To: J3/B
From: Malcolm Cohen
Subject: Argument association restructuring.
Date: 2006/07/31

1. Introduction

The requirements for argument association for dummy data objects are fiendishly complicated. But worse, there is no coherent structure to the subclause explaining them. This subclause needs to be rewritten.

2. Structure

There are many possible structures:

- 12.5.1.4 Ordinary dummy variables
  12.5.1.5 Allocatable and pointer dummy variables

- 12.5.1.4 Ordinary dummy variables
  12.5.1.5 Allocatable and pointer dummy variables
  12.5.1.6 Allocatable dummy variables
  12.5.1.7 Pointer dummy variables

- 12.5.1.4 Dummy data objects
  12.5.1.4.1 General rules
  12.5.1.4.2 Allocatable dummy arguments
  12.5.1.4.3 Pointer dummy data objects
  12.5.1.4.4 Ordinary dummy variables

This revision of this paper explores the second structure listed above; there is very little duplication introduced by this structure, though that subclause does get longer by virtue of the extra space taken up by the headings.

3. Edits to 06-007 (illustrative)

```
[312-316] 12.5.1.4 Actual arguments associated with dummy data objects
  Change heading to
    "Ordinary dummy variables"
  and introduce new subclauses
    "12.5.1.4a Allocatable and pointer dummy variables"
    "12.5.1.4b Allocatable dummy variables"
    "12.5.1.4c Pointer dummy variables".

[312:5-] Insert new initial paragraphs to 12.5.1.4 and the new subclauses
  "The requirements in this subclause apply to actual arguments associated
   with nonpointer nonallocatable dummy data objects."
  For 12.5.1.4a
  "The requirements in this subclause apply to actual arguments associated
   with either allocatable or pointer dummy data objects."
  For 12.5.1.4b
  "The requirements in this subclause apply to actual arguments associated
   with allocatable dummy data objects."
```

```
  For 12.5.1.4c
  "The requirements in this subclause apply to actual arguments associated
   with dummy data pointers.".

[312:5-9] Replace with
  "The dummy argument shall be type compatible or bits compatible with the
   associated actual argument."
  For 12.5.1.4a
  "The actual argument shall be polymorphic if and only if the dummy
   argument is polymorphic, and either both the actual and dummy arguments
   shall be unlimited polymorphic, or the declared type of the actual argument
   shall be the same as the declared type of the dummy argument.".

[312:9+-10-] Move note 12.21 to 12.5.1.4a.

[312:10-13] Leave this paragraph in 12.5.1.4 as is, and copy the first half
            (up to but not including the ", except") into 12.5.1.4a, viz
  "Unless the actual argument and the corresponding dummy argument are bits
   compatible, the type parameter values of the actual argument shall agree
   with the corresponding ones of the dummy argument that are not assumed or
   deferred.".

[312:14-18] Leave as is.

[312:19-20] Leave in 12.5.1.4a as is, plus copy into 12.5.1.4a.

[312:21-22] Move into 12.5.1.4a, deleting "that is allocatable or a pointer".

[312:23-24] Leave as is, together with the following note and UTIs.

[313:1-4] Move into 12.5.1.4c, changing
  "If the dummy argument is a pointer that" -> "If the dummy argument".
{This also improves the clarity of the Otherwise sentence in this paragraph.}

[313:5-8 plus j3 note at top of 314] Move into 12.5.1.4c, changing
  "If the dummy argument and actual argument are pointers, the" -> "The"
{I think this also fixes a technical flaw - at least I cannot understand how
 the auto-targetting works if the ranks differ, since pointer assignment to a
 plain named pointer doesn't allow that.}

[314:1-4 plus j3 note before 314:5] Move into 12.5.1.4b, changing
  "If a dummy argument is allocatable, the" -> "The".

[314:5-6] Move to 12.5.1.4c, rewriting.
```

et cetera.

4. Edits for restructuring (by result)

These start on page 312. Note that I've left the J3 internal notes in.

12.5.1.5 Ordinary dummy variables

The requirements in this subclause apply to actual arguments that are associated with nonallocatable nonpointer dummy data objects.

The dummy argument shall be type compatible or bits compatible (4.3.1.3) with the associated actual argument.

Unless the actual argument and the corresponding dummy argument are bits compatible, the type parameter values of the actual argument shall agree with the corresponding ones of the dummy argument that are not assumed, except for the case of the character length parameter of an actual argument of type default character associated with a dummy argument that is not assumed shape.

