
Towards a new IDL to Ada(2005) Mapping

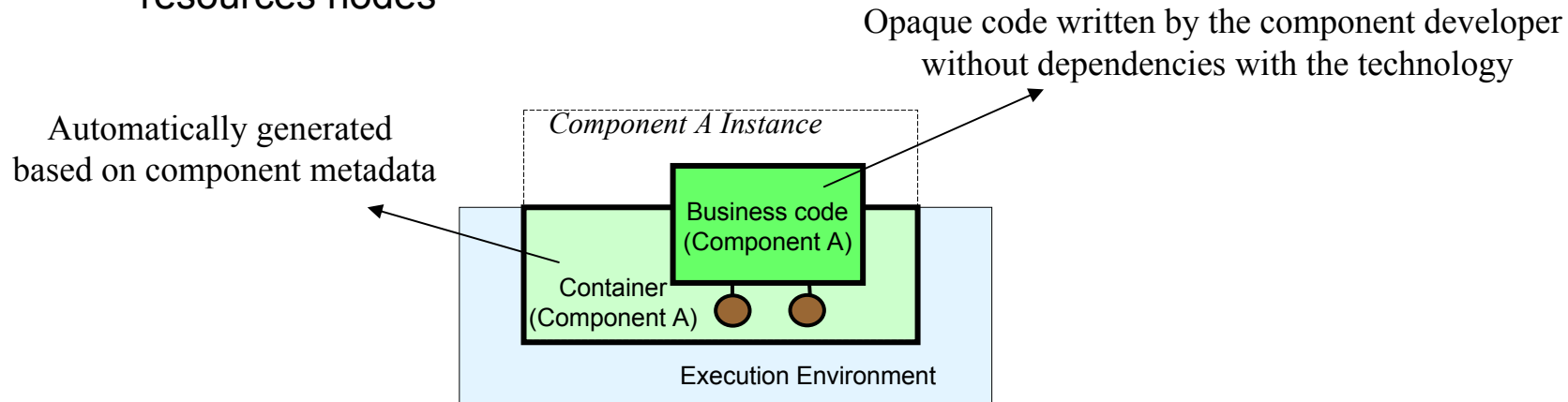
Julio Medina, Patricia López, Pablo Pacheco, and José M^a Drake
Universidad de Cantabria



