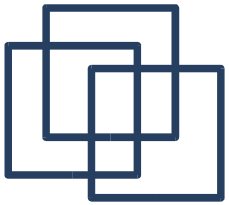
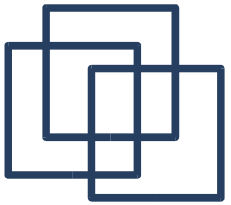


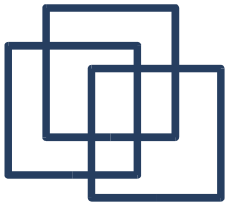
CORBA/ORBit2

Emmanuel Fleury
B1-201
fleury@cs.aau.dk

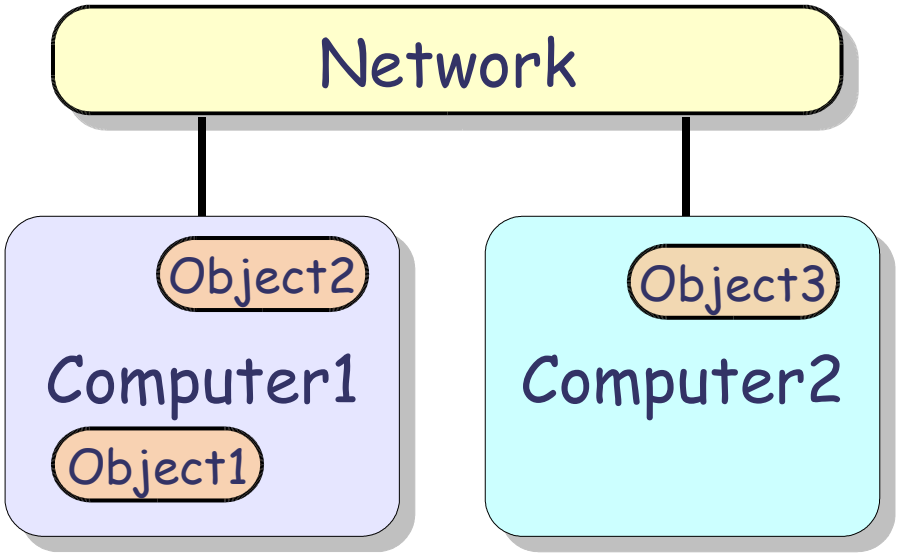


Summary

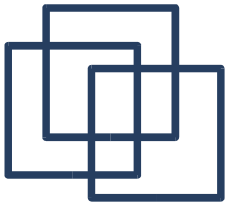
- Basics of CORBA
- Interface Definition Language
- The ORBit2 Project
- Developing for ORBit2
- Examples



Generic Problem



- Object Oriented Programs
- Objects are spread over the network
- Objects might be coded in different languages
- Objects connect to each other via an abstract interface



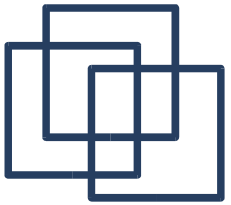
Basic Idea of CORBA

CORBA =

Common Object Request Broker Architecture

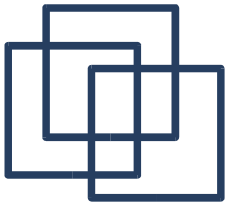
CORBA is an Open Distributed Object Infrastructure developed by the Object Management Group (OMG) which provides the developer with services as:

- Object Registration
- Object Location
- Object (Un)Marshalling
- Object Activation
- ...



Object Management Group

- Founded in 1989
- Aim at setting standards in Object-Oriented Programming
- Released CORBA 1.0 in 1991
- Released CORBA 3.0 in 2003 (last specs)
- In charge of UML, MOF and XMI
- Website: http://www.omg.org



Distributed Object

A Distributed Object is given by:

- An Identity

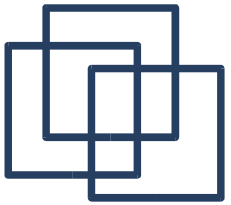
  IOR (Interoperable Object Reference)

- An Interface

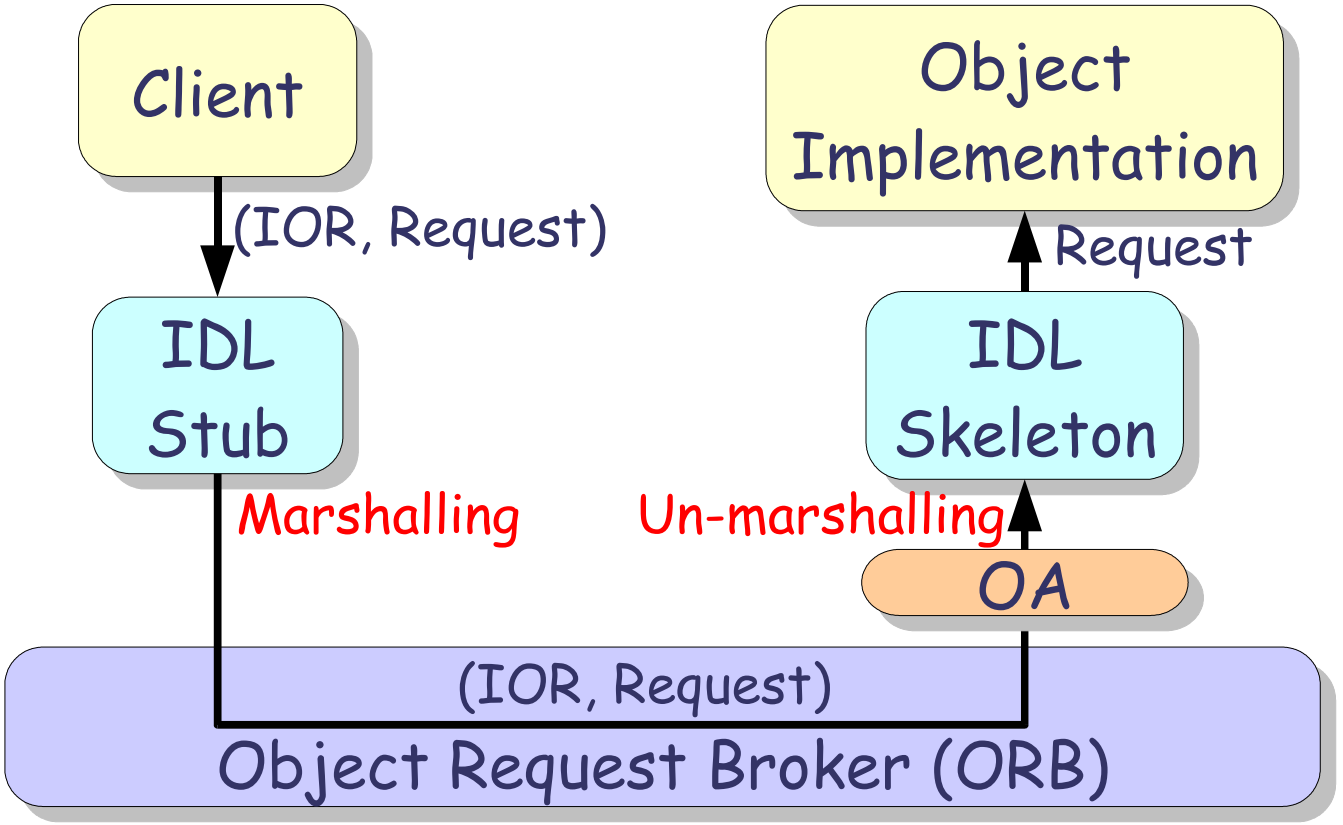
  Defined in IDL (Interface Description Language)

