Date: Tue, 5 Mar 91 15:18:17 EST
From: edbark@cme.nist.gov (Ed Barkmeyer)
Subject: RPC, CLIPC and subsetting the CLI Datatypes

In a recent meeting on another international standard, a request for an additional feature was denied because a majority of those in attendance felt that it was unimportant and would be infrequently used. Those siding with the request were heard to say: I know how I will EXTEND the standard to support this in MY implementation. I thought these statements sufficiently menacing to re-raise the issue in the committee. If there are going to be several products which extend the standard in a common direction, it is wise to standardize the feature BEFORE they are all different. I am beginning to have similar concerns about subsetting the CLID in the RPC and CLIPC standards.

There are several reasons for declining to add a requested feature to a standard:

1) the purpose of the feature is inappropriate to the standard;

2) the feature is adequately supported by existing features;

3) the feature is too complex and affects the standard in too many ways for it to go in without more analysis and experience, seriously delaying the standardization process;

4) there is no committee consensus on how to add the feature or how to implement it;

5) the feature would place a significant burden on implementors (without significantly affecting most users);

6) the feature is unlikely to be used;

7) the feature is only one of many of its kind and the feeling is that there is a kind of "level of completeness" at issue.

Vis-a-vis the CLI datatypes and their inclusion in RPC/CLIPC, the motivation for subsetting that has been stated is of the 5 and 6 variety. I'd like to reconsider the inclusion/exclusion decisions, in the light of probable user-group extensions and the arguments against their inclusion:

1. Boolean - included.

2. State - excluded. Treated as equivalent to Enumerated, which is the type in programming languages. => (6) will not be used. The problem is that there should be only one datatype, but it should be "State" not "Enumerated". Enumerated forces the representations to be ordered, so that the "named integer" convention of Pascal (and Fortran, by the way) is the paradigm. This makes the "character selector" conventions of COBOL record-types and the "named bitstring" conventions of real-time programming unrepresentable as Enumerated datatypes, although they are clearly State datatypes. I would argue for eliminating the Enumerated datatype from the CLID and giving the name Enumerated to the current State datatype. This will confuse the C programmers into believing they got what they wanted, while giving them what they need and what many others can use.

3. Enumerated - included, see State.

4. Character - included.

5. Ordinal - excluded, reason 6. I agree. Ordinal is for conceptual schemas and precise mathematical descriptions. In practice, Integer will do.

6. Date-and-Time - excluded, reason "it's not in programming languages", which I assume amounts to "not used". But clearly "not used" is false. Dates and times are used by many applications - business and financial transactions, manufacturing control and scheduling, activity, personnel and resource scheduling in hundreds of disparate industries. The CLID datatype is probably too general, and needs to be restricted in CLIPC/RPC, but this is one type which will suffer from multitudinous mutually incompatible user

definitions if it is not standardized. Date will be MMDDYY, YYMMDD, and DDMMYY in CharacterString and Array of Integer form, and in some of the character string forms, dots or slashes will separate the elements - 12 or more mutually incompatible representations of a common information unit. And Time is even worse. The INTERNAL representation of dates and times should indeed be up to the program and language, but there should be a common INTERCHANGE notion and representation for so common an information unit.

7. Bit, included, sort of, via "Octet". In fact, Octet has the semantics of Private, and Bit/BitString are not included at all. Reason 5. I'm not sure that Bit is distinct from Boolean, conceptually or otherwise, and therefore I think the right reason is 2, but it certainly doesn't need to be included. On the other hand, BitString, whether it is Array of Bit or List of Boolean, should be considered (see below).

8. Integer, included.

9. Rational, excluded, reasons 5 and 6. I agree. Rational is for conceptual and documentary purposes, I guess, or perhaps its the "level of completeness" argument.

10. Scaled, excluded, in lieu of Real, I suppose, which is reason 2. This is erroneous and the exclusion is a mistake. All implementations of Scaled are fixed-point (machine Integer) or (packed or zoned) decimal, in order to get the required accuracy. The absence of Scaled in ASN.1 has caused two applications to my knowledge to choose to represent Scaled as Integer or CharacterString (even though ASN.1 Real really does support this). Scaled is the principal datatype of COBOL and it is REQUIRED for many financial transactions, manufacturing and defense measurements. Again, the INTERNAL representation in some languages and programs would have to be selected by the application programmers, but when the language is appropriate to the application (COBOL, PL/I, SQL), it will be directly supported. Users should not have to determine an exchange form for a datatype common to several languages and many applications. If they do, they will end up with several incompatible conventions.

11. Real, included, but converted to "scientific number", not to say floating-point. I believe this is the proper conceptualization of the datatype in the CLID. Mathematical REAL is NOT a "computational" datatype.

