

# CORBA

## Common Object Request Broker Architecture

Refs: Distributed Object Computing With CORBA  
(Steve Vinoski)

Check the home page for additional references  
(CORBA links).

# Review of some benefits of XDR & RPC

- XDR takes care of providing translation between differing data representations.
- RPC provides a portable, high-level programming interface.
  - The remote procedure interface defines all communication.
  - Clients and servers can be anywhere.
  - Formal interface for finding servers/services (portmapper).

# RPC for OOP

- We could extend RPC to handle C++.
  - Calling an object method is like calling a remote procedure.
  - What about data members?
    - We could make sure every object knows how to package up all data members (called marshaling) and send to the remote method.
  - Virtual Functions?

# Before looking at CORBA

- Think about developing a challenging network application.
- You will need to do something similar for test#2!
- Our example will start simple and grow.

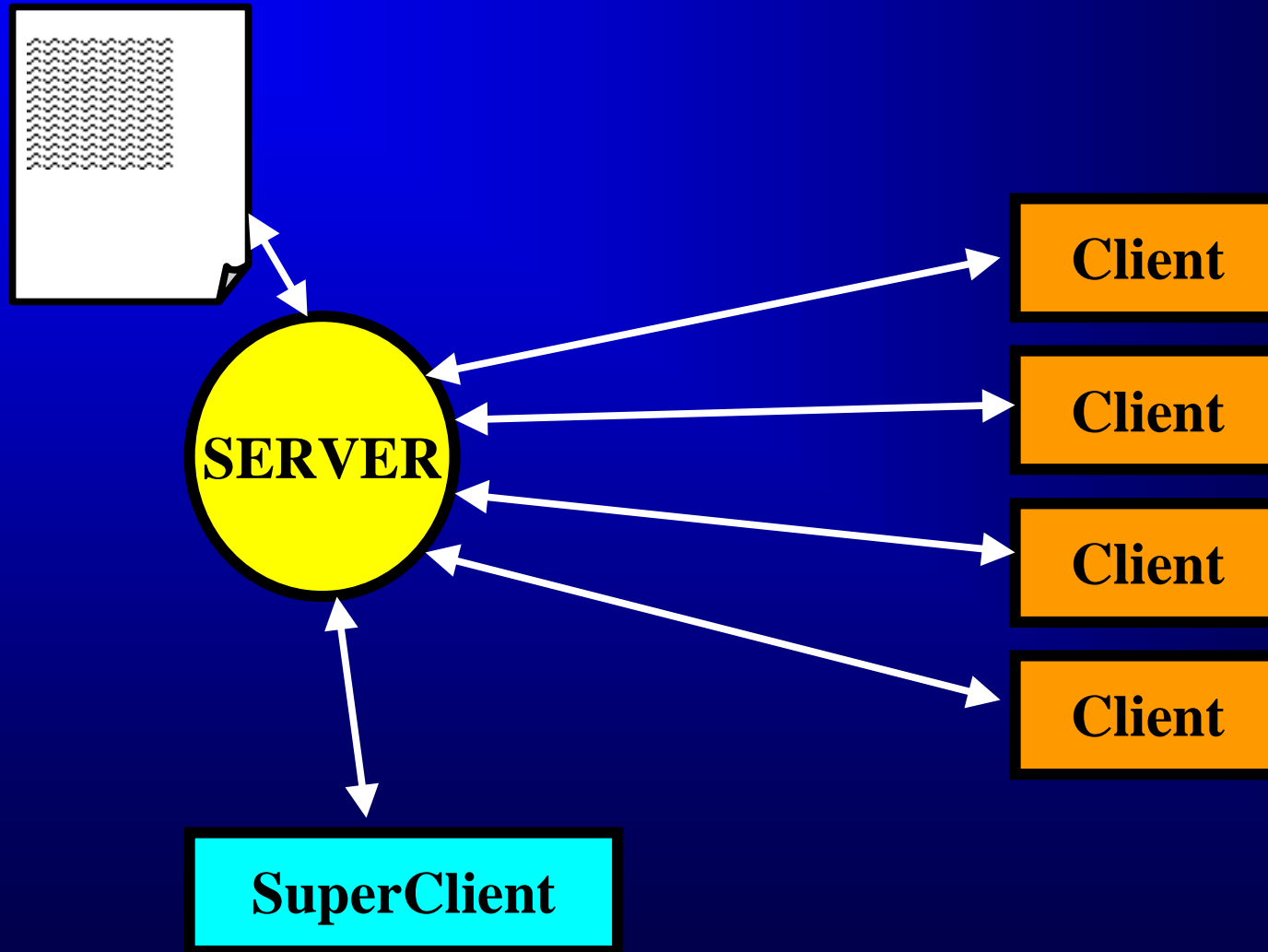
# A sample application - multiedit

- Consider a network application that supports shared document editing.
- Initially we will assume that the only thing in the document is text.
- We want to support many users:
  - All users can see the current document.
  - All users can change the current document.

# Some Issues

- Many of the same issues we talked about when discussing multi-user chat.
- Probably need some support for determining who can make changes and when (to avoid chaos).
- Where is the document (does it exist when everyone logs off?).

# A Server-Centric system



# Server-Centric System

- The Server has the document.
- Anyone can view the document.
- SuperClient determines who can change the document.
- Updates (changes to the document sent to all clients) could be:
  - Initiated by the server - this is called a “push”.
  - Initiated by the client - this is called a “pull”.

# Server software requirements

- Database of documents.
- Handle client requests
  - create a new document
  - get a copy of a document
  - make a change to a document
  - request recent changes
- Handle SuperClient requests
  - Establish current privileged client.
  - Undo changes ?

# Client software

- Need to be able to display (render) a document.
- Request privileged status.
- Translate user actions into a change request and send to server.
- Ask for a document.
- Ask for recent changes.