- An Implementation

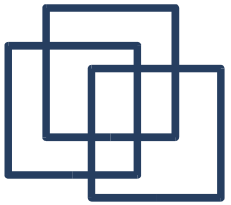
  Coded in C, C++, Python, Java, ...



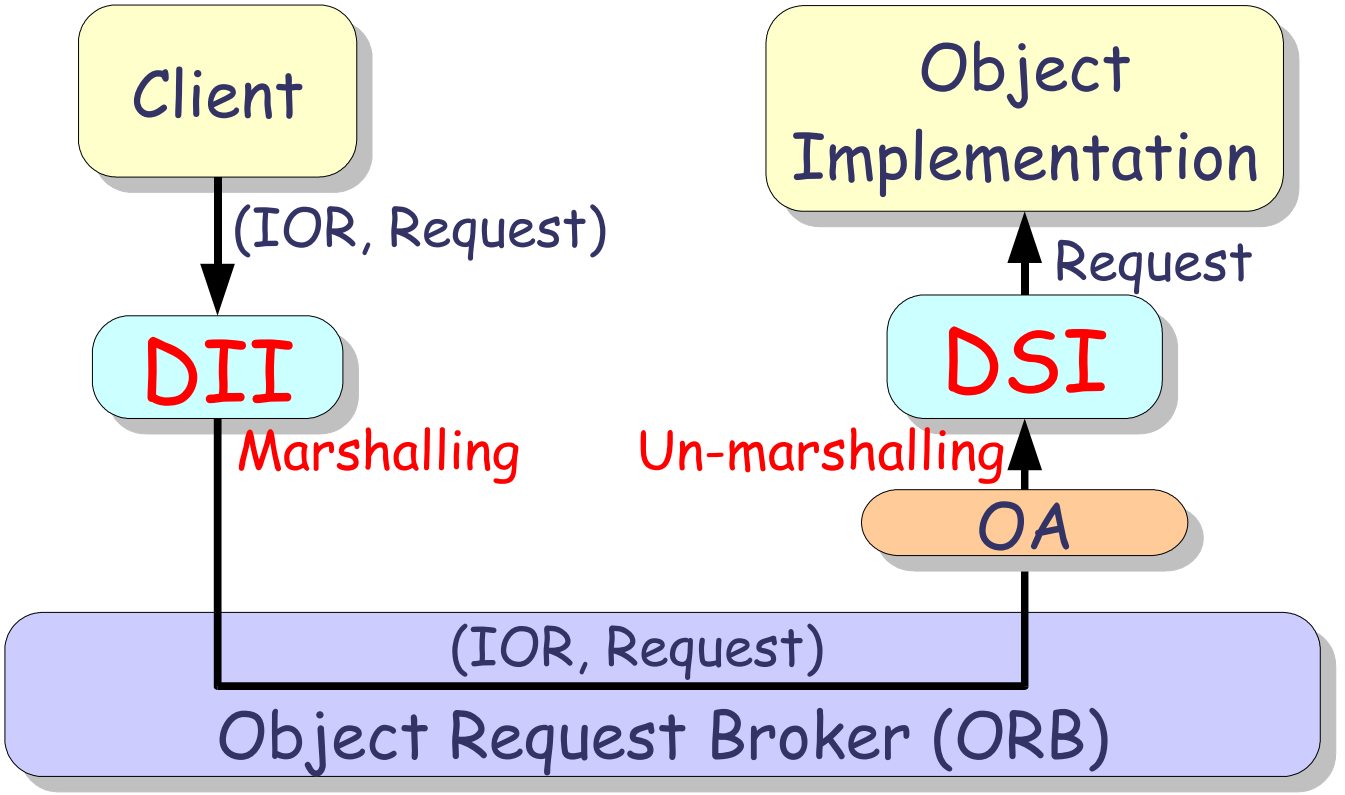
CORBA Architecture



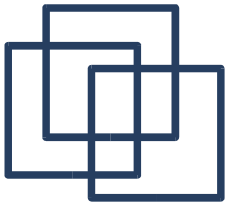
1. A Client call a method from an object IOR from the IDL Stub
2. ORB get the request from IDL Stub and forward it to IDL Skeleton through the Object Adapter (OA).
3. Object Implementation get the request from IDL Skeleton



CORBA Architecture (Alt)



1. A Client call a method from an object IOR directly from the ORB through the Dynamic Invocation Interface
2. The ORB get the request from the Client and forward it to the Object Implementation through the Object Adapter and the Dynamic Skeleton Interface



Interfaces

- Stub:

  Mapping between the language of implementation of the client to the ORB core

- Skeleton:

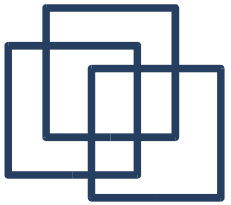
  Mapping between the language of implementation of the ORB core and the Object implementation

- Dynamic Invocation Interface (DII):

  Alternative to Stubs

- Dynamic Skeleton Interface (DSI):

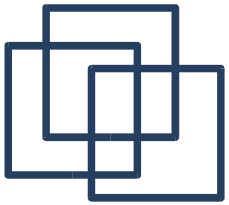
  Alternative to Skeletons



Object Request Broker

The ORB is a distributed service which implements:

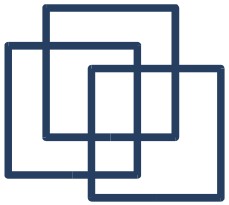
- Location transparency
  (Stub/Skel, DII/DSI)
- Language independence
  (Languages Binding)
- Interoperability
  (Generic Inter-ORB Protocol (GIOP))
  (Internet Inter-ORB Protocol (IIOP))



