Distributed Computing with Ada and CORBA using PolyORB

Frédéric Praca

Ada-France

FOSDEM'19, 2nd of February 2019, Brussels

About the author

Frédéric Praca holds a MSc in computer science. After working in the defence industry in a European aeronautics company, he now works for the energy industry developing information systems. Usually coding in Java/Python at work, he started coding in Ada in 2003 in his spare time. Now, he tries to advocate people to use Ada.

Goal of the presentation

The goal of this presentation is to show you a way to distribute computing thanks to Ada.

Table of Contents

According to Wikipedia

A distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to each other

According to Wikipedia

A distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to each other

Message passing is not defined and several technologies exist for this task.

Different technologies appeared in the last 40 years:

Different technologies appeared in the last 40 years:

- SOAP

Different technologies appeared in the last 40 years:

- SOAP
- REST

Different technologies appeared in the last 40 years:

- SOAP
- REST
- CORBA

Different technologies appeared in the last 40 years:

- SOAP
- REST
- CORBA
- DSA

Different technologies appeared in the last 40 years:

- SOAP
- REST
- CORBA
- DSA
- RMI

Different technologies appeared in the last 40 years:

- SOAP
- REST
- CORBA
- DSA
- RMI
- DCOM

Different technologies appeared in the last 40 years:

- SOAP
- REST
- CORBA
- DSA
- RMI
- DCOM

Among others. . .

Several candidates are possible with Ada

Several candidates are possible with Ada

- REST

Several candidates are possible with Ada

- REST
- SOAP

Several candidates are possible with Ada

- REST
- SOAP
- DSA

Several candidates are possible with Ada

- REST
- SOAP
- DSA
- CORBA

CORBA : Common Object Request Broker Architecture

CORBA : Common Object Request Broker Architecture

- Standard from the OMG (Object Management Group)

CORBA : Common Object Request Broker Architecture

- Standard from the OMG (Object Management Group)
- Released first time in 1991

CORBA : Common Object Request Broker Architecture

- Standard from the OMG (Object Management Group)
- Released first time in 1991
- Object oriented

CORBA : Common Object Request Broker Architecture

- Standard from the OMG (Object Management Group)
- Released first time in 1991
- Object oriented
- Using a definition language (IDL)

CORBA : Common Object Request Broker Architecture

- Standard from the OMG (Object Management Group)
- Released first time in 1991
- Object oriented
- Using a definition language (IDL)
- Language agnostic but standard mappings were defined

Table of Contents

“ CORBA and Ada are not very trendy but together, they do a great job ”

“ CORBA and Ada are not very trendy but together, they do a great job ”

F. Praca

" *CORBA and Ada are not very trendy but together, they do a great job* "
F. Praca 2nd of February at FOSDEM 2019, Brussels

" *CORBA and Ada are not very trendy but together, they do a great job* "
F. Praca 2nd of February at FOSDEM 2019, Brussels

Code is available on GitHub

https://github.com/FredPraca/distributed_cbsg Go

Suppose we have a program which is :

Suppose we have a program which is :

- very useful

Suppose we have a program which is :

- very useful
- easy to use and integrate

Suppose we have a program which is :

- very useful
- easy to use and integrate
- but not designed for remote access

Suppose we have a program which is :

- very useful
- easy to use and integrate
- but not designed for remote access
- neither for scalability

Suppose we have a program which is :

- very useful
- easy to use and integrate
- but not designed for remote access
- neither for scalability

Distribution is the solution to our problem

CBSG stands for *Corporate Bullshit Generator*

Aim of the project

Providing the user sentences built against a vast vocabulary and sentence constructions harvested during long boring meetings.

Thanks to Gauthier de Montmollin for this essential piece of software 100% written in Ada

CBSG through CGI ▸ GO!

