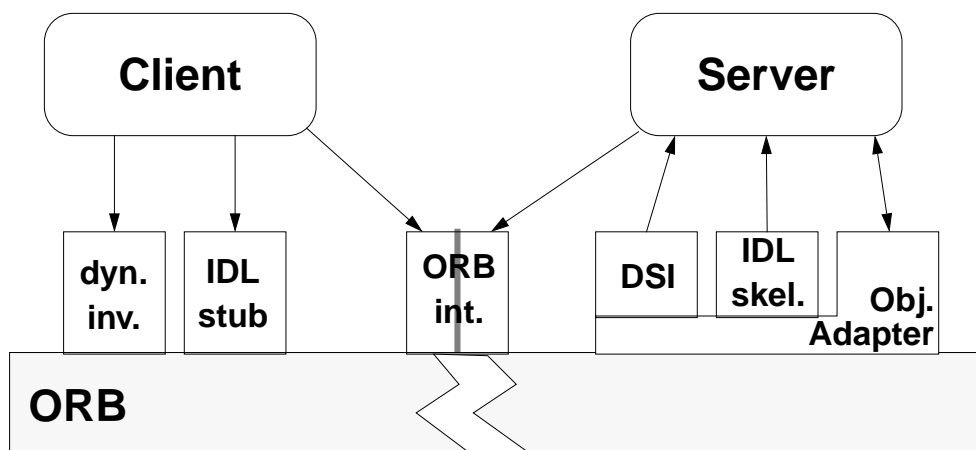


Review — What is CORBA?

- **Common *Object Request Broker* Architecture**
 - developed by OMG
 - supports distributed object computing
 - distributed computing + OO computing
- It uses a *broker*
 - an intermediary handling requests in a system
 - separates a component's interface from its implementation
 - flexible system composition and evolution
- Mutable client-server relationships
 - clients and servers not hard-wired to one another
 - ORBs are interaction intermediaries
- Supports both synchronous and deferred synchronous communication

Review — CORBA Architecture



Review — Object Request Broker

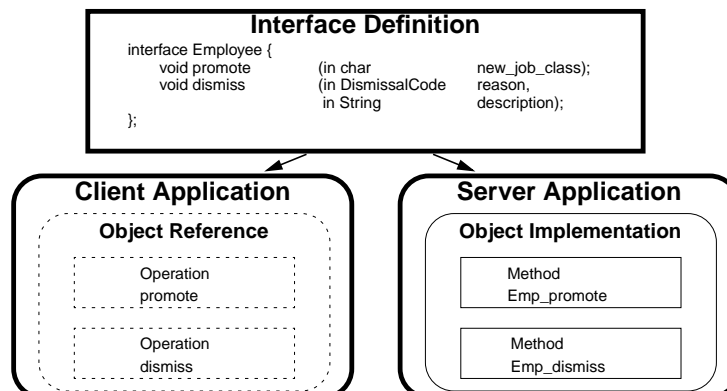
- Performs delivery of client requests and server responses
 - provides a layer of transparency between clients and servers
- The ORB hides object, location, implementation, execution state, and communication mechanisms

Review — OMG IDL

- Specifies (implementation-independent) object interface
- Supports
 - built-in types
 - constructed types
 - template types

Review — Stubs and Skeletons

- Used in CORBA's static invocation
- PL-specific counterparts to IDL definitions
- **Stubs** create and issue requests on the client side
- **Skeletons** receive and forward requests to objects on the server side



Review — Interface Repository

- Used for accessing objects whose interface is not known at compile time
 - knowing interfaces of all objects *a priori* may be impractical
 - IR allows access to the IDL type system at runtime
 - IR is a CORBA object, invoked like any other object
 - supports CORBA's dynamic invocation

Review — Dynamic Invocation and Dispatch

- Dynamic invocation interface (DII) — generic stub
 - dynamic requests on the client side
- Dynamic skeleton interface (DSI) — generic skeleton
 - dynamic request dispatch to objects on the server side

Review — Object Adapters

- Serve as the glue between an object and the ORB
 - no need for direct attachments between objects and ORBs
- OA responsibilities
 - object registration, activation, reference generation, upcalls
 - server process activation
 - request demultiplexing

