Improving Software Development by using Safe Object Oriented Development : OTCD

Xavier Méhaut, Pierre Morère

Ada & Safety Skill Center

AONIX
66-68 avenue Pierre Brossolette
F-92247 Malakoff - FRANCE
xavier.mehaut@aonix.fr
pierre.morere@aonix.fr

Abstract. Starting in early 90's with the first COTS certifiable executive (C-SMART), and continuing through the end of the decade with the first certifiable implementation of the Ravenscar profile (ObjectAda Raven™), Aonix has had a long history of experience in Ada language based Safety Critical applications, especially in transportation (avionics or rail). Aonix has developed a solution which takes advantage of a fully object oriented approach related to three major emerging trends in this domain: the use of Object Oriented methodology (in analysis, design and coding) in the real-time embedded market, the birth of new areas for certification such as the space industry (for both ground-based and on-board missions) and the increasing complexity of the applications which need to be certified. The main point of this process is called OTCD- Optimized Technique for Certification of Dispatching. This paper explains the main phases of the process and the potential cost reduction benefits.

Introduction

The use of an Object Oriented approach is not new in embedded mission critical applications. Some companies in the military aircraft, space, and ground transportation industries have used and continue to use an OMT or HOOD design approach[8], although the use of Ada83 features during implementation may not allow those companies to translate their designs in straightforward and natural ways. With Ada95, there exists greater traceability between the design and the implementation (eg. benefiting from the use of data extension features). Aonix has been involved in several certification projects and in particular in the SAE (Safe Ada Executive) project which is currently in use on the ATV (Automatic Transport Vehicle) under the name T-SMART [1]. Aonix is now working on a definition of a UML subset and an extension to the Ada95 subset known as the Ravenscar's Profile [2]. In this paper we will first describe the state of art regarding the use of object oriented in Safety Critical system. In a second step we will present a simple system used for the purpose of illustration. We will describe the current way to implement

this kind of system using Ada83 approach. We will propose an implementation using new Ada95 features. We will compare the two kinds of implementation. Finally, we will describe the OTCD approach; an approach primarily designed for applications needing to be certified at the highest levels (Level A and B of DO-78B [3], or SIL 4 and SIL 3 of EN-50128 [4]).

Use of Object Oriented Programming in Safety Critical System

1994 [4] points out that with regard to safety aspects, it is not clear whether object-oriented languages should be preferred to conventional ones. It is amazing to note that 6 years later this is still not clear. On the one hand, the use of abstraction and encapsulation is fully accepted and even highly recommended in several standards or norms. On the other hand, the use of any non-deterministic and dynamic feature linked to object-oriented programming is forbidden. Therefore, constructors/destructors as well as dynamic object creation are not allowed. Moreover the use of access types is restricted to static structures only.

Today the Ada community agrees on the fact that the dynamic features of Ada95 should be excluded for use in High Integrity Applications [9]. Nevertheless a large number of applications which use these kinds of features have been certified. This is due to the fact that some of these feature are safe as a result of special implemention characteristics. In general, high level restrictions (based on language semantics) may be eased due to implementation details. In this paper we will see how an a-priori forbidden feature (dynamic dispatching) might be used because the characteristics of its implementation.

Object Oriented System Description

The implementation of a protocol has been chosen in order to illustrate the advantages of using Object Oriented programming for improving the software development process. Two distant systems communicate through a message passing mechanism. The messages may be of various lengths, containing different kinds of frames (see Fig 1.).