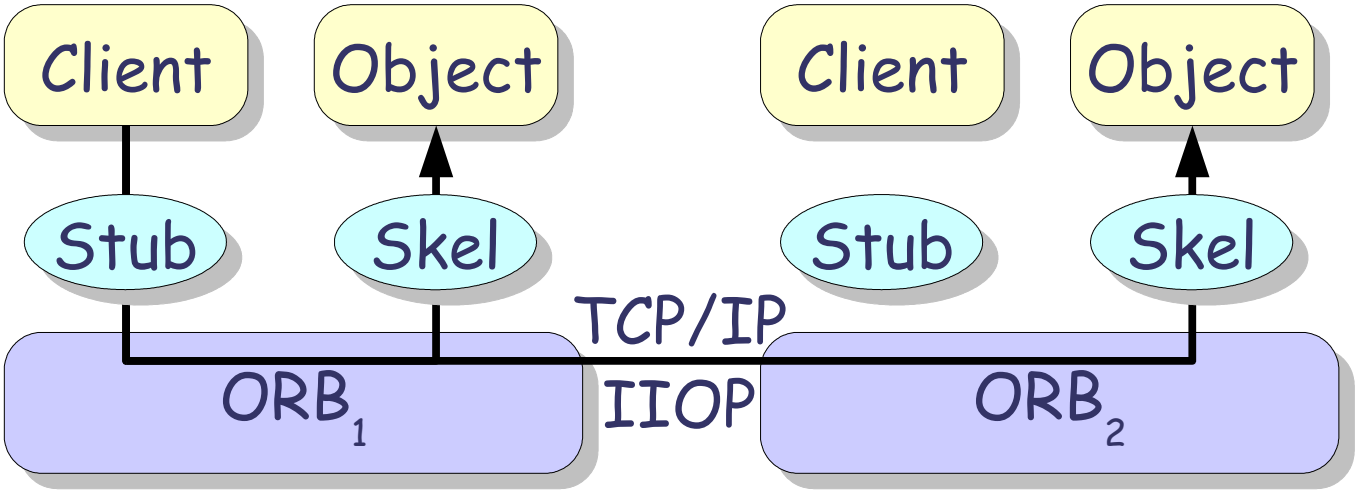
Object Adapters

The OA assist the ORB with delivering requests to the object, it can be running in several modes:

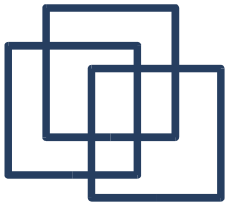
- Shared Server
  Multiple clients share the same object server.
- Unshared Server
  Each client has his own object server.
- Server-per-Method
  Each client request create a new object server.
- Persistent Server
  Server is started by another entity than the OA.



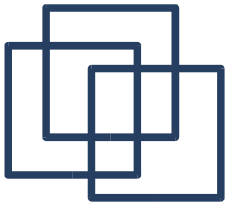
Object Request Broker



- In CORBA 2.0 specifications, the OMG added interoperability
- Different ORB implementations should be able to interoperate through the Internet Inter-ORB Protocol (IIOP)
- This interoperability should be working over any TCP/IP network (including Internet)



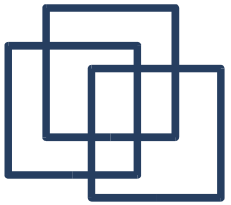
Interface Language Definition



Interface Language Definition

This language define the class hierarchy, the attributes and the signatures of the methods inside a module.

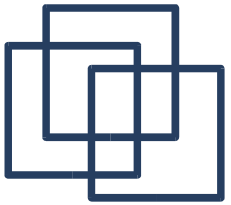
- Basic Types
  - boolean, char, short, strings, ...
  - float, double, ...
- Constructed Types
  - enum, struct, union, ...
  - array, ...



Basic Types (C)

- void
- boolean (TRUE, FALSE)
- octet/char
- wchar (wide char)
- string
- wstring (wide string)
- any (assumed safe until runtime)

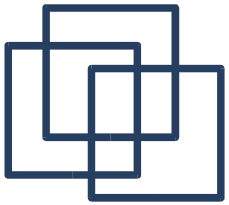
- short (-2^{15} ... $2^{15}-1$)
- long (-2^{31} ... $2^{31}-1$)
- long long (-2^{63} ... $2^{63}-1$)
- unsigned short (0 ... $2^{16}-1$)
- unsigned long (0 ... $2^{32}-1$)
- unsigned long long (0 ... $2^{64}-1$)
- float (IEEE specs)
- double (IEEE specs)
- long double (IEEE specs)



Constructed Types (C)

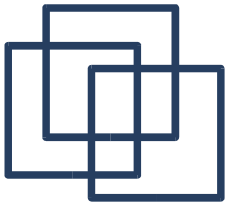
- enum
- struct
- union
- array[size]
- typedef

As in C Language



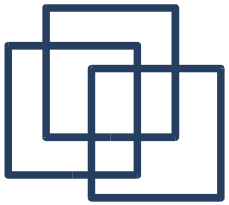
Basic Types (C/C++)

IDL	C	C++
short	CORBA_short	CORBA::Short
long	CORBA_long	CORBA::Long
long long	CORBA_longlong	CORBA::LongLong
unsigned short	CORBA_ushort	CORBA::UShort
unsigned long	CORBA_ulong	CORBA::ULong
unsigned long long	CORBA_ulonglong	CORBA::ULongLong
float	CORBA_float	CORBA::Float
double	CORBA_double	CORBA::Double
long double	CORBA_longdouble	CORBA::LongDouble



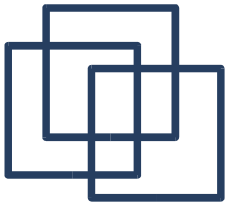
Basic Types (C/C++)

IDL	C	C++
char	CORBA_char	CORBA::Char
wchar	CORBA_wchar	CORBA::WChar
string	CORBA_string	CORBA::String
wstring	CORBA_wstring	CORBA::WString
boolean	CORBA_boolean	CORBA::Boolean
octet	CORBA_octet	CORBA::Octet
any	CORBA_any	CORBA::Any



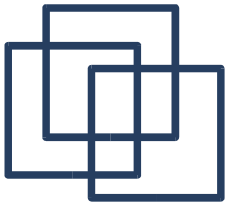
Parameter Declaration

- in
  (parameter passed from client to server)
- out
  (parameter passed from server to client)
- inout
  (parameter passed in both directions)



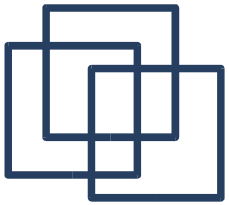
Interface Declaration

```
interface foo {
    // Private types
    enum material_t {rubber, glass, metal};
    struct position_t { float x, y;};

    // Attributes
    attribute float radius;
    attribute material_t material;

    // Methods
    float get_radius();
    void set_radius(in float r);
};
```



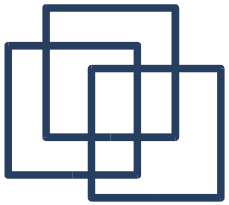
Module Declaration

```
#include <orb.idl>

#ifndef _MYMODULE_IDL_
#define _MYMODULE_IDL_

module MyModule {
    typedef long MyLong;

    interface MyInterface; // Forward declaration
    interface MyInterface { }; // Empty declaration
};
#endif // _MYMODULE_IDL_
```