Review — Inter-ORB Protocols

- General ORB interoperability architecture
 - based on the General Inter-ORB Protocol (GIOP)
 - Environment-Specific Inter-ORB Protocols (ESIOP)
- Standard object reference format
 - Interoperable Object Reference (IOR)

Review — CORBA Deficiencies

- Does not exploit known operational regularities
 - complex marshalling even in a homogeneous case
 - no alternative transports (e.g., shared memory)
 - application footprint is large
- Deadlock detection not supported
- Garbage collection must be addressed
- Local vs. remote transparency is not always good
 - remoteness cannot be hidden w.r.t. time and reliability
 - programs are forced to treat all objects as remote
- CORBA's object naming assumptions are not scalable
 - all object names are kept in a global namespace
 - all objects are represented in clients via object references
 - problem for very large object-based applications

Microsoft's Component Object Model

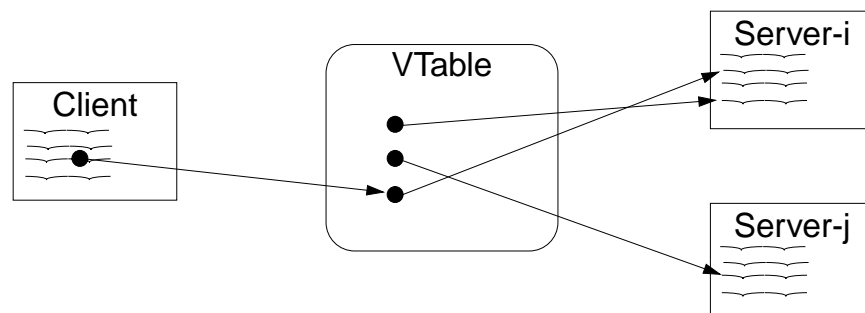
- Middleware infrastructure
 - in many ways similar to CORBA
- Defines a binary standard for component interoperability
 - PL independence
- Platform independent
 - Windows (95, 98, NT)
 - Mac
 - Unix
- Distribution transparency
 - does exploit operational characteristics
- Dynamic component loading and unloading

Basic COM Features

- Binary standard for component interactions
- Support for interfaces
 - defined in an IDL different from CORBA's
- Base interface with introspection support
 - dynamic discovery of other components' interfaces
 - garbage collection via reference counting
- Unique component ID mechanism
- Communication is synchronous by default
 - asynchronous communication supportable by callbacks and connection points

Binary Standard

- Component operation invocation via virtual tables — *vtables*
 - works for languages that support function pointers
 - e.g., C, C++, Smalltalk
 - the client contains a pointer to the vtable
 - the vtable contains a pointer to the server function
 - one layer of indirection over standard function calls



COM Components

- Compiled code that provides some service to a system
- A component may support a number of interfaces
 - an interface is a collection of semantically related functions
 - COM interfaces begin with “I” by convention
- All access to component services is via interface pointers
 - allows service reimplementations
- Each component supports base interface *IUnknown*
- Transparent remote access via proxy-stub pairs
 - similar to CORBA
- Every component has a globally unique identifier (GUID)
 - 128 bit integer
 - used to refer to component's TypeLibrary (metadata repository)

Notes on COM Interfaces

- Interface \neq class
 - they contain no implementation
 - multiple implementations of an interface possible
- Interface \neq component
 - interfaces are the (binary) component interaction standard
- Heterogeneous COM clients and servers interact via interface pointers
- A single COM component can implement multiple interfaces
- Interfaces are strongly typed
 - every interface has a GUID
- Interfaces are immutable
 - e.g., no subtyping, subclassing, inheritance

Interface *IUnknown*

- Must be implemented by all COM components
- Has three methods
 - QueryInterface
 - > provides introspection capabilities
 - > allows runtime discovery of component interface
 - > delivers interface pointer to a client
 - AddRef
 - > called when another component is using the interface
 - > increments reference count
 - Release
 - > called when another component stops using the interface
 - > decrements reference count
- Supports garbage collection
 - a component can be unloaded when its reference count is 0

Benefits of COM Components & Interfaces

- Shared with CORBA
 - PL independence
 - evolvable functionality
 - > interface reimplementations or extensions
 - interface reuse
 - local/remote transparency
 - > remote handled by DCOM via RPC
- Unique
 - low overhead of object interaction
 - > negligible for in-process communication
 - > cross-process and -network is handled by DCOM