If a scalar dummy argument is of type default character, the length *len* of the dummy argument shall be less than or equal to the length of the actual argument. The dummy argument becomes associated with the leftmost *len* characters of the actual argument. If an array dummy argument is of type default character and is not assumed shape, it becomes associated with the leftmost characters of the actual argument element sequence (12.5.1.8) and it shall not extend beyond the end of that sequence.

The values of assumed type parameters of a dummy argument are assumed from the corresponding type parameters of the associated actual argument.

If the actual argument is a co-indexed object, the dummy argument shall not be a co-array and shall have the INTENT(IN) or the VALUE attribute.

J3 internal note

Unresolved Technical Issue 50
It doesn't seem like forcing the user to write

```
localvar = x[i]
CALL sub(localvar)
x[i] = localvar
```

is a substantial improvement in, well, anything. Not to mention that the user cannot do that if x[i] is undefined or only partially defined before the call to sub. It doesn't seem to be any more work for the processor than what it already does for

```
CALL sub(a(1:n:2))
```

for explicit-shape dummies. And, *MUCH WORSE*, this makes defined assignment unusable with co-arrays. Can we say "not even integrated with Fortran 90"?
Subgroup said: "this requires more complications to the memory consistency rules since the copy out may overwrite other changes to the variable."
The editor says: That is just so untrue it is not funny.
Is everyone *REALLY* that ignorant of our dummy argument aliasing rules?
We've not repealed them, and there are no edits to subvert them for co-arrays. If they need to be subverted (something I doubt, but maybe the go-faster department want to go slower), something needs to be done.
Either way this is a serious technical flaw.

NOTE 12.1

If the actual argument is a co-indexed object and the corresponding dummy argument is not a co-array, it is likely that a processor will make a copy on the executing image of the actual argument, including copies of any allocated allocatable subcomponents, before argument association occurs.

NOTE 12.1 (cont.)

J3 internal note

Unresolved Technical Issue 51
That doesn't seem likely for the vast majority of Fortran processing systems, which are going to be shared-memory small-cpu-count. They are, presumably, going to do the "right thing" here. That does not involve making any copies.
Anyway, what's this "and the ... dummy is not a co-array." guff. Last time I looked, a co-indexed object was not a valid actual argument to a co-array dummy.

Except in references to intrinsic inquiry functions, if the dummy argument is not nonoptional and the actual argument is allocatable, the corresponding actual argument shall be allocated.

If the dummy argument has the VALUE attribute it becomes associated with a definable anonymous data object whose initial value is that of the actual argument. Subsequent changes to the value or definition status of the dummy argument do not affect the actual argument.

NOTE 12.2

Fortran argument association is usually similar to call by reference and call by value-result. If the VALUE attribute is specified, the effect is as if the actual argument is assigned to a temporary, and the temporary is then argument associated with the dummy argument. The actual mechanism by which this happens is determined by the processor.

If the dummy argument does not have the TARGET attribute, any pointers associated with the actual argument do not become associated with the corresponding dummy argument on invocation of the procedure. If such a dummy argument is used as an actual argument that is associated with a dummy argument with the TARGET attribute, whether any pointers associated with the original actual argument become associated with the dummy argument with the TARGET attribute is processor dependent.

If the dummy argument has the TARGET attribute, does not have the VALUE attribute, and is either a scalar or an assumed-shape array that does not have the CONTIGUOUS attribute, and the corresponding actual argument has the TARGET attribute but is not a co-indexed object or an array section with a vector subscript then

(1) any pointers associated with the actual argument become associated with the corresponding dummy argument on invocation of the procedure, and

(2) when execution of the procedure completes, any pointers that do not become undefined (16.5.2.2.3) and are associated with the dummy argument remain associated with the actual argument.

If the dummy argument has the TARGET attribute and is an explicit-shape array, an assumed-shape array with the CONTIGUOUS attribute, or an assumed-size array, and the corresponding actual argument has the TARGET attribute but is not an array section with a vector subscript then

(1) on invocation of the procedure, whether any pointers associated with the actual argument become associated with the corresponding dummy argument is processor dependent, and

(2) when execution of the procedure completes, the pointer association status of any pointer that is pointer associated with the dummy argument is processor dependent.

