

# Sections 9.1 & 9.2

## Corba & DCOM

John P. Daigle

Department of Computer Science  
Georgia State University

05.16.06

# Outline

- 1 Introduction
- 2 CORBA
  - Overview
  - Communication
  - Processes
  - Naming
  - Other Design Concerns
- 3 Distributed COM
  - Overview
  - Naming
  - Fault Tolerance & Security

# Outline

- 1 Introduction
- 2 CORBA
  - Overview
  - Communication
  - Processes
  - Naming
  - Other Design Concerns
- 3 Distributed COM
  - Overview
  - Naming
  - Fault Tolerance & Security

# CORBA and DCOM

## CORBA

- CORBA stands for Common Object Request Broker Architecture.
- Specification Developed by the Object Management Group (OMG Authors)
- The essence of CORBA is “separation of interface from implementation” (from OMG web site)

## DCOM

- Distributed Component Object Model
- Microsoft Technology

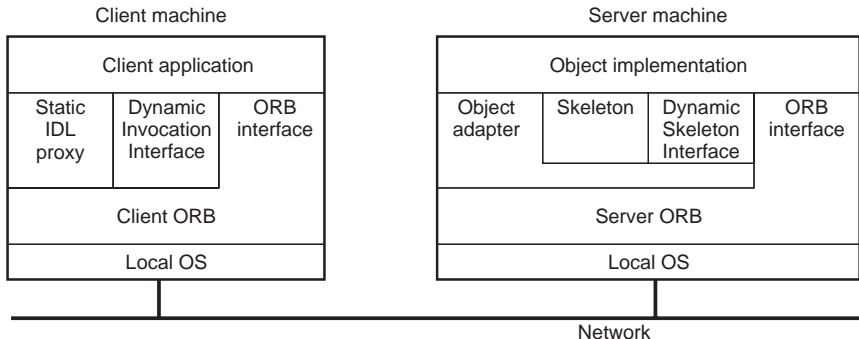
# What are these things?

CORBA and DCOM are  
Distributed Object Based  
Systems

# Outline

- 1 Introduction
- 2 CORBA
  - Overview
  - Communication
  - Processes
  - Naming
  - Other Design Concerns
- 3 Distributed COM
  - Overview
  - Naming
  - Fault Tolerance & Security

# Object Model



**ORB** Object Request Broker: the Core of a CORBA system, enables communication between objects and their clients.

**IDL** Interface Definition Language provides a precise syntax for describing methods and parameters, static

**Dynamic Invocation Interface** Runtime tool for clients to construct an invocation request at runtime.

# Interface, services

## Interface & Implementation Repository

**Interface Repository** Stores all interface definitions

- IDL compiler assigns a **repository identifier** to each interface on compilation
- Standardized to IDL syntax

**Implementation Repository** Stores all information needed to implement objects. Closely coupled to OS and ORB (which is non standard)

## CORBA Services

Conceptually, CORBA services are the “operating system” of the CORBA distributed platform.

- Services are general purpose
- Services are independent of application

# List of CORBA services (p. 500)

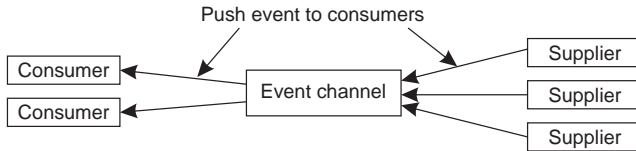
<b>Service</b>	<b>Description</b>
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide naming of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services an object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

# Object Invocation Models

How does the client connect to an object?

<b>Request Type</b>	<b>Failure Semantics</b>	<b>Description</b>
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred Synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

# Event & Notification Services



## Push Model

Objects wait passively to be informed when an event occurs.

