****
J3/03-113r1

Date: 12 March 2003
To: J3
From: Aleksandar Donev
Subject: Reinstating Deferred Bindings via Abstract Types
Reference: J3-007R3, based on J3/02-296r3, also see J3/02-271r1
****

Summary

I propose to bring back deferred bindings into F2x, by introducing abstract derived types into the language, much alike abstract types in Java. These are essential in creating Abstract Programming Interfaces and are much needed in Fortran 2002.

Many thanks to Van, Malcolm, Rob and Kurt for comments and also help with the edits.

Important Note:

The syntax proposed here for deferred bindings should be chosen according to the resolution of comment UKTC4. At present generic type-bound procedures are nameless, and so the syntax uses NULL(interface-name):

```
PROCEDURE, DEFERRED :: binding=>NULL(interface-name)
GENERIC, DEFERRED :: binding=>specific_procedure, NULL(specific-interface-name)
```

But if UKTC4 is passed, then we only need one syntax, for specific type-bound procedures, and in this case even the deferred keyword will not be needed:

```
PROCEDURE(interface-name) :: binding
```

I give both sets of edits, together with the things in common.

Motivation

We want several things from deferred bindings:
1. They must be part of the syntax so that from the declaration of an extensible type it is clear that a particular type-bound procedure is deferred.
2. They must provide compile-time checks when extending the type, forcing the user to do something with the deferred procedure (either defer it again or override).
3. They should provide a runtime protection against envoking the deferred procedure.

Solution

An abstract derived type is one which has the newly introduced ABSTRACT attribute. One cannot instantiate objects of abstract type. Such types are extensible and are used as base types in forming ``real'' types which can be instantiated. One can never get to an object of abstract type, even if using polymorphic pointers of abstract type, since there is never a target of abstract type. This provides a runtime protection against trying to envoke deferred type-bound procedures. The only tricky possibility is using parent component selection to get to an object of abstract type and try to envoke a deferred binding this way, but this is detectible at compile time and is forbidden explicitly in the edits below.

The main use of abstract types is when forming base types which have no default or natural implementation for certain operations and are therefore incomplete. An example is given at the end of this paper. Deferred type-bound procedures can appear only in such derived types. They represent operations whose implementation is deferred to extensions of the abstract type. Only when there are no more deferred bindings can the ABSTRACT attribute be removed. This provides the compile-time reminder for the user that he must override all bindings if he wants to declare a nonabstract type.

I note that the prohibition against instantiation is useful in itself even without any

deferred bindings. Therefore one can have an abstract type which has no deferred bindings.

Edits

As usual, I kindly ask for help with the edits, as it is likely I have forgotten or misworded something. See the example below for an illustration of the syntax:

Part 1: Add ABSTRACT as a derived type attribute

42:2+ Add ABSTRACT:
or ABSTRACT

42:9+ Add:
C417a (R425) If the type definition contains any deferred bindings (4.5.1.5) or inherits (4.5.3.1) any deferred bindings, ABSTRACT shall appear.

42:10 Add ABSTRACT:
If ABSTRACT, EXTENDS or EXTENSIBLE appears...

Part 2: Prohibit instantiation of ABSTRACT types

For constructors: [57:29+] Add:
C468a (R449) The <derived-type-spec> shall not specify an abstract type (4.5.3).

For declarations: [67:24+] Add:
C503a (R503) The <derived-type-spec> or the <type-alias-name> shall not specify an abstract type (4.5.3).

Internal Note to J3: CLASS(abstract-type) can of course appear. Also note that type-spec in ALLOCATE is automatically covered. Also note that C_F_POINTER is also covered since we do not allow polymorphic FPTR.

Part 3: Add DEFERRED bindings:

VERSION 1: If UK TC4 is accepted:

44: 17 Replace PROCEDURE with:
PROCEDURE [(<deferred-binding-interface>)]

44: 19+ Add:
R441a: <deferred-binding-interface> is <interface-name>

44: 20+ Add constraint:
C444a (R441) =><binding> shall appear only if <deferred-binding-interface> does not appear.

44: 20-21 Replace "If...appears," by:
If neither =><binding> or <deferred-binding-interface> appear,

45: 11+ Add:
C455a (R440) NON_OVERRIDABLE shall appear in the <binding-attr-list> of a deferred binding (4.5.1.5)

49: 4 Add to the end of the sentence:
or the abstract interface specified in <deferred-binding-interface>.

49:5+ Add:
A binding that specifies a <deferred-binding-interface> is a <<deferred binding>>. A <binding> is not given for deferred bindings. Deferred bindings can only appear in abstract types (4.5.3.3).

Also add glossary term for <<deferred binding>>:

<<deferred binding>> (4.5.1.5): A binding that specifies a <deferred-binding-interface> (4.5.1.5). Deferred bindings can only appear in abstract types (4.5.3.3).

VERSION 2: If generic tbps stay as they are:

_____

Discussion:
The basic syntax for specific bindings is:
PROCEDURE, DEFERRED :: binding_name => NULL(abstract_interface_name)
The basic syntax for generic bindings is:
GENERIC, DEFERRED :: generic_spec=>procedure_name, NULL(abstract_interface_name)
One can only put NULL(abstract_interface_name) is DEFERRED is there. In this version I require the DEFERRED attribute if the binding is still deferred (this makes a difference for generic bindings), but this can be changed.

