CORBA and Perl

Abram Hindle

Victoria Perl Mongers

abez@abez.ca

May 20, 2003

This Presentation

- What am I going to cover?
  - CORBA Introduction
  - ORBit Introduction
  - ORBit and Perl

This Presentation

- What am I not going to cover?
  - Indepth IDL details
  - Non-Orbit based perl CORBA implementations
  - Using ORBit in any other language.
  - ORBit2
  - Programming OO in Perl.

CORBA Introduction

- What is CORBA?
  - Common Object Request Broker Architecture
  - Multi Vendor Communication Architecutre
  - A interoperable architecutre well suited to distributed computering.
  - Framework to provice an object level abstraction for remote procedure calls and the underlying communications.
  - Defined and maintained by the Object Management Group

CORBA Introduction

- Why CORBA?
  - Interopability.
  - Portability - client side.
  - Reuse.
  - Multiple language support.
  - Scalable.

CORBA Introduction

- What are the components of a CORBA System?
  - ORB - Object Request Broker. This is your agent which communicates on your behalf to fufill and request remote methods.
  - IDL - Interface Definition Language. You define your object's external interfaces in IDL.
  - COS - Common Object Services e.g. NameServer
  - IOP - Inter Orb Protocal, enable communication between orbs.

CORBA Example

- A Trivial CORBA Clock Example
- Clock.idl

```
module VictoriaPM {
        interface Clock {
                long getTime();
        };
}
```

