Distributed Systems
Principles and Paradigms

Chapter 09
(version 27th November 2001)

Maarten van Steen

Vrije Universiteit Amsterdam, Faculty of Science
Dept. Mathematics and Computer Science
Room R4.20. Tel: (020) 444 7784
E-mail:steen@cs.vu.nl, URL: www.cs.vu.nl/∼steen/

01 Introduction
02 Communication
03 Processes
04 Naming
05 Synchronization
06 Consistency and Replication
07 Fault Tolerance
08 Security
09 Distributed Object-Based Systems
10 Distributed File Systems
11 Distributed Document-Based Systems
12 Distributed Coordination-Based Systems

Distributed Object-Based Systems

- CORBA
- DCOM
- Globe

CORBA

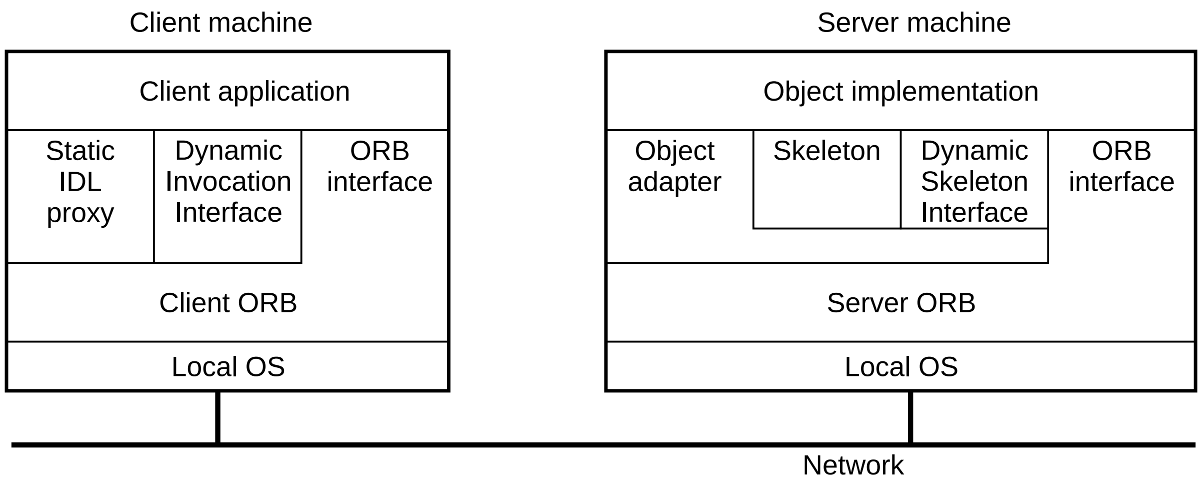
CORBA: Common **O**bject **R**equest **B**roker **A**rchitecture

Background:

- Developed by the **Object Management Group (OMG)** in response to industrial demands for object-based middleware
- Currently in version #2.4 with #3 (almost) done
- CORBA is a *specification*: different implementations of CORBA exist
- Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like

Essence: CORBA provides a simple distributed-object model, with specifications for many supporting services $\Rightarrow$ it may be here to stay (for a couple of years)

CORBA Overview (1/2)



Object Request Broker (ORB): CORBA's object broker that connects clients, objects, and services

Proxy/Skeleton: Precompiled code that takes care of (un)marshaling invocations and results

Dynamic Invocation/Skeleton Interface (DII/DSI): To allow clients to "construct" invocation requests at runtime instead of calling methods at a proxy, and having the server-side "reconstruct" those request into regular method invocations

Object adapter: Server-side code that handles incoming invocation requests.

CORBA Overview (2/2)

Interface repository: Database containing interface definitions and which can be queried at runtime

Implementation repository: Database containing the implementation (code, and possibly also state) of objects. Effectively: a server that can launch object servers.

