Enhancing Exception Support in Ada 95: a Workshop Position Paper

S. Tucker Taft
Chief Technology Officer, stt@avercom.net
AverCom Corp., A Titan Company
12 Oak Park Drive, Bedford, MA 01730

This workshop position paper discusses two alternative approaches to enhancing exception support. The first approach is based on a syntax extension, and is very similar to an approach proposed early in the Ada 9X process. The second approach is based on adding a child package to Ada.Exceptions to support raising an exception and associating an Ada object of any type that is an extension of a particular root type.

I. A Syntax-Based Approach

With this approach, exceptions may have parameters (sort of like discriminants) and may be derived from other exceptions. For example:

```
My_Exception : exception(X : Integer := 0; Y : Boolean := False);
My_Other_Exception : exception is new My_Exception
    with (A : Character := ' '; Z : String_Ptr := null);
```

When handling exceptions, one may handle a specific exception, or a class of exceptions. For example:

```
when My_Exception => -- Handles only My_Exception
when My_Exception'Class => -- Handles My_Exception and its descendants
```

The first exception handler that "covered" the type of the exception occurrence would be chosen (so order would matter). We could make it illegal for a more inclusive handler to precede a more specific handler, in the same way that “when others =>” must come last now.

Parameters of exceptions may be provided when an exception is raised, either positionally or named. If not specified, the default values are used. For example:

```
raise My_Exception(Y => True);
```

Parameters of exception are available as components of the exception occurrence:

```
when E : My_Exception =>
    if E.X > 35 and then E.Y = True then ...
when E : My_Exception'Class =>
    if E.X > 35 and then E.Y = True then ...
```

To avoid implicit dynamic storage allocation and other nasties, exception parameters would have to be of elementary types, which includes access values. Note that defaults should probably be required for all exception parameters, so an exception can be raised given just the exception identity. This requirement was not part of the original Ada 9X proposal, but at that point exception identities weren't manipulable values.

II. A Package-Based Approach

A significantly simpler approach which wouldn't require any new syntax has been discussed more recently. This involves adding a child package to Ada.Exceptions, perhaps Ada.Exceptions.Objects:

```
package Ada.Exceptions.Objects is
        -- This package provides a way of associating an arbitrary
        -- tagged object with an exception occurrence
        type Root_Exception_Object is tagged null record;
        Null_Exception_Object : constant Root_Exception_Object :=
                (null record);
        procedure Raise_With_Object(
                Id : Exception_Identity; Obj : Root_Exception_Object'Class);
                -- This raises the given exception, and passes along the
                -- specified object. Implementations may place limits
                -- on the 'Size of Obj, but must support at least an
                -- Obj'Size of 1000 bits. Storage_Error is raised
                -- if the Obj'Size is too great.
        function Exception_Object(Occ : Exception_Occurrence)
                return Root_Exception_Object'Class;
                -- This returns the object associated with the given
                -- exception occurrence. It returns the Null_Exception_Object
                -- if there is no associated object.
        function Image(Obj : Root_Exception_Object)
                return String;
                -- This is the only dispatching operation of the
                -- root exception object type. Of course others could
                -- be added in descendants. This function presumably returns
                -- some string appropriate for printing out in a general
                -- purpose exception handler (e.g. when E:others => ...).
                -- It returns the emptry string by default (i.e. if
                -- not overridden).
end Ada.Exceptions.Objects;
```

With this latter approach, you could use a membership test on the result of calling the "Exception_Object" function to disambiguate among various descendants of Root_Exception_Object, and/or add dispatching operations to perform type-specific exception processing.

III. Conclusion

I think keeping the exception capability relatively simple is justified. It is easy to spend a lot of energy, without adequate return on investment in this area. Using exceptions to accomplish clever communication from the point of an error sounds good in practice, but I doubt many programmers will actually make good use of it. My expectation is that in most systems, exceptions are for debugging, or at most error logging and resetting at a fairly high level. Raising an "object" would allow a standard "logging" type which could be application-specific. Much more than that is probably overkill.

So the bottom line at this point is that I would probably recommend the raise-with-object approach, and not bother making changes to the basic exception handling model. The exception hierarchy approach seems very elegant, but I'm just not sure it is worth the language complexity.