If the dummy argument has the TARGET attribute and the corresponding actual argument does not have the TARGET attribute or is an array section with a vector subscript, any pointers associated with the dummy argument become undefined when execution of the procedure completes.

If the dummy argument has the TARGET attribute and the VALUE attribute, any pointers associated with the dummy argument become undefined when execution of the procedure completes.

If the actual argument is scalar, the corresponding dummy argument shall be scalar unless the actual argument is of type default character, of type character with the C character kind (15.2), or is an element or substring of an element of an array that is not an assumed-shape, pointer, or polymorphic array. If the procedure is nonelemental and is referenced by a generic name or as a defined operator or defined assignment, the ranks of the actual arguments and corresponding dummy arguments shall agree.

If a dummy argument is an assumed-shape array, the rank of the actual argument shall be the same as the rank of the dummy argument; the actual argument shall not be an assumed-size array (including an array element designator or an array element substring designator).

Except when a procedure reference is elemental (12.8), each element of an array actual argument or of a sequence in a sequence association (12.5.1.8) is associated with the element of the dummy array that has the same position in array element order (6.2.2.2).

NOTE 12.3

For type default character sequence associations, the interpretation of element is provided in 12.5.1.8.

A scalar dummy argument of a nonelemental procedure may be associated only with a scalar actual argument.

If a dummy argument has INTENT (OUT) or INTENT (INOUT), the argument associated entity shall be definable. If a dummy argument has INTENT (OUT), the argument associated entity becomes undefined at the time the association is established, except for components of an object of derived type for which default initialization has been specified. If the dummy argument is not polymorphic and the type of the actual argument is an extension type of the type of the dummy argument, only the part of the actual argument that is of the same type as the dummy argument becomes undefined.

If the actual argument is an array section having a vector subscript, the dummy argument is not definable and shall not have the INTENT (OUT), INTENT (INOUT), VOLATILE, or ASYNCHRONOUS attributes.

NOTE 12.4

Argument intent specifications serve several purposes. See Note 5.16.

NOTE 12.5

For more explanatory information on argument association and evaluation, see subclause C.9.5. For more explanatory information on targets as dummy arguments, see subclause C.9.6.

J3 internal note

Unresolved Technical Issue 55
Probably need this constraint:
C1201 An actual argument that is a co-indexed object shall not be associated with a dummy argument that has the ASYNCHRONOUS attribute.
It is true that allowing VALUE+ASYNCHRONOUS makes a mockery of the restrictions on ASYNCHRONOUS dummies and note (04-007) 12.26 about their purpose. But should co-arrays try to be consistent or should they copy this inconsistency?

C1202 (R1223) If an actual argument is an array section or an assumed-shape array, and the corresponding dummy argument has either the VOLATILE or ASYNCHRONOUS attribute, that dummy argument shall be an assumed-shape array.

C1203 (R1223) If an actual argument is a pointer array, and the corresponding dummy argument has either the VOLATILE or ASYNCHRONOUS attribute, that dummy argument shall be an

assumed-shape array that does not have the CONTIGUOUS attribute or a pointer array.

NOTE 12.6

The constraints on actual arguments that correspond to a dummy argument with either the ASYNCHRONOUS or VOLATILE attribute are designed to avoid forcing a processor to use the so-called copy-in/copy-out argument passing mechanism. Making a copy of actual arguments whose values are likely to change due to an asynchronous I/O operation completing or in some unpredictable manner will cause those new values to be lost when a called procedure returns and the copy-out overwrites the actual argument.

12.5.1.6 Allocatable and pointer dummy variables

The requirements in this subclause apply to actual arguments that are associated with either allocatable or pointer dummy data objects.

The actual argument shall be polymorphic if and only if the associated dummy argument is polymorphic, and either both the actual and dummy arguments shall be unlimited polymorphic, or the declared type of the actual argument shall be the same as the declared type of the dummy argument.

NOTE 12.7

The dynamic type of a polymorphic allocatable or pointer dummy argument may change as a result of execution of an allocate statement or pointer assignment in the subprogram. Because of this the corresponding actual argument needs to be polymorphic and have a declared type that is the same as the declared type of the dummy argument or an extension of that type. However, type compatibility requires that the declared type of the dummy argument be the same as, or an extension of, the type of the actual argument. Therefore, the dummy and actual arguments need to have the same declared type.

