



AdaControl

A free ASIS based tool

J-P. Rosen
Adalog

Origin

- Initial development sponsored by Eurocontrol
 - 👉 Organization that manages air traffic over the whole (geographical) Europe.
 - 👉 1,1 MSloc Ada
 - 👉 Not life critical, but business critical
 - 👉 Need to automate programming rules checking
- More rules sponsored by Belgocontrol
 - 👉 Other industrials interested
- Decision by Eurocontrol to make it free
 - 👉 No interest in marketing the product
 - 👉 Benefit from improvements provided by the community
 - 👉 GMGPL: everything can be reused in any context
- Based on ASIS

Why Asis?

Ada is a very rich language...
... and very difficult to compile

Syntax alone doesn't tell much:

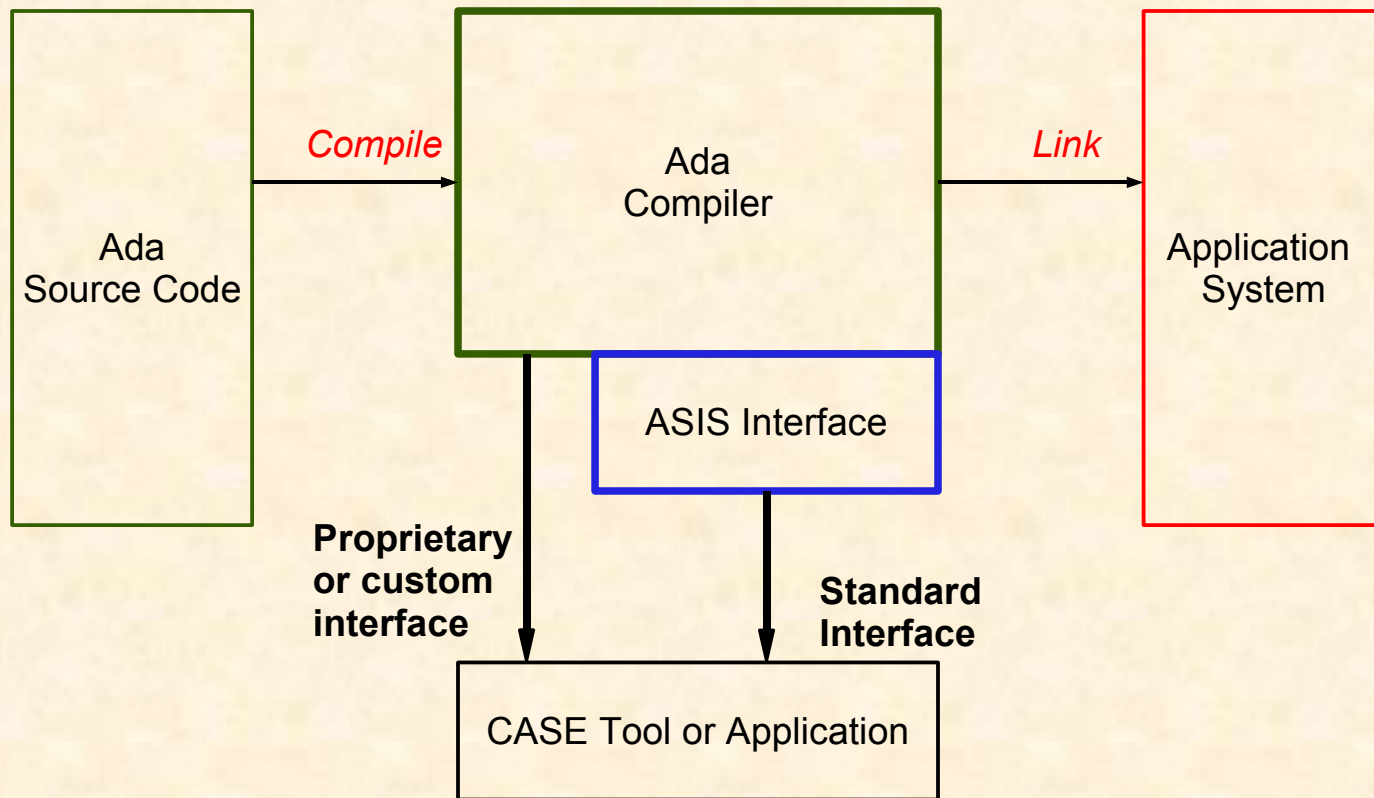
- A (B)
 - 👉 A procedure call (and overloading resolution is needed to tell which one)
 - 👉 A type conversion
 - 👉 An array indexing