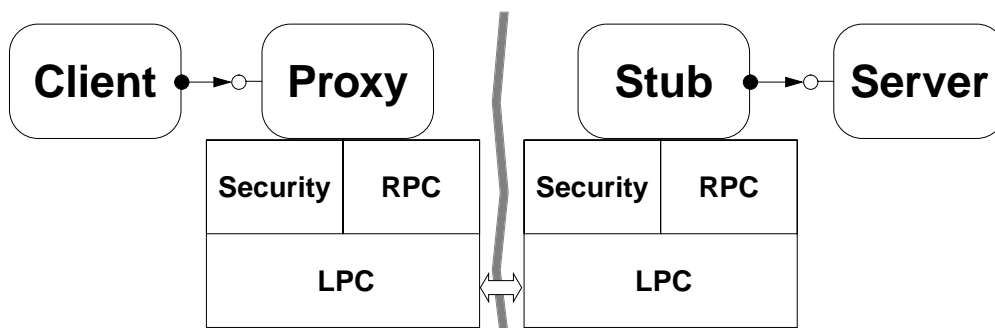
Distributed COM

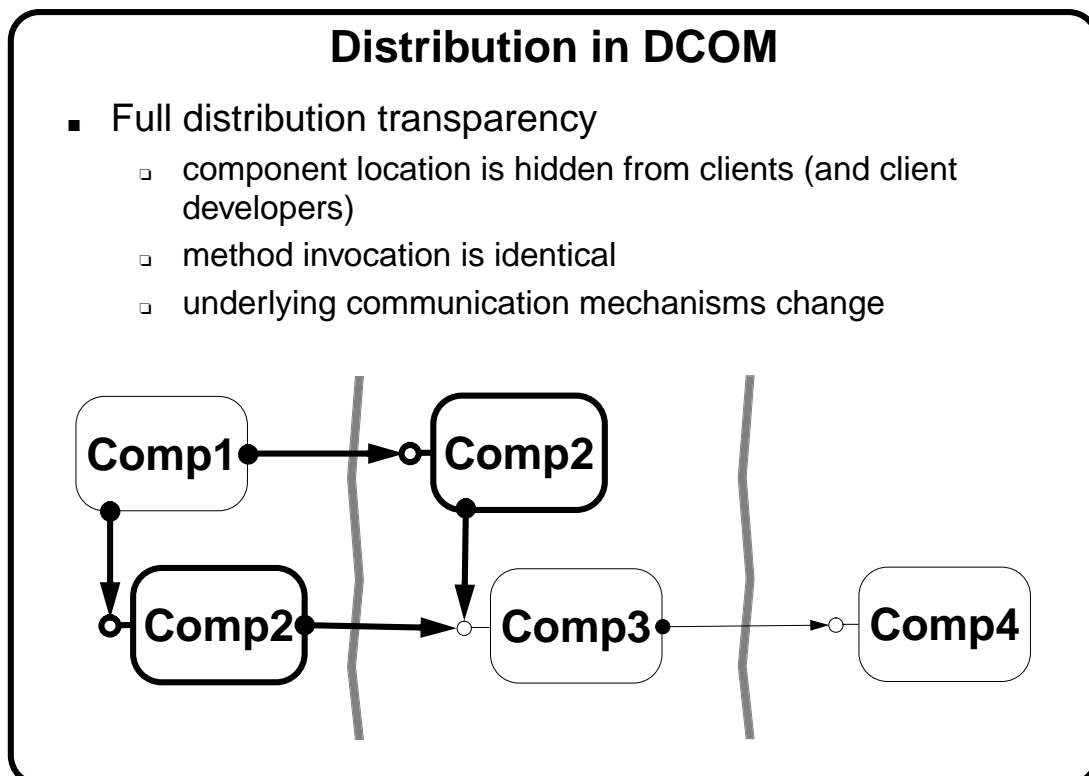
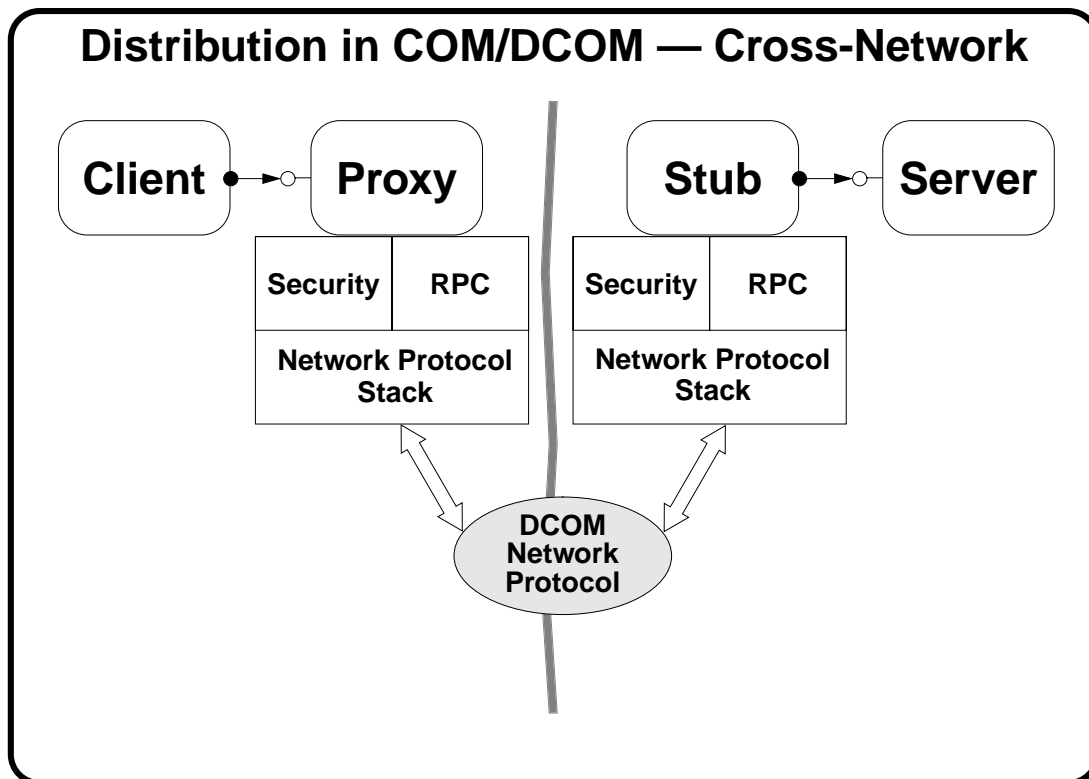
- DCOM = COM binary standard
 - + runtime infrastructure for communicating across distributed address spaces
 - initially only on Windows
 - recently adding Mac and Unix
- Uses OSF's DCE RPC as a basis for remote interaction
 - proxy/stub mechanism
- Attempts to address challenges of distributed computing
 - interacting components should be "close" to one another
 - some components' locations are fixed
 - inverse relationship between component size and flexibility
 - direct relationship between component size and network traffic

