From: Ed Barkmeyer

To: WG11

Subject: The further discourse with Tom Turba

Date: Thu, 18 Jul 91 12:26:35 EDT

On receipt of Tom's direct emailed comments, I sent back a request for clarification asking:

1. what "object-oriented" features he thought should be included, and
2. what aspect of the Complex datatype definition, apart from the syntax for complex-literal, led him to say that it was dependent on Cartesian representation.

Attached hereto I present his reply. He also commented on Ed Greengrass's response, which comment is also attached.

-Ed Barkmeyer

>>From s5000!turba@uunet.UU.NET Tue Jul 16 17:36:23 1991
To: edbark@cme.nist.gov
Subject: RE: Re: Comments on CLIDT
Message-Id: <9107161634.AA16991@s5000.RSVL.UNISYS.COM>
Date: 16 Jul 91 16:34:06 CDT (Tue)
From: turba@s5000.RSVL.UNISYS.COM
Status: RO

Ed,

Sorry for not being able to answer your message sooner. I have been on vacation since July 3rd.

On object types, the main thing is the underlying concepts of inheritance and subtyping. A subtype inherits all of the properties of the type it is derived from, i.e., the interface for all visible operations on objects of that type are preserved. A subtype may also add new operations to those it inherits and add new fields to the basic object.

These are the main concepts, and it should be noted that subtyping with objects serves two purposes, extension and classification. This is different than subtyping in non-object languages.

Sets and lists are orthogonal to objects. Their true relationship is with generators, which I think was correctly stated in the document, although I have not reread it to check.

On the complex data type, this was brought up by David Joslin at the meeting. Although I did not take specific notes on this, I believe it is just a matter of how the "Values:" specification was done. It is not a matter of having literals, which we have in an equivalent form. David could probably tell you exactly which parts of the definition should be changed.

On Null and Undefined, I have gotten 2 messages from Ed Greengrass on this via Ken Edwards. I totally disagree with his rationale for having them and will be writing a reply. I'll send you a copy.

T. N. T.

>>From s5000!turba@uunet.UU.NET Wed Jul 17 14:26:24 1991
To: edbark@cme.nist.gov
Subject: Comments on Ed Greengrass's Comments
Message-Id: <9107171324.AA23822@s5000.RSVL.UNISYS.COM>
Date: 17 Jul 91 13:24:03 CDT (Wed)
From: turba@s5000.RSVL.UNISYS.COM
Status: RO

Ed,

I sent the following to Ken Edwards who had forwarded to me some comments by Ed Greengrass. I have not included his comments here because they probably will be obvious from my responses. However, if you want, I'll send them to you.

T. N. T.
- - - - -
Ken,

Thanks for sending me the comments by Ed Greengrass. The following is a response to his two sets of comments.

****

In Greengrass's comments he states that there are two reasons for having the null type

1. To define variations on the same record, and

2. To define two or more different types of record that may appear in the same position.

He gives reasons that are based on application specific problems; however, the capabilities desired can be achieved by other means that do not require the invention of the null type (which to the best of my knowledge does not exist in any common programming language).

The solution to number 1 above is simply to use subclasses to define the variations that are desired. This is a much cleaner and more descriptive solution. It has been adopted by language designers such as Wirth in place of a choice construct. (See the Oberon language specification.)

The solution to number 2 is simply to use a tag field that indicates which record type is present. It is a solution that is found and used in numerous programming languages.

****

Greengrass goes on to say that an instance of a null type is needed to represent absent data. Again, this is only one possible solution to the problem. A better solution is to use existing facilities found in programming languages rather than invent a new one. The solution here is better served by a tag field, which can not only represent the absence of data, but also the partial presence of data.

****

Greengrass indicates that the undefined type is also useful to indicate a distinction between null, undefined, and present. Again, there is no need to invent a new type (undefined) that is not found in any known programming language. Its use is simply that of a choice in a variant. Because such a tag field is needed anyway, it is better to just use the tag field rather than invent something new. Tag fields also give more flexibility because partially known data can also be expressed as well as other application specific information.

All of the uses for null types and undefined types proposed by Greengrass seem better satisfied by tag fields and subclasses. This is especially true when considering the definitional problems introduced by null and undefined types.

****

There seems to be considerable confusion as to how object concepts relate to CLIDT. In part this is probably due to there being more than one object model, and also to the fact that some terms (e.g., subtype) are used in different ways. (A subtype or subclass is not a partitioning of values or objects, but rather a statement about a relationship between types, values, and objects.)

The concepts of types and classes can be, and have been, merged in many languages. Oberon, for example, simply extends the type and record concepts to include class definitions. A record type can simply be extended by inheritance. This is also true in Beta, Eiffel, C++, and many other languages. It is type concepts like this that should be included in CLIDT, most probably in a separate subsection between 7.2 and 7.3.

Generators are a separate issue for objects just as for any other type. This already appears to be properly separated.

****

The comments by Greengrass do not provide sufficient rationale as to why Choice and List are being required for direct compliance. There are many other datatypes for which the same comments could be said that are not included in Annex A. There are also missing types for which stronger arguments could be made.

****

T. N. T.

----- End Included Message -----