You don't want to rewrite half a compiler with each tool!

What is ASIS ?

- The **A**da **S**emantic **I**nterface **S**pecification is an *interface* between an *Ada environment* and any *tool* requiring information from it.
- ASIS is an *open and published (ISO/IEC 15291:1999) callable interface* .
- ASIS has been designed to be *independent of underlying Ada environment implementations*.
 - 👉 Implemented by (almost) every compiler

An interface to the Ada compiler



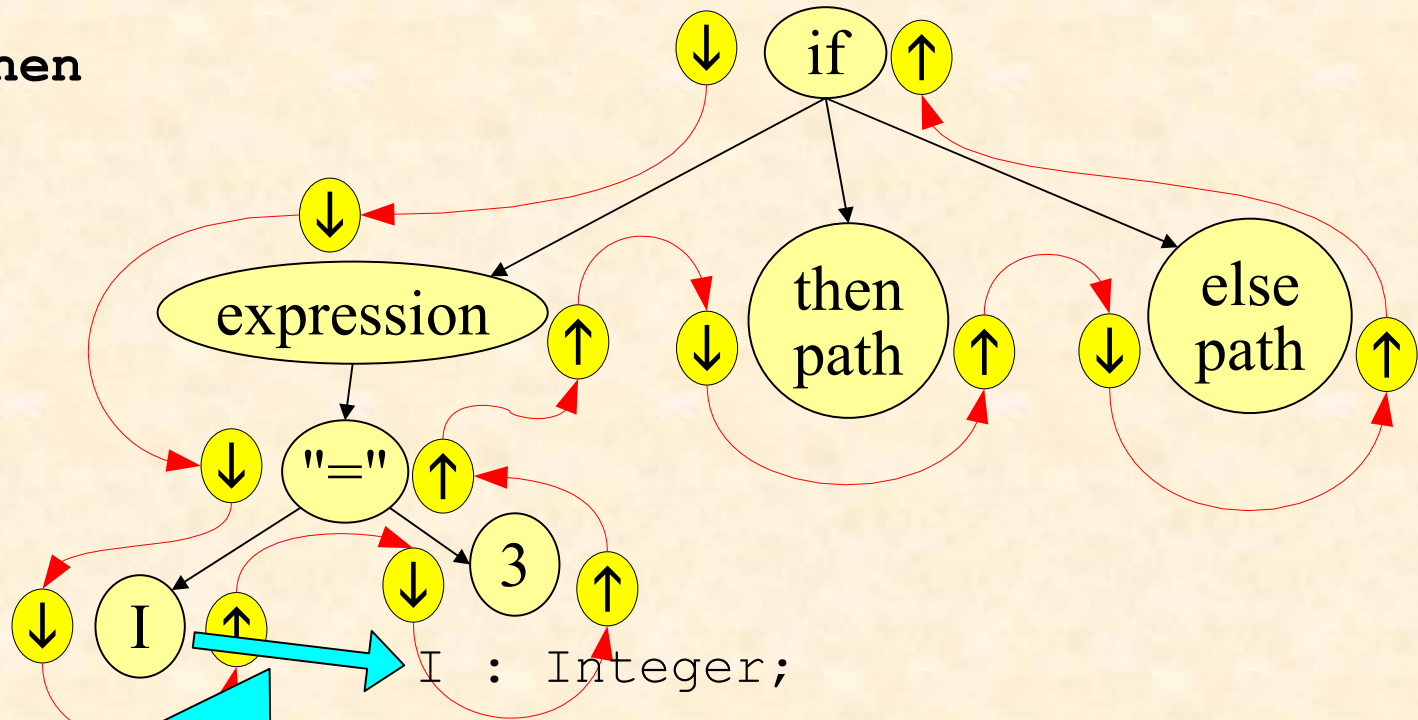
A semantic interface

- API to walk-through and query the decorated syntactic tree of an Ada program.

```

if I=3 then
...
else
...
end if;

```



Corresponding_Name_Declaration

Why bother with checking rules?



- A language provides facilities for all possible programs
 - 👉 In a given program, you don't need everything
 - 👉 Some useful features in one context are dangerous in other contexts
- Common style helps readability, understandability, maintenance
 - 👉 The one who writes the code is *never* the one who reads it.
- Projects have special requirements

A rule without enforcement is worth nothing

Current rules (development version)

Allocators	Max_Blank_Lines	Representation_Clauses
Declarations (x30)	Max_Line_Length	Side_Effect_Parameters
Default_Parameter	Max_Nesting	Silent_Exceptions
Entities	Named_Notation	Simplifiable_Expressions (x4)
Entity_Inside_Exception	Naming_Convention	Special_Comments
Exception_Propagation (x3)	No_Tabs	Specification_Entities (x5)
Global_References	Not_Elaboration_Calls	Statements (x14)
Header_Comments	Parameter_Aliasing	Style (x5)
If_For_Case	Potentially_Blocking_Operations	Unnecessary_Use_Clause
Instantiations	Pragmas	Unsafe_Paired_Calls
Local_Hiding	Real_Operators	Use_Clauses