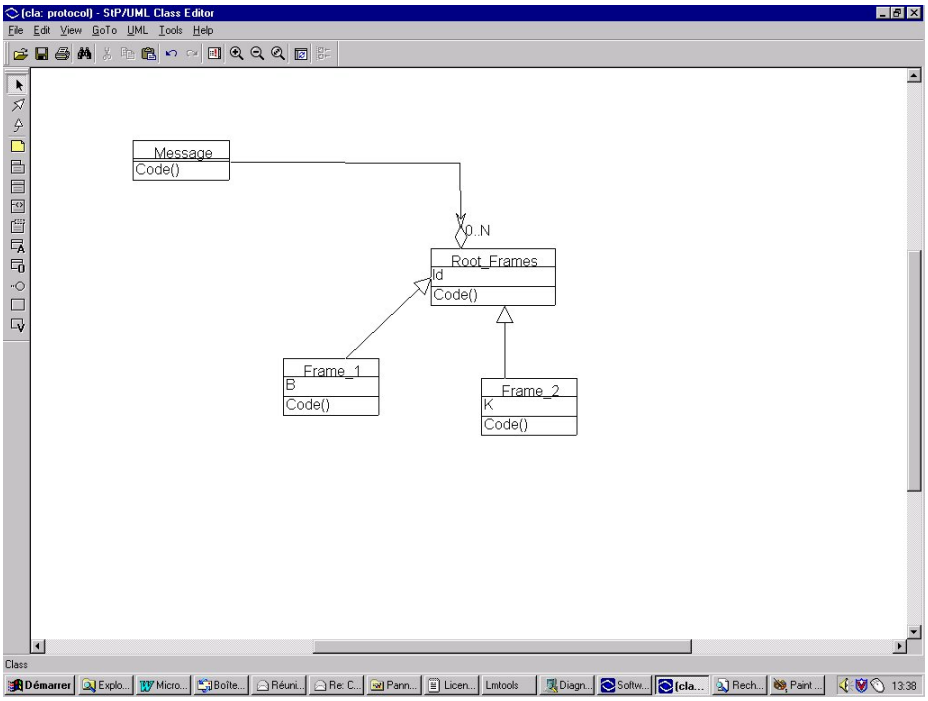

Fig. 1. UML Classes diagram

Development without dispatching

There are several features in Ada83 which support Object Oriented Design such as, encapsulation and abstraction (private and limited types), polymorphism (overloading of subprograms) and inheritance without data extension (derived types). In order to ensure upward compatibility these features have been kept in Ada95. The use of unconstraint records with discriminants with default values can be used to implement type hierarchies. The use of such a feature requires a specific approach with respect to certification. This is mainly due to compiler behavior. For instance, during the life of the application, it is possible for objects declared with discriminants with default values to undergo structural changes as a result of complete assignments. Given the fact that the size of such objects can change, a compiler might choose to create these objects :

1. in the heap : If the size changes, the compiler allocates a new structure and frees the old one. Of course such a solution is forbidden for use in safety applications. For this reason, this kind of type is not permitted to be used in high integrity software.
2. on the stack : The maximum size is allocated on the stack, and all the structural changes of the object are done in this area. Of course this solution is memory

consuming, but with such an approach, the use of these kinds of types is allowed by the certification authority.

In the solution explained hereafter, we assume that the second solution[1] is supported by the compiler, and is acceptable to the certification authority.

First Version

The code of the high level package which describes the frame type is the following.

```
package All_Frames is

    type Frame_Kind is (Frame_1, Frame_2);

    type Frame (Kind : Frame_Kind := Frame_1) is record

        Id : Integer;

        case Kind is

            when Frame_1 => B : Boolean;

            when Frame_2 => K : Integer;

        end case;

    end record;

    type Index is range 1 .. 2;

    type Frames_Set is array (Index) of Frame;

    procedure Code (F : in Frame);

end All_Frames;
```

For objects of type Frame_Set, the maximum size is used for each component of Frame type. This solution is memory consuming but allows the declaration of arrays with component of variable size.

Evolution

In order to accommodate the introduction of a new kind of frame, the enumerated type used for the definition of the record needs to change. This implies a

[1] A compiler could use these two approaches, for example in AdaWorld cross 68k (Aonix compilers), transient records with a maximum size less than 8 kilo-bytes are allocated on the stack whereas objects with a maximum size greater than 8 kilo-bytes are allocated in the heap. A technical note on this subject is available upon demand.

recompilation of the entire system, and all the code needs to be re-tested. One consequence of such an approach is that those objects which require no functional changes are also impacted by this change. This impact is shown in the following figure.