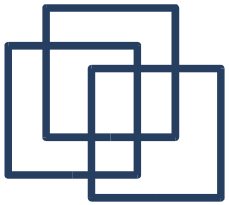
Inheritance

```
module Example {
    interface base;       // Forward declaration
    interface base { }; // Empty declaration

    interface inherited: base { };

    interface multipleinherited: base, inherited { };
};
```

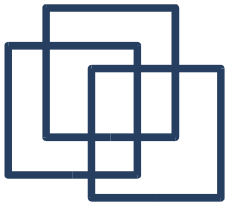


Inherited Types

```
interface A {
    typedef enum {one, two, three} numbers;
};

interface B: A {
    attributes A::numbers SerialNumber;
    attribute string      Name;
};
```

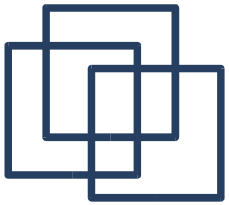


The ORBit2 Project



The ORBit2 Project

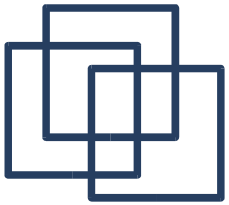
- Related to GNOME
  (GNU Network Object Model Environment)
- GNOME was intended to provide a full desktop environment over network
- Most of the ORB implementations where not supporting C language as a binding
- The GNOME team decided to code his own (project started in late 1999)
- The last version of ORBit is ORBit2 supporting CORBA 2.2 (see: http://orbit-resource.sourceforge.net/)



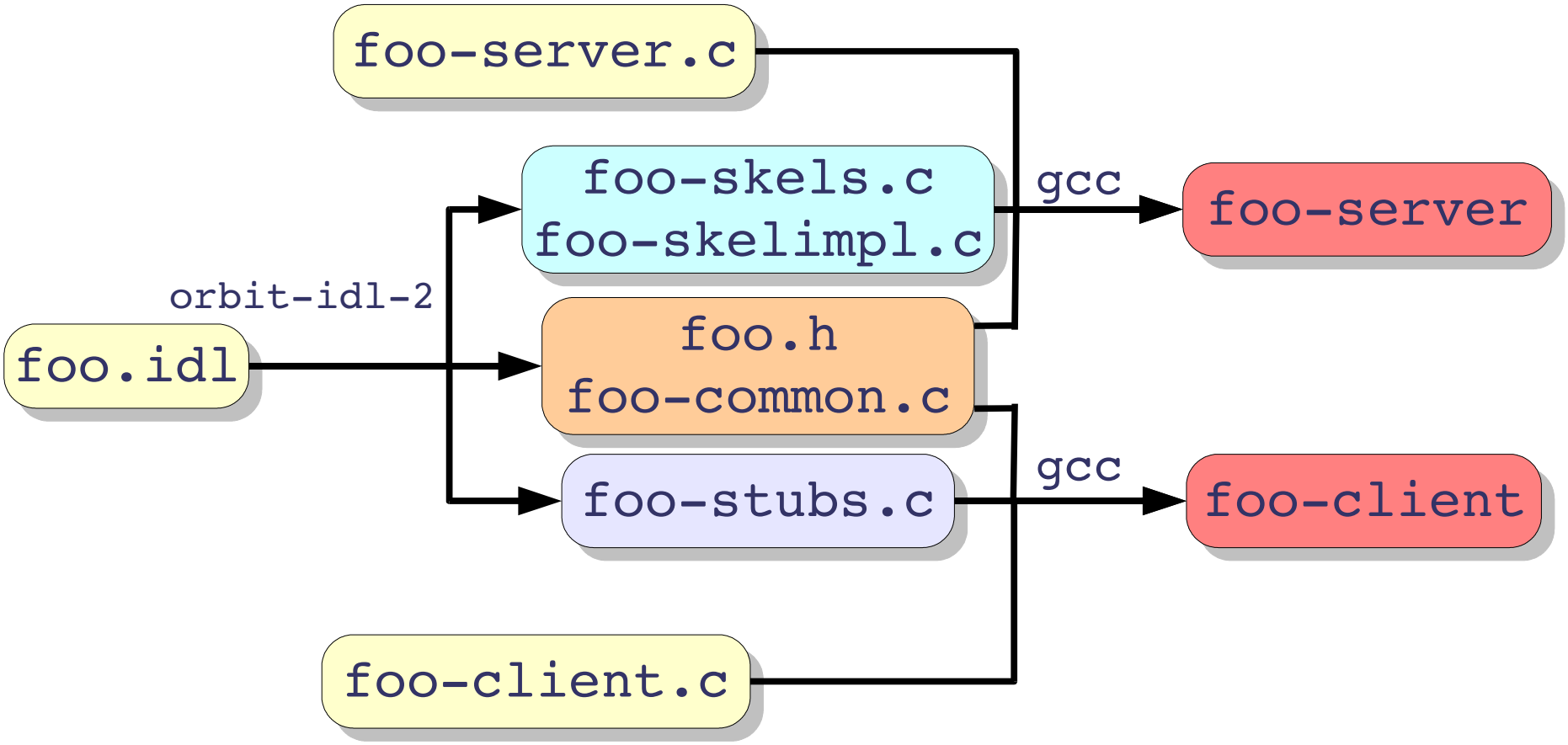
Developing for ORBit2

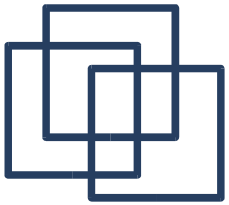
The development environment of ORBit2 provides:

- An IDL to C compiler (`orbit-idl-2`)
  To provide the implementation of the IDL
- An Event Server (`orbit-event-server`)
  To resolve the IOR when called
- An Interface Repository Daemon (`orbit-ird`)
  To store/cache the IOR of the interfaces
- A Name Server (`orbit-name-server`)
  To handle more meaningful references than IOR



IDL To Binaries

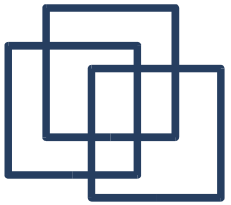




IDL To C Compiler

Given foo.idl, the IDL compiler provide:

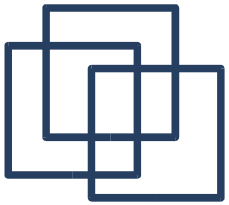
- foo.h: Header file of the "foo" component.
- foo-common.c: Common functions for client and server.
- foo-stub.c: Functions needed by the client to call the stub.
- foo-skel.c: Functions needed by the server to get requests from the skeleton.
- foo-skelimpl.c: Template for user code



