Developing a Profile for Using Object-Oriented Ada in High-Integrity Systems

Jean-Pierre Rosen
Adalog
2 rue du Docteur Lombard
92441 ISSY LES MOULINEAUX CEDEX
FRANCE
+33 (1) 45 29 21 52
rosen@adalog.fr

ABSTRACT

This paper reports on the workshop that was held on Oct 27th, 2010 during the SIGAda conference in Fairfax, VA.

The goal of the workshop was to start an effort for defining a restricted profile for Object Oriented programming with Ada for high integrity systems, following the example of the Ravenscar profile for the concurrency aspects.

Categories and Subject Descriptors

D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification – *Reliability.*

General Terms: Measurement, Reliability, Verification.

Keywords: Ada, safety high reliability systems, Ada 2005, Static analysis, AdaControl.

1. INTRODUCTION

Object Oriented Technologies (OOT) introduce a number of features that provide increased flexibility and reusability in software. However, in the context of high reliability systems such as those subject to DO178B[1] (for airborne systems) or EN-50128[2] (for railway systems), those same features may make certification more difficult, at least for the highest integrity levels.

Ada provides all the necessary tools for object-oriented programming, but does not require it. It features also the notion of restricted *profiles* that define sets of restrictions, intended for example to make behavior more deterministic and/or provable.

Such a successful profile is the Ravenscar profile**Error! Reference source not found.**, which defines a subset of Ada's tasking features that make concurrency acceptable up to the highest integrity levels. The goal of the workshop was to assess the possibility of defining a similar profile for OO features.

2. OOTiA, DO178-C, AND THE PUSH FOR OOT IN AIRBORNE SYSTEMS

The current issue of the DO-178, DO-178B defines objectives, activities intended to fulfill these objectives, and way of assessing that the objectives have been met. It does not require nor forbid any particular technique for meeting the objectives; as such it does not preclude the use of OOT. It does not provide special guidance either, and in practice, applying the requirements of DO-178B to OO software is quite difficult.

In order to address these issues, two workshop were held under the sponsorship of FAA and NASA in April 2002 and March 2003 to produce a document called OOTiA[3] (*Object Oriented Technologies in Aviation*). This document addresses the issues of using OOT in airborne systems, and provides various directions for solving the problems; it is not intended to give definite answers, bur rather to explore several possible ways.

A revision effort was started to update DO-178B, with the aim of producing a new version, to be called DO-178C. At the time of writing., this effort is still going on, but approaching completion. There is no major change to the spirit of DO-178B, but mainly the incorporation of various clarifications, and the addition of four annexes related to model based design, object oriented technology, formal methods, and tool qualification.

Although there is nothing (officially) specific to any programming language, these documents address mainly the concerns of the C++ and Java communities, with few considerations for the specificities of Ada's OO model. For example, dynamic memory management and virtualization (the use of a virtual machine) are part of the OO supplement, while tasking or function pointers were left out due to not being specific to OOT (which is true, but not less than virtualization or memory management).

3. ADA'S PLACE

Ada features some unique possibilities for OO programming. Foremost is the fact that OO does not, in itself, depend on dynamic memory. The distinction between a specific type and a class-wide type, allows a clear distinction between real *methods* (i.e. operation whose realization depends on the class of the object it applies to) and *class operations* (i.e. operations that operate the same on any object belonging to a certain inheritance tree, generally a combination of calls to methods).[1]

Unlike "pure" OO languages (like Java or C#, but not C++), Ada doesn't require everything to be expressed as classes and objects.

4. TO REDISPATCH OR NOT?

But why is the use of OOT in airborne systems a problem in the first place? The main issue relates to the testing of dispatching calls.

[1] To be honest, it is possible to define class operations in other languages, but the lack of clearly identified class-wide types makes it more difficult – and rarely a regular programming style in practice.

The starting hypothesis is that, by the time of certification, the program is complete, and all the classes involved are known (i.e. dynamic loading of classes is a no-no for certification). Given this precondition, a dispatching call is equivalent to a "big case", where each branch would contain a static call to each possible implementation of the method. This was the main path advocated by OOTiA.