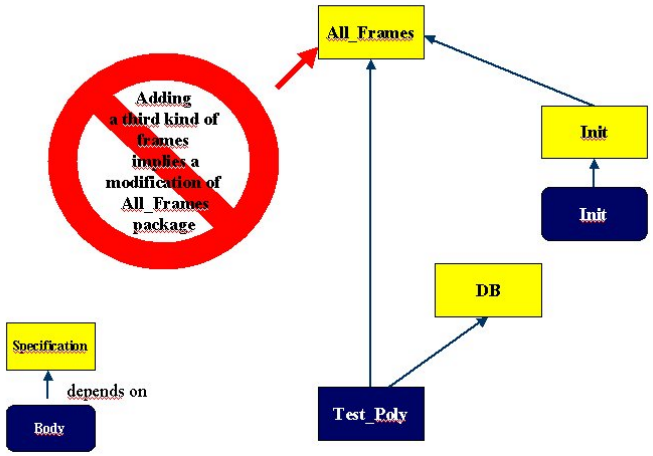

Fig. 2. Dependence graph for Ada83 implementation of the protocol sample

Using new Ada95 features

Ada95 provides new functionality such as tagged type (equivalent to the class notion in C++), inheritance, polymorphism and dispatching. The polymorphism is the fact that the same operation may behave different on different object classes. In conjunction with polymorphism the notion of dispatching is supported by Ada95. The dispatching is the capability to call the correct depending on the actual type of an object addressed through an access. The dispatching could be static (the actual type of the object is known at compile time) or dynamic (the actual type of the object is known at execution time). In the following implementation we will use all these new features.

First Version

The design implementation is conceived in such a way as to minimize the dependence between Ada units (see Fig 3). This is achieved for example, by introducing an abstract type which encompasses all the attributes and methods common to all frames and the methods that will be used to encode and decode messages.

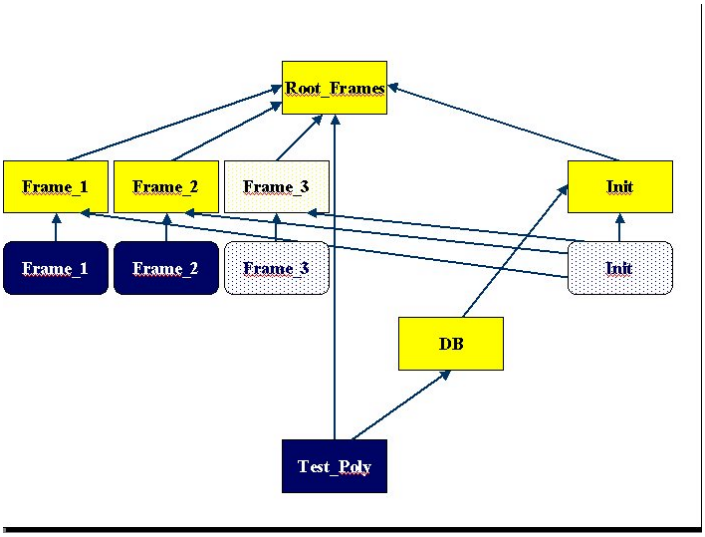

Fig. 3. Dependence graph for Ada95 implementation of the protocol sample

Only the code for the units directly involved in the main subprogram : Root_Frames, DB (for DataBase) and Test_Poly is given

- Root_Frames specification package :

```
package Root_Frames is

    type Frame is abstract tagged record

        Id : Integer;

    end record;

    -- use for the dispatching

    type A_Frame is access all Frame'Class;

    type Index is range 1 .. 2;

    type Frames_Set is array (Index) of A_Frame;

    procedure Code (M : in Frame);

end Root_Frames;
```

- DB specification package

```
with Root_Frames;use Root_Frames;
```

```
with Init; use Init;

package DB is

   Table : constant Frames_Set :=Init_Message_Array;

end DB;
```

- Test_Poly subprogram body :

```
with DB; use DB;

with Root_Frames;use Root_Frames;

procedure TestPoly is

    PA : A_Frame;

begin

    for I in Table'Range loop

        PA := Table (I);

        Code (PA.all);

    end loop;

end;
```

In the Test_Poly main program, the correct Code subprogram is chosen at runtime using the tag of the object which is managed by the compiler.

Evolution

Adding a new kind of frame implies only a modification of the package body Init. The units which have not been modified do not need to be re-tested (unitary, functional, coverage tests). - with the exception of Test_Poly. For this unit, we need only to test that the new kind of frame is properly handled. In this simple sample, only one unit out of nine need be revalidated. The same evolution using the Ada83 solution implies a new complete validation of the application.

Comparison between the two Implementations

Although the first implementation is still used in Ada95, we will call it the Ada83 solution. We have chosen several criteria covering the entire life cycle.

