In complex modern systems often more than half of 
the application code is devoted to dealing with errors. 
In contrast modern systems were designed to run with 
minimal human intervention.

Exception handling should have an associated:

- Exception handling should lead to a clearer design of the software.
- Exception handling should make it possible to hide errors from the user.
- Exception handling should improve the robustness of the software.
- Exception handling should allow developers to focus on other aspects of the software.

Exception handling can be used to:

- Trap and handle errors.
- Provide graceful recovery from errors.
- Improve the reliability of the software.
- Reduce the complexity of the software.

Exception handling is often used to:

- Handle system errors.
- Handle user errors.
- Handle interface errors.
- Handle resource errors.

Exception handling can lead to:

- Increased developer productivity.
- Increased software reliability.
- Increased software maintainability.
- Increased software understandability.

Exception handling is often used to:

- Trap and handle system errors.
- Trap and handle user errors.
- Trap and handle interface errors.
- Trap and handle resource errors.

Exception handling is often used to:

- Trap and handle system errors.
- Trap and handle user errors.
- Trap and handle interface errors.
- Trap and handle resource errors.
The problem is exacerbated in the context of tagged types. One solution would be to declare all external exceptions that procedures can propagate in the package specification. Unfortunately, this approach does not work for subprogram specifications, and gives no guarantee that all external exceptions are declared. Moreover, the link between a particular subprogram and exceptions is lost. For example, with several tagged types declared in the same package, it is impossible to distinguish between the external exceptions of the different types.

Ada does not automatically make the base type exceptions visible to the clients of the derived types [16]. Moreover, these exceptions are not part of the abstraction represented by the base type as there is no link between the interface exceptions of the parent and the child. Interface exceptions declared in the parent specification are not derived together with the child type. The caller cannot know all exceptions that can be signaled just by looking in the specification of the class, the environment in which they operate always functions correctly. Second, there are many situations where exceptions can be avoided only at the cost of obscure work-arounds (such as the notorious return codes).

2. Ada Exception Handling

Ada 83 has a number of well-known problems with exceptions [6, 14, 16]. Ada 95 does not really solve them but rather introduces additional problems when it provides new features such as classes and protected types without offering exception handling features that are sophisticated enough to back them. In this section we discuss features that we believe should be improved.

2.1. Exception Propagation

A number of anomalies are related to exception propagation [16]. They include uncontrolled propagation of unhandled exceptions, unnoticed task completion and propagation of an exception outside the scope in which it is visible as an exception. This propagation is implicit and can easily get out of the programmer's control. It is impossible to learn the origin of an exception.


differentiate between internal and external exceptions and even exceptions that are not visible in the containing scope can be propagated. This propagation is implicit and can easily get out of the programmer's control. It is impossible to learn the origin of an exception.

shortcomings of the existing exception-handling mechanisms and methodologies for their use, as well as techniques for proving and analyzing exception-handling mechanisms and methodologies.
When others choice allows a treatment of an exception of its own. Moreover, when a task is completed because of an exception one cannot learn the reason as new language features that allow exception propagation by terminating with an exception, and this exception is not a part of the task interface. We would not need such a feature if we could guarantee that errors are contained in the accept blocks and that recovering only one of the tasks is always enough. But there is much evidence that cooperative concurrent systems need cooperative recovery because erroneous information can be smuggled to a number of tasks and because errors can be caused by erroneous patterns of cooperative behaviour [10].

3. Bad Practice

In this section we briefly analyse the main ways of misusing Ada exception handling. There is some solid research on the topic [4, 6, 16]. Clearly the problems discussed in sections 2 and 3 are related because sound language features prevent misuse. The challenge is to improve the exception handling mechanism in ways that make misuse more difficult. We explore there three different ways to improve the Ada exception handling mechanisms in ways that avoid the problems in section 2 and the problems expressed in section 3. The two approaches are:

- The combination of exception handling and OO programming is unnecessary. Exception handling and OO programming are orthogonal concepts which are not strictly related. The combination of exception handling and OO programming is unnecessary.
- The combination of exception handling and OO programming is useful. Exception handling and OO programming are natural companions and should be combined to improve the Ada exception handling facilities. The combination of exception handling and OO programming is useful.

3.1. When Others

The unspecified exception that can only be very general and imprecise [5]. Although a firewall, there is no way to learn what the exception was, and the clause complicates system verification [16]. Programmers are recommended to use raises a predefined exception in the caller. It is not difficult to see why this is unsuitable for exception propagation.

- An attempt at a rendezvous with a non-existing task.
- An exception propagated outside a rendezvous block.
- Another problem is discussed in [5]: When several tasks are involved in the handling of an exception.

A serious restriction is that exceptions are not classes and are not objects. In some situations, it is impossible to determine which call has resulted in signaling a particular exception. Similarly, operations on different types can propagate the same predefined exceptions, which basically contradicts the idea of typing. The solution is to introduce predefined exceptions as classes, in which case each exception will belong to an instance of a type. This makes it possible to identify which call returns a predefined exception and to design different handlers for different calls.

In an interesting research effort, Ada was extended to allow exception handlers to be introduced at the object (package) level [9]. A number of Ada applications were analysed to demonstrate that object level exception handling is useful. This result is widely cited by proponents of exception handling and OO programming.