Unfortunately, a simplistic testing based only on the "big case" analogy would quickly lead to a combinatorial explosion of test cases. Consider the following example:

```
procedure P (X : T); -- a primitive operation of T

procedure Q (X : T) is -- even if not primitive
begin
        P (T'Class (X)); -- Redispatching (1)
end Q;
```

Although Q appears to be a simple operation of T (it is not a method), it must still be tested for all descendant classes or T, since it is possible to write:

```
type D is new T with ... ;
overriding procedure P (X : D); -- Overriding of P

V : D;
...
Q (T (V));
```

A call to Q would dispatch at (1) to the new definition of P, therefore requiring testing of Q itself with objects of type D. The root of the problem comes from the "forced" dispatching call. These forced dispatching calls are what we call "redispatching".

In its current state, DO178C proposes to solve this issue by relying on proofs of substitutability, as theorized by the LSP (Liskov Substitution Principle)**Error! Reference source not found.**. In short, if it can be demonstrated that all "contracts" (in the general sense) of the parent class are obeyed by descendant classes, this implies that tests can be run only for the parent class, and that success of the tests will imply that the behavior is also valid for all descendent classes, without having to test explicitly with descendent objects.

While this approach has the merit of curing the combinatorial explosion problem, it has also weak points, since it relies on the precise definition of a contract to be observed by the whole inheritance tree, including real-time properties such as WCET or blocking properties; it remains to be seen if this approach would be acceptable for the highest safety levels (A and B), since it relies on formal proofs for accepting combination of parameters that have not been tested.

Ada may offer alternative solutions. With the Ravenscar profile, the notion of a well defined set of restrictions has proved successful for allowing the use of a (restricted) concurrency model even in the most demanding contexts. Could it be possible to define a similar profile that would limit redispatching in a way that would make full testing practical?

Such a strategy could be, for example, to forbid redispatching from within methods, while still allowing it in class-wide operations. This way, methods would need testing only for their associate *specific* types. Class-wide operations would need testing for all possible objects in the class-wide type, but without the combinatorial explosion effect.

Such a design pattern might seem limiting – and it is. But we are talking here about level A-B software; allowing OOT at these levels, even with those limitations, would be a huge step forward compared to the current situation.

5. NEXT STEPS, AND HOW TO CONTRIBUTE

The general idea of exploring if and how the definition of a restricted profile would make OOT in Ada more acceptable for the highest integrity levels received general agreement during the workshop.

The goal is eventually to produce a document in the same form as the Ravenscar profile: a profile that defines a set of restrictions, plus a rationale to explain the restrictions and provide example on how to program within the restrictions. If this is successful, the document could be proposed as a technical report to ISO. Note however that since implementations are allowed to add implementation-defined restrictions, the profile could be implemented in compilers independently of any ISO blessing.

A mailing list has been established; if you are interested, whether to actively participate or just observe, send a mail to rosen@adalog.fr to be added to the list.

Clyde Roby (our faithful webmaster) has set up a WiKi at http://wiki.acm.org/sigada-HROOT. You'll need a user account to access it, please get in touch with me if interested.

Activities include:

- Studying the feasibility of a design pattern that would still allow extensive (so-called pessimistic) testing, while retaining a substantial part of OO benefits.
- Gathering possible restrictions. Everybody is welcome to send proposed restrictions, preferably with some rationale, examples, etc. First proposals, and a template for sending new proposals, are available from Adalog's site at http://www.adalog.fr/hr-oo-workshop/
- Liaising with other interested parties to get feedback from other communities.

The Ada-Europe 2011 conference in Edinburgh will feature a panel on OO techniques for high reliability systems, where this approach will be presented.

6. REFERENCES

[1] RTCA/EuroCAE, DO-178B, ``Software Considerations in Airborne Systems and Equipment Certification"

[2] EN-50128:2001. Railway applications. Communications, signaling and processing systems. Software for railway control and protection systems. ISBN 058037584 6

[3] Alan Burns, Brian Dobbing and Tullio Vardanega. "Guide for the use of the Ada Ravenscar Profile in high integrity systems". ACM SIGAda Ada Letters XXIV (2): 1–74.OOTiA

[4] AdaControl web site, http://www.adalog.fr/adacontrol2.htm

[5] B. Liskov and J. Wing, ``A Behavioral Notion of   Subtyping'', ACM Transactions on Programming Languages and Systems, Nov. 1994, vol. 16, no. 6, ACM, New-York.