****

J3/02-296

Date: October 2002
To: J3
From: Aleksandar Donev
Subject: Reinstating Deferred Bindings
Reference: J3-007R3, J3/02-271r1

****

Summary

I propose to do the work necessary to bring back deferred bindings into F2x, which were unfortunately deleted at J3 meeting 162 (J3/02-271) due to lack of time.

I argue why they are an integral part of the OOP framework in the language and how they help the user write better, correct and safer code. Actual edits to do the reinstallation are not provided here, since I need feedback on the choice of the model, for which I make specific proposals here. I will provide them once I have guidance and approval.

I stress that all OOP languages I know of have some form of deferred (abstract, virtual, etc.) methods.

Motivation

The last-minute deletion of deferred bindings at meeting 162 in my opinion ruins the rather elegant OOP structure of the language. OOP is a framework for how to develop structured programs in which the relation between concepts and implementations is clean and clear. In particular, one of the cornerstones of OOP thinking is the separation between specification and implementation. The language needs to provide a framework for writing codes in that manner. So the proposed justification for the deletion, "you can make a deferred binding [specification] by writing an empty procedure [implementation]" is an irrelevant reasoning. It is not a question of whether I can find a bad way of writing a working program, but a way to write a maintainable, safe and working program.

In an API (Abstract Programming Interface) it must be clear what is real and what is abstract. Even if deferred bindings do not make it into the language, any program I would write would have comments in the body of the definition of the derived-type emphasizing what is abstract and what is real. Otherwise, one has to look at actual codes to see which procedures do something meaningful, and which are empty.

Furthermore, with the deletion of deferred bindings, we are asking an API writer to also be an implementor. Even if empty, a real procedure is an implementation--this has no part in an API. Moreover, putting STOP in the body of a deffered procedure is a bad idea--one would want to put more meaningful error-generation mechanisms. Again, this has no place in an API. It would also be desireable to allow the user to
invoke his own error-handling mechanisms when such a deferred procedure is called, while also preserving the ability of the compile-time checks/reminders to do something with deferred bound procedures.

Solution

Only a few changes are needed from the original syntax in 007R2 to make deferred bindings a great feature in F2x. A good syntax would add a keyword DEFERRED, which means that in a type extension one needs to do something with the bound procedure other then simply inherit it, to be a compile-time check. But we should
also allow a binding to a specific procedure to go along with this, to allow the user to trap his errors. So for a plain procedure (not a generic), we have the possibilities:

1. No specific binding--the user must ensure this method is not invoked. If it is, it can produce a segfault or with some debugging flag produce useful tracebacks to the user to help him fix the bug.

```
PROCEDURE(abstract_interface_name), DEFERRED :: binding_name
```

2. A specific binding (to a user-defined error-handling procedure) is also present--the method must be deferred again
or bound to a real procedure when extending the base type. There is no abstract interface.

```
PROCEDURE, DEFERRED :: binding_name=>error_handling_procedure_binding
```

Notice that there is no NULL() here, which was mentioned as a bad point for the old design of this language feature in J3-007R2.

As for deferred operators, I don’t have a strong preference, but it is probably wise to keep the same syntax:

```
GENERIC(abstract_interface_list), DEFERRED :: generic_spec
```

or

```
GENERIC, DEFERRED :: generic_spec=>real_procedures_binding_list
```

Notice that this is different from the original proposal which proposed to have GENERIC(abstract_interface_name). I find the above more consistent.

Edits

Will be done only after the solution proposed above is supported.

! EOF