Fnorb: a CORBA 2.0 ORB for Python

Thomas Huining Feng

MSDL, McGill

`http://msdl.cs.mcgill.ca/people/tfeng/`

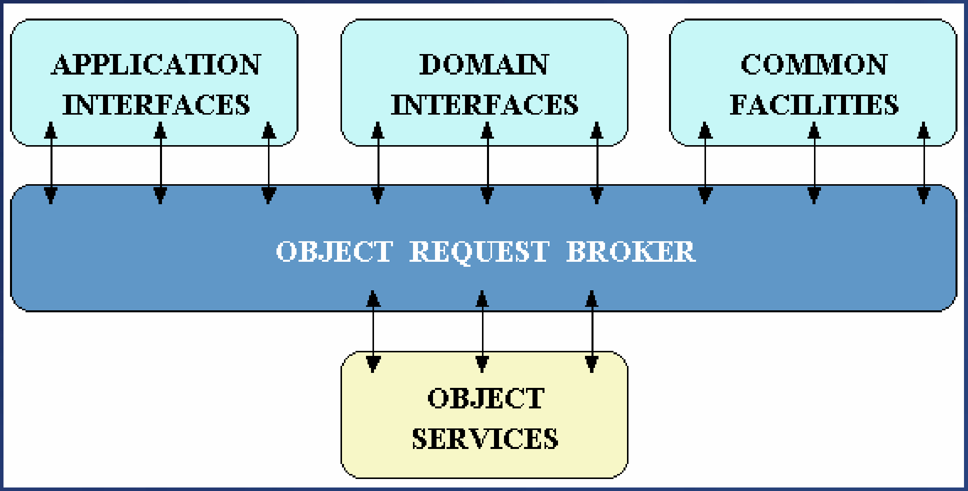
`hfeng2@cs.mcgill.ca`

CORBA Overview

The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing infrastructure being standardized by the Object Management Group (OMG).

CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching.

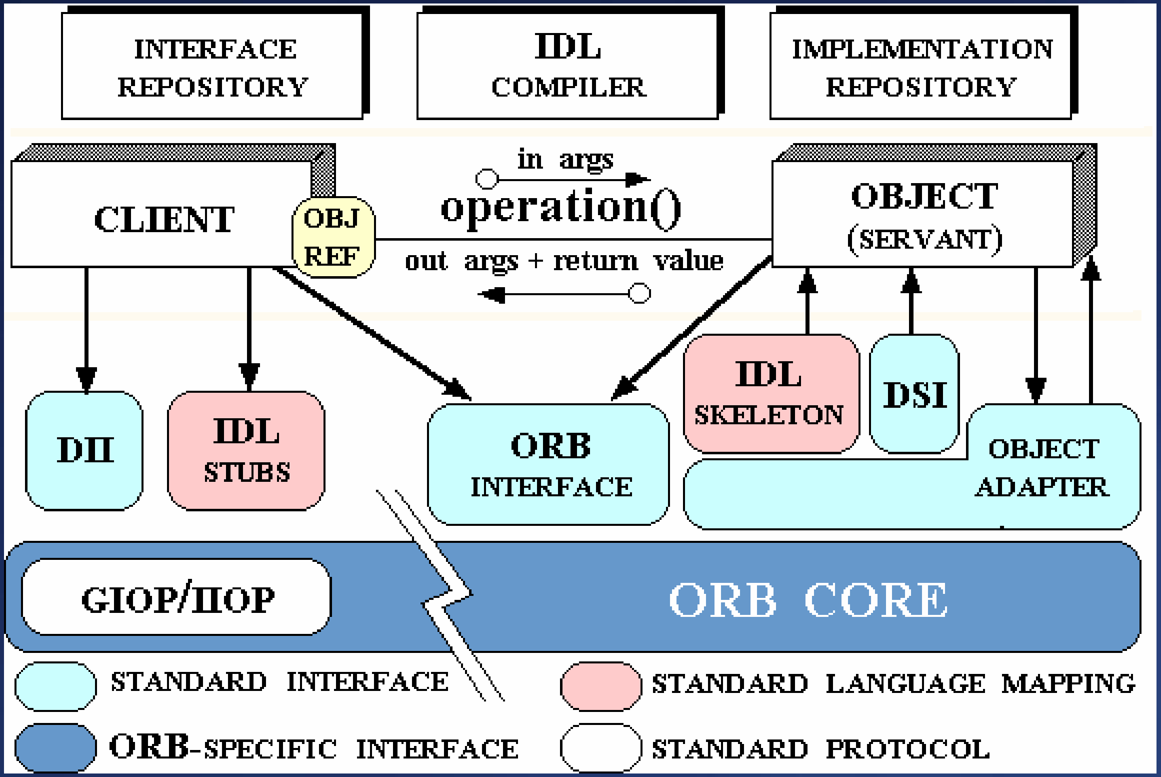
The following figure illustrates the primary components in the OMG Reference Model architecture.