CORBA Example

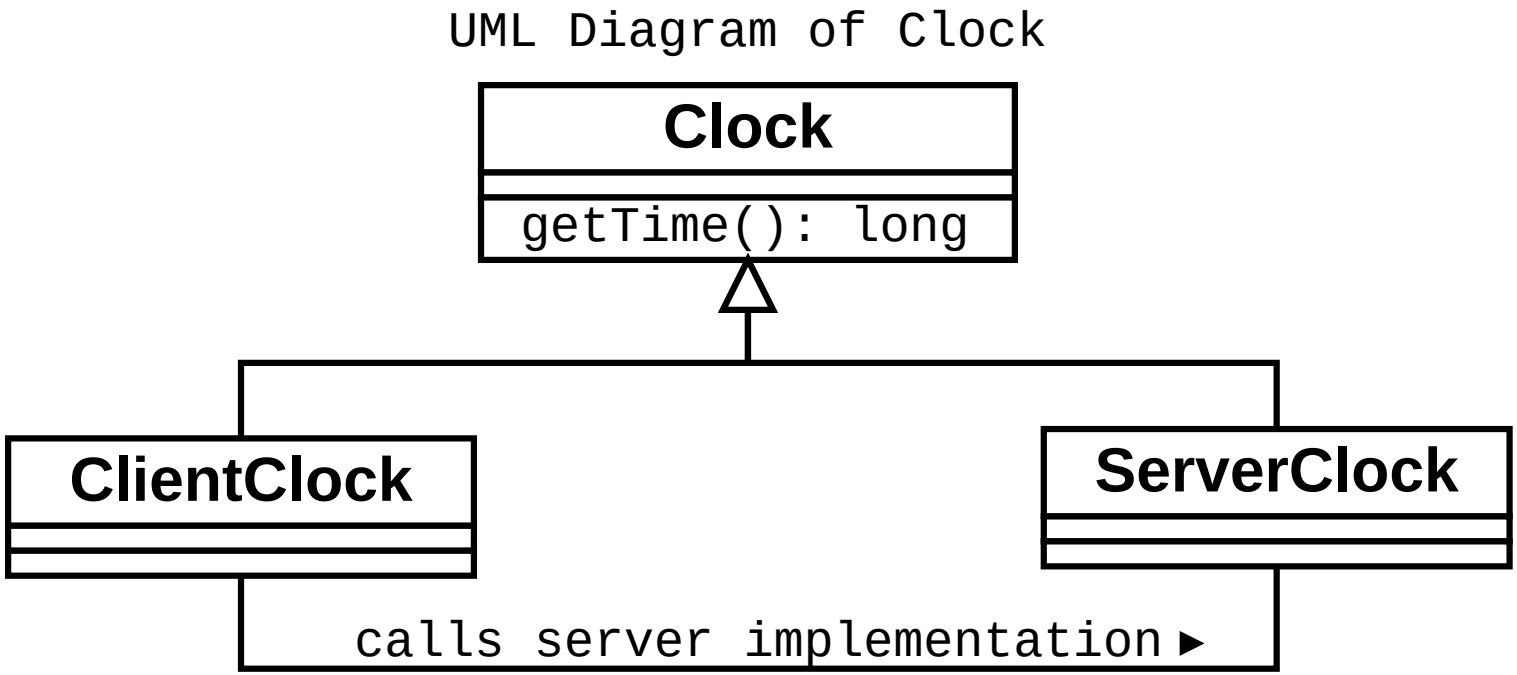

Figure 1: Class Diagram of Clock

CORBA Example

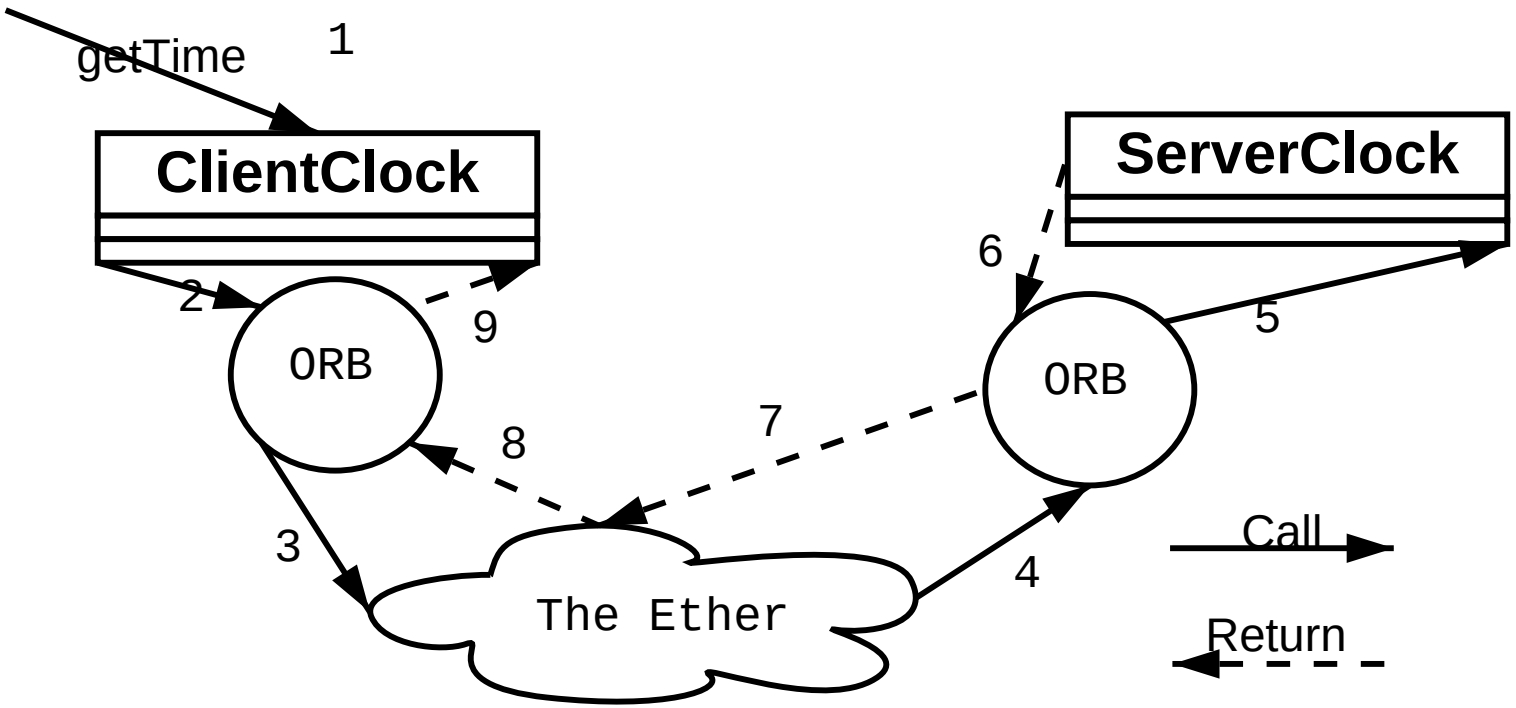

Figure 2: Flow control of a call to Clock (Notice the communication through the ORBs)

CORBA Examples

- Why would I use CORBA?
  - You want OO remote method invocation.
  - You want to create a multi platform, multi language networked system.
  - You want to make a heterogenous component based system.
  - You want to abstract legacy applications with a developer friendly OO interface.
  - You want networking but you don’t want to program “effecient networking” let alone be concerned with it.