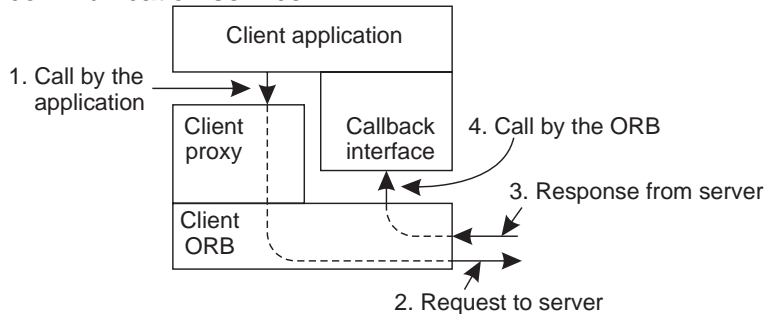


## Pull Model

Objects poll the event channel to check for events. Both models are inherently unreliable, and neither allows filtering of messages.

# Messaging Service

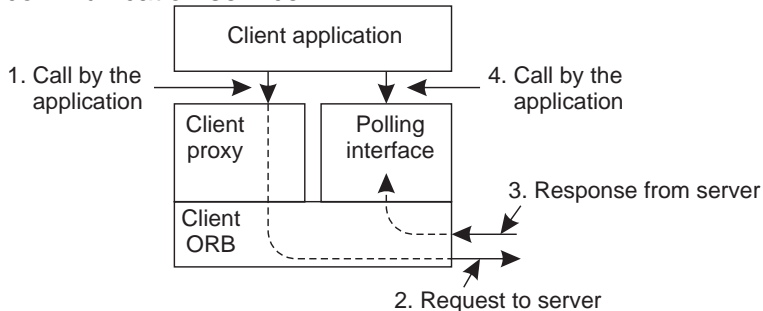
CORBA also provides a more reliable communication method for persistent communication. This is an object oriented communication service.



Callback model: application driven, application supported

# Messaging Service

CORBA also provides a more reliable communication method for persistent communication. This is an object oriented communication service.



Polling Model: application driven, ORB supported

# Interoperability

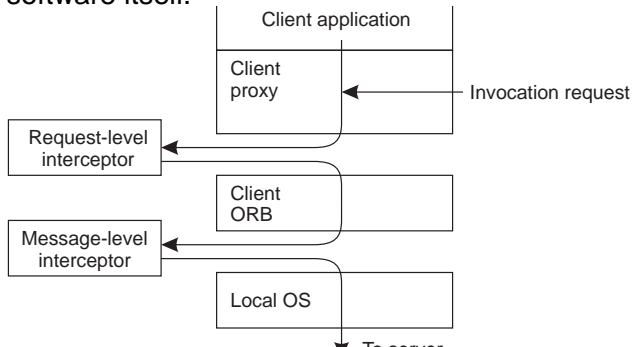
## GIOP

There is a need in CORBA for different ORB's on different platforms to be able to communicate.

**GIOP** General Inter-ORB protocol: must run on top of an existing transport protocol, such as TCIP.

# Clients

One unique aspect of client software in CORBA is the idea of an *interceptor*, which allows client side software to be modified as needed by modifying the input or output, rather than the software itself.



# Object References

**Important:** In CORBA, it is essential to distinguish specification-level and implementation-level (object references

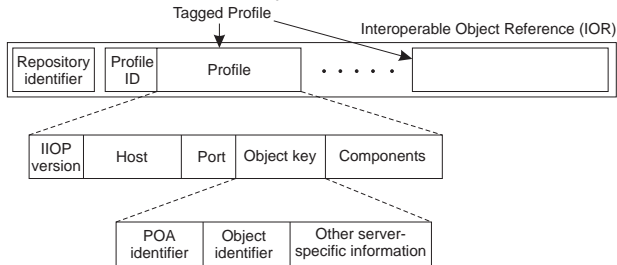
**Specification level:** An object reference is considered to be the same as a proxy for the referenced object → having an object reference means you can directly invoke methods; there is no separate client-to-object binding phase

**Implementation level:** When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client's address space

**Conclusion:** Object references in CORBA used to be highly **implementation dependent**: different implementations of CORBA could normally not exchange their references.

