

How does the Unknown type, as discussed above, relate to Turba's Undefined status? One can certainly create a personnel record, assign values to its fields but NOT to the field "name of spouse". Does this mean that "name of spouse" is Undefined in Turba's sense? Not necessarily. To decide, it is necessary to examine the state and logic of the program involved. For example, suppose program P creates a personnel record and immediately assigns values to all its fields. If P is stopped or interrupted before the assignment to "name of spouse" can be executed, then "name of spouse" is indeed Undefined in Turba's sense. However, suppose P creates personnel records by obtaining its data from a heterogeneous collection of databases. Further, suppose that all of these databases record marital status but some of them do not record data about spouses. If P tries to create a record about married employee X and finds that data about X is available only in a database that does not support spouse data, then AT THAT POINT "name of spouse" passes from Undefined to Unknown; P should express this by setting "name of spouse" to type Unknown, value = unavailable.

The distinction is not trivial. The fact that "name of spouse" is Undefined only tells us something about P, i.e., that P has not gotten around to assigning a value to "name of spouse" for X. The fact that "name of spouse" for X is Unknown tells us something EXTERNAL to P, something about the "real world" or at any rate about the environment in which P operates. Hence, Unknown is a value of a datatype. Undefined is a status of a variable corresponding to the current state (the logical program counter) of program P. It follows that to compare two Undefined variables only tells us something about P (if that). Comparing two variables of type Unknown tells us something meaningful about the world outside of P itself.