CORBA Components

- **Object Services**. Domain-independent interfaces that are used by many distributed object programs.
  - Naming Service – which allows clients to find objects based on names;
  - Interface Repository – which allows clients to dynamically look for interfaces;
  - Trading Service – which allows clients to find objects based on their properties.
- **Common Facilities**. Horizontally-oriented interfaces, oriented towards end-user applications.
- **Domain Interfaces**. Similar to Object Services and Common Facilities, but oriented towards specific application domains.
- **Application Interfaces**. Interfaces developed specifically for a given application.

CORBA ORB Architecture



- **Object**. A CORBA programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *servant*.

- **Servant**. An implementation programming language entity that defines the operations that support a CORBA IDL (Interface Definition Language) interface.

- **Client**. The program entity that invokes an operation on an object implementation.
- **Object Request Broker (ORB)**. A mechanism for transparently communicating client requests to target object implementations.
- **ORB Interface**. A specification defining an abstract interface for an ORB.
- **CORBA IDL stubs and skeletons**. The "glue" between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.
- **Dynamic Invocation Interface (DII)**. The interface allowing a client to directly access the underlying request mechanisms provided by an ORB.

- **Dynamic Skeleton Interface (DSI)**. The server side's analogue to the client side's DII.

- **Object Adapter**. An interface associating object implementations with the ORB.

So much for theory. A more detailed overview can be found at: `http://www.cs.wustl.edu/~schmidt/corba-overview.html`

Fnorb Introduction

Fnorb is a pure Python implementation of CORBA 2.0 ORB. It has been successfully used in both Python and Jython.

Current stable version is 1.2, but it does NOT run on our lab machines (Python 2.2, RedHat 9). Use version 1.3, which can be downloaded from CVS.

Website

`http://www.fnorb.org/`

Download (CVS)

```
cvs -d:pserver:anonymous@cvs.fnorb.sourceforge.net:/cvsroot/fnorb login

cvs -z3 -d:pserver:anonymous@cvs.fnorb.sourceforge.net:/cvsroot/fnorb co fnorb
```

Documentation

`http://www.fnorb.org/docs/1.2/Fnorb-Guide/`

Installation

```
python setup.py install
```

Fnorb Services

Though CORBA 2.0 standard specified much more services, Fnorb only supports the two most common services: naming service and interface repository. Other services such as trading service (which I need in the first place) are not supported.

- **Naming service**. The service resolving the names of*modules* and *interfaces*.

  A *module* is a logical entity used to organize interfaces in a tree form. *Interfaces* are leaves of the tree, which cannot have sub-interfaces or sub-modules. However, a module may have sub-modules or interfaces inside it.

  Looking for an interface dynamically by its name (and possibly its ancesters' names) is to traverse the tree until a leave is reached.

  Fnorb program `fnaming` runs in the background and provides naming service. Environment variable `FNORB_NAMING_SERVICE` must be set before using any client can access the naming service.

  For example:

  ```
  $ fnaming --ior &
  ```

`IOR:000000...` (a very long string)

`$ export FNORB_NAMING_SERVICE=IOR:000000...` (copy the same string here)

or alternatively:

```
$ fnaming --ior > $HOME/.NameService &
$ export FNORB_NAMING_SERVICE=file:$HOME/.NameService
```

- **Interface repository**. The service handling queries of modules and interfaces by their names or name patterns.

  A client application may dynamically traverse the whole tree structure and obtain related information, such as all the interfaces of a module, all the operations or attributes of an interface, and all the parameters and their types of an operation.

  Similarly, program `fnifr` runs in the background. Environment variable `FNORB_INTERFACE_REPOSITORY` is used.

An Easy Way to Start Both Services

First, set both the environment variables:

```
$ setenv FNORB_NAMING_SERVICE file:$HOME/.NameService
$ setenv FNORB_INTERFACE_REPOSITORY file:$HOME/.InterfaceRepository
```

Then, create script `fnstart`:

```
#!/bin/bash
fnaming --ior > ${FNORB_NAMING_SERVICE#file:} &
fnifr --ior > ${FNORB_INTERFACE_REPOSITORY#file:} &
```

and script `fnstop`:

```
FNAMING=`ps -Cfnaming | grep fnaming | awk '{print $1}'`
if [[ $FNAMING ]]; then
  kill $FNAMING
fi
FNIFR=`ps -Cfnifr | grep fnifr | awk '{print $1}'`
if [[ $FNIFR ]]; then
  kill $FNIFR
fi
```

CORBA IDL

IDL is a system implementation independent language used to describe interfaces of servants.