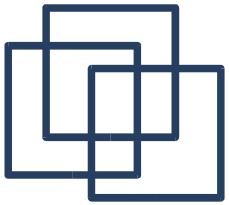
Client/Server Files

It is up to the developer to code:

- foo-client.c (Client side code)
  - Initialisation of the ORB
  - Getting the object from the stub
  - Using the object once fetched
- foo-server.c (Server side code)
  - Initialisation of the server
  - Waiting for the request
  - Providing the service

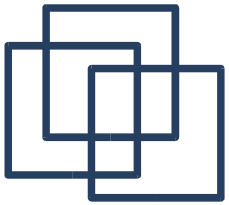


The Echo Component (IDL)

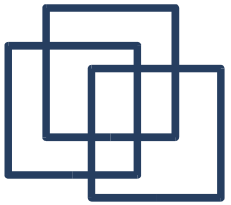


The Echo IDL

```
// The Echo component
//
// All this does is pass a string
// from the client to the server.

module ModuleEcho {

   interface Echo {
       void echoString(in string input);
   };
};
```

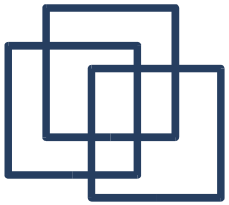


The Echo Component (C)



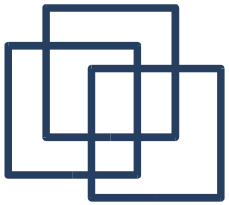
The Echo Client (C)

- Core functions:
  - `client_init`
  - `client_run`
  - `client_cleanup`
- Helper functions:
  - `abort_if_exception`
  - `import_object_from_file`
  - `client_shutdown`



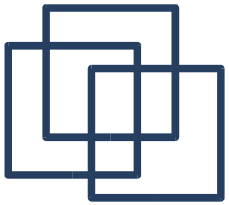
client_init

```
static void client_init (int *argc_ptr,
                         char *argv[],
                         CORBA_ORB *orb,
                         CORBA_Environment *ev) {

  /* init signal handling */
  signal (SIGINT, client_shutdown);
  signal (SIGTERM, client_shutdown);

  /* create Object Request Broker (ORB) */
  (*orb) =
      CORBA_ORB_init (argc_ptr, argv,
                      "orbit-local-orb", ev);
  if (ev->_major != CORBA_NO_EXCEPTION)
    return;
}
```



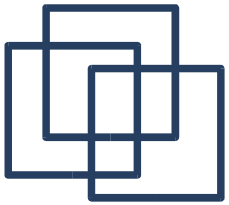
client_run

```
static void client_run (ModuleEcho_Echo echo_service,
                        CORBA_Environment *ev) {
  char msg[1024 + 1];

  printf ("Type messages to the server, a dot terminate input\n");

  while (fgets (msg, 1024, stdin))
    {
      if (msg[0] == '.' && msg[1] == '\n')
        break;

      /* chop the newline off */
      msg[strlen (msg) - 1] = '\0';

      /* using the echoString method in the Echo object this
       * is defined in the echo.h header, compiled from echo.idl */
      ModuleEcho_Echo_echoString (echo_service, msg, ev);
      if (ev->_major != CORBA_NO_EXCEPTION)
        return;
    }
}
```



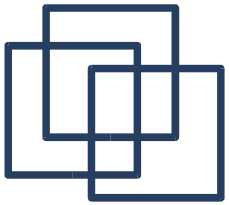
client_cleanup

```
static void client_cleanup (CORBA_ORB orb,
                            CORBA_Object service,
                            CORBA_Environment *ev) {

  /* releasing managed object */
  CORBA_Object_release (service, ev);
  if (ev->_major != CORBA_NO_EXCEPTION)
    return;

  /* tear down the ORB */
  if (orb != CORBA_OBJECT_NIL)
    {
      /* going to destroy orb.. */
      CORBA_ORB_destroy (orb, ev);
      if (ev->_major != CORBA_NO_EXCEPTION)
        return;
    }
}
```



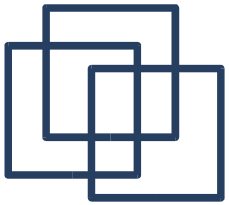
abort_if_exception

```
static void
abort_if_exception (CORBA_Environment *ev,
                    const char *mesg)
{
  if (ev->_major != CORBA_NO_EXCEPTION)
    {
      g_error ("%s %s", mesg,
               CORBA_exception_id (ev));

      CORBA_exception_free (ev);
      abort ();
    }
}
```



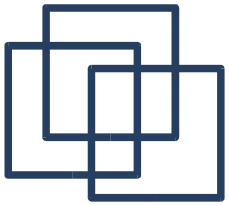
import_object_from_file

```
/* Import an IOR from a file and get the object from the ORB */
static CORBA_Object
import_object_from_file (CORBA_ORB orb,
                         char *filename, CORBA_Environment *ev) {
  CORBA_Object obj = NULL;
  FILE *file = NULL;
  gchar *objref = NULL;

  /* open the file */
  if ((file = fopen (filename, "r")) == NULL)
    g_error ("could not open %s\n", filename);

  /* read objref from the file */
  fscanf (file, "%as", &objref);

  /* get the object from the ORB */
  obj =
    (CORBA_Object) CORBA_ORB_string_to_object (global_orb, objref, ev);

  /* cleaning */
  free (objref);
  fclose (file);

  return obj;
}
```



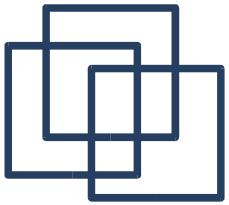
client_shutdown

```
/* Is called in case of process signals. it invokes
 * CORBA_ORB_shutdown() function, which will terminate
 * the processes main loop.
 */
static void client_shutdown (int sig)
{
  CORBA_Environment local_ev[1];
  CORBA_exception_init (local_ev);

  if (global_orb != CORBA_OBJECT_NIL)
    {
      CORBA_ORB_shutdown (global_orb, FALSE, local_ev);
      abort_if_exception (local_ev, "caught exception");
    }
}
```



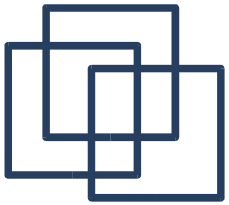
Main

```
int main (int argc, char *argv[]) {

  CORBA_char filename[] = "echo.ref";
  ModuleEcho_Echo echo_service = CORBA_OBJECT_NIL;
  CORBA_Environment ev[1];
  CORBA_exception_init (ev);

  /* Client initialization */
  client_init (&argc, argv, &global_orb, ev);
  abort_if_exception (ev, "init failed");
  printf ("Reading service reference from file \"%s\"\n", filename);

  /* Get the object from the ORB */
  echo_service =
    (ModuleEcho_Echo) import_object_from_file (global_orb, filename, ev);
  abort_if_exception (ev, "import service failed");

  /* Run the service */
  client_run (echo_service, ev);
  abort_if_exception (ev, "service not reachable");

  /* Clean-up after running */
  client_cleanup (global_orb, echo_service, ev);
  abort_if_exception (ev, "cleanup failed");

  return 0;
}
```