Traceability: both solutions provides good traceability between the design and the code. Nevertheless the Ada95 solution allows a better mapping between a UML class and an Ada class which is defined in only one package.
Modularity: in the Ada83 solution a message needs to have access to the definition of the frame. In general this implies the creation of a large package containing all the type definitions needed for the declarations of the Message type.
Performance: the dispatching code of Ada98 is faster than the code used in the Ada83 implementation.
Control flow determinism: the Ada83 implementation is fully deterministic. The Ada95 implementation requires the use of procedure addresses.
Execution determinism: with the Ada95 solution the time needed to execute the call to the correct operation (depending on the actual type of the object) is constant. This is not the case with the Ada83 solution where the execution time for the selection of the correct operation depends on the evaluation order within the case statement.
Maintainability: the Ada95 solution provides better maintainability as a result of the loose coupling modules.
Certification: the main advantage of the Ada83 solution is that several applications using this approach have been previously certified.
Miscellaneous: Another advantage of the Ada95 solution is its look &feel is familiar to developers who are used to using other Object-Oriented Programming languages such as C++.

As this short non-exhaustive list shows, the main inconvenience of an Ada95 implementation is that it is a new approach whose dynamic aspects are not sufficiently field tested to give confidence to certification authorities. OTCD is a technique which solves this issue by reducing the dynamic aspects of the approach.

OTCD Optimized Technique for Certification of Dispatching

OTCD is based on a knowledge of ObjectAda code generation. The use of this technique has an impact on both the design of the system as well as on the coding phase of development. The main steps are explained hereafter, the whole approach is available upon demand.

Design and coding

There are two main aspects which need to be taken into account during the design phase. The first one is limiting the dependence between data structures and various units. The second one is the handling of static data. The OTCD approach implies that each object subject to dispatching is accessed through reference rather than directly. The link between the message and the frame is created during the elaboration phase of execution, this is done in order to reduce dependencies between units and to avoid unnecessary recompilation. With such an approach, the implementation of a Message object is functionally equivalent to a large static structure (contiguous representation)

which declares all the potential frames within. This kind of representation is not compatible with the dispatching capability of Ada95.

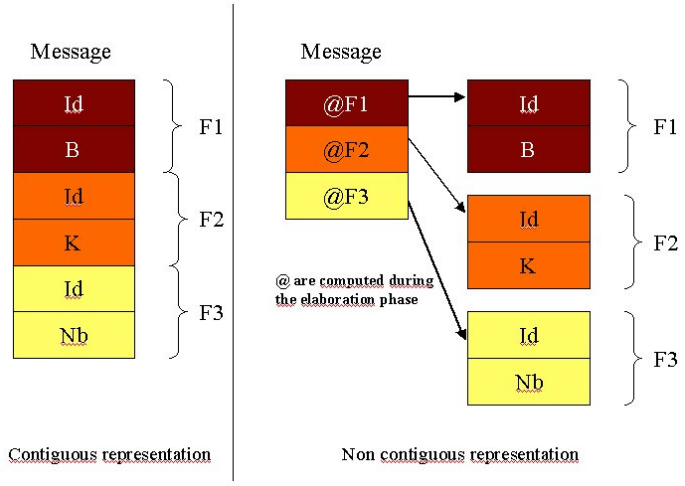

Fig. 4. Comparison between two ways to implement composition

Static Verification

Static verification of the program, requires generating the assembly file corresponding to the Ada units. For example, the following code, using ObjectAda Raven for PowerPC, is generated for the Root_Frames.Frames_1 specification package.

```
# ************* Constant segment :
    |                 .align 8
    |                 .global root_frames__frames_1__frame_1___dispatch_table
    | 000000          root_frames__frames_1__frame_1___dispatch_table:
    |                 .word   root_frames__frames_1__frame_1___expanded_name
    |                 .word   root_frames__frames_1__frame_1___external_tag
    |                 .word   root_frames__frames_1__frame_1___ancestor_table
    |                 .word   root_frames__frames_1__frame_1___su_size
    |                 .word   0x00000000
    |                 .word   0x00000000
    |                 .word   root_frames__frames_1__frame_1___equal
    |                 .word   root_frames__frames_1__frame_1___assign
    |                 .word   0x00000000
    |                 .word   0x00000000
    |                 .word   0x00000000
    |                 .word   0x00000000
    |                 .word   root_frames__frames_1__code__2
```

