SC22/WG11/N261

From: Ed Greengrass 6/4/91

To: WG11

Subject: Response to Barkmeyer's Suggested CLID and IDN changes per Arles
(WG11/N259)

The referenced message contains some comments on generated types, primitives, pointers and procedures. Some points of disagreement:

(1) Is the difference between primitive and generated purely syntactic? It may be true that a generated type is specified syntactically in terms of its base or component types. However, it is not purely syntactic that for most of those types, one can obtain uniquely the values of the base types from a value of the generated type. (I once called that decomposition.) That seems to me an essential semantic property, especially of so-called "aggregate" types; in fact, it is exactly what justifies calling them aggregate types. This leads to:

(2) Should one distinguish the dichotomy simple/generated from the dichotomy primitive/aggregate? First, a point of terminology. I introduced the term "generator" precisely to get away from terms like "construct", "aggregate", "compose", etc, which seemed to me to have an implication of PHYSICAL combination. I suppose one could use the term "logical aggregate" as Barkmeyer at one point suggests, but that is a trifle more awkward. On the other hand, one can continue to use "generator" for the semantic "aggregations" and use "parameterized" types for the purely syntactic case. Which brings me to:

(3) What is Pointer in terms of these categories? Is Pointer a primitive? In my latest opus, SC22/WG11 N258 (does anybody read these things in spite of their length?), I explain very clearly why Pointer is NOT a primitive type. Specifically, I point out that there is a crucial difference between the primitive type UID (Unique Identifier) and the type Pointer which draws its value space from UID's. A UID-valued field in

Record of (intval:Integer, nextval:Pointer)

is type Pointer because it makes sense to apply a dereference operation to it to obtain the object to which it points. But, a UID-valued field in

Table of (T, UID)

is type UID because the UID functions as a key, identifying the object (the table row) in which it occurs, not pointing to any other object. Hence, dereference would make no sense here and the type is NOT pointer even though the value space may be the same. (Note: In the April X3T2 meeting in Gaithersburg, and subsequently in my SC22/WG11 N247, it was pointed out that each element - "row" - in Table of (T, UID) is an Object of type T.)

(4) If Pointer is not a primitive type, is it an "aggregate" type? Yes, because from a value of the base type Object (instance) of type T (a T, UID pair drawn from an Object-of-type-T Table), one can generate a unique value of the "generated" type, i.e., the UID that identifies (points to) the given object. And one can then "assign" the UID to a Pointer, i.e., associate the UID with an instance of type Pointer. From the value of a Pointer one can obtain (by dereference) the unique object pointed to. The term "aggregate" is less satisfactory here because a Pointer points to (is generated from) a single object. However, the fact that one can go from the pointer value back to the unique object it identifies is a semantic, not a purely syntactic property of pointer. (As is the fact that multiple pointers can point to the same object.) Hence, Pointer is a "logical aggregate" or as I would prefer, a "generator".

(5) Is Procedure a primitive? In SC22/WG11 N228, I argued that Procedure is neither a generator nor a

primitive but a "generated type". I was wrong as Note 5 under Procedure in SC22/WG11 N233 (CLIDT WD 5) points out. However, a small part of my N228 argument on Procedure seems to me salvageable. I now think that Procedure is a generator/aggregate/whatever.

CLIDT WD 5 regards Procedure as a primitive type because Apply operates on the entire value of a procedure to produce a value (or values) of what may be an entirely different type. However, it seems to me quite a fundamental characteristic of type Procedure to be able to examine and work with the components separately, e.g., to analyze the formal computational properties of its algorithm or to ask what datatypes the procedure requires, etc. This would be analogous to selecting fields of a record.

There is an important distinction between asking for the datatypes of the formal parameters of a Procedure and asking whether an Integer is prime. Prime is a property that some integers possess but the boolean "prime = yes" or "prime = no" is not one of the defining components out of which one generates an integer. On the other hand, the algorithm and formal parameters are precisely what one generates (defines) a procedure out of.

It is necessary to keep clearly in mind the distinction between a mapping from algorithm, datatypes of formal parameters, etc, to a value of datatype Procedure (as defined in WD 5) AND the mapping between the values of the operands in a given procedure call (as specified to Apply) and the result values returned by that procedure call. It is definitely possible to perform the first kind of mapping in both directions, just as one can go from values of components (field values) of the generator Record (more precisely, a generated record type formed by specifying the field types) to the corresponding value of a generated-record and then retrieve the values of the individual fields from the value of the generated record. Hence, there is some justification for calling Procedure (in the WD 5 sense) an aggregate.

What about the second kind of mapping (which is performed by executing Apply for a given procedure and set of operand values)? One can certainly perform the mapping from operand values to procedure result (except for operand values for which the procedure is not defined or doesn't terminate). That is exactly what Apply does. On the other hand, one can't (in general) map uniquely from procedure results to input operand values. Hence, Apply does not define an aggregate (uniquely decomposable) kind of mapping but Procedure does.

Hence, I disagree with the statement in Note 5 that "[t]he values of a procedure type ... are not in any wise composed of the values of their argument types, nor are the argument values (in the general case) in any wise accessible by operations on the procedure values." This is a true statement as applied to the mapping performed by Apply; one can't, in general, obtain the argument values of a procedure call from the output of the procedure. However, one CAN obtain, e.g., the types of the formal parameters or the algorithm, from the definition of a procedure - which is what a value of Procedure (in WD 5) is.