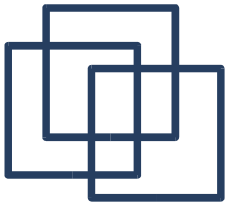
The Echo Server (C)

- Core functions:
  - `server_init`
  - `server_activate_service`
  - `server_run`
  - `server_cleanup`
- Helper functions:
  - `abort_if_exception`
  - `export_object_to_file`
  - `server_shutdown`



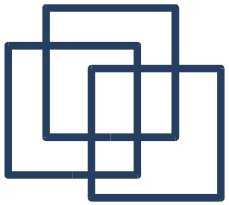
server_init (1)

```
/* Inits ORB @orb using @argv arguments for configuration. For each
 * ORBit options consumed from vector @argv the counter of @argc_ptr
 * will be decremented. Signal handler is set to call
 * echo_server_shutdown function in case of SIGINT and SIGTERM
 * signals. If error occures @ev points to exception object on return.
 */
static void server_init (int *argc_ptr,
                         char *argv[],
                         CORBA_ORB *orb,
                         PortableServer_POA *poa,
                         CORBA_Environment *ev)
{
  PortableServer_POAManager poa_manager = CORBA_OBJECT_NIL;
  CORBA_Environment local_ev[1];

  /* init of exceptions */
  CORBA_exception_init (local_ev);

  /* init signal handling */
  signal (SIGINT,  server_shutdown);
  signal (SIGTERM, server_shutdown);

 ... to be continued ...
```



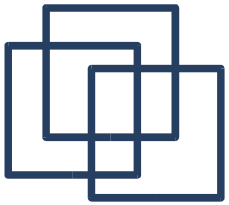
server_init (2)

```
  /* create Object Request Broker (ORB) */
  (*orb) = CORBA_ORB_init (argc_ptr, argv, "orbit-local-mt-orb", ev);
  if (ev->_major != CORBA_NO_EXCEPTION) goto failed_orb;

  (*poa) = (PortableServer_POA)
    CORBA_ORB_resolve_initial_references (*orb, "RootPOA", ev);
  if (ev->_major != CORBA_NO_EXCEPTION) goto failed_poa;

  poa_manager = PortableServer_POA__get_the_POAManager (*poa, ev);
  if (ev->_major != CORBA_NO_EXCEPTION) goto failed_poamanager;

  PortableServer_POAManager_activate (poa_manager, ev);
  if (ev->_major != CORBA_NO_EXCEPTION) goto failed_activation;

  /* everything went fine */
  CORBA_Object_release ((CORBA_Object) poa_manager, ev);
  return;

failed_activation:
failed_poamanager:
  CORBA_Object_release ((CORBA_Object) poa_manager, local_ev);
failed_poa:
  CORBA_ORB_destroy (*orb, local_ev);
failed_orb:
  return;
}
```



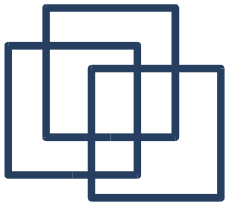
server_activate_service

```
/* Creates servant and registers in context of ORB @orb.
 * The ORB will delegate incoming requests to specific
 * servant object. @return object reference. If error
 * occures @ev points to exception object on return.
 */
static CORBA_Object
server_activate_service (CORBA_ORB orb,
                         PortableServer_POA poa,
                         CORBA_Environment *ev)
{
  ModuleEcho_Echo ref = CORBA_OBJECT_NIL;

  ref = impl_ModuleEcho_Echo__create (poa, ev);
  if (ev->_major != CORBA_NO_EXCEPTION)
    return CORBA_OBJECT_NIL;

  return ref;
}
```



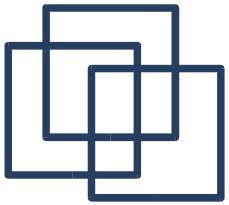
server_run

```
/* Entering main loop @orb handles incoming request and
 * delegates to servants. If error occures @ev points to
 * exception object on return.
 */
static void
server_run (CORBA_ORB orb, CORBA_Environment *ev)
{
  /* enter main loop until SIGINT or SIGTERM */

  CORBA_ORB_run (orb, ev);
  if (ev->_major != CORBA_NO_EXCEPTION)
    return;

  /* user pressed SIGINT or SIGTERM and in signal
   * handler CORBA_ORB_shutdown(.) has been called
   */
}
```



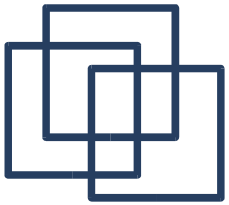
server_cleanup (1)

```
/* Releases @servant object and finally destroys @orb.
 * If error occures @ev points to exception object on
 * return.
 */
static void server_cleanup (CORBA_ORB orb,
                            PortableServer_POA poa,
                            CORBA_Object ref,
                            CORBA_Environment *ev)
{
  PortableServer_ObjectId *objid = NULL;

  objid =
    PortableServer_POA_reference_to_id (poa, ref, ev);
  if (ev->_major != CORBA_NO_EXCEPTION)
    return;

  ... to be continued ...
```



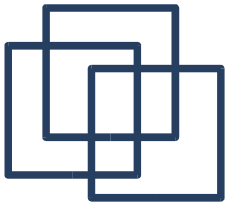
server_cleanup (2)

```
    /* Servant: deactivation - will invoke   __fini destructor */
    PortableServer_POA_deactivate_object (poa, objid, ev);
    if (ev->_major != CORBA_NO_EXCEPTION) return;

    PortableServer_POA_destroy (poa, TRUE, FALSE, ev);
    if (ev->_major != CORBA_NO_EXCEPTION) return;

    CORBA_free (objid);

    CORBA_Object_release ((CORBA_Object) poa, ev);
    if (ev->_major != CORBA_NO_EXCEPTION) return;

    CORBA_Object_release (ref, ev);
    if (ev->_major != CORBA_NO_EXCEPTION) return;

    /* ORB: tear down the ORB */
    if (orb != CORBA_OBJECT_NIL) {
        /* going to destroy orb.. */
        CORBA_ORB_destroy (orb, ev);
        if (ev->_major != CORBA_NO_EXCEPTION) return;
      }

    return; /* everything went fine */
}
```