# SuperClient software

- Determine current privileged client.
- Set (change) privileged client.
- Request previous document version (undo changes).

# Assumptions

- We haven't discussed everything (security, network efficiency, robustness, ...) but - *let's assume we have developed multi-edit and it works great.*
- Now assume we want to provide support for more elaborate documents.

# Document Entity Types

- We want to be able to use multi-edit to create and modify documents containing more than plain old text.
  - Tables
  - Graphs
  - Drawing Primitives (lines, arcs, shapes, etc).

# Adding new features

- Changes we need include:
  - Server: not much, possibly document storage and retrieval, and new change request protocol to support new types of data.
  - Client:
    - GUI needs to support creation/modification of new document entity types.
    - Renderer needs to know about new document entity types.

## We finish Version 2.0

- Multi-edit now supports text, tables, graphs and drawing primitives.
- We had to add code for each new entity type.
- Some additional code needed to manage the document (entity placement, etc).

# Manager from hell

- Now, out of the blue, our Manager wants the following (by next week):
  - for Ford Motor Co: a special version that includes support for CAD drawings.
  - For MTV: a special version allows the documents to include audio and video.
  - For the gov't: a special version that allows parts of a document to be encrypted with a proprietary encryption algorithm that we can't ever see.

# Creeping Feature-itis

- Adding on new entity types one-by-one becomes tedious and repetitive.
- We eventually get bored, make mistakes, lose our job and our friends, our toaster oven is repossessed, and worst of all - we lose access to the Internet.
- There must be a better way!

# OOP to the Rescue

- View document entities as objects.
- Each entity could now:
  - Know how to render itself.
  - Provide information to document manager about special features
  - Know how to save/retrieve itself from a file.
- Using objects, we focus on each object separately and never get bored.

# Using the Network.

- Developing an OOP based multi-edit that could be extended easily by adding new entity objects is nothing new.
- What we really want is to be able to provide an application framework and publish the interface so that anyone could develop new entity objects, and our customers could incorporate these new objects (over the network).

# Potential Problems

- How do we make sure everyone using multi-edit has a version that knows about all the new entity objects (requires recompilation).
- What happens to our documents when somebody changes an entity object definition?

# More Problems/Limitations

- Inheritance can only be used if everyone has the same base classes.
- Every client would need to be capable of rebuilding itself.
- We end up needing a fancy source code management and distribution system, since every client would need to have everything.

# Super OOP to the Rescue

- A better solution might be to provide some way to build a system based on dynamic, distributed objects.
- Object definitions are not static, they can be established and referenced at run time.
- For each object we can provide a single implementation that acts as a server.

# The General Idea

- Our document can contain any kind of entity as long as somewhere there is a server that can provide all the functionality of the entity object.
- In this case the *server* could be a DLL or a remote procedure call(s).

# CORBA

- The notion of having objects distributed across the network has been around for a while.
- The Object Management Group (OMG) was formed in 1989 to create a set of standards that would facilitate the development of distributed object-oriented applications.

# Buzzword Warning

- CORBA is a rich source of acronyms and buzzwords.
- OMG is now the largest standards body that has ever existed (on this planet).
- First buzzword: *Middleware - software that hides the details of network programming from programmers, so they can worry about the application.* CORBA is middleware.

# Important message from our sponsor

IDL does not provide a complete definition of OMA, nor does it facilitate the use of DII in conjunction with an ORB. Only with the aid of a BOA or alternative OAs as described in the RFPs and RFIs will it be possible to make use of any IIOP compliant system.

In other words: “my toaster oven was just repossessed by the Budwiser™ Frogs”.

# Object Management Group

- OMG creates specifications, not implementations.
- Some Key Specifications:
  - OMA: Object Management Architecture.
  - CORBA: Common Object Request Broker Architecture.

# OMA Object Model

- Objects provide services.
- Clients makes a request to an object for a service.
- Client doesn't need to know where the object is, or anything about how the object is implemented!
- Object interface must be known (public) - provides signature for each object method.

# Object References

- Clients don't *have* objects, they just have object references.
- Object references can be persistent (saved for use later).

# Accessing Remote Methods

- Clients can call remote methods in 2 ways:
  - Static Invocation: using stubs built at compile time (just like with RPC).
  - Dynamic Invocation: actual method call is created on the fly. It is possible for a client to discover new objects at run time and access the object methods.

# Interface Definition Language

- IDL is the language used to describe object interfaces, the same basic idea as a protocol definition file for RPC.
- IDL is a declarative language, it only describes object interfaces.
- IDL is language neutral - there are mappings for many object oriented languages (C++, Smalltalk, Java).

# Inheritance

- IDL supports interface inheritance
  - all operations are effectively virtual.
- C++ programmers can get confused
  - C++ supports implementation inheritance
  - IDL doesn't say anything about implementation!

# Interface Repository

- An IR provides persistent storage of IDL interface declarations.
- IR serves 2 purposes:
  - tool for programmers. Basically a database of object interfaces and inheritance hierarchy.
  - Support dynamic invocation interface (DII).

# Object Adapters

- Object Adapters provide a layer between object method requests and the servers that service the requests. Functions include:
  - generation of object references
  - starting up the actual server program(s)
  - handling security

# Basic Object Adapter

- Simplest Object Adapter, can support a number of different implementations:
  - one server that always is running
  - one program that can handle requests for multiple objects.
  - one program per object implementation.
  - one program for each object method.

# Object Request Broker

- The ORB is an abstract entity that acts as the middleman in all remote method invocations.
- The ORB finds a server that can handle a method invocation, passes the request to the server, receives the response and forwards it to the client.
- The functions handled by an ORB are actually implemented in both client and server.

# Some (old) OMG propoganda