CORBA

with

Python

Dr Duncan Grisby
duncan@grisby.org

(These slides at `www.omniorb.org/~dpg1/python10`)

Outline

1. Introduction
2. What is CORBA?
3. CORBA myths
4. Python and CORBA
5. Facilities of CORBA
6. Comparison with web services
7. When to use CORBA

About me

- BA and PhD at the University of Cambridge Computer Laboratory.
- Work for AT&T Laboratories Cambridge (`www.uk.research.att.com`).
- Working on CORBA systems — ways to make CORBA easier to use.
- Main author of omniORBpy.

What is CORBA?

Common Object Request Broker Architecture.

- i.e. a common architecture for object request brokers.
- A framework for building *object oriented* distributed systems.
- Cross-platform.
- Language neutral.
- An extensive open standard, defined by the Object Management Group.
  - `www.omg.org`

Object Management Group

- Founded in 1989.
- The world's largest software consortium with around 800 member companies.
- Only provides *specifications*, not implementations.
- As well as CORBA core, specifies:
  - Services: naming, trading, security, . . .
  - Domains: telecoms, health-care, finance, . . .
  - UML: Unified Modelling Language.
- All specifications are available for free.

CORBA Myths

- There are many myths about CORBA.
  - Ignorance or malice?
- These myths taken from various places:
  - ‘Understanding SOAP’, Kennard Scribner and Mark Stiver.
  - ‘Special Edition Using SOAP’, John Paul Mueller.
  - `www-106.ibm.com/developerworks/webservices/library/ws-arc3/`
  - `www.infoworld.com/articles/tc/xml/01/07/16/010716tcsoap.xml`
  - …

CORBA Myths

Myth: CORBA is comparable to SOAP / XML-RPC.

Truth: SOAP and XML-RPC are *wire protocols*. CORBA includes a wire protocol, GIOP, but it also has an object model, an interface definition language, standard language mappings, standard services, . . .

CORBA client and server code is *portable* between different CORBA implementations.

CORBA Myths

Myth: CORBA is too complex.

Truth: Distributed systems are fundamentally complex things. CORBA does an amazingly good job at managing that complexity for you.

CORBA Myths

Myth: CORBA is too hard to use.

Truth: CORBA makes simple things simple (see example later). Complex things are only as complex as they have to be, but there is an inevitable learning curve.

Some language mappings are ugly (and so are the languages). With Python, everything is beautiful.

CORBA Myths

Myth: CORBA is bloated.

Truth: A good CORBA ORB is around 1 or 2 MB of shared library code. If CORBA is too bloated for your system, there's a good chance Python is too. Either way you look at it, it's not *unnecessarily* bloated.

CORBA Myths

Myth: CORBA is not interoperable.

Truth: CORBA really *is* interoperable.
Interoperability bugs are almost unheard-of.
People have been working on getting it right since 1995.

CORBA Myths

Myth: CORBA is not Unicode aware.

Truth: CORBA has had a wstring type since 1997. Some ORBs were slow to support it, but most have it now.

CORBA Myths

Myth: CORBA is not ‘firewall friendly’.

Truth: CORBA uses IIOP, a protocol above TCP. It is not hard for a service provider to open a suitable port on a firewall.

The problem is *callback* objects. They require a connection back to the client. CORBA partially solves this with bi-directional IIOP.

Remember that firewalls are put there for a reason.

CORBA Myths

Myth: CORBA does not scale.

Truth: CORBA has *proven* scalability in real applications. ORB designers have had many years to work on this.

CORBA Myths

Myth: CORBA does not scale because IIOP is ‘stateful’.

Truth: CORBA clients and servers can decide to close IIOP connections at any time.

Opening a TCP connection is expensive and latency bound. Usually, you *want* to cache open connections.

CORBA Myths

Myth: CORBA does not support ‘loose coupling’.

Truth: CORBA *does* support loose coupling, through the Interface Repository, Dynamic Invocation Interface, and Dynamic Skeleton Interface.

Almost nobody uses it. Loose coupling sounds good until you try to use it.

CORBA Myths

Myth: CORBA is expensive.

Truth: Many open source ORBs: omniORB, ORBit, MICO, TAO, JacORB, OpenORB, . . .

CORBA Myths

Myth: CORBA is not supported by scripting languages.

Truth: All the major scripting languages have at least one CORBA implementation. Python has four implementations.

CORBA Myths

Myth: CORBA is obsolete.

Truth: CORBA is the *only* solution for cross-platform, cross-language, object-oriented distributed systems.

The only question is whether your application needs these things. . .

CORBA Myths

Truth: CORBA is not the current ‘in thing’.

Myth: That this matters.