44: 20-21 Replace "If...appears," by:
If =><binding> does not appear and the binding does not override an inherited deferred binding,

45: 3+ Add:
or DEFERRED

45: 11+ Add:
C455a (R441) DEFERRED shall appear if and only if <deferred-binding> appears.

C455b (R442) DEFERRED shall appear if and only if <deferred-binding> appears or any of the inherited generic bindings with the same <generic-spec> are deferred.

C455c (R440) DEFERRED and NON_OVERRIDABLE shall not both appear in the same <binding-attr-list>.

45: 19+ Add:
or <deferred-binding>

45: 18+ Add:
R445a: <deferred-binding> is NULL(<deferred-binding-interface>)
R445b: <deferred-binding-interface> is <interface-name>

49: 4 Add to the end of the sentence:
or the abstract interface specified in <deferred-binding-interface>.

49:5+ Add:
A binding that specifies the DEFERRED attribute is a <<deferred binding>>. Deferred bindings can only appear in abstract types (4.5.3.3).

Also add glossary term for <<deferred binding>>:
<<deferred binding>> (4.5.1.5): A binding that specifies the DEFERRED attribute (4.5.1.5). Deferred bindings can only appear in abstract types (4.5.3.3).

For both VERSION 1 and 2:

_____

56: 11+ Add new subsection:

4.5.3.3 Abstract types

An <<abstract type>> is a type that has the ABSTRACT attribute. If a type has any deferred type-bound procedures (4.5.1.5) it shall have the ABSTRACT attribute. If a type has the ABSTRACT attribute an object of the type shall not be declared, constructed or allocated.

Also add glossary entry for <<abstract type>>:
<<abstract type>> (4.5.3.3): A type that has the ABSTRACT attribute. An object of abstract type shall not be declared, constructed or allocated.

Add NOTE to end of 4.5.3.3:
Abstract types can be extended into nonabstract types once all the procedure bindings (including inherited ones) are nondeferred. Deferred bindings (4.5.1.5) delegate the

implementation of a type-bound procedure to extensions of the type, and can only appear in abstract types. Objects of abstract type cannot be instantiated (brought into existence), and deferred type-bound procedures cannot be invoked. Abstract types are common in Abstract Programming Interfaces (APIs). An example is given in Note XXX.

Add another NOTE with example (I can extend the example for section C if accepted):

The following API defines an object that can be displayed in an X window:

```
TYPE, EXTENSIBLE, ABSTRACT :: DRAWABLE_OBJECT
   INTEGER, DIMENSION(3) :: RGB_COLOR=(/0,0,0/) ! White
   REAL, DIMENSION(2) :: POSITION=(/0.0,0.0/) ! Centroid
CONTAINS
```

Either VERSION 1:

```
PROCEDURE(RENDER_X), PASS(OBJECT) :: RENDER
```

or VERSION 2:

```
   PROCEDURE, DEFERRED, PASS(OBJECT) :: RENDER=>NULL(RENDER_X)
END TYPE DRAWABLE_OBJECT

INTERFACE
   SUBROUTINE RENDER_X(OBJECT, WINDOW)
      CLASS(DRAWABLE_OBJECT), INTENT(IN) :: OBJECT
      CLASS(X_WINDOW), INTENT(INOUT) :: WINDOW
   END SUBROUTINE RENDER_X
END INTERFACE
```

We can declare a nonabstract type by extending the abstract type:

```
TYPE, EXTENDS(DRAWABLE_OBJECT) :: TRIANGLE ! Not ABSTRACT
   REAL, DIMENSION(2,3) :: VERTICES ! In relation to centeroid
CONTAINS
   PROCEDURE, PASS(TRIANGLE) :: RENDER=>RENDER_TRIANGLE_X ! Not shown here
END TYPE TRIANGLE
```

Part 4: Prohibit invoking deferred bindings:

Discussion: There is a danger of referencing a deferred type-bound procedure by using the parent component selection (type casting) mechanism:
CALL object_of_nonabstract_type%abstract_parent_component%abstract_binding()
which would result in a run-time error. We prohibit this explicitly. Also dangerous are data pointers, if they somehow end up pointing to an object of abstract type:
polymorphic_pointer_of_abstract_type=>object_of_nonabstract_type%abstract_parent_component
Another possibility is for a procedure pointer to end up pointing to a deferred type-bound procedure. This seems impossible in the current standard, so we are OK, but let's forbid it just in case.

[262:27+] Insert:
C1225a (R1219) If <data-ref> is not polymorphic, <binding-name> shall not be the name of a deferred binding.

[262:27+] Note 12.16a [in a box]: A deferred binding specified by accessing a parent component shall not be invoked.

[143:18+] Add:
C724a (R739) The type of <data-target> shall not be abstract (4.5.3.3).

[143:28+] Add:
C728a (R741) The <proc-target> shall not be a deferred binding.

5. Ammend NULL's syntax

VERSION 2 only, nothing for 1:

293:27 Add deffered:
NULL ([MOLD]) Returns disassociated, unallocated, or deferred result

334:18+ Add to end of sentence:
or designates a deferred type-bound procedure binding.
and remove the second or

Internal J3 Note: NOT ALL EDITS OF NULL ARE FINISHED. Use 007-R2 as guide. It depends on what the syntax ends up being.

! EOF