First step : Define a simple IDL for CBSG

First step : Define a simple IDL for CBSG

Let's see the Ada spec first

```
generic
   Paragraph_Mark      : String;
   Paragraph_End_Mark  : String;
   Dialog_Mark         : String;
package Corporate_Bullshit is

   function Sentence return String;

   function Workshop return String;

   function Short_Workshop return String;

   function Financial_Report return String;

end Corporate_Bullshit;
```

Now our IDL will look like

Now our IDL will look like

```
module CorbaCBSG {

  struct timestamped_Sentence {
    long timestamp;
    string sentence;
  };

  interface CBSG {
    timestamped_Sentence
        createTimestampedSentence();
    string createSentence();
    string createWorkshop();
    string createShortWorkshop();
    string createFinancialReport();
  };
};
```

PolyORB

PolyORB is a polymorphic, reusable infrastructure for building object-oriented distributed systems.

PolyORB

PolyORB is a polymorphic, reusable infrastructure for building object-oriented distributed systems.

It provides several middlewares through application personalities

PolyORB

PolyORB is a polymorphic, reusable infrastructure for building object-oriented distributed systems.

It provides several middlewares through application personalities

- DSA
- CORBA
- MOMA

PolyORB

PolyORB is a polymorphic, reusable infrastructure for building object-oriented distributed systems.

It provides several middlewares through application personalities

- DSA
- CORBA
- MOMA

and protocol personalities

- GIOP
- SOAP
- SRP

PolyORB

PolyORB is a polymorphic, reusable infrastructure for building object-oriented distributed systems.

It provides several middlewares through application personalities

- DSA
- CORBA
- MOMA

and protocol personalities

- GIOP
- SOAP
- SRP

PolyORB is maintained by AdaCore and available on Github.

Building PolyORB for CORBA

```
fred@Tatooine:~/Dev/Ada/PolyORB$ support/reconfig
[snip]
fred@Tatooine:~/Dev/Ada/PolyORB$ ./configure \
  --prefix=/opt/gnat/ --with-proto-perso="giop" \
  --with-appli-perso="corba" \
  --with-corba-services="naming event ir notification
      time"
  --with-gprbuild=gprbuild
[snip]
fred@Tatooine:~/Dev/Ada/PolyORB$ make && make install
```

Now that we have PolyORB, prepare the development by generating Ada server code from IDL

```
fred@Tatooine:~/Dev/Ada/dcbsg$ iac -o Ada/server \
                  -ada -i cbsg.idl
```

So what do we get ?

- `corbacbsg_cbsg_hash.ad[sb]`: Utilities used by PolyORB internally

So what do we get ?

- `corbacbsg_cbsg_hash.ad[sb]`: Utilities used by PolyORB internally
- `corbacbsg-cbsg-skel.ad[sb]`: Skeleton which is the glue between ORB and implementation

So what do we get ?

- `corbacbsg_cbsg_hash.ad[sb]`: Utilities used by PolyORB internally
- `corbacbsg-cbsg-skel.ad[sb]`: Skeleton which is the glue between ORB and implementation
- `corbacbsg-cbsg-impl.ad[sb]`: The implementation


Skeleton
?

