From: Brian Meek, 16 July 1991

To: WG11

Subject: Brian Meek's comments on Tom Turba's piece on CLID

>> Comments dated 16 July 1991

>> Tom, thanks very much for this contribution. It is exactly the kind
>> of feedback needed if the language groups and WG11 are to reach mutual
>> understanding.

1. Introduction

>> No comments, so omitted here

2. General Comment

None of the datatype concepts of object-oriented programming, e.g., subclassing and inheritance, are addressed in this document.

>> inheritance is certainly outside the scope of CLID though CLID does not
>> preclude distinguishing between datatypes by inheritance rules

This is extremely unfortunate,

>> disagree! it was a conscious decision - reached after quite a debate!
>> - not to attempt to venture into the OOPL area. My estimate is that
>> doing that would delay the CLID standard by at least 2 years, probably
>> more, because it is an open-ended can or worms. This wouldn't matter
>> except that CLID as it is can do a lot of useful work in a whole raft of
>> areas where OOPS is unheard-of (in that form). I'd support the concept
>> of an OOCLID for a 1995 supplement or 1997-ish revision, but not now.
>> I actually think experience of a non-OO CLID would bring that goal closer.

especially when common terms such as "subtype" are used in this area but have a definition that is more inclusive.

>> I do not understand what the last two words are supposed to mean.
>> However, clearly we should try to specify CLID notions in a way that will
>> not constrain any languages, OO or not. I am delighted that Keld Simonsen
>> has been appointed WG21 liaison to WG11 and hope that he, Tom and others
>> will help us through any difficulties in this area.

3. Undefined Datatype 7.1.14

Undefined is not a datatype!!!

>> We-e-e-ll.... See below after Null. I don't understand the emphasis
>> here as opposed to the muted objection to Null

The notes under Outstanding Issues suggests that there is an underlying confusion here. Maybe the comments here can add light to the issue.

By your own definition of a datatype in 6.1, "a datatype is a collection of distinct values, a collection of properties on those values and a collection of characterizing operations on those values", 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.

Either the definition of a datatype must be warped into something I cannot even imagine so that it can accommodate undefined, or more correctly, undefined should not be classified as a datatype.

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.

>> see below about the 'state' argument. I sympathise, but WG11 has had this
>> argument.

4. Null Datatype 7.1.13

Null should not be a datatype!

>> On the contrary, it is ESSENTIAL - though that does NOT necessarily
>> invalidate the remarks that follow! Again, see below!

From the comments under Outstanding Issues it is clear that there is considerable confusion here. This is probably, in part, due to the fact that there are several distinct semantic concepts that are attempting to be rolled into one.

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.

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.

>> OK let's see if we can sort this out.

>> There are essentially 3 approaches to the vexed NULL/UNDEFINED problem.

>> One is to say that NULL and UNDEFINED are states of an entity which would
>> normally have a value of a given (possibly Choice) datatype, i.e. aren't
>> values at all. The problem with this is that you can then envisage
>> different kinds of (particularly) undefined state, e.g. things never
>> defined (e.g. declared but not initialised), and becoming undefined in
>> various ways that some people might want to distinguish, cf. PL/I ON
>> conditions; and can even make out a case for user-defined states.
>> WG11 did seriously consider this option, indeed I advocated it for a
>> time. It was finally abandoned primarily I think because it was so
>> open-ended. You still need a null datatype as well, with this approach
>> (for Choices).

>> Another is to say that every datatype has NULL and UNDEFINED in its
>> value set. This means redefining the "conventional" - and I think that
>> means "mathematical" - definitions of certain familiar dataypes to include

>> them. Some people get very uncompfortable with that though I agrees,
>> personally, that a null or undefined Array 1:10 of Real is still an
>> Array 1:10 of Real, not any old null. However, again you need a null
>> datatype as well, as before.

>> The third is that used in the CLID CD, of just having Null and Undefined
>> datatypes. The disadvantage is that actual datatypes in actual languages
>> mave to be mapped taking this into account, e.g. Integer maps not to CLID
>> Integer of some unspecified range, but Choice of that or Null or
>> Undefined. People find this hard to get their heads round; it seems
>> unnatural. IS unnatural, some would say. Its advantages are that it is
>> not openended like 'state of', and it applies Occam's Razor, you don't
>> use a multiplicity of concepts when one will do. The 'null here is not
>> just any old null' doesn't in fact apply - if the null arises from a
>> Choice of, the Choices determine what it's a nulll of in a particular
>> case. Furthermore the same old Null does duty in Choice itself!

>> Neverthless my PERSONAL preference is for the second possibility rather
>> than what we have now. CLID value spaces are computational, not
>> mathematical it worries me not one jot that there's a null and an
>> undefined for every datatype. However, I'm also a polymorphist, saying
>> that Datatype is itself a datatype, i.e. objects can take on Datatype
>> as a value, and among those values, for orthogonality, MUST be Null,
>> we knew that already, and indeed Undefined as well.

>> I can actually live with any of the three, if they are carried through
>> consistently. What I dislike are the ideological arguments that say it
>> HAS to be one or another to be acceptable! (I'd better explain that
>> wasn't at all meant personally!)

5. Procedure Datatype 7.1.16

There is considerably more syntax than semantics and I can only guess at what some things might be. For example, what is an exception-argument-list in exception? Considerable more description is needed.

The rules for subtypes disagree with some current languages such as C++.

I would hope that the "call" operation would be allowed for values of type procedure.

>> CLID is about concepts rather than specific syntax and what a procedure
>> call actually does is a matter for CLIP, not CLID. Characterising
>> operations are only for identification purposes, I've no objection to
>> CALL being there but it is really redundant in the CLID context.

>> Personally I favour procedure and pointer types being separated out
>> and aggregate types grouped together. Aggregates naturally go together,
>> but procedures and pointers (which can themselves be regarded as
>> specialist procedures) are things which manufacture new values of (in
>> general) different datatypes from those you start with, a kind of
>> packaging which is dynamic rather than static. I suspect that kind of
>> reorganisation, taking Procedure out of primitive and the aggregates
>> out of generated into a new 'aggregated' category, might help to clarify

>> this aspect of CLID.

6. Complex Datatype 7.1.12

This datatype assumes an implementation strategy based on Cartesian coordinates,

>> This statement is untrue; hence the rest, though true, is irrelevant.

which is not the only means by which complex numbers can be implemented. This is unnecessary and conflicts with the definition of the complex datatype in languages such as Extended Pascal.

7. Annex A

Requiring that Choice be a required datatype generator seems a bit strong, especially for languages like FORTRAN that only indirectly support the concept. What is the rationale for requiring it?

>> The rationale is given above, you need it to do mappings including Null
>> etc.

Also, why are Lists a required datatype generator? They are not nearly as essential as records or arrays, and many languages do not directly support them.

>> I agree, but if things like character strings are defined as List of
>> Character etc (which I believe is wrong but that's a different issue)
>> then it has to be there; though it is no reason for excluding arrays etc.

>> However, I am not actually convinced we should have Annex A at all,
>> any more. As with language features generally, defining a 'core' which
>> will command general consensus is very difficult.

>> However, languages not directly supporting anything is not in itself a
>> reason for not having something in the required list, all it means is
>> that if you don't have it you have to define a simulation in the
>> mapping standard (so that all processors supporting it do so in a
>> consistent way). The argument for an Annex A is that it ensures that
>> in ANY mapping you can always rely on having at least these available,
>> in one form or another.