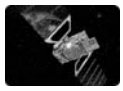
AdaCore
The GNAT Pro Company

Safety, Security, and OOP: Inheritance, Dynamic Binding, Testing

Franco Gasperoni gasperoni@adacore.com
Jose F. Ruiz ruiz@adacore.com

DASIA 2006
2006 Berlin, 22-25 May 2006

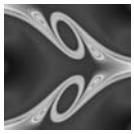
Safety & Security





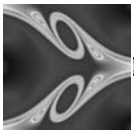


Software Technologies



chaos

Software Technologies

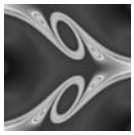


SP

SP = structured programming

chaos

Software Technologies



SP

OOP

OOP = object-oriented programming

chaos

Safety & Security Standards

- **DO-178B / ED-12B (Airborne SW)**
- **DO-278 / ED-109 (Air Traffic Management)**
- **IEC 61508:1999 (Trains, control machinery, …)**
- **ISO/IEC 15408:2005 (Common Criteria)**
- **DEF-STAN 00-56**
- **…**

Have not much to say about OOP vs. SP

AdaCore

Up to Now ....

Most "safe & secure" software has used SP ...

- **... with some trying OOP**

Things are changing rapidly

- **Companies want to explore the of use OOP**
- **OOTiA (Handbook published October 26, 2004)**
  - Object-oriented Technology in Aviation
  - http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/oot/
- **DO-178C due 3 years from now (or so)**

AdaCore

DO-248: SP vs. OOP

DO-248 (2001 - clarification of DO-178B) – 3.32:

- Avoidance of non-determinism: Non-determinism may be reduced through one or more of the following practices:
  1. Choose a language with a well-defined standard.
  2. Do not permit self-modifying code.
  3. Minimize memory paging and swapping.
  4. Avoid use of dynamic binding, memory allocation, and memory deallocation.
  5. Do not assume initial values for variables that are not explicitly initialized.
- Avoidance of complexity: Complexity may be reduced through the following practices:
  1. Maximize cohesion; minimize coupling.
  2. Make cautious use of overloading of operators and variable names.
  3. Use a single point of entry and exit for subprograms.
  4. Minimize use of interrupt-driven processing.
  5. Minimize use of multi-tasking/multi-processing in the architecture.
  6. Beware of side effects in the use of built-in functions and compiled libraries.
  7. Control the class library depth (inheritance tree depth).

AdaCore

DO-248 In a Nutshell

- **Feel free to use OOP**
- **But be careful at what OOP features you use**
- **And at how you use them**

AdaCore

DO-248 In a Nutshell

- **Feel free to use OOP**
- **But be careful at what OOP features you use**
- **And at how you use them**

Feel free to eat the pie as long as there is no intake of fat

AdaCore

That is a reasonable approach

- **Use static OOP in safety/security-critical SW**

- **See SPARK (http://sparkada.com)**

SPARKAda
Praxis High Integrity Systems

AdaCore

DO-248: SP vs. OOP

DO-248 (2001 - clarification of DO-178B) – 3.32:

- Avoidance of non-determinism: Non-determinism may be reduced through one or more of the following practices:
  1. Choose a language with a well-defined standard.
  2. Do not permit self-modifying code.
  3. Minimize memory paging and swapping.
  4. Avoid use of dynamic binding, memory allocation, and memory deallocation.
  5. Do not assume initial values for variables that are not explicitly initialized.
- Avoidance of complexity: Complexity may be reduced through the following practices:
  1. Maximize cohesion; minimize coupling.
  2. Make cautious use of overloading of operators and variable names.
  3. Use a single point of entry and exit for subprograms.
  4. Minimize use of interrupt-driven processing.
  5. Minimize use of multi-tasking/multi-processing in the architecture.
  6. Beware of side effects in the use of built-in functions and compiled libraries.
  7. Control the class library depth (inheritance tree depth).

AdaCore

Coupling

- **Control coupling**
  - The manner or degree by which one software component influences the execution of another software component
- **Data coupling**
  - The dependence of a software component on data not exclusively under the control of that software component

Slide 13

AdaCore

SP vs. OOP

Types	Operations					
	op1()	op2()	op3()	op4()	op5()	op6()
A	X		X		X	
B	X	X		X		X
C	X	X	X			X
D		X	X		X	

Slide 14

AdaCore

SP: A centralized view of SW

A subprogram

Types	Operations					
	op1()	op2()	op3()	op4()	op5()	op6()
A	X		X		X	
B	X	X		X		X
C	X	X	X			X
D		X	X		X	

Slide 15

AdaCore

OOP: A distributed view of SW

A class

Types	Operations					
	op1()	op2()	op3()	op4()	op5()	op6()
A	X		X		X	
B	X	X		X		X
C	X	X	X			X
D		X	X		X	

Slide 16

AdaCore

OOP: A distributed view of SW

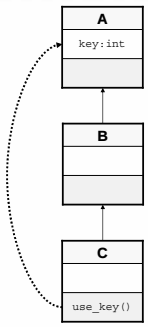
Does this chopping increase coupling?