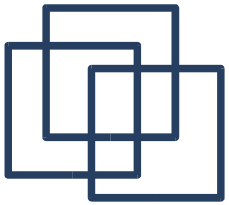
export_object_to_file

```
void export_object_to_file (CORBA_ORB orb, CORBA_Object servant,
                            char *filename, CORBA_Environment *ev)
{
  CORBA_char *objref = NULL;
  FILE *file = NULL;

  /* write objref to file */
  if ((file = fopen (filename, "w")) == NULL)
    g_error ("could not open %s\n", filename);

  /* get objref */
  objref = CORBA_ORB_object_to_string (orb, servant, ev);
  if (ev->_major != CORBA_NO_EXCEPTION) return;

  /* print ior to the file */
  fprintf (file, "%s\n", objref);
  fflush (file);

  /* cleaning */
  CORBA_free (objref);
  fclose (file);

  return;
}
```



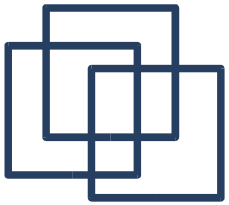
server_shutdown

```
/* Is called in case of process signals. it invokes
 * CORBA_ORB_shutdown() function, which will terminate
 * the processes main loop.
 */
static void server_shutdown (int sig)
{
  CORBA_Environment local_ev[1];

  CORBA_exception_init (local_ev);

  if (global_orb != CORBA_OBJECT_NIL)
    {
      CORBA_ORB_shutdown (global_orb, FALSE, local_ev);
      abort_if_exception (local_ev, "caught exception");
    }

  return;
}
```



main

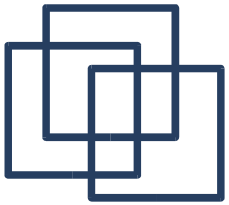
```
int main (int argc, char *argv[])
{
  CORBA_Object servant = CORBA_OBJECT_NIL;
  CORBA_char filename[] = "echo.ref";
  CORBA_Environment ev[1];

  CORBA_exception_init (ev);
  server_init (&argc, argv, &global_orb, &root_poa, ev);
  abort_if_exception (ev, "failed ORB init");

  servant = server_activate_service (global_orb, root_poa, ev);
  abort_if_exception (ev, "failed activating service");

  printf ("Writing service reference to: %s\n\n", filename);

  export_object_to_file (global_orb, servant, filename, ev);
  abort_if_exception (ev, "failed exporting IOR");

  server_run (global_orb, ev);
  abort_if_exception (ev, "failed entering main loop");

  server_cleanup (global_orb, root_poa, servant, ev);
  abort_if_exception (ev, "failed cleanup");

  return 0; }
```



Implementing the Skeleton

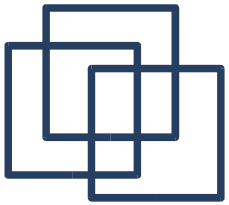
The IDL compiler (`orbit-idl-2`) can generate a template for the implementation of the skeleton when the option `--skeleton-impl` is given. This file is ready to use for implementing your skeleton. It contains:

- The Create/Destroy Templates:
  - `impl_ModuleEcho_Echo_create`
  - `impl_ModuleEcho_Echo_destroy`
- The Methods Templates:
  - `impl_ModuleEcho_Echo_echoString` (derived from the idl file)



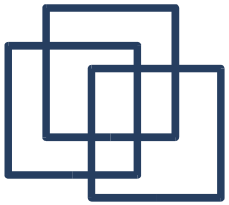
impl_ModuleEcho_Echo__create

```
static ModuleEcho_Echo
impl_ModuleEcho_Echo__create(PortableServer_POA poa, CORBA_Environment *ev)
{
   ModuleEcho_Echo retval;
   impl_POA_ModuleEcho_Echo *newservant;
   PortableServer_ObjectId *objid;

   newservant = g_new0(impl_POA_ModuleEcho_Echo, 1);
   newservant->servant.vepv = &impl_ModuleEcho_Echo_vepv;
   newservant->poa =
      (PortableServer_POA) CORBA_Object_duplicate((CORBA_Object) poa, ev);
   POA_ModuleEcho_Echo__init((PortableServer_Servant) newservant, ev);
   /* Before servant is going to be activated all
    * private attributes must be initialized.  */

   /* ------ init private attributes here ------ */
   /* ------ ---------- end ------------ ------ */

   objid = PortableServer_POA_activate_object(poa, newservant, ev);
   CORBA_free(objid);
   retval = PortableServer_POA_servant_to_reference(poa, newservant, ev);

   return retval;
}
```



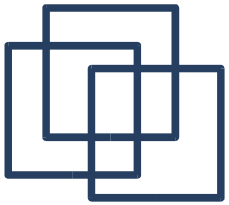
impl_ModuleEcho_Echo__destroy

```
static void
impl_ModuleEcho_Echo__destroy(impl_POA_ModuleEcho_Echo *servant,
                              CORBA_Environment *ev)
{
   CORBA_Object_release((CORBA_Object) servant->poa, ev);

   /* No further remote method calls are delegated to
    * servant and you may free your private attributes. */
   /* ------ free private attributes here ------ */
   /* ------ ---------- end ------------ ------ */

   POA_ModuleEcho_Echo__fini((PortableServer_Servant) servant, ev);

   g_free(servant);
}
```



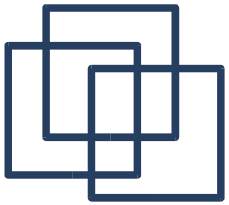
impl_ModuleEcho_Echo__echoString

```
static void
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
    /* ------   insert method code here   ------ */
    /* ------ ---------- end ------------ ------ */
}
```



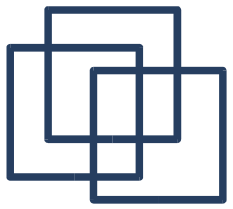
impl_ModuleEcho_Echo__echoString

```
static void
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
   /* ------   insert method code here   ------ */
   printf (">> %s\n", input);
   /* ------ ---------- end ------------ ------ */
}
```



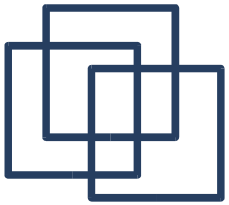
Compiling Client & Server

```
CC=gcc
ORBIT_IDL=/usr/bin/orbit-idl-2
CFLAGS=$(shell pkg-config ORBit-2.0 --cflags)
LDFLAGS=$(shell pkg-config ORBit-2.0 --libs)

TARGETS=echo-client echo-server
IDLOUT=echo-common.c echo-stubs.c echo-skels.c echo.h

all: idl echo-client echo-server

idl: $(IDLOUT)

echo-server.o: echo-server.c echo-skelimpl.c
echo-client : echo-client.o echo-stubs.o echo-common.o
echo-server : echo-server.o echo-skels.o echo-common.o

$(IDLOUT): echo.idl
        $(ORBIT_IDL) --skeleton-impl echo.idl

clean:
        rm -rf *.o *~ $(IDLOUT) *.ior *.ref

distclean: clean
        rm -rf echo-skelimpl.c echo-client echo-server
```

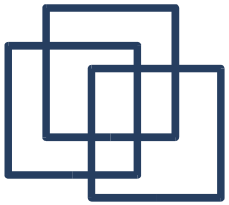


Running the Echo Component

Demo Time !!!