Example: (testfnorb.idl)

```
#pragma prefix "msdl.cs.mcgill.ca"
// with this, the ID of this module is:
// IDL:msdl.cs.mcgill.ca/TestFnorb:1.0
module TestFnorb
{
  interface TestFnorbIF
  {
    string test_string(in string s);
  };
};
```

To compile the IDL definition, run:

```
$ fnidl testfnorb.idl
```

Two folders will be generated automatically, which contains Python mapping for the IDL:

- `TestFnorb`, which contains the client-side mapping. It is a Python module, which every client requires to import.

- `TestFnorb_skel`, which contains the skeleton for the server-side.

Note that the name (`TestFnorb`) of the two folders comes from the module name specified in the IDL.

Python Implementation of the Server

This server (`server.py`) runs in the background and provides implementation for interface `Test.TestFnorb.TestFnorbIF`.

```
# import required Python modules
from Fnorb.orb import CORBA, BOA
from Fnorb.cos.naming import CosNaming
# import server skeleton
import TestFnorb_skel
# class definition of the implementation
class TestFnorb_Impl (TestFnorb_skel.TestFnorbIF_skel):
   def test_string(self, s):
      print 'Server receives:  '+s
      return 'success'
# initialize ORB and BOA
orb=CORBA.ORB_init()
boa=BOA.BOA_init()
```

```
# build the context to a certain level
ctx=orb.resolve_initial_references("NameService")
path=[CosNaming.NameComponent('Test', ''),
      CosNaming.NameComponent('TestFnorb', '')]
for i in range(len(path)):
  try:
    ctx.bind_new_context(path[:i+1])
  except CosNaming.NamingContext.AlreadyBound:
    pass
# create an object responsible for calls
testfnorb=TestFnorb_Impl()
# create a ref (What's ref?  God knows.)
ref=boa.create('TestFnorbIF', TestFnorb_Impl._FNORB_ID)
# the path of the interface is the old path plus 1 level
path.append(CosNaming.NameComponent('TestFnorbIF', ''))
ctx.rebind(path, ref)
# notify BOA the implementation is ready for calls
boa.obj_is_ready(ref, testfnorb)
# start the CORBA mainloop, which will block the current thread
boa._fnorb_mainloop()
```

Python Implementation of Client 1

This client uses the naming service to locate the server, and invokes its `test_string` operation.

```
# import required Python modules
from Fnorb.orb import CORBA
from Fnorb.cos.naming import CosNaming
# import client-side mapping
import TestFnorb
# initialize ORB and naming service
orb=CORBA.ORB_init()
ctx=orb.resolve_initial_references('NameService')
# the path where the server is located
testfnorb=ctx.resolve([CosNaming.NameComponent('Test', ''),
       CosNaming.NameComponent('TestFnorb', ''),
       CosNaming.NameComponent('TestFnorbIF', '')])
```

```
# simple type-checking
if testfnorb._is_a(TestFnorb.TestFnorbIF._FNORB_ID):
  # invoke server operation
  print 'Client 1 receives:  '+testfnorb.test_string('Hello server!')
else:
  print 'Error!'
```

Executing the server and the client:

```
$ python server.py &
$ python client1.py
Server receives:  Hello server!
Client 1 receives:  success
```

Python Implementation of Client 2

This client uses the interface repository to dynamically list all the modules, their interfaces, the attributes and operations of those interfaces, and the parameters of each operation.

```
from Fnorb.orb import CORBA
def dump_module(ctx, level):
  modules=ctx.contents(CORBA.dk_Module, 0)
  interfaces=ctx.contents(CORBA.dk_Interface, 0)
  for m in modules:
    print ' '*level+'+ '+m._get_name()
    dump_module(m, level+1)
  for i in interfaces:
    print ' '*level+'- '+i._get_name()
    desc=i.describe_interface()
    opers=desc.operations
    attrs=desc.attributes
    for at in attrs:
      print ' '*(level+1)+'at '+at.name
    for op in opers:
      print ' '*(level+1)+'op '+op.name
      params=op.parameters
      for p in params:
        print ' '*(level+2)+p.type.kind().name()+' '+p.name
orb=CORBA.ORB_init()
ctx=orb.resolve_initial_references('InterfaceRepository')
dump_module(ctx, 0)
```

Executing the server and the client:

```
$ fnfeed testfnorb.idl
$ python server.py &
$ python client2.py
+ TestFnorb
  - TestFnorbIF
    op test_string
      tk_string s
```

That’s it!

Thank you for your attendance!

Any question or concern, pleace contact:
hfeng2@cs.mcgill.ca