# Interoperable Object References

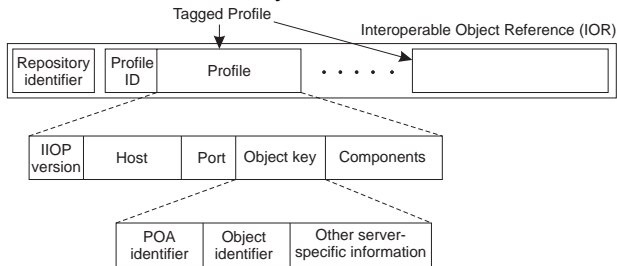
Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.



*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

# Interoperable Object References

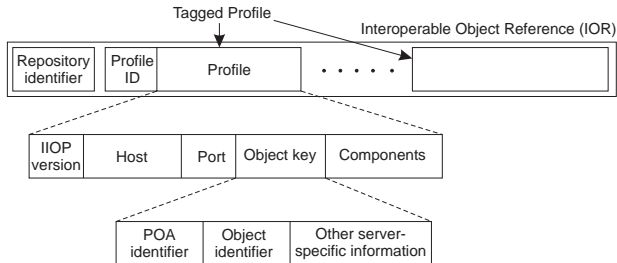
Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.



*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

# Interoperable Object References

Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.

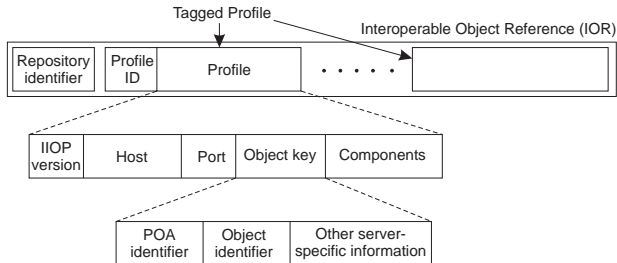


*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

*IIOP version* The version Number for this profile

# Interoperable Object References

Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.

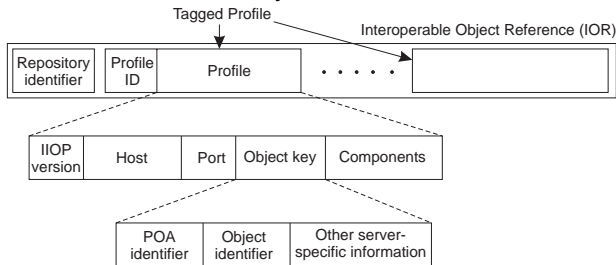


*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

*Host* Location of this object (for example, an IP or DNS)

# Interoperable Object References

Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.

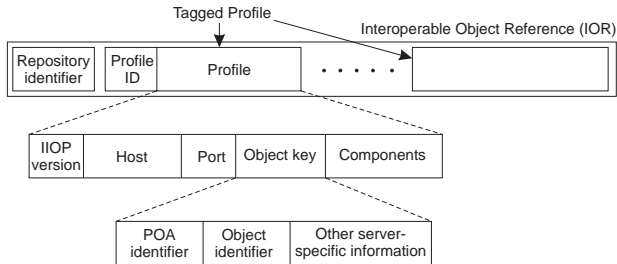


*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

*Object Key* Server specific information for demultiplexing incoming requests to the appropriate object.

# Interoperable Object References

Because object references are implementation dependent, CORBA requires a way to package object references to allow them to be universally understood.



*ProfileID Field* The part of the **tagged profile** that contains the complete information needed to invoke the object.

*Components* Additional information such as security information.