IDL

- **IDL**: Interface Definition Language is used to describe your interfaces to CORBA objects. (Like prototypes)
- **Module**: A Package or Module encapsulates all the interfaces and structs into a neat package. (Like packages)
- **Interface**: A definition of the corba methods and attributes a interface implementer provides. (Like classes)
- **Methods**: single return valued methods with parameters that can act as inputs, outputs or both (in , out , inout).
- **Attributes**: attributes an interface is expected to provide.

IDL Example

- An example dictionary service. This will be used in further CORBA::ORBit examples. IDL maps differently to each language. Not all languages support all the features IDL supports and vice-versa.
- dictionary.idl

```
module VictoriaPM {
        typedef sequence<string> List;
        interface Dictionary {
                // put a key value pair into the dictionary.
                void put(in string key, in string val);
                // Given a key get a value
                string get(in string key);
                // Given a list of keys get a list of values
                List getList(in List keys);
        };
};
```

How do I use CORBA and PERL?

- Choose a CORBA library to use:
  - **COPE**: - CORBA implemented in Perl (100% perl)
  - **CORBA::MICO**: CORBA bindings to MICO (uses C)
  - **CORBA::ORBit**: CORBA bindings to ORBit (uses C)

ORBit?

- ORBit is a CORBA ORB destined for use on the desktop but can be used for network communication. It is heavily used in the GNOME project.
- We'll use ORBit because it is commonly found on many linux systems.
- CORBA::ORBit bindings are being maintained and work quite well.
- IDL's do not need to be precompiled with CORBA::ORBit (other languages "precompile" IDLs) thus saving Perl Programmers some unnecesseary hassle.

NameService

- The ORBit nameservice interface is defined by the “CosNaming.idl”.
- The NameService is reachable by using it’s object reference to resolve it.
- The nameservice allow the binding of object references to strings.
- NamingContext’s can be created to create a name hierarchy.
- **bind**: bind a object reference to a name.

```
$ns->bind([ { id=>’nametobind’, kind=>’’} ], $poa->
     servant_to_reference($servant));
```

- **resolve**: resolve a name to a object reference.

```
my $object = $ns->resolve( [{id => "nametobind", kind => ""}] );
```

NameService...

- Unfortunately ORBit lacks an easy to use method to retrieve this Object Reference (IOR).
- The nameservice prints it's IOR, you can save this and provide it at the commandline to enable your ORB to resolve the nameservice.
- e.g. `perl dict-client.pl -ORBNamingIOR=IOR:01000000280...`
- or `perl dict-client.pl -ORBNamingIOR='cat ns.ior'` if you use a file to hold the object reference.

How to Make a Server

- Include the necessary IDL's. (They will be "compiled" at runtime). Include this at the start of anything that needs to access your IDL interfaces. (Also include Error if you want to try and use exceptions).

```
use CORBA::ORBit idl => [ qw(CosNaming.idl dictionary.idl) ];
use Error qw(:try);
```

- The next step is to resolve a reference to an ORB, the RootPOA (Portable Object Adapter) and the NameService (to resolve future objects).

```
#get our ORB
my $orb = CORBA::ORB_init("orbit-local-orb");
#get our POA so we can act as a server
my $poa = $orb->resolve_initial_references("RootPOA");
#get our nameservice
my $ns = $orb->resolve_initial_references("NameService");
$ns or  die "No Nameserver found!";
```

How to Make a Server ...

- The next step will be to make a new Object and associate it with our POA (Portable Object Adapter)

```
my $servant = new Dictionary();
#Activate our Dictionary with COBRA
my $id = $poa->activate_object ($servant);
#get a corba object reference
my $ref = $orb->object_to_string ($poa->id_to_reference ($id));
#start the poa manager
$poa->_get_the_POAManager->activate;
```

- Now we can bind our newly made object to a name in the nameserver and start our object server (nothing will run after we call “run()”;

```
#we want to make our new dictionary object map to a key in the
     nameserver
my $binding = [ { id=>’dictionary’, kind=>’’} ];
#using the CORBA bindings for the namserver bind our new object to
$ns->bind($binding, $poa->servant_to_reference($servant));
#start CORBA Event Loop
$orb->run();
```