Local_Instantiation	Reduceable_Scope	When_Others_Range

91 possible checks...and still growing!

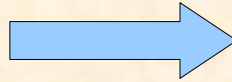
Some rules are simple...

- Textual rules
 - ➡ Max line length
 - ➡ No tabs
 - ➡ Special comments (-- TBSL)
- Use of special constructs
 - ➡ goto
 - ➡ tasking
 - ➡ pointers
- Use of Ada entities

Some very simple rules are extremely useful!

Some are sophisticated

- Programming style
 - 👉 if for case
 - 👉 reduceable scope
- Special patterns
 - 👉 silent exceptions
 - 👉 unsafe paired calls
- Safe programming
 - 👉 Parameter aliasing
 - 👉 Exception propagation
 - 👉 Potentially blocking operations



```
if I = 1 then
    ...
elsif I = 2 then
    ...
elsif I = 3 then
    ...
else
    ...
end if;
```

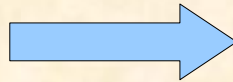
```
case I is
    when 1 =>
        ...
    when 2 =>
        ...
    when 3 =>
        ...
    when others =>
        ...
end case;
```

Some are sophisticated

- Programming style

- 👉 if for case

- 👉 reduceable scope



```
type Counter is range 1..100;  
procedure Counting is  
    -- Counter used only here  
begin  
    ...  
end Counting;
```

- Special patterns

- 👉 silent exceptions

- 👉 unsafe paired calls

- Safe programming

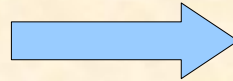
- 👉 Parameter aliasing

- 👉 Exception propagation

- 👉 Potentially blocking operations

Some are sophisticated

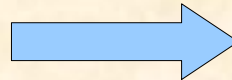
- Programming style
 - ☞ if for case
 - ☞ reduceable scope
- Special patterns
 - ☞ silent exceptions
 - ☞ unsafe paired calls
- Safe programming
 - ☞ Parameter aliasing
 - ☞ Exception propagation
 - ☞ Potentially blocking operations



```
begin
  ...
exception
  when Data_Error =>
    Report ("Bad data");
  when Device_Error =>
    raise Failure;
end;
```