Is it best to base your application on something new, rapidly changing, untested, and in the press, or on something mature and stable?

CORBA designers have had many years to think about, and solve, problems that web services designers are only just discovering. Wheel reinvention is everywhere.

CORBA Myths

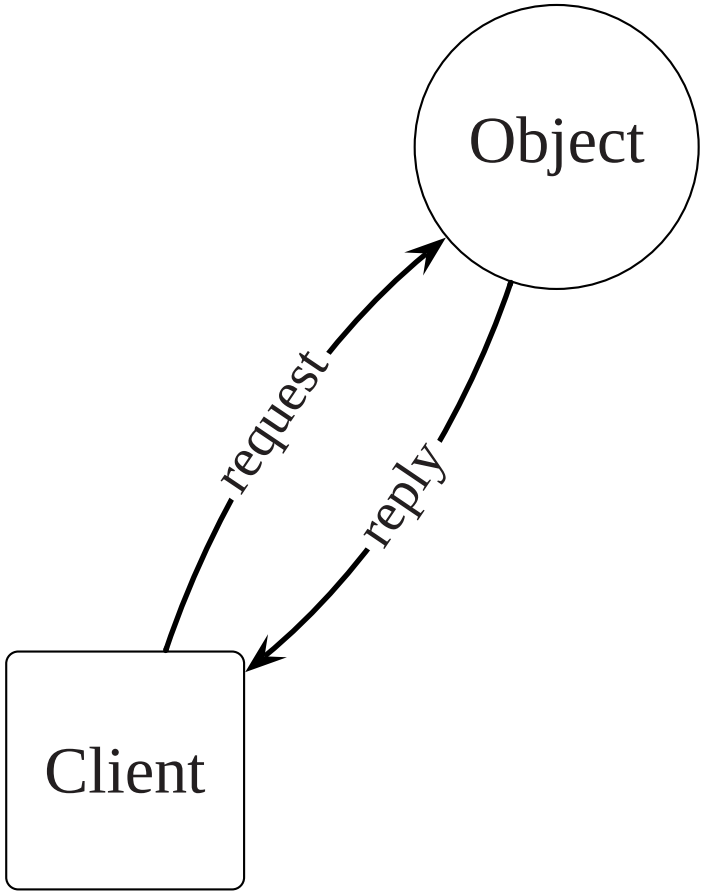
Myth: CORBA is the solution to all problems.

Truth: Of course it isn't! Nothing ever is. A good system designer chooses the best tools for the job.

Python ORBs

- omniORBpy
  - Based on C++ omniORB. Multi-threaded. Free (LGPL).
  - `www.omniorb.org/omniORBpy`
- orbit-python
  - Based on C ORBit. Single-threaded. Free (LGPL).
  - `projects.sault.org/orbit-python/`
- Fnorb
  - Mostly Python, with a small amount of C. Multi-threaded. Newly open source (Python style). Back from dead?
  - `www.fnorb.com`
- ILU
  - Based on C ILU. More than just CORBA. Open source. Dead?
  - `ftp.parc.xerox.com/pub/ilu/ilu.html`

A CORBA call



- A classical object model
  - the client sends request messages to the object; the object sends replies back.
- The client does not care where the object is
  - because the ORB deals with it.
- The client knows what messages it can send, because the object has an *interface*
  - specified in CORBA IDL. . .

Interface Definition Language

- IDL forms a ‘contract’ between the client and object.
- Mapped to the target language by an *IDL compiler*.
- Strong typing.
- Influenced by C++ (braces and semicolons — sorry!).

```
module Snake {
  interface Adder {
    long accumulate(in long a);
    void reset();
  };
};
```

Python client

```
>>> import sys, CORBA, Snake
>>> orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
>>> adder = orb.string_to_object("corbaname:rir:#adder.obj")
>>> adder.accumulate(5)
5
>>> adder.accumulate(6)
11
>>> adder.accumulate(42)
53
>>> adder.reset()
>>> adder.accumulate(10)
10
```

Python server

```
2  import sys, CORBA, CosNaming, Snake, Snake__POA
3
4  class Adder_i (Snake__POA.Adder):
5      def __init__(self):
6          self.value = 0
7
8      def accumulate(self, a):
9          self.value = self.value + a
10         return self.value
11
12     def reset(self):
13         self.value = 0
14
15 orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
16 poa = orb.resolve_initial_references("RootPOA")
17
18 adderServant = Adder_i()
19 poa.activate_object(adderServant)
20 adderObjref  = adderServant._this()
21
22 nameRoot = orb.resolve_initial_references("NameService")
23 nameRoot = nameRoot._narrow(CosNaming.NamingContext)
24 name     = [CosNaming.NameComponent("adder", "obj")]
25 nameRoot.rebind(name, adderObjref)
26
27 poa._get_the_POAManager().activate()
28 orb.run()
```

IDL Facilities

- All types and interfaces are specified in IDL.
- Base types:
  - integers, floating point, strings, wide strings.
- Constructed types:
  - enumerations, sequences, arrays, structures, discriminated unions, fixed point, interfaces.
- Interfaces:
  - operations, attributes, exceptions.
- Dynamic types:
  - Any, TypeCode.