How to Provide a ServerSide Implementation

- To return a list/sequence of objects, return a arrayref (sequences of octets map to a single scalar).
- To accept a list/sequence as a parameter, accept a single arrayref
- Structs are simply hashreferences e.g. `{key => name}`
- Classes are Perl Objects (blessed hashrefs).
- Your implementation has to use POA_ModuleName::Interface as a base class.
- short, long, float, double, octect, boolean, char map to perl scalars. To learn more about mapping run `‘‘perldoc CORBA::ORBit::mapping’’`

Dictionary Implementation

```
package Dictionary;
use base qw(POA_VictoriaPM::Dictionary);
sub new {
        my $type = shift;
        $type = ref($type) || $type;
        my $self = {dict=>{}};
        bless($self,$type);
        return $self;
}
sub put {
        my ($self,$key,$val) = @_;
        $self->{dict}->{$key} = $val;
}
sub get {
        my ($self,$key) = @_;
        return $self->{dict}->{$key};
}
sub getList {
        my ($self,$list) = @_;
        return [map { $self->{dict}->{$_} } @$list];
}
```

How to Run The Server

- We need to run the server and we also need a nameservice running. In the following example we save the nameserver's reference to a file and load the the reference through the commandline for our server. You don't need to pass arguements to the ORB_init, it will be done automatically.

```
orbit-name-server  > ns.ior &
sleep 2
perl dict-server.pl -ORBNamingIOR=‘cat ns.ior‘  &
```

How to Make a Client

- A client is much easier to create. All we need to do is to initalize our ORB then optionally resolve our nameserver. Once we have the nameserver we can resolve other objects by their name.
- To Run the client simply use:

```
#!/bin/sh
perl dict-client.pl -ORBNamingIOR=‘cat ns.ior‘
```

dict-client.pl

```
#!/usr/bin/perl -w
use CORBA::ORBit idl => [ qw(dictionary.idl CosNaming.idl) ];
use Error qw(:try);
use strict;
#init the ORB
my $orb = CORBA::ORB_init("orbit-local-orb");
#get nameservice
my $ns = $orb->resolve_initial_references("NameService");
$ns or die "No NameService!";

#retrieve the dictionary
my $name = [{id => "dictionary", kind => ""}];
my $server = $ns->resolve($name);

#interact with the dictionary object.
$server->put("joe","123");
$server->put("janice","13.5");
$server->put("jacky","99.5");
print "Got: ",$server->get("joe"),"\n";
print "Got: ",join(",",@{$server->getList([qw(joe janice jacky)])}),"\n";
```

How to Use CORBA Objects

- To read an attribute "`_get_attribute`" methods can be used.
- To set an attribute "`_set_attribute`" methods can be used.
- Parameters for methods can be multidirectional. The perl mapping is done in a particular manner:

```
#IDL: string method(in string instring, inout string inoutstring, out
     string outstring)
my ($return,$outstring) = $obj->method($instring,\$inoutstring);

#IDL: typedef sequence<string> strlist; strlist getList()
my $list = $obj->getList();
```

Concerns

- When `$orb->run` is called the server blocks and enters the ORBit event loop. You cannot run any more code in the same thread. If you want to mix your clients and server (e.g. listeners) you'll have to use more than one thread.
- As specified before the nameserver is a pain to access.
- Avoid the site "`http://people.redhat.com/otaylor/corba/orbit.html`" as it hosts an old version of CORBA::ORBit that is not able to use the CosNaming.idl module.
- POA and NameService are all defined by IDLs in CORBA. CORBA is quite consistent this way.
- Often times you have to refer to the C documentation for help rather than any Perl documentation.

Possible Issues

- If you get errors regarding marshalling and the nameserver in perl make sure you have the latest CORBA::ORBit from CPAN. (0.4.7+)
- Make sure "`Error.pm`" is installed for exception handling. Sometimes it's not automatically installed.
- Currently only ORBit 1 seems really supported. I personally have not tested ORBit2 with CORBA::ORBit.
- If you have a newer version of ORBit1 (e.g. a CVS copy) you can use urls like "`corbaloc://`" to describe where the nameserver is.
- Make sure your nameserver is running.