Grupo de Computadores y Tiempo Real. Universidad de Cantabria. Spain

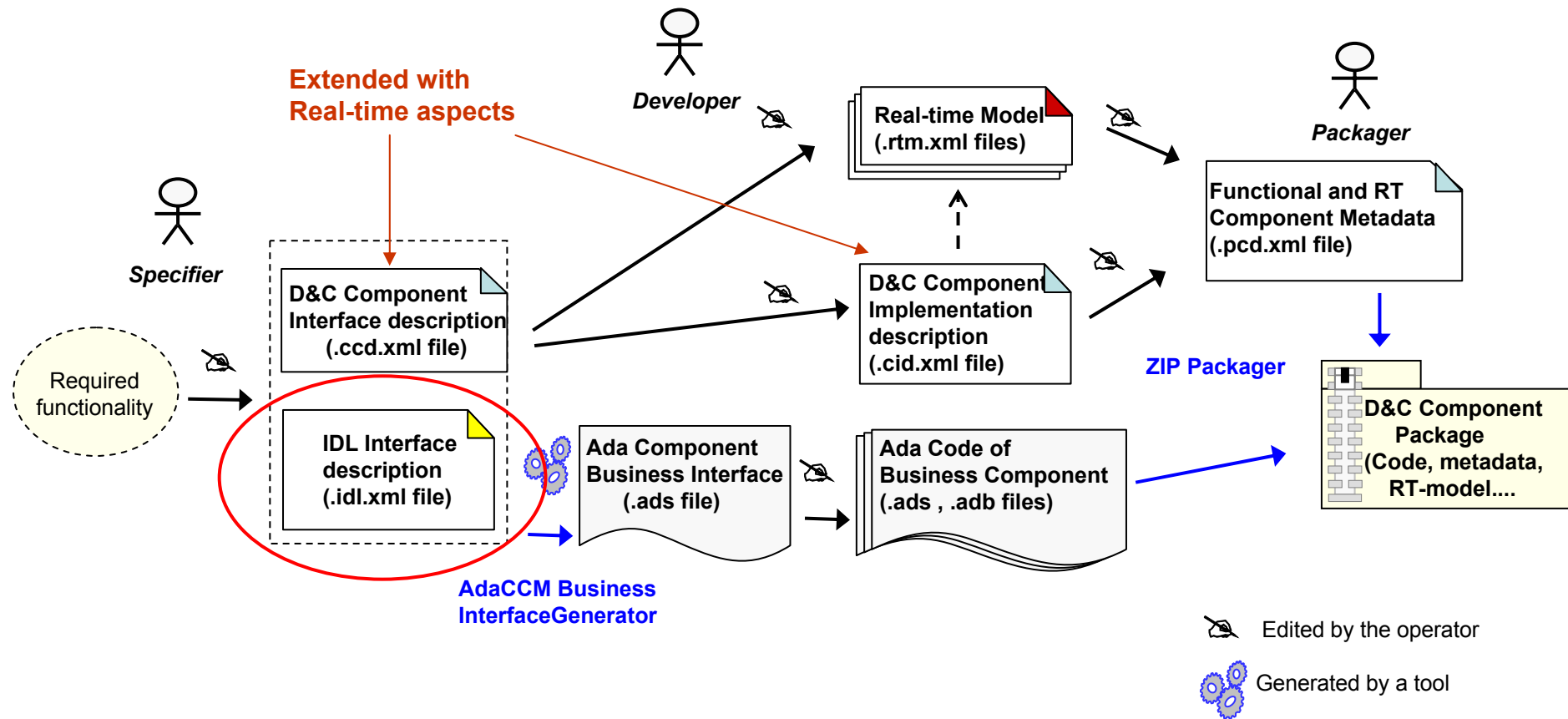
Motivation: The RT-Component

- Current CBSE technologies (EJB, .NET, COM) not suitable for embedded and real-time systems
- Some EU projects (COMPARE, FRESCOR) try to find a component-based technology for embedded real-time and distributed systems:
 - ➔ based on Container Component Model,
 - ➔ OMG's LwCCM but not relying on CORBA => Too heavy for limited resources nodes



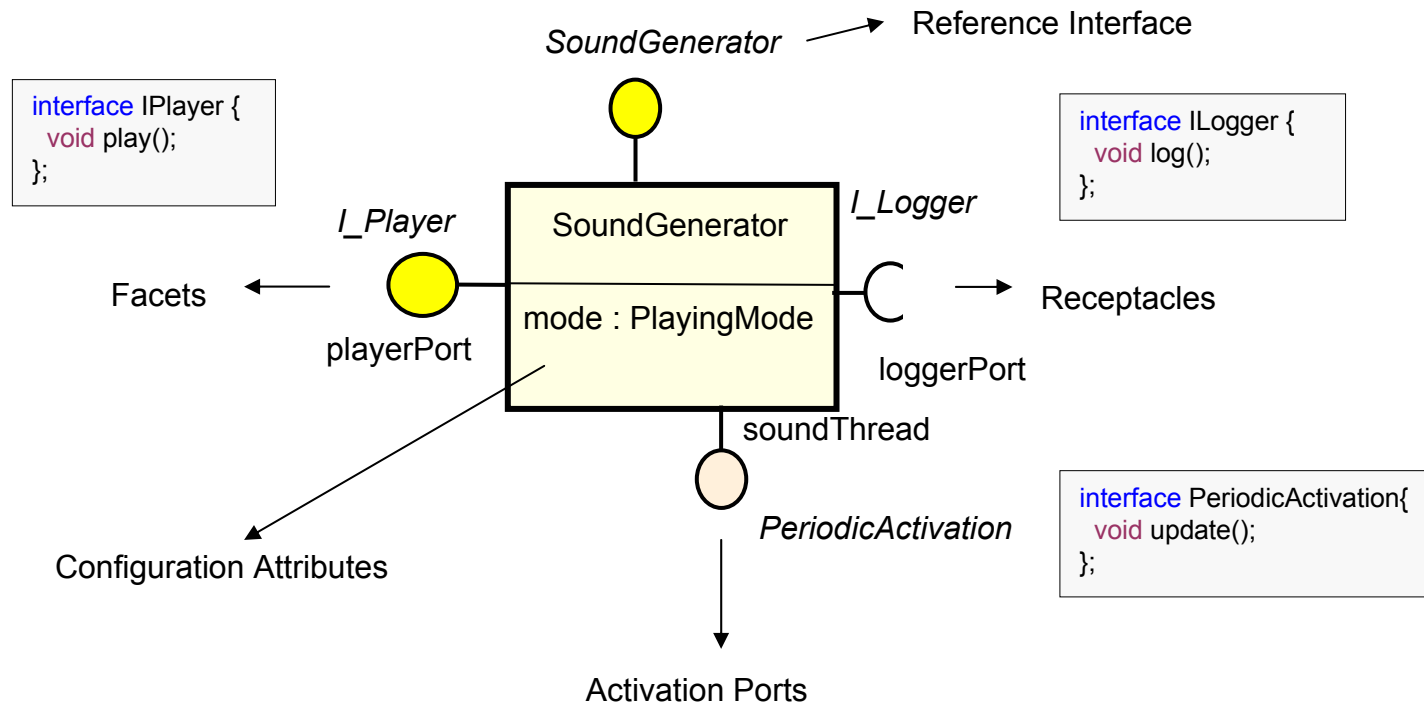
- **Ada 2005 is a significantly relevant option to develop an Ada native component-based technology for real-time systems**
 - ➔ Taking advantage of the Ada native support for real-time systems
 - ➔ The multiple inheritance provided by the usage of interfaces is a key aspect in a component-based technology