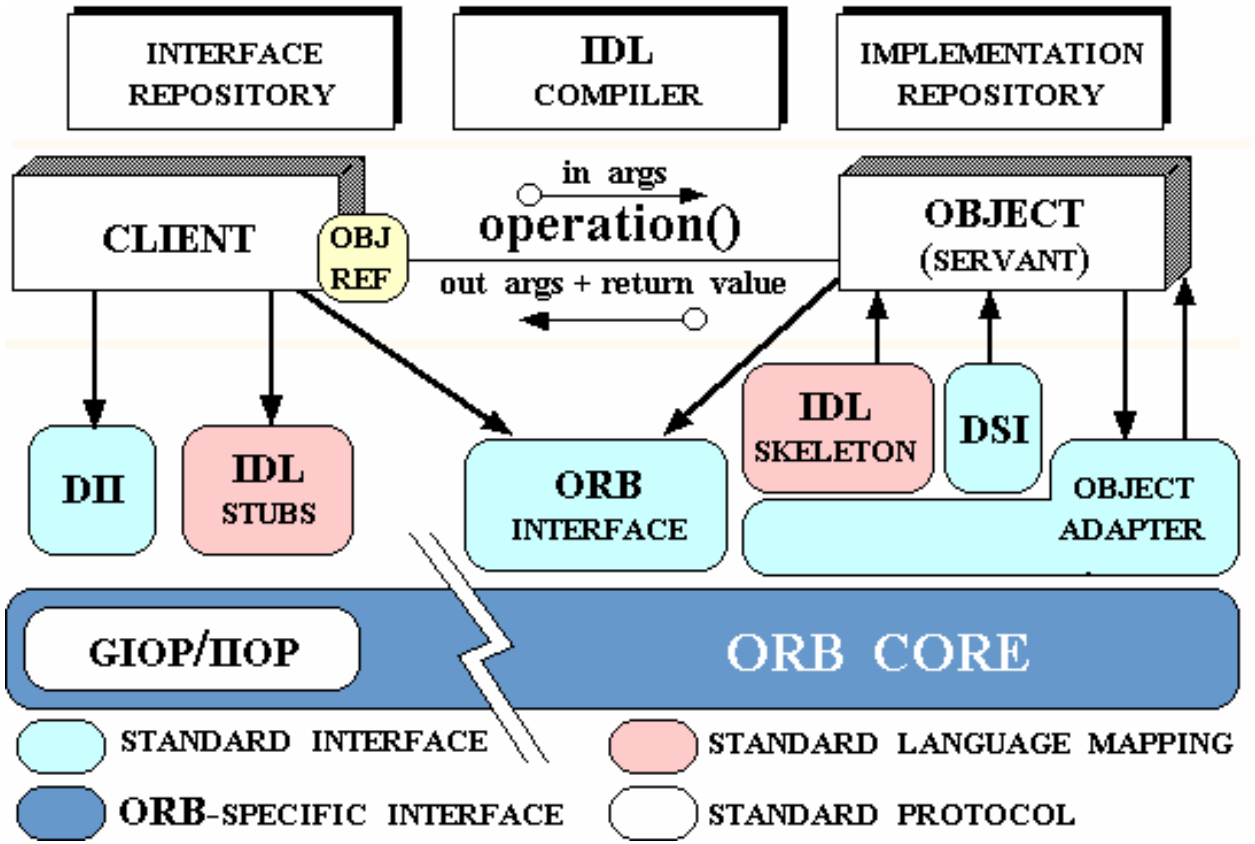

Figure courtesy of Douglas C. Schmidt, Vanderbilt University

The only part to change is the implementation.

The only part to change is the implementation.

```
package Simple_Generator is
new Corporate_Bullshit(Paragraph_Mark => "",
                       Paragraph_End_Mark => "",
                       Dialog_Mark => "");
-----------------------------
-- createSentence --
-----------------------------
function createSentence
  (Self : not null access Object)
  return CORBA.String
is
   Generated_Sentence : String :=
       Simple_Generator.Sentence;
begin
   return CORBA.To_CORBA_String(Generated_Sentence);
end createSentence;
```

We have the implementation code called servant but not the program using it.
So we need to implement a server program. Here are the steps:

- Initialization of the ORB
- Retrieval of the Root POA
- Activation of the Root POA
- Creation of the servant reference
- ORB main loop start

```
CORBA.ORB.Init (CORBA.ORB.To_CORBA_String ("ORB"),
    Argv);
declare
   Root_POA : PortableServer.POA.Local_Ref;

   Ref : CORBA.Object.Ref;

   Obj : constant CORBA.Impl.Object_Ptr
                  := new CorbaCBSG.CBSG.Impl.Object;

   begin

    Root_POA := PortableServer.POA.Helper.To_Local_Ref
      (CORBA.ORB.Resolve_Initial_References
      (CORBA.ORB.To_CORBA_String ("RootPOA")));
    PortableServer.POAManager.Activate
      (PortableServer.POA.Get_The_POAManager
          (Root_POA));
```

```
Ref := PortableServer.POA.Servant_To_Reference
       (Root_POA, PortableServer.Servant (Obj));

Put_Line ("'"
& CORBA.To_Standard_String
    (CORBA.Object.Object_To_String (Ref))
  & "'");

CORBA.ORB.Run;
```

The last line will output the *corbaloc* string which is an unique identifier for the object and can be used to reference the onject from another computer.