Get HELP

- perldoc CORBA::ORBit::mapping - how does CORBA map to perl.
- See the examples provided with the source code of CORBA::ORBit.
- http://www.gnome.org/projects/ORBit2/
- http://www.lausch.at/gnome/programming/gnome-corba-programming.html

Conclusions

- CORBA is extremely helpful for building distributed systems.
- CORBA is often difficult to interface because of it’s specific “order of operations” and server event loop.
- CORBA::ORBit is great as you don’t have to deal with generated perl code.
- You will benefit if you use CORBA in more than one language. You will notice how pleasant it is to use PERL and CORBA can be.

References

- http://search.cpan.org/author/HROGERS/CORBA-ORBit-0.4.7/ORBit.pm
- http://www.omg.org/gettingstarted/corbafaq.htm
- http://adams.patriot.net/ tvalesky/freecorba.html
- http://www.gnome.org/projects/ORBit2/
- http://www.lausch.at/gnome/programming/gnome-corba-programming.html
- http://www2.lunatech.com/research/corba/cope/

CodeListing: dict-client.pl

```
#!/usr/bin/perl -w
use CORBA::ORBit idl => [ qw(dictionary.idl CosNaming.idl) ];
use Error qw(:try);
use strict;
#init the ORB
my $orb = CORBA::ORB_init("orbit-local-orb");
#get nameservice
my $ns = $orb->resolve_initial_references("NameService");
$ns or die "No NameService!";

#retrieve the dictionary
my $name = [{id => "dictionary", kind => ""}];
my $server = $ns->resolve($name);

#interact with the dictionary object.
$server->put("joe","123");
$server->put("janice","13.5");
$server->put("jacky","99.5");
print "Got: ",$server->get("joe"),"\n";
print "Got: ",join(",",@{$server->getList([qw(joe janice jacky)])}),"\n";
```

CodeListing: dict-server.pl

```
#!/usr/bin/perl -w

use CORBA::ORBit idl => [ qw(CosNaming.idl dictionary.idl) ];
use strict;

package Dictionary;
use base qw(POA_VictoriaPM::Dictionary);
#constructor (nothing special)
sub new {
        my $type = shift;
        $type = ref($type) || $type;
        my $self = {};
        $self->{dict} = {};
        bless($self,$type);
        return $self;
}
sub put {
        my ($self,$key,$val) = @_;
        $self->{dict}->{$key} = $val;
}
sub get {
        my ($self,$key) = @_;
        return $self->{dict}->{$key};
```

```
}
sub getList {
        my ($self,$list) = @_;
        return [map { $self->{dict}->{$_} } @$list];
}
package main;
use Error qw(:try);
#get our ORB
my $orb = CORBA::ORB_init("orbit-local-orb");
#get our POA so we can act as a server
my $poa = $orb->resolve_initial_references("RootPOA");
#get our nameservice
my $ns = $orb->resolve_initial_references("NameService");
$ns or  die "No Nameserver found!";

my $servant = new Dictionary();
#Activate our Dictionary with COBRA
my $id = $poa->activate_object ($servant);
#get a corba object reference
my $ref = $orb->object_to_string ($poa->id_to_reference ($id));
#start the poa manager
$poa->_get_the_POAManager->activate;
#we want to make our new dictionary object map to a key in the nameserver
my $binding = [ { id=>'dictionary', kind=>''} ];
#using the CORBA bindings for the namserver bind our new object to
```

```
#‘‘dictionary’’
#$ns->bind($binding,
$ns->bind($binding, $poa->servant_to_reference($servant));
#start CORBA Event Loop
$orb->run ();
exit(0);
```

CodeListing: dictionary.idl

```
module VictoriaPM {
        typedef sequence<string> List;
        interface Dictionary {
                // put a key value pair into the dictionary.
                void put(in string key, in string val);
                // Given a key get a value
                string get(in string key);
                // Given a list of keys get a list of values
                List getList(in List keys);
        };
};
```

CodeListing: runserver

```
#!/bin/sh
echo "KILL NAME SERVER"
killall orbit-name-server
echo "START NAME SERVER"
orbit-name-server  > ns.ior &
sleep 2
echo "DICT-SERVER"
perl dict-server.pl -ORBNamingIOR=‘cat ns.ior‘  &
```

CodeListing: runclient

```
#!/bin/sh
perl dict-client.pl -ORBNamingIOR=‘cat ns.ior‘
```