Seminar demonstration files

Overlays (II)

Denis Girou
June 2002

1 – Introduction

- If the talk is related to computing science, we must often show the contents of some programs. The ‘listings’ package is very useful and powerful for such tasks, used alone or both with the ‘fancyvrb’ one as we do here.
- This is also useful to be able to use **overlays** to emphasize some lines of the codes. This is possible both with ‘fancyvrb’ and ‘listings’[a], using their escape mechanisms to execute some commands put inside verbatim material.
- We also add a macro to be able to put annotations on a text previously shown. This is specially useful to comment interactively things like equations and codes, putting them also in overlays. We give examples for this two cases.

[a] Thanks to Carsten HEINZ to have added in his package a special mechanism to interact with the overlay feature of Seminar.

2 – Equations with (cumulative) annotations

A formula for Π from Leonhard Euler

= 2.44949

$$\Pi = \sqrt{6} \times \sqrt{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots}$$

1 $\Longrightarrow$ 2.44949

1.25 $\Longrightarrow$ 2.73861

1.36111 $\Longrightarrow$ 2.85774

1.42361 $\Longrightarrow$ 2.92261

$$= \left(6 \sum_{n=1}^{\infty} \frac{1}{n^2}\right)^{\frac{1}{2}}$$

Demonstration of overlays (ii)

3 – Listing with (progressive) annotations

- From a manual to introduce to parallel programming with the MPI library
- First with overlays but without annotations, just using the features of the 'fancyvrb' package

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

  print *,'I am process ',rank,' among ',nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```


Demonstration of overlays (ii)

- Then the same code, but using both the features of the 'fancyvrb' and 'listings' packages, with an automatic *pretty printing* of the code, after definition of a new language to emphasize the MPI-1 constants and subroutines (see the file `lstlang0.sty`)
- We could also use the 'listings' package alone

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

  print *,'I am process ',rank,' among ',nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```


Demonstration of overlays (ii)

- And now always the same code, but adding external annotations, using PSTricks nodes. This time, all annotations are shown together, without using overlays.

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

  print *,'I am process ',rank,' among ',nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```

Initialization of MPI environment

***Number** of processes for the current execution*

***Rank** of the process among all of them*

Exit of MPI environment

Demonstration of overlays (ii)

- Then finally the same code again, with the same annotations, but shown in a progressive way, using overlays

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

  print *,'I am process ',rank,' among ',nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```

Initialization of MPI environment

***Number** of processes for the current execution*

***Rank** of the process among all of them*

Exit of MPI environment

4 – (Cumulative) listing with (cumulative) annotations (I)

```
program WhoAmI
  implicit none
  include 'mpif.h'
  integer :: nb_procs,rank,code

  call MPI_INIT(code)

  call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code)
  call MPI_COMM_RANK(MPI_COMM_WORLD,rank,code)

  print *,'I am process ',rank,' among ',nb_procs

  call MPI_FINALIZE(code)
end program WhoAmI
```

Initialization of MPI environment

***Number** of processes for the current execution*

***Rank** of the process among all of them*

Exit of MPI environment

5 – Listing with (cumulative) annotations (II)

☞ From a manual to introduce to distributed programming with CORBA

```
#include <iostream.h>
#include <fstream.h>

#include <OB/CORBA.h>
#include <export_skel.h>

class ClassMatrix : virtual public POA_Exporte {

  private :
   TypeMatrix A;

  public :
   ClassMatrix(double init);
   ~ClassMatrix();

   virtual void MultiplyVector(CORBA::Double    alpha,
                               TypeVector_slice *vector)
     throw(CORBA::SystemException);
};
```

File of CORBA required headers

*File of headers relative to the **skeleton** generated from the IDL interface by the IDL compiler*

The class ClassMatrix must now be known inside the CORBA POA

Constructor

Destructor

Definition of a service to multiply a matrix by a scalar and a vector, with a management of the exceptions done by CORBA


```
// Implementation of the methods

ClassMatrix::ClassMatrix(double cste) {
  long long i, j;

  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      A[i][j] = 0.0;}}
  for (i = 0; i < N; i++) {
      A[i][i] = cste;}
}

ClassMatrix::~ClassMatrix() {
  cout << "Destruction of the object" << endl;
}

void ClassMatrix::MultiplyVector(CORBA::Double    alpha,
                                 TypeVector_slice *vector)
  throw(CORBA::SystemException) {

  long long i, j;
  TypeVector tmp;

  for (i = 0; i < N; i++) {
    tmp[i] = 0.0;
    for (j = 0; j < N; j++) {
      tmp[i] = tmp[i] + alpha * A[i][j] * vector[j];
    }
  }
  for (i = 0; i < N; i++) vector[i] = tmp[i];
}
```

*Implementation of the **constructor**: initialization of the matrix to the identity matrix*

*Implementation of the **destructor**: generally memory deallocation*

Service to compute the product of a matrix (multiplied by a constant) with a vector, with a management of the exceptions done by CORBA


```
// Main program

int main(int argc, char* argv[])
{
  // Initialization of the CORBA ORB and POA
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
  CORBA::Object_var poaObj = orb -> resolve_initial_references("RootPOA");
  PortableServer::POA_var RootPOA = PortableServer::POA::_narrow(poaObj);

  ClassMatrix Matrix((double) (1.0));

  // Writing of the "universal pointer" IOR in the file "reference"
  CORBA::String_var str = orb -> object_to_string( Matrix._this() );
  ofstream out("reference");
  out << str << endl;
  out.close();

  RootPOA -> the_POAManager() -> activate();
  orb -> run();

  orb -> destroy();
}
```

Declaration and initialization of the ORB

Declaration and initialization of the POA

Creation of an object Matrix

Local characters string, used to store the generated reference

*Writing of this **server reference** in the file reference*

Activation of the ORB (which will "listen")

Destruction of the ORB (it will never occur here)