4. Conclusion

Adopting object-oriented concepts (or for that matter object-oriented versions of constructs like exception handling will pay enough attention to exception handling nor understand it well. Practical guidelines can improve language usage. They should contain realistic sample programs, typical examples of misuse and common pitfalls. In the following, we discuss some typical forms of misuse and some problems with the current Ada exception handling constructs.
4.1. Tools

4.2. Related Research on Exception Handling

A tool for static analysis of exception handling is discussed in [22]. It detects all exceptions that can be propagated from a segment of code (including handlers). When used in conjunction with a set of design and code

4.3. Improper Handling

When used in conjunction with a set of design and code-provenance tools, it can help to improve the quality of exception handling. When others raise exceptions, they can help to identify the exception.

3.3. Improper Handling

Several examples of misuse of the Ada exception handling facility can be found in [14,16]. Improper handling causes several errors, some of which can hardly be detected, and can lead to unexpected behavior.
Many researchers emphasise the importance of exception handling mechanisms in software development. Another approach is to develop new exception handling mechanisms or to adjust existing ones to concurrent, object-oriented or safe programming environments.

4.4. Specification Languages

The specification language Anna [19] introduces formal specification into Ada programs to make it easier for programmers to design software prior to implementation and to maintain and explain software. The specification is developed as a set of annotations inserted into the program code as Ada comments. It is symptomatic that the authors introduce exception propagation annotation to be used in specifying subprograms. The idea is to formally specify both the state of the calling environment when an exception is propagated and a condition that an exception must be propagated.

5. Possible Solutions

One approach would be to develop a number of conventions and methodologies helping programmers to avoid problems and bad practice [2]. The conventions can be backed by tools such as a pre-compiler. A body of abstractions using standard exception handling falls into this category (see section 4.3). Another idea is to develop design patterns [13] to avoid some of the problems. These directions need additional efforts, particularly in the context of Ada 95.

Recently a number of fault tolerance schemes have been developed using standard Ada, all of which rely on a set of programming conventions and on reusable code that is adjusted for a particular application. Researchers working in fault tolerance have long realised that the best way to put their schemes into practice is not to introduce new language constructs but to develop conventions and provide reusable components. Some of these schemes introduce new approaches to handling exceptions, and system partitioning with re-execution is an example.

4.2. Analysing the Experience

It is becoming invaluable to share experience of using Ada exception handling and show realistic examples, both of the experience of programmers trying to use the language and the difficulties they encountered. Often the most difficult to use features are also those that one would expect to use in normal programming.

4.1. Exception Handling Techniques

As mentioned earlier, a number of papers discuss patterns of Ada exception handling, and some of the best techniques are described in these papers. This section describes some of those patterns and discusses their applicability to Ada.

5. Possible Solutions

Exception handling can be adjusted to develop a number of conventions and methodologies helping programmers to avoid problems and bad practice [2]. The conventions can be backed by tools such as a pre-compiler. A body of abstractions using standard exception handling falls into this category (see section 4.3). Another idea is to develop design patterns [13] to avoid some of the problems. These directions need additional efforts, particularly in the context of Ada 95.

Recently a number of fault tolerance schemes have been developed using standard Ada, all of which rely on a set of programming conventions and on reusable code that is adjusted for a particular application. Researchers working in fault tolerance have long realised that the best way to put their schemes into practice is not to introduce new language constructs but to develop conventions and provide reusable components. Some of these schemes introduce new approaches to handling exceptions, and system partitioning with re-execution is an example.

4.2. Analysing the Experience

It is becoming invaluable to share experience of using Ada exception handling and show realistic examples, both of the experience of programmers trying to use the language and the difficulties they encountered. Often the most difficult to use features are also those that one would expect to use in normal programming.

4.1. Exception Handling Techniques

As mentioned earlier, a number of papers discuss patterns of Ada exception handling, and some of the best techniques are described in these papers. This section describes some of those patterns and discusses their applicability to Ada.

5. Possible Solutions

Exception handling can be adjusted to develop a number of conventions and methodologies helping programmers to avoid problems and bad practice [2]. The conventions can be backed by tools such as a pre-compiler. A body of abstractions using standard exception handling falls into this category (see section 4.3). Another idea is to develop design patterns [13] to avoid some of the problems. These directions need additional efforts, particularly in the context of Ada 95.

Recently a number of fault tolerance schemes have been developed using standard Ada, all of which rely on a set of programming conventions and on reusable code that is adjusted for a particular application. Researchers working in fault tolerance have long realised that the best way to put their schemes into practice is not to introduce new language constructs but to develop conventions and provide reusable components. Some of these schemes introduce new approaches to handling exceptions, and system partitioning with re-execution is an example.

4.2. Analysing the Experience

It is becoming invaluable to share experience of using Ada exception handling and show realistic examples, both of the experience of programmers trying to use the language and the difficulties they encountered. Often the most difficult to use features are also those that one would expect to use in normal programming.
6. Conclusions

It is our belief that the general solution for the problems discussed lies in the following:

• Improving exception handling support in the run-time environment by means of interfaces, class inheritance, and method overriding.
• Sharing good practice and supporting it by means of guidelines, best practices, and software tools.
• Developing new exception handling mechanisms.
• Extending the Ada programming language with new elements.

7. Acknowledgment

The research described in this paper was partially supported by the European Commission through the IST Project DOTS, under Grant Agreement IST-2000-28193.

References:


Keywords:

- Exception handling
- Ada
- Exception propagation
- Run-time environment
- Software reliability