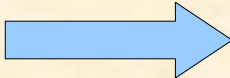
Some are sophisticated

- Programming style
 - ☞ if for case
 - ☞ reduceable scope
- Special patterns
 - ☞ silent exceptions
 - ☞ unsafe paired calls
- Safe programming
 - ☞ Parameter aliasing
 - ☞ Exception propagation
 - ☞ Potentially blocking operations



```
declare
  My_Lock : Lock_Type;
begin
  P (My_Lock);
  -- Do something
  V (My_Lock);
exception
  when others =>
    V (My_Lock);
    -- handle exception
end;
```

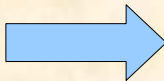
Some are sophisticated

- Programming style
 - ☞ if for case
 - ☞ reduceable scope
- Special patterns
 - ☞ silent exceptions
 - ☞ unsafe paired calls
- Safe programming
 - ☞ Parameter aliasing 
 - ☞ Exception propagation
 - ☞ Potentially blocking operations

```
procedure Merge
  (Left  : in out Element;
   Right : in out Element);


  X : array (1..10) of Element;
  Y : Element renames X (3);
begin
  Merge (X (3), Y); -- Certain
  Merge (X (1), Y); -- Possible
```

Some are sophisticated

- Programming style
 - ☞ if for case
 - ☞ reduceable scope
- Special patterns
 - ☞ silent exceptions
 - ☞ unsafe paired calls
- Safe programming
 - ☞ Parameter aliasing
 - ☞ Exception propagation 
 - ☞ Potentially blocking operation

```
procedure Call_Back;  
pragma Export (C, Call_Back);  
  
procedure Call_Back is  
    ...  
begin  
    ...  
    -- no exception handler  
end Call_Back;
```

Some are sophisticated

- Programming style
 - 👉 if for case
 - 👉 reduceable scope
- Special patterns
 - 👉 silent exceptions
 - 👉 unsafe paired calls
- Safe programming
 - 👉 Parameter aliasing
 - 👉 Exception propagation
 - 👉 Potentially blocking operations 

```
protected Lock is  
  procedure P;  
end Lock;  
  
protected body Lock is  
  procedure P is  
  begin  
    Put_Line ("In P");  
    ...  
  end P;  
end Lock;
```