IDL Example

```
module Example {

  struct Person {
    string name;
    unsigned short age;
  };

  enum DwellingKind { house, flat, cottage, castle };

  struct Dwelling {
    DwellingKind kind;
    Person owner;
    unsigned long number_of_rooms;
  };

  interface Auction {
    readonly attribute Dwelling lot;
    readonly attribute float high_bid;
    boolean bid(in Person who, in float amount);
  };

  interface AuctionHouse {
    Auction SellDwelling(in Dwelling to_sell, in float reserve);
  };
};
```

IDL to Python

- Standard Python language mapping:
  - `www.omg.org/technology/documents/formal/python_language_mapping.htm`
- Map IDL to Python with an *IDL compiler*. . .

```
$ omniidl -bpython example.idl
```

- Use the mapped types from Python. . .

```
>>> import Example
>>> fred = Example.Person("Fred Bloggs", 42)
>>> residence = Example.Dwelling(Example.cottage, fred, 3)
>>> residence.number_of_rooms
3
>>> auctioneer = # Get AuctionHouse object from somewhere
>>> auction = auctioneer.SellDwelling(residence, 1000.0)
>>> auction.bid(Example.Person("Joe Smith", 28), 2000.0)
>>> auction._get_high_bid()
2000.0
```

ORB and POA

- The Object Request Broker (ORB) holds everything together.
  - Not a stand-alone process—library code in all CORBA applications.
  - Provides basis for network-transparency, object model, etc.
- The Portable Object Adapter (POA) supports server code.
  - Supports activation of *servants*—i.e. implementation objects.
  - On-demand activation, default servants, flexible servant locators.

Standard CORBA services

- Naming
  - Tree-based hierarchy of named objects.
  - Supports federation.
- Notification
  - Asynchronous event filtering, notification.
- Interface repository
  - Run-time type discovery.
- Trading
  - Object discovery by properties.
- Security, Transaction, Concurrency, Persistence, Time, . . .

Web service comparison

- CORBA is *object-oriented*
  - Object references are first-class data types.
  - Application entities can be modelled as objects.
  - Managing large numbers of objects can be tricky.
- XML-RPC and SOAP are *procedural*
  - No implicit state in function calls.
  - Using explicit state in all calls can become tricky.

Web service comparison

- CORBA is *statically typed*.
- XML-RPC and SOAP are *dynamically typed*.
- Dynamic typing in Python is (usually) a good thing. What about in a distributed application?
  - Dynamic typing can help rapid application development. . .
  - . . . but type errors change from being a debugging issue to a security issue.

Dynamic type security

```
def credit_account(self, account_name, amount):
    account = database.start_transaction(account_name)
    account.balance = account.balance + amount
    account.commit_transaction()
```

- With dynamic typing, `amount` might be a string, not a number, so Python raises a `TypeError` and the transaction is never committed!
- Web services code must therefore explicitly check all types in application code.
- The CORBA run-time knows your IDL, so it prevents this kind of error. The application can safely assume the types it receives are what the IDL said.

Web service comparison

- CORBA uses a compact binary format for transmission.
  - Efficient use of bandwidth.
  - Easy to generate and parse.
- XML-RPC and SOAP use XML text.
  - Egregious waste of bandwidth.
  - Easy-ish to generate, computationally expensive to parse.
  - ‘Easy’ for a human to read
    - not this human!
- CORBA is 10–100 times more compact, 100–500 times faster.

When to use CORBA

- Use CORBA if
  - object orientation and complex types are important.
  - interoperability is important.
  - performance is important.
  - CORBA's services solve many of your problems.
- Use XML-RPC if
  - your requirements are *really* simple.
  - performance is not a big issue.

When to use CORBA

- Use SOAP if
  - you like tracking a moving ‘standard’ :-)
  - you want to be buzzword-compliant.
- Use sockets if
  - you need to stream binary data.
  - you can’t afford *any* infrastructure.
- Use something else if
  - it fits neatly with your application.
- Use a combination of things if
  - it makes sense to do so.

Conclusion

- CORBA is here to stay.
- It is the best solution to many real-world problems.
- The value of web services is not as a replacement for CORBA, but an addition.
- Web services proponents could learn a lot from CORBA, if only they looked.