Motivation: RT Components Development Process



Motivation: RT Component Specification

- **A component is a software module with a business functionality defined by:**
 - The set of services that it offers (facets)
 - The set of services that it requires (receptacles)
 - The configuration attributes that it admits (attributes)
- **Ada-CCM extends the specification of a component to control the schedulability of applications:**
 - In our technology the business code of a component is formulated as passive code
 - The required threads are supplied and managed by the container
 - They are required by means of activation ports: PeriodicActivation or OneShotActivation



Topics in agenda

- **Elements in the current IDL-Ada mapping that can be enhanced**
- **The usage of Interfaces in Ada 2005**
- **Practical approach to do the mapping**
- **Action lines towards its standardization**
- **Conclusions and future work**

Summary of IDL to Ada

Table 1-1 Summary of IDL Constructs to Ada Constructs

IDL construct	Ada construct
Source file	Library package
Module	Package (Child Package if nested)
Interface	Package with Tagged Type (Child Package if nested)
Operation	Primitive Subprogram
Attribute	"Set_attribute" and "Get_attribute" subprograms
Inheritance: Single Multiple	Tagged Type Inheritance Tagged Type Inheritance for first parent; cover functions with explicit widening and narrowing for subsequent parents
Data types	Ada types
Exception	Exception and record type

Example of the current mapping

The following IDL specification:

```
// IDL - file barn.idl  
typedef long measure;  
interface Feed {  
    attribute measure weight;  
};  
interface Horse : Animal{  
    void trot(in short distance);  
};
```

Is mapped to these Ada interface packages:

```
with CORBA;  
package Barn_IDL_FILE is  
    type Measure is new CORBA.Long;  
end Barn_IDL_FILE;  
  
with CORBA; CORBA.Object;  
with Barn_IDL_FILE;  
package Feed is  
    type Ref is new CORBA.Object.Ref  
        with null record;  
    procedure Set_Weight  
        (Self : in Ref;  
         To : in Barn_IDL_FILE.Measure);  
    function Get_Weight  
        (Self : in Ref) return  
        Barn_IDL_FILE.Measure;  
end Feed;
```

Example of the current mapping (cont.)

The following IDL specification:

```
interface Animal {  
    enum State {SLEEPING, AWAKE};  
    boolean eat(inout Feed bag);  
    // returns true if animal is full  
    attribute State alertness;  
    readonly attribute Animal parent;  
};  
  
interface Horse : Animal{  
    void trot(in short distance);  
};
```

Is mapped to these Ada interface packages:

```
with CORBA.Object;  
with Feed;  
package Animal is  
    type Ref is new CORBA.Object.Ref with null  
    record;  
    type State is (SLEEPING, AWAKE);  
    procedure Eat (Self : in Ref;  
                  Bag : in out Feed.Ref;  
                  Returns : out Boolean);  
    -- returns true if animal is full  
    procedure Set_Alertness (Self : in Ref;  
                            To : in State);  
    function Get_Alertness (Self : in Ref) return  
    State;  
    function Get_Parent(Self : in Ref) return  
    Ref^CLASS;  
end Animal;  
  
with Animal;  
package Horse is  
    type Ref is new Animal.Ref with null record;  
    subtype State is Animal.State;  
    procedure Trot (Self : in Ref; Distance : in  
                  CORBA.Short);  
end Horse;
```

Example of the current mapping (cont.)