Dynamic type information is not maintained for a nonpolymorphic allocatable or pointer dummy argument. However, allocating or pointer assigning such a dummy argument would require maintenance of this information if the corresponding actual argument is polymorphic. Therefore, the corresponding actual argument needs to be nonpolymorphic.

Unless the actual argument and the corresponding dummy argument are bits compatible, the type parameter values of the actual argument shall agree with the corresponding ones of the dummy argument that are not assumed or deferred.

The values of assumed type parameters of a dummy argument are assumed from the corresponding type parameters of the associated actual argument.

The actual argument shall have deferred the same type parameters as the dummy argument.

12.5.1.7 Allocatable dummy variables

The requirements in this subclause apply to actual arguments that are associated with allocatable dummy data objects.

The actual argument shall be allocatable, the ranks shall agree, and either the nondeferred type parameters shall agree or the actual argument and the dummy argument shall be bits compatible. It is permissible for the actual argument to have an allocation status of unallocated.

J3 internal note

Unresolved Technical Issue 53
Argument-associating allocatables of different types does not seem to be in accordance with the spirit of our current rules. It requires a processor to use the same representation for allocatables of different kinds or to do copyin/copyout conversions. Why are we doing this?
Furthermore, the very concept of having allocatable entities of REAL and INTEGER types being associated with one another is blood-curdling. Just how far the implications of this would go I am not sure; certainly they are not good for reliability and portability, whether performance suffers as well I cannot say without spending considerable effort.
This butchery of our association rules does not seem at all necessary for BITS to be useful.

If the dummy argument does not have the TARGET attribute, any pointers associated with the actual argument do not become associated with the corresponding dummy argument on invocation of the procedure. If such a dummy argument is used as an actual argument that is associated with a dummy argument with the TARGET attribute, whether any pointers associated with the original actual argument become associated with the dummy argument with the TARGET attribute is processor dependent.

If the dummy argument has the TARGET attribute, does not have the VALUE attribute, and is either a scalar or an assumed-shape array that does not have the CONTIGUOUS attribute, and the corresponding actual argument has the TARGET attribute but is not a co-indexed object or an array section with a vector subscript then

(1) any pointers associated with the actual argument become associated with the corresponding dummy argument on invocation of the procedure, and
(2) when execution of the procedure completes, any pointers that do not become undefined (16.5.2.2.3) and are associated with the dummy argument remain associated with the actual argument.

If a dummy argument has INTENT (OUT) or INTENT (INOUT), the argument associated entity shall be definable. If a dummy argument has INTENT (OUT), the argument associated entity becomes undefined at the time the association is established, except for components of an object of derived type for which default initialization has been specified.

12.5.1.8 Pointer dummy variables

The requirements in this subclause apply to actual arguments that are associated with dummy data pointers.

If the dummy argument does not have the INTENT(IN) attribute, the actual argument shall be a pointer. Otherwise, the actual argument shall be a pointer or a valid target for the dummy pointer in an assignment statement. If the actual argument is not a pointer, the dummy pointer becomes pointer-associated with the actual argument.

The ranks shall agree and either the nondeferred type parameters shall agree or the actual argument and the dummy argument shall be bits compatible. If the dummy pointer has the CONTIGUOUS attribute, the actual argument shall have the CONTIGUOUS attribute.

J3 internal note

Unresolved Technical Issue 52
Argument-associating pointers of different types does not seem to be in accordance with the spirit of our current rules. It requires a processor to use the same representation for pointers of different kinds or to do copyin/copyout conversions. Why are we doing this?
Furthermore, this "pseudo-polymorphism" for bits leads directly to INTEGER pointers being associated with REAL targets. This will be a serious inhibition on optimisation. It is an open invitation to programming errors and nonportable hacks.
NOTE: A similar comment would apply to pointer assignment if one is allowed to turn a bits pointer into another pointer (the term "bits compatible" not being well-defined as yet, I don't know whether this is the case). Fiddling with the bits is one thing, but removing the type restrictions on our pointers is simply unacceptable both from a software engineering point of view and from a performance point of view.

If the dummy argument has INTENT(OUT), the pointer association status of the associated actual argument becomes undefined on invocation of the procedure.

If the dummy argument is nonoptional and the actual argument is allocatable, the actual argument shall be allocated.

NOTE 12.8

For more explanatory information on pointers as dummy arguments, see subclause C.9.6.