Let's start with an Ada client... For the moment :)

Let's start with an Ada client... For the moment :)
First as a reminder

Let's start with an Ada client... For the moment :)
First as a reminder

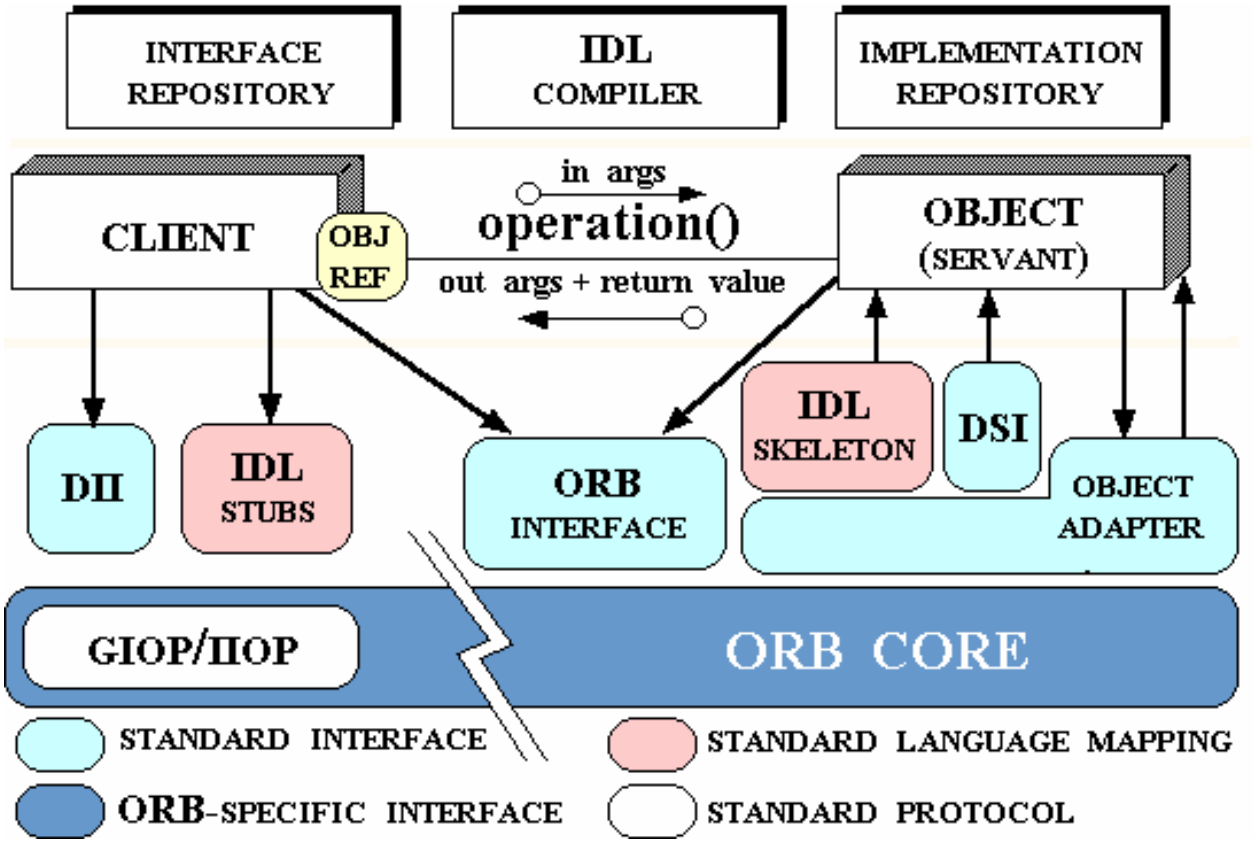

Figure courtesy of Douglas C. Schmidt, Vanderbilt University

The steps are quite simple:

The steps are quite simple:

- Generate the stub

The steps are quite simple:

- Generate the stub
- Init the ORB

The steps are quite simple:

- Generate the stub
- Init the ORB
- Get the object from the *corbaloc*

The steps are quite simple:

- Generate the stub
- Init the ORB
- Get the object from the *corbaloc*
- Test that the returned object is ok

The steps are quite simple:

- Generate the stub
- Init the ORB
- Get the object from the *corbaloc*
- Test that the returned object is ok
- Call the method

The steps are quite simple:

- Generate the stub
- Init the ORB
- Get the object from the *corbaloc*
- Test that the returned object is ok
- Call the method

And that's all :)

```
fred@Tatooine:~/Dev/Ada/dcbsg$ iac -o Ada/client \
-c -ada -i cbsg.idl
```

```
Rcvd_Bullshits : CORBA.String;
Bullshit_Generator : CorbaCBSG.CBSG.Ref;

begin
  CORBA.ORB.Initialize ("ORB");
  CORBA.ORB.String_To_Object
    (CORBA.To_CORBA_String (Ada.Command_Line.Argument
       (1)), Bullshit_Generator);

  if CorbaCBSG.CBSG.Is_Nil(Bullshit_Generator) then
     Put_Line ("main : cannot invoke on a nil
        reference");
     return;
  end if;

  Rcvd_Bullshits := CorbaCBSG.CBSG.createSentence
     (Bullshit_Generator);
```

then use `CORBA.To_Standard_String` to translate to Ada String

Our C++ client is using OmniORB4

Our C++ client is using OmniORB4
The steps are also simple:

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub
- Init the ORB

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub
- Init the ORB
- Get the object from the IOR

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub
- Init the ORB
- Get the object from the IOR
- Test that the returned object is ok

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub
- Init the ORB
- Get the object from the IOR
- Test that the returned object is ok
- Call the method

Our C++ client is using OmniORB4
The steps are also simple:

- Generate the stub
- Init the ORB
- Get the object from the IOR
- Test that the returned object is ok
- Call the method

And that's all :)

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
if( argc != 2 )
  {
    cerr << "usage: client <object reference>"
         << endl;
    return 1;
  }

// We create a CORBA object from the provided
    string
CORBA::Object_var obj =
    orb->string_to_object(argv[1]);

//Then cast it to CBSG reference
CorbaCBSG::CBSG_var cbsgRef =
    CorbaCBSG::CBSG::_narrow(obj);
```

```
// We then check the object is correct
if( CORBA::is_nil(cbsgRef) )
  {
    cerr << "Can't narrow reference to type CBSG
        (or it was nil)." << endl;
    return 1;
  }

// And go, call the method
cout << "The generator said : " <<
    cbsgRef->createSentence() << endl;

// Then we stop the ORB
orb->destroy();
```

Finally, the demo !!

Table of Contents

Why using Ada with CORBA ?

- Using our favorite language

Why using Ada with CORBA ?

- Using our favorite language
- Put the safety and readiness of Ada in heterogen environment

Why using Ada with CORBA ?

- Using our favorite language
- Put the safety and readiness of Ada in heterogen environment
- Using PolyORB

Further reading: Several useful links on CORBA

- OMG, CORBA Standard
- Douglas C. Schmidt, Distributed Object Computing with CORBA Middleware
- Ciaran McHale, CORBA explained simply

And PolyORB

- Adacore, PolyORB GitHub repository
- Adacore, PolyORB User's Guide