Types	Operations					
	op1()	op2()	op3()	op4()	op5()	op6()
A	X		X		X	
B	X	X		X		X
C	X	X	X			X
D		X	X		X	

Slide 17

AdaCore

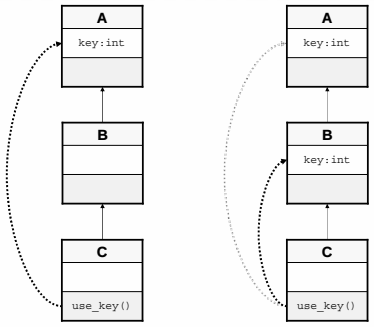
An example of what can go wrong



Slide 18

AdaCore

An example of what can go wrong



Slide 19

AdaCore

DO-248: SP vs. OOP

DO-248 (2001 - clarification of DO-178B) – 3.32:

- Avoidance of non-determinism: Non-determinism may be reduced through one or more of the following practices:
  1. Choose a language with a well-defined standard.
  2. Do not permit self-modifying code.
  3. Minimize memory paging and swapping.
  4. Avoid use of dynamic binding, memory allocation, and memory deallocation.
  5. Do not assume initial values for variables that are not explicitly initialized.
- Avoidance of complexity: Complexity may be reduced through the following practices:
  1. Maximize cohesion; minimize coupling.
  2. Make cautious use of overloading of operators and variable names.
  3. Use a single point of entry and exit for subprograms.
  4. Minimize use of interrupt-driven processing.
  5. Minimize use of multi-tasking/multi-processing in the architecture.
  6. Beware of side effects in the use of built-in functions and compiled libraries.
  7. Control the class library depth (inheritance tree depth).

Slide 20

AdaCore

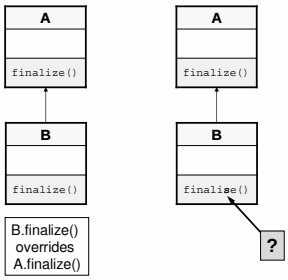
Overloading

- **Overloading is a fundamental part of OOP**
- **When you mix inheritance and overloading**
- **You get a powerful and explosive mix**
- **Which can lead to serious hazards in a safety & security context**

Slide 21

AdaCore

An example of what can go wrong



Slide 22

AdaCore

A Possible Solution ….

Ada 2005

- **No hiding of field names along a derivation chain**
- **Developer can state whether a method**
  - does **override**
  - does **not override**
  - the corresponding method in the ancestor

Slide 23

AdaCore

DO-248: SP vs. OOP

DO-248 (2001 - clarification of DO-178B) – 3.32:

- Avoidance of non-determinism: Non-determinism may be reduced through one or more of the following practices:
  1. Choose a language with a well-defined standard.
  2. Do not permit self-modifying code.
  3. Minimize memory paging and swapping.
  4. Avoid use of dynamic binding, memory allocation, and memory deallocation.
  5. Do not assume initial values for variables that are not explicitly initialized.
- Avoidance of complexity: Complexity may be reduced through the following practices:
  1. Maximize cohesion; minimize coupling.
  2. Make cautious use of overloading of operators and variable names.
  3. Use a single point of entry and exit for subprograms.
  4. Minimize use of interrupt-driven processing.
  5. Minimize use of multi-tasking/multi-processing in the architecture.
  6. Beware of side effects in the use of built-in functions and compiled libraries.
  7. Control the class library depth (inheritance tree depth).

Slide 24

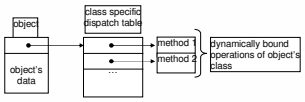
AdaCore

The Challenges of Dynamic Binding

- **Control coupling (e.g. re-dispatching)**
- **Implementation issues: safety & security**
- **Code coverage and testing**

AdaCore

Implementation of Dynamic Binding



- **Correct initialization?**
  - Dispatch tables
  - vtable pointers
- **No unintentional / malicious modification?**
  - Dispatch tables
  - vtable pointers
- **Source-based tools?**
  - Implementation invisible at source level

AdaCore

A Possible Solution

In secure / safety-critical contexts

- **The compiler could implement dynamic binding**
- **With (visible) case statements**
- **This can only be done at bind time**
- **When the entire hierarchy is available**

AdaCore

Testing under Dynamic binding

- **M : number of virtual methods in the program**
- **D : number of dispatching calls**

Two extremes

Code coverage — **Test coverage**

O (M + D)

Case statement
per virtual method signature

???

O (D × M)

Case statement
per dispatching call

AdaCore

Who is looking at these issues

- **DO-178C (http://ultra.pr.erau.edu/SCAS/)**
  - SC-205 / WG-71: SG5: Object-Oriented Technology
- ***ISO/IEC Project 22.24772: Guidance for Avoiding Vulnerabilities through Language Selection and Use* (http://www.aitcnet.org/isai/)**
  - SC 22