Command Language

- Rules' syntax

👉 `[name:] Search|Check|Count <Rule> [(<parameters>)];`

- Naming entities

👉 `all My_Func`

👉 `My_Package.My_Func`

👉 `My_Package.My_Func {Integer return Float}`

- Other commands

👉 `Go, Quit, Help`

👉 `Message`

👉 `Clear`

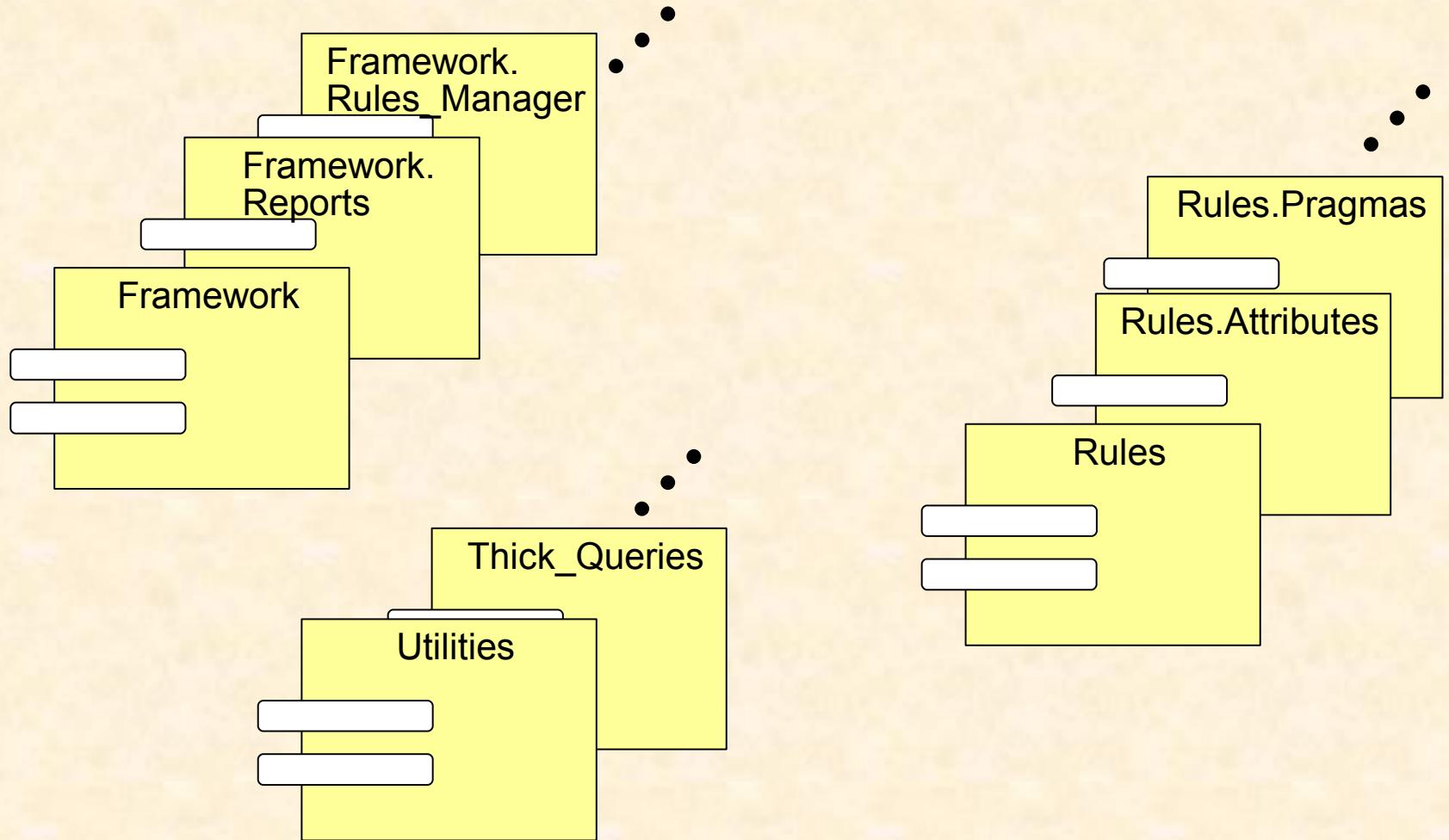
👉 `Inhibit`

👉 `Set (Output, flags)`

Other features

- Rules derogation
 - 👉 Global, single rule, named rule
 - 👉 Over block of source or single line
 - 👉 Possibility to ignore derogations
- Inhibition
 - 👉 "Unplugging" a rule for a given unit
- Interactive mode
- Using Emacs (.adp) projects files
 - 👉 Not (yet) GPS (.gpr) projects
- Integration into GPS
- Return codes

AdaControl's structure



Documentation

- User's Guide
 - 👉 How to use AdaControl
- Programmer's Guide
 - 👉 How to write new rules
- Formats:
 - 👉 Texinfo source
 - 👉 Info
 - 👉 HTML
 - 👉 PDF

Business with AdaControl

- AdaControl is a *commercial* product of Adalog
 - 👉 Not opposed to free!
- Adalog services
 - 👉 Product maintenance
 - 👉 Development of new rules
 - 👉 Help in designing coding rules
 - 👉 Help in checking rules

Lessons learned

- Increases visibility of company
- Show case of Adalog's know-how
 - 👉 Clients asked for custom development of ASIS-based tools.
- The "consortium effect"
- Sponsoring free software is good
 - 👉 For the client who gets more than was paid for
 - 👉 For the company whose business increases