Distribution in COM/DCOM — In-Process



Distribution in COM/DCOM — Inter-Process





Garbage Collection in COM/DCOM

- Networks are fragile
 - connections may break for many reasons
 - if a connection to a client is broken, a server component should not needlessly consume resources
- COM/DCOM uses a pinging protocol to detect (in)active clients
 - a ping message is sent every two minutes from client to server
 - distributed garbage collection
 - transparent to the application (developer)
 - reference count is decremented if multiple (>3) ping periods pass without receiving a message
- The protocol is efficient
 - ping messages are piggybacked onto existing COM calls

Performance and Scalability in DCOM

- Adjust expectations with the size of the application
 - small applications => faster performance
 - achieved via vtables
 - different from CORBA
- Performance is addressed by adjusting component deployment to processors
 - components are redeployed dynamically
- Scalability is addressed by versioned interfaces
 - no subclassing
 - dynamic interface query
 - newly added functionality does not affect old clients/servers

Performance by the Numbers

Configuration	4 bytes calls/sec*	4 bytes ms/call	50 bytes calls/sec*	50 bytes ms/call
Pentium (in-process)	3,200,000	0.0003	3,300,000	0.0003
DEC Alpha (in-process)	2,800,000	0.00035	2,800,000	0.00035
Pentium (inter-process)	2,400	0.4	2,000	0.5
DEC Alpha (inter-process)	1,900	0.5	1,600	0.6
Alpha to Pentium (cross-network)	375	2.7	300	3.3

* 2,000 calls/sec satisfies most performance requirements