# The CORBA Naming Service

**Essence:** CORBA's naming service allows servers to associate a name to an object reference, and have clients subsequently bind to that object by resolving its name

**Observation:** In most CORBA implementations, object references denote servers at specific hosts; naming makes it easier to relocate objects

**Observation:** In the naming graph all nodes are objects; there are no restrictions to binding names to objects: CORBA allows arbitrary naming graphs

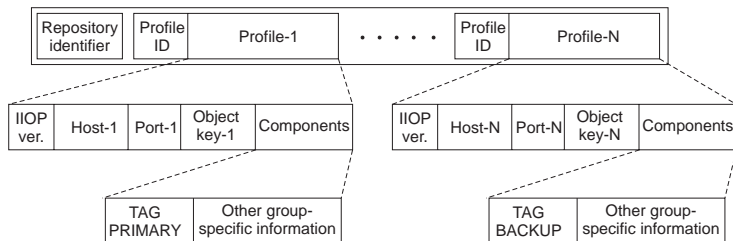
**Observation:** There is no single root; an initial context node is returned through a special call to the ORB. Also: the naming service can operate *across* different ORBs

# Fault Tolerance

Remember that replication is the key to fault tolerance.

## Interoperable Object Group Reference

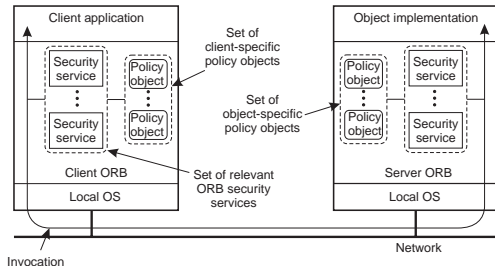
Interoperable Object Group Reference (IOGR)



**Note:** IOGRs have the same structure as IORs; the main difference is that they are *used* differently. In IORs an additional profile is used as an alternative; in IOGR, it denotes another replica.

# Security

**Essence:** Allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time; the ORB does the rest. Specific policies are passed to the ORB as (local) objects and are invoked when necessary:



**Examples:** Type of message protection, lists of trusted parties.

# Outline

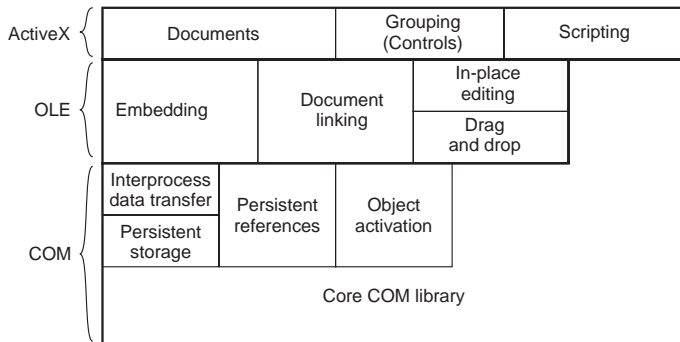
- 1 Introduction
- 2 CORBA
  - Overview
  - Communication
  - Processes
  - Naming
  - Other Design Concerns
- 3 Distributed COM
  - Overview
  - Naming
  - Fault Tolerance & Security

# Acronyms

**COM** Component Object Model

**OLE** Object Linking & Embedding

**Active X** Everything that isn't COM



# Object Model

- Interfaces are Binary, essentially, a DCOM interface is a collection of pointers to implementations of methods.
- Each interface is typed, and therefore has a globally unique **interface identifier**
- A client always requests an implementation of an interface:
  - Locate a class that implements the interface
  - Instantiate that class, i.e., create an object
  - Throw the object away when the client is done
- The DCOM equivalent to the Interface Repository is the *type library*

# DCOM Services

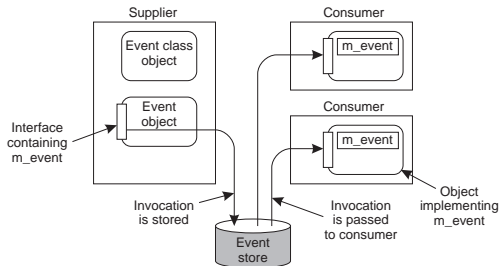
CORBA	DCOM/COM+	Windows 2000
Collection	ActiveX Data Objects	–
Query	None	–
Concurrency	Thread concurrency	–
Transaction	COM+ Automatic Transactions	Distributed Transaction Coordinator
Event	COM+ Events	–
Notification	COM+ Events	–
Externalization	Marshaling utilities	–
Life cycle	Class factories, JIT activation	–
Licensing	Special class factories	–
Naming	Monikers	Active Directory
Property	None	Active Directory
Trading	None	Active Directory
Persistence	Structured storage	Database access
Relationship	None	Database access
Security	Authorization	SSL, Kerberos
Time	None	None