Fig. 5. Constant section used for the support of dispatching operations

This table needs to linked to the code generated for the dispatching operation done in the test_poly subprogram. The ***root_frames__frames_1__code_2*** is located at the offset 48 of the constant section known under the generic name of "dispatch table".

```
#  Source Line # 8        Code (PA.all);
   8| 00003C 80610008      lwz     r3,8(sp)       # STATIC_FRAME_SEG PA
   8| 000040 90610010      stw     r3,16(sp)      # STATIC_FRAME_SEG
   8| 000044 81810010      lwz     r12,16(sp)     # STATIC_FRAME_SEG
   8| 000048 81610010      lwz     r11,16(sp)     # STATIC_FRAME_SEG
   8| 00004C 814B0000      lwz     r10,0(r11)
   8| 000050 812A0030      lwz     r9,48(r10)
   8| 000054 386C0000      addi    r3,r12,0
   8| 000058 7D2903A6      mtspr   ctr,r9
   8| 00005C 4E800421      bctrl
```

Fig. 6. Code generated for the dispatching operation

In the above code the offset used in order to call the correct subprogram is not computed but is hardcoded. This is due to the technology of the ObjectAda compiler (and is platform-independant). It is easy to verify, in a static way, that all the offsets correspond to the correct offset in the constant table. This is very important because one of the issues when using class type programming and dispatching in high integrity software, as pointed out in [6], is that no static analysis can be in such a case. Using OTCD, the data structures are static, the dispatching tables are static and the offsets are known at compile time.

Test

Of course code coverage should be performed in conjunction with static verification,. This code coverage needs to be performed at the assembly level as mandated in chapter 6.2.4 of [3] in order to be compliant with the level A of criticality. This can be achieved by using a qualified tool such as AdaCover[2] which is fully integrated with ObjectAda Raven technology. One should note that that the generated low level code is sequential, greatly simplifying the coverage test design. The complexity of testing (and of verification) can be compared to the complexity involved in an Ada83 like approach. In the integration phase, the tests need to verify that correct coverage is achieved regarding the actual contents of a message. In case of modification only the new class need be tested.

Conclusion

OTCD allows for an incremental approach for developing safety related software while streamlining cumbersome testing phases at each development iteration. This

[2] AdaCover is a coverage tool qualified against the DO-178B to the CC2 level.

approach is mainly suitable for applications which are data oriented (protocol, data base) and which have a degree of predictability in their future modifications. Using dispatching (as described by OTCD) enables direct traceability between design (eg using UML) and implementation. OTCD also exists for less critical systems whose rules are relaxed in regards to low level coverage and data organization.
It is important to note that the generation of static offsets guaranteed by the ObjectAda technology is linked to the fact that Ada95 does not explicitly support multiple inheritance (see [7]). For other object oriented languages which support multiple inheritance, the offset can not be known at compile time and so no static analysis may be performed.
Related work is currently underway by Aonix in order to describe OTCD as a certifiable design pattern and to associate this approach with a specific test design pattern.

1. Morère, P.: Certifiable Multitasking Kernel: from T-SMART to Raven, DASIA'99, Lisboa 99
2. Dobbing, B.: Real-Time and High Integrity Features of Ada95 using Ravenscar Profile, Ada-Belgium'98, Decmber 1998
3. DO-178B/ED-12B, « *Software considerations in airborne systems and equipment certifications* », RTCA/EUROCAE, December 1992.
4. EN 50128, "Software for railway control & protection systems (CENELEC)."
5. « *Ada 95 Reference Manual* », International Standard ANSI/ISO/IEC-8652:1995, January 1995.
6. "Ada 95 Quality and Style: Guidelines for Professional Programmers"SPC-94093-CMC Version 01.00.10 October 1995 AJPO.
7. "Ada 95 Rationale", January 1995, Intermetrics. Inc
8. Méhaut, X. and Richard-Foy, M.: Enhancing critical software Development using HOORA/HOOD and UML/ROOM, DASIA'99, Lisboa 99
9. HRG Working Group: [GUIDANCE] "*Guidance for the use of the Ada Programming Language in High Integrity Systems*" September 1997