CORBA Object Model

Essence: CORBA has a “traditional” remote-object model in which an object residing at an object server is remote accessible through proxies

Observation: All CORBA specifications are given by means of interface descriptions, expressed in an IDL. CORBA follows an interface-based approach to objects:

- Not the objects, but interfaces are the really important entities
- An object may implement one or more interfaces
- Interface descriptions can be stored in an interface repository, and looked up at runtime
- Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, Cobol, Ada, and Java.

CORBA Services

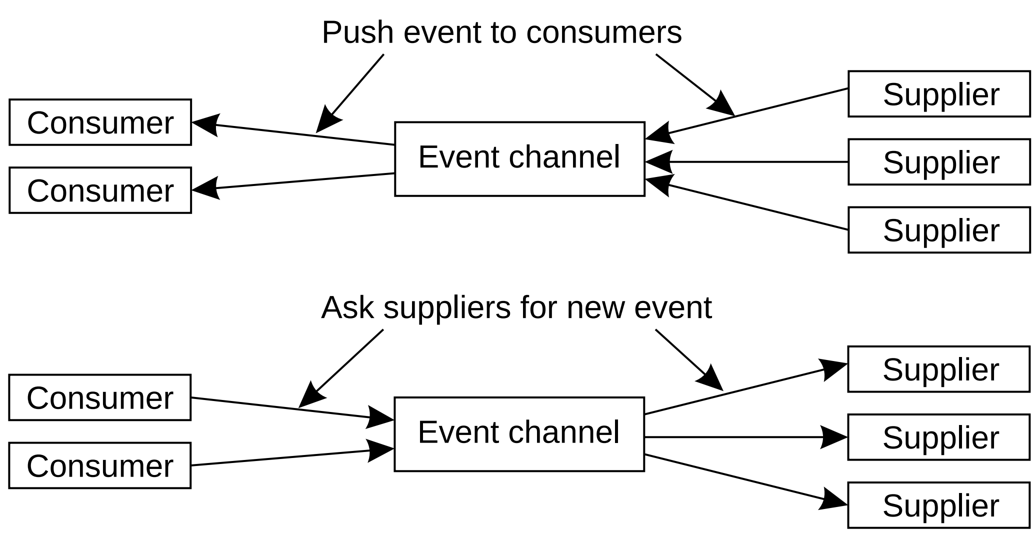
Service	**Description**
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over 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

Communication Models (1/2)

Object invocations: CORBA distinguishes three different forms of direct invocations:

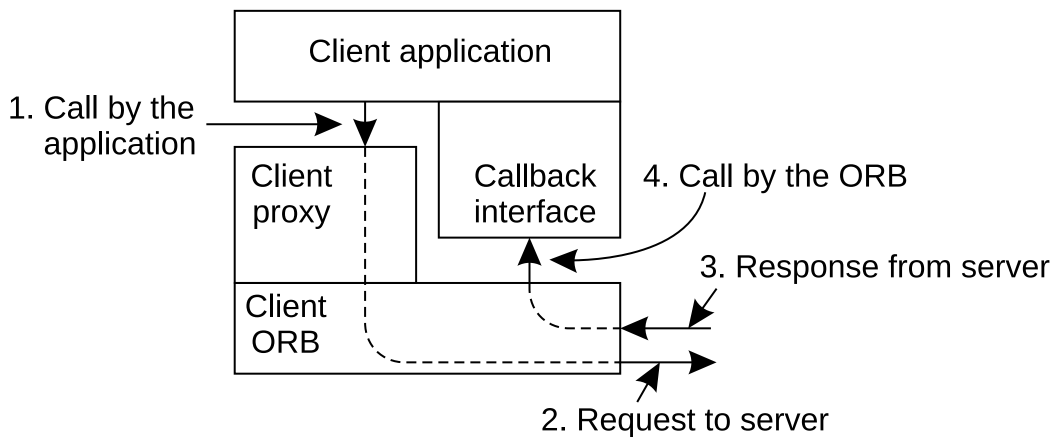
Request type	Failure sem.	Description
Synchronous	At-most-once	Caller blocks
One-way	Unreliable	Nonblocking call
Deferred synchronous	At-most-once	Nonblocking, but can pick-up results later