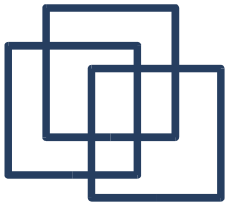


Getting an Output (C)



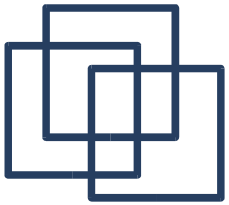
The Echo IDL

```
// The Echo component
//
// All this does is pass a string
// from the client to the server.

module ModuleEcho {

   interface Echo {
       string echoString(in string input);
   };
};
```



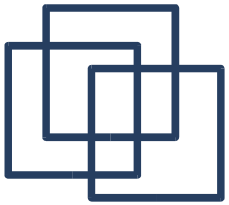
echo-client.c

```
static void client_run (ModuleEcho_Echo echo_service,
                        CORBA_Environment *ev) {
  char msg[1024 + 1];

  printf ("Type messages to the server, a dot terminate input\n");

  while (fgets (msg, 1024, stdin))
    {
      if (msg[0] == '.' && msg[1] == '\n')
        break;

      /* chop the newline off */
      msg[strlen (msg) - 1] = '\0';

      /* using the echoString method in the Echo object this
       * is defined in the echo.h header, compiled from echo.idl */

Changed [ printf("Returned>> %s\n",
                 ModuleEcho_Echo_echoString (echo_service, msg, ev));

      if (ev->_major != CORBA_NO_EXCEPTION)
        return;
    }
}
```



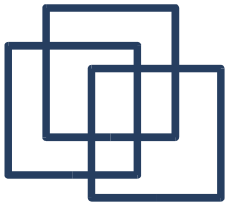
echo-skelimpl.c

```
static CORBA_string
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
    CORBA_string retval;

    /* ------   insert method code here   ------ */
    /* ------ ---------- end ----------- ------ */

    return retval;
}
```

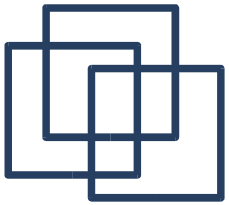


echo-skelimpl.c

```
static CORBA_string
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
    CORBA_string retval;

    /* ------   insert method code here   ------ */
    strcat(retval, "123");
    retval = input;
    /* ------ ---------- end ------------ ------ */

    return retval;
}
```

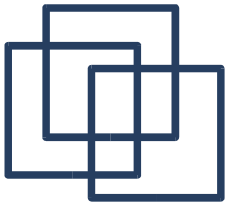


Attributes (C)



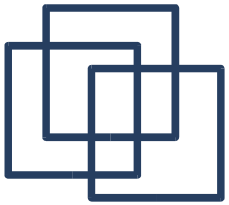
The Echo IDL

```
module ModuleEcho {
   interface Echo {
      readonly attribute string last_msg;

      string echoString(in string input);
      string recallLast();
   };
};
```



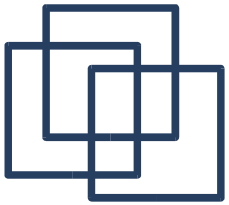
echo-skelimpl.c

```
typedef struct
{
    POA_ModuleEcho_Echo servant;
    PortableServer_POA poa;
    CORBA_string attr_last_msg;

    /* ------ add private attributes here ------ */
    /* ------ ---------- end ---------- ------ */
} impl_POA_ModuleEcho_Echo;
```



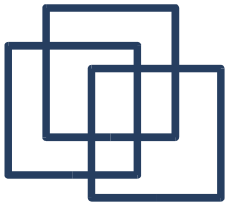
echo-skelimpl.c

```
static CORBA_string
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
   CORBA_string retval;

   /* ------   insert method code here   ------ */
   /* ------ ---------- end ------------ ------ */

   return retval;
}

static CORBA_string
impl_ModuleEcho_Echo_recallLast(impl_POA_ModuleEcho_Echo *servant,
                                CORBA_Environment *ev)
{
   CORBA_string retval;

   /* ------   insert method code here   ------ */
   /* ------ ---------- end ------------ ------ */

   return retval;
}
```



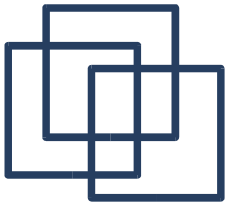
echo-skelimpl.c

```
static CORBA_string
impl_ModuleEcho_Echo_echoString(impl_POA_ModuleEcho_Echo *servant,
                                const CORBA_char *input,
                                CORBA_Environment *ev)
{
    CORBA_string retval;

    /* ------   insert method code here   ------ */
    attr_last_msg = input;

    retval = input;
    strcat(retval, "123");
    /* ------ ---------- end ------------ ------ */

    return retval;
}
```

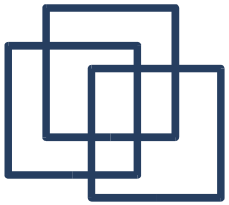


echo-skelimpl.c

```
static CORBA_string
impl_ModuleEcho_Echo_recallLast(impl_POA_ModuleEcho_Echo *servant,
                                CORBA_Environment *ev)
{
    CORBA_string retval;

    /* ------   insert method code here   ------ */
    retval = attr_last_msg;
    /* ------ ---------- end ------------ ------ */

    return retval;
}
```

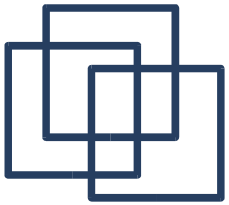


Nameserver (C)

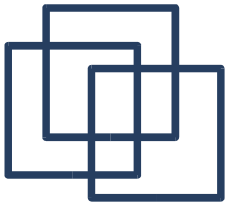


The Echo IDL

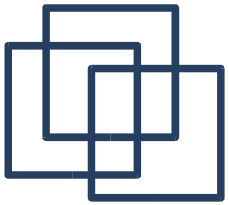
```
module ModuleEcho {

   module NameResolve {

      interface Echo {
        readonly attribute string last_msg;

        string echoString(in string input);
        string recallLast();
      };
   };
};
```



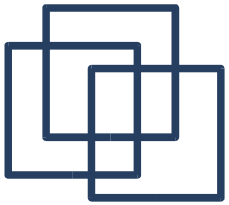
Changing the POA Strategy (C)



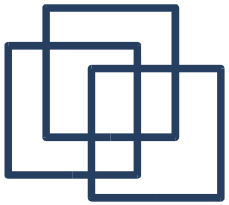
The Echo Component (C++)



Memory Management in CORBA



Questions ?



Next Week