**Note:** COM+ is effectively COM plus services that were previously available in an ad-hoc fashion

# Communication

**Object invocations:** Synchronous remote-method calls with at-most-once semantics. Asynchronous invocations are supported through a polling model, as in CORBA.

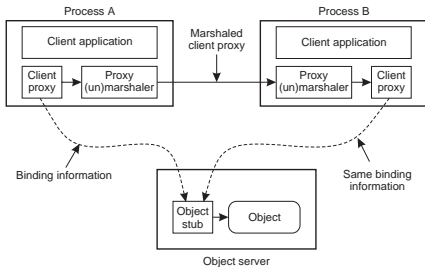
**Event communication:** Similar to CORBA's push-style model:



**Messaging:** Completely analogous to CORBA messaging.

# Processes

- Client side processes in DCOM are generally handled as if all objects are running locally.
- Passing object references between machines requires passing unique ID information, location, and other binding information. The wrapping is done by the proxy:



# Monikers

## DCOM Objects are Temporary

To accommodate objects that can outlive their client, something else is needed. A Moniker is a persistent reference, a hack to support real objects

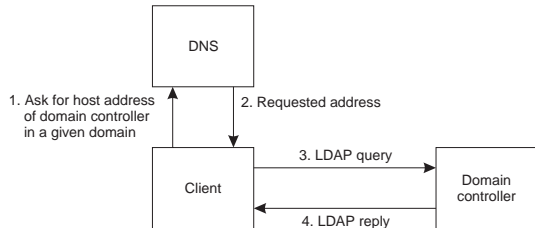
- A moniker associates data (e.g., a file), with an application or program
- Monikers can be stored
- A moniker can contain a **binding protocol**, specifying how the associated program should be “launched” with respect to the data.

1	Client	Calls BindMoniker at moniker
2	Moniker	Lookup CLSID and tell SCM to create object
3	SCM	Loads class object
4	Class object	Creates object, returns int. pointer
5	Moniker	Instructs object to load previously stored state
6	Object	Loads its state from file
7	Moniker	Returns interface pointer of object to client

# Active Directory

A worldwide distributed directory service, but one that does not provide location transparency.

**Basics:** Associate a directory service (called **domain controller**) with each domain; look up the controller using a normal DNS query:



Domain controllers can be grouped into trees and forests, and indexed at that level.

# Fault Tolerance

**Automatic transactions:** Each class object (from which objects are created), has a transaction attribute that determines how its objects behave as part of a transaction:

Attr. value	Description
REQUIRES_NEW	A new transaction is always started at each invocation
REQUIRED	A new transaction is started if not already done so
SUPPORTED	Join a transaction only if caller is already part of one
NOT_SUPPORTED	Never join a transaction (no transaction support)
DISABLED	Never join a transaction, even if told to do so

**Note:** Transactions are essentially executed at the level of a method invocation.

# Declarative Security I

DCOM Security is based on roles. The registry keeps an access control list for its entries, specifying the access privileges for each group.

Auth. level	Description
NONE	No authentication is required
CONNECT	Authenticate client when first connected to server
CALL	Authenticate client at each invocation
PACKET	Authenticate all data packets
PACKET_INTEGRITY	Authenticate data packets and do integrity check
PACKET_PRIVACY	Authenticate, integrity-check, and encrypt data packets

**Delegation:** A server can impersonate a client depending on a level:

Impersonation	Description
ANONYMOUS	The client is completely anonymous to the server
IDENTIFY	The server knows the client and can do access control checks
IMPERSONATE	The server can invoke local objects on behalf of the client
DELEGATE	The server can invoke remote objects on behalf of the client