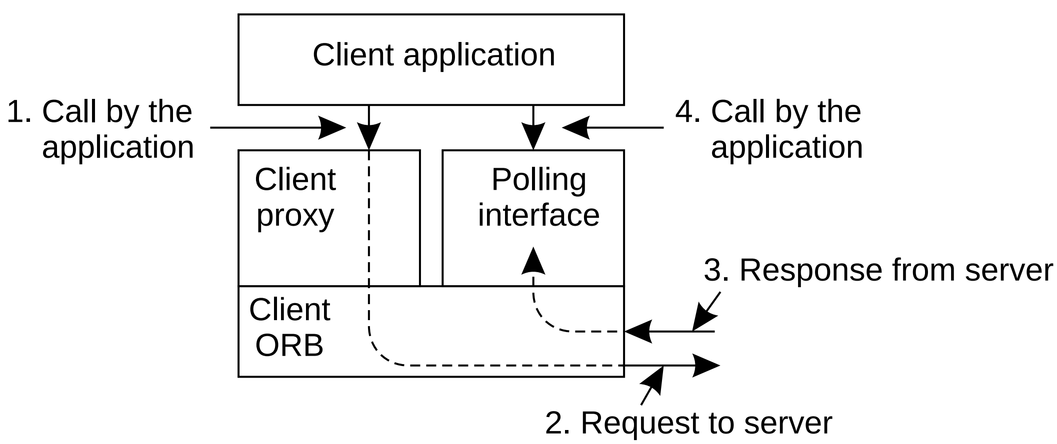
Event communication: There are also additional facilities by means of **event channels**:



Communication Models (2/2)

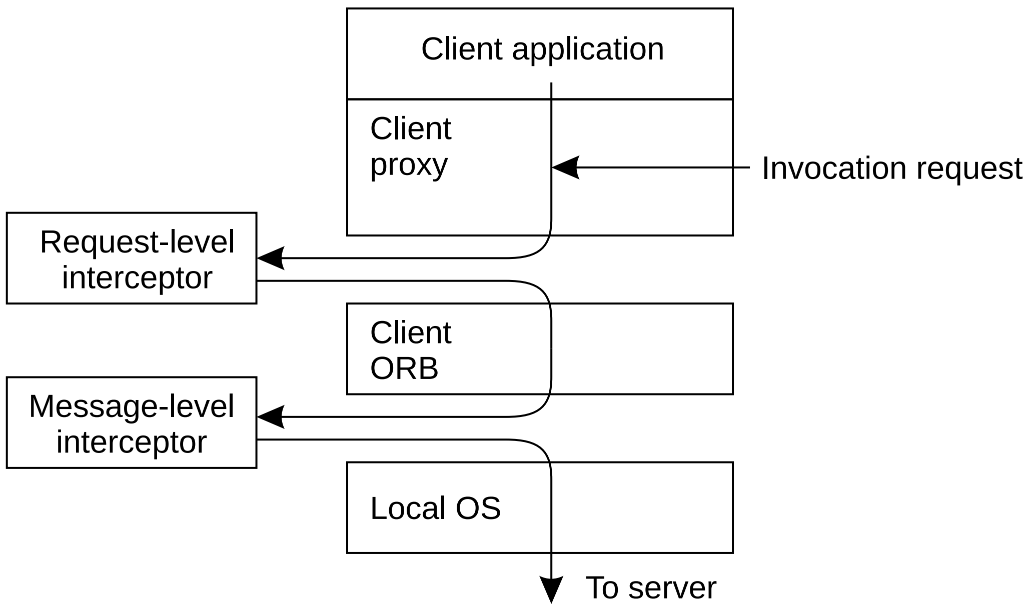
Messaging facilities: reliable asynchronous and persistent method invocations:





Processes

Most aspects of processes for in CORBA have been discussed in previous classes. What remains is the concept of **interceptors**:



Request-level: Allows you to modify invocation semantics (e.g., multicasting)

Message-level: Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)

Naming

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 $\Rightarrow$ having an object reference means you can directly invoke methods; there is no separate client-to-object binding phase