12. Complex, excluded. I have heard reasons 2, 5 and 6. Since there are multiple possible representations of Complex, which have different properties, reason 2 is erroneous: Complex is NOT the pair of reals (real, imaginary). Reason 5 is just plain nonsense: adding Complex to the list of interchange datatypes is a drop in the bucket compared to the cost of an RPC implementation. There is merit in 6 - Complex is infrequently used. But the fact that many believe the LCCAPS is worth doing indicates that there is a Complex user community, and the type IS supported in Fortran and Ada. Complex is less significant than Scaled, but it should be supported in the CLIPC/RPC.

13. Null and Undefined, excluded. These need to be cleared up in the CLID, but some form of support for the concepts will be absolutely necessary in procedure calling. What is the value of an OUT parameter which is not returned?

14. Private, included under the name "Octet(string)". Naming it for its implementation is crude, but what do you expect from C programmers? The name should be changed, in both documents, to "Opaque". Length, in bits or octets, is a required implementation attribute of opaque types.

15. Procedure, included. There are syntactic problems with the RPC concept of "procedure type" when viewed from a CLID perspective. These can be ironed out, but some work is needed, largely in the CLID.

16. Subtyping, somewhat included. The IDN compiler should support subtype declarations of all types. However, in my view, little or none of the subtyping should be used to constrain the RPC implementation. It appears that Range is useful in the RPC implementation, but other subtypes should probably be

considered in CLIPC and RPC to be the same as the base type for implementation purposes. The documentary value in the IDN may still be significant.

17. Choice, included.

18. Record, included.

19. Pointer, included, with restrictions. The restrictions are important to avoid problems of type 4 and 5.

20. Set, excluded. It appears that the reason is 2: everything can be supported by Array. That is conceptually false - arrays have significantly different semantics and mapping Pascal/Ada SET types into arrays prevents their use in package interfaces. RPC/CLIPC should support Set, but map it into arrays in the implementation if that is what is wanted. If you force the programmer to do the mapping, you will get many undesirable results, and the default mapping of Set into Pascal and Ada may be different.

21. List, excluded. The reason is stated to be 2: it can be done with arrays. I believe that this is almost correct, but backwards. The general object is a List - the array is a special case. Specifically the Array is a List of values, with the subscript mapping function attached. In CLID terms, that is a NEW List datatype. The concept List is what is really supported in the RPC and actually passed between caller and receiver, along with (sometimes) the parametric definition of the subscripting function. This also needs to be spelled out in the CLID itself, and applies to Table as well as Array.

22. Bag, excluded, reasons 2 and 6. Again, my view is that the fact that it can be done with arrays (or Lists) doesn't make it semantically a List. The IDN compiler should accept Bag, but CLIPC and RPC should implement it as a curious spelling of List.

23. Array, included, but modified to multidimensional. See my comments under List. Multiple dimensions is a property of the attached subscripting function. A subscripting function logically maps a tuple of values of the index types into an ordinal value which is used to select an element from the List. This model, by the way, differs from the Array of Array model, in which each index value selects from a separate list.

24. Table, excluded. A Table is just a List with a different kind of "subscripting" function, but it probably needs to be modelled as a List of pairs (Record(key, element)), regardless of how it's implemented. This should be either supported by the RPC/CLIPC or made into a declared datatype in the CLID. The problem here is reason 6: will anyone use it?

25. Declared types and declared generators, included. The RPC syntax document makes this somewhat fuzzy.

26. BitString, excluded per se, but Array of Boolean is supported. BitString is defined in the CLID precisely for the purpose of giving implementations a hook on which to hang optimization of (interchanges of) "array of bit/boolean". If Bit and BitString are not part of the RPC/CLIPC, it's not clear what function they serve in the CLID - they are not very conceptual datatypes.

27. CharacterString, included.

28. Switch, Cardinal, Integer Modulo and Currency seem to be exercises in standardizing names for datatypes which are adequately supported by the CLID primitive datatypes, and whatever implementations thereof may be developed.

29. Stack, excluded, presumably supported by Array. Stack can only be supported by Lists which are allowed to vary in size between the call and return of a procedure. It is not clear that this is supported by the RPC "array" concept. The same argument can be made for Lists, but since List is the generic datatype,

the "variable-length" concept is subject to several interpretations. Stack makes the problem quite clear.

30. Modulo, excluded, reason 6. Modulo provides cyclic enumerated datatypes. Someone will probably want one in the next 10 years, but don't hold your breath. Modulo should be deleted from the CLID, along with enumerated datatypes.

It also appears that the RPC effort is defining a whole new collection of "Pragmata" or "attributes" or "annotations", and those which are not specific to communication should definitely be part of some common lexicon, along with whatever remains of the list in Annex D of the CLID.