Final Session Summary:
How to Evolve Exception Handling in Ada

Chair: Alan Burns

Rapporteur: Alexander Romanovsky

1. Agenda

The agenda of the final session included:

- a discussion on improving the ways of using exception handling in modern application development;
- a brief analysis of the following two topics: exception handling in systems with concurrency and object-orientation; exception handling in distributed systems;
- a wrap-up to discuss possible Ada Interpretations that can be developed further in the nearest future.

2. What is exception handling for?

The chair started the session by initiating a discussion on the ways exception handling is used. Generally speaking, there are three views promoting the use of exception handling for: debugging, software fault tolerance and dealing with abnormal events in general. Various branches of industry and organizations take different approaches to it. For example, in nuclear power industry Ada is used because of its exception handling facilities. On the other hand, in avionics exception handling is usually not used: the view here is that developers should rid their software of all exceptions statically. The existing guidelines very often contradict each other. More universal guidelines sometimes offer several paradigms of using exception handling. For example, the HRG developed a guidance for the use of the Ada programming language in high integrity systems (ISO/IEC TR 15942. Programming Languages - Guide for the Use of the Ada Programming Language in High Integrity Systems. March 2000) in which three models restricting the general exception handling model provided by Ada are discussed: do not use exception handling at all, raise and handle all exceptions locally, handle all exceptions at the outmost level only. Bill Bail and Currie Colket stated that to the best of their knowledge DoD and MITRE do not usually use exception handling for fault tolerance but use it mainly for debugging. After a brief discussion we all agreed that dealing with abnormal events usually means tolerating faults. For example, switching modes, degradation, safe stop are typical examples of fault tolerance.

Many workshop participants have agreed that it is bad practice to use exception handling for debugging (this can be only explained by the absence of good debuggers) but not to use it for tolerating faults. It is clear to us that it is not practically possible to rid software of all possible faults (including software and hardware ones, operators' mistakes, etc.) while developing complex modern applications. Tullio Vardanega gave a brief presentation that showed a positive experience of using exception handling in ESA.

The group's general feeling was that there is clearly a lack of education and understanding in the area (e.g., many users implicitly equate a language implementation with the language itself), and that many industrial guides and standards are too conservative with respect to exceptions. Existing Ada exception handling mechanisms should be backed by good user guides. Some of the topics that need education are: making system developers understand that each program (or, a subprogram) that propagates an exception should be left in a known state to make recovery simpler, and showing the best ways of using exception handling for delivering quality of service.

Tullio Vardanega presented general requirements based on a number of interviews he had held with the ESA programmers before the workshop. This is the list:

- better integration with the typing model
- exception parameters that go beyond string- or stream-based attributes

- ability to distinguish between different occurrences of the same exception type
- generic parameters to include an exception type
- better user control of the exception-raising overhead
- users should know better what state the raising of a predefined exception leaves a block in.

The following discussion used this list to raise again important questions addressed during the workshop.

It was emphasized that the failing unit has an obligation to remove any ambiguous state (even by deleting itself if necessary) and that the unit developers should think in terms of maintaining the unit (e.g. object) invariants while applying exception handling.

All participants agreed that the ultimate solution to many of the problems discussed in the workshop is to make exceptions extendable types similar to tagged types. This would allow system developers to build hierarchies of exceptions and to add information at different levels of these hierarchies. The two workshop position papers (by Pascal Leroy and by Thomas Wolf) offer interesting solutions to these problems. All participants realised that achieving compatibility with the existing Ada exception handling is vital. At the same time it was clear that there always will be a difference between tagged types and extendable exceptions as there are exception-specific functionalities and restrictions.

The discussion then moved on to interface exceptions and to including exceptions into the signature of the object. The existing concept of the idealized fault-tolerance component (developed many years ago at University of Newcastle upon Tyne) was briefly outlined to show the difference between the local, interface and failure exceptions. After this, approaches to including all (interface) exceptions that a subprogram can propagate into its signature were discussed. The solution should make it possible for the compiler to do static checks, in which case all local exceptions should have handlers inside a subprogram. All workshop participants realise that the solution should be compatible with Ada 95. One of the ideas is to treat all existing (Ada 95) exceptions as unchecked: if there is no exception clause in the signature, the subprogram can only propagate the unchecked exceptions. Another idea was to use a pragma to control exception propagation and to distinguish between exceptions of different types.

3. OO and concurrency

The idea of developing better exception handling suitable for programming concurrent object-oriented systems was briefly discussed. Ada 95 promotes different programming paradigms which makes it difficult to introduce a "general" exception handling mechanisms that is equally suitable for programming all types of systems. In the future, developing a general mechanism that works for tasks, objects and classes will be an important issue.

4. Distribution

The existing distribution model supports distribution of packages (not of objects) and therefore exceptions that can be propagated between partitions via remote procedure calls are not associated with objects. The discussion of this topic did not go too far as it is not our intention to re-think the Ada distribution model. It was mentioned that it may be important to be able to specify the list of interface exceptions in the signatures of the remotely callable subprograms.

5. Wrap-up

In this part of the session an attempt was made to rank the proposals discussed during the workshop with respect to their chances to make it into Ada 200Y and to their importance for improving the current situations in Ada 95. This is the list:

- extended finalize
- exceptions as types
- exceptions in signatures
- exceptions as formal generic parameters
- propagation of exceptions from tasks
- task groups.

The group agreed that extended finalize should be incorporated into the language as soon as possible, as it allows programmers to smoothly deal with many problems discussed in the session on exceptions and concurrency. Another reasonable approach that could provide some solutions to these problems is to introduce additional task attributes indicating how the task has terminated.

A considerable amount of time was devoted to clarifying the idea of making exceptions types (this was a continuation of our work within session on Object Orientation and Exception Handling – see the session summary). We have tried to develop a way to compare existing proposals by analysing if they allow for the building of exception hierarchies and/or for introducing extensible exceptions. It was clear that we need features for classifying exceptions when building complex applications. Some of the important questions that should be discussed when developing the final solution are: how can exceptions be passed

as parameters, allocated and assigned. The solution should take into account that allowing pointers to exceptions can become a hazard for system safety, and would therefore be ironically counter-productive. A possibility of introducing exceptions as limited (extensible) types was briefly mentioned; this approach would allow one to build exception hierarchies and to add information to an exception.

Regarding the declaration of exceptions in subprogram signatures, our conclusions were that this is a very important feature that should be included into a future version of Ada, and that it is possible to develop an approach that would be compatible with Ada 95 by treating all "old" exceptions as unchecked ones. The participants emphasised that this proposal needs further development and discussions, as soon as possible.

All workshop participants agreed that the workshop had been very successful and that there was a clear need to organise a second workshop sometime in 2002 to follow up on the progress with proposed language enhancements.