The following IDL specification:

```
interface Asset {  
...  
void op1();  
void op2();  
...  
};  
interface Vehicle {  
...  
void op3();  
void op4();  
...  
};  
interface Tank : Vehicle, Asset {  
...  
};
```

Is mapped to :

```
with CORBA;  
package Asset is  
  type Ref is new CORBA.Object.Ref with  
    null record;  
    procedure op1 (Self : Ref);  
    procedure op2 (Self : Ref);  
end Asset;  
  
with CORBA;  
package Vehicle is  
  type Ref is new CORBA.Object.Ref with  
    null record;  
    procedure op3 (Self : Ref);  
    procedure op4 (Self : Ref);  
end Vehicle;  
with CORBA;  
with Vehicle, Asset;  
package Tank is  
  type Ref is new Vehicle.Ref with null  
    record;  
    procedure op1 (Self : Ref);  
    procedure op2 (Self : Ref);  
end Tank;
```

Reasoning

- **The introduction of interfaces as a native concept in Ada enable the mapping of both formalisms to be much closer in style and notation.**
- **The mechanisms to implement single and multiple inheritance result now simpler and seamlessly to denote with the usage of interfaces.**
- **This makes much simpler not only the code generation from IDL specifications but also the reverse engineering processes and hence potentially the certification process of both: the generator and the code generated.**
- **A new native collection types library is now provided, so the overheads in the various implementation may be reduced.**

Ada Interfaces in IDL constructs

- **Interface:**

- ⇒ Interface

- **Inheritance:**

- ⇒ Single

- ✓ Tagged Type implementing an interface
 - ✓ Extending (composing) interfaces

- ⇒ Multiple

- ✓ Tagged Type implementing multiple interfaces
 - ✓ Interfaces extended with (composed of) multiple interfaces

Interface definition

Package P1 is

type Int1 is interface;

procedure Op1(X: Int1) is abstract;

procedure N(X: Int1) is null;

end P1;

type DT is new Int1 with record ... end record;

procedure Op1(NX: DT);

Simple Inheritance

- **Implementation of an interface**

```
type DT is new Int1 with record ... end record;  
procedure Op1(NX: DT);
```

- **Extending/composing Interfaces**

```
type Int3 is interface and Int1;
```

Multiple Inheritance

- **Implementing multiple interfaces**

**type DT2 is new Int1 and Int2 with
record ... end record;**

type DT3 is new T1 and Int1 and Int2 with ...

- **Extending/Composing multiple interfaces**

type Int4 is interface and Int1 and Int2 and Int3;

Practical experience

- **A set of templates have been defined to convert some IDL specifications into Ada2005 constructs.**
- **A number of procedures to handle the templates**

Component template

```
include [PackageName]
[...]
component [ComponentName] : [FatherComponent] supports [SupportedInterface]
{
    typedef [nativeType] [defined_Att_Type]
    [...]
    provides [InterfaceName] [PortName];
    [...]
    uses [InterfaceName] [ReceptacleName];
    uses multiple [InterfaceName] [ReceptacleName];
    [...]
    attribute [defined_Att_Type] [attributeName];
    [...]
}
home [HomeName] manages [ComponentName]
{
    factory [FactoryName]([ParametersFactory]);
};
```

```
<With>  
with @Package@;</With>  
<Use> use @Package@;</Use>
```

```
<PackageHead>  
package @DomainName@ is</PackageHead>
```

```
<EnumTypeHead>  
type @TypeName@ is (  
</EnumTypeHead>
```

```
<EnumValue>    @EnumValue@,  
</EnumValue>
```

```
<LastEnumValue>    @EnumValue@);  
</LastEnumValue>
```

```
<SimpleTypeHead>  
type @TypeName@ is new @IdType@;</SimpleTypeHead>
```

```
<With>  
with @Package@;</With>
```

```
<Use> use @Package@;</Use>
```

```
<InterfaceHead>  
package @DomainName@.@InterfaceName@ is
```

```
-- Interface definition  
type Face is interface;  
type Ref is access all Face'Class;</InterfaceHead>
```

```
<InterfaceWithInheritanceHead>  
package @DomainName@.@InterfaceName@ is
```

```
-- Interface definition  
type Face is interface and </InterfaceWithInheritanceHead>
```

```
<InterfaceWithInheritanceBase>@DomainName@.@BaseInterface@ and  
</InterfaceWithInheritanceBase>
```

```
<InterfaceWithInheritanceBaseLast>@DomainName@.@BaseInterface@;  
type Ref is access all Face'Class;</InterfaceWithInheritanceBaseLast
```

Sequence Types

- **Include the Ada 2005 containers library for using Vectors:**

```
<Head_WithSecuence>  
with Ada.Containers.Vectors;  
</Head_WithSecuence>
```

- **Declaration of Vectors:**

```
<SequenceTypeHead>  
package @SequenceName@_pkg is new  
  Ada.Containers.Vectors(Positive,@SequenceType@);  
  use @SequenceName@_pkg;  
type @SequenceName@ is  
  new @SequenceName@_pkg.Vector with null  
  record;</SequenceTypeHead>
```

A sub-set of basic common types is offered

```
with Ada.Strings.Unbounded;  
with Interfaces;
```

```
package Common_Types is
```

```
type Short      is new Interfaces.Integer_16;  
type Long       is new Interfaces.Integer_32;  
type Long_Long  is new  
  Interfaces.Integer_64;  
type Unsigned_Short is new  
  Interfaces.Unsigned_16;  
type Unsigned_Long is new  
  Interfaces.Unsigned_32;  
type Unsigned_Long_Long is new  
  Interfaces.Unsigned_64;  
type Float      is new  
  Interfaces.IEEE_Float_32;  
type Double     is new  
  Interfaces.IEEE_Float_64;  
type Long_Double is new  
  Interfaces.IEEE_Extended_Float;  
subtype Char     is Standard.Character;  
subtype Wchar    is  
  Standard.Wide_Character;  
type Octet       is new  
  Interfaces.Unsigned_8;  
subtype Boolean  is Standard.Boolean;
```

```
type String      is  
  new  
  Ada.Strings.Unbounded.Unbounded_String;  
function To_Standard_String  
  (Source : String)  
  return Standard.String;  
  
function To_String  
  (Source : Standard.String)  
  return String;  
  
type Any is private;  
  
function Any_To_String(val:Any) return String;  
  
function String_To_Any(val:String) return Any;  
function Any_To_Standard_String(val:Any) return  
  Standard.String;  
  
private  
  
type Any is new String;  
  
end Common_Types;
```

Action lines towards its standardization

- Identification of any other aspects of the current mapping that might be aligned to Ada2005 with the minimal impact on backwards compatibility
- Revision of usage of reference types and naming conventions

Table 4-1 Reference Type Names and Ancestor Types

IDL Construct	Name of Reference Type	Ancestor Type
abstract interface	Abstract_Ref	CORBA.AbstractBase.Ref
unconstrained interface	Ref	CORBA.Object.Ref
local interface	Local_Ref	
stateful valuetype	Value_Ref	CORBA.Value.Base
abstract valuetype	Abstract_Value_Ref	

- Revision of the impact of new native Containers library

Concluding remarks

- **A first IDL to Ada2005 converter has been built, it addresses the proposed mappings, though it is not complete.**
- **About now it is still an academic project, a publication that include these results has been made:**
[“An Ada 2005 Technology for Distributed and Real-Time Component-based Applications”. P. López, J. M. Drake, P. Pacheco, and J. Medina. In Proceedings of Ada-Europe 2008: International Conference on Reliable Software Technologies, May 2008.]
- **Still expect prospective industrial interest in the look for a future/possible RFC or its inclusion in the IDL standalone standardization process.**

Further directions

- **As a general criteria it seems interesting to observe the IDL-Java mapping to improve portability.**
- **A code update guide/tool/stub, from the current Ada95 elements generated by the current mapping to the Ada2005 one, would help to accept it.**
- **Promoting this as an activity to the members of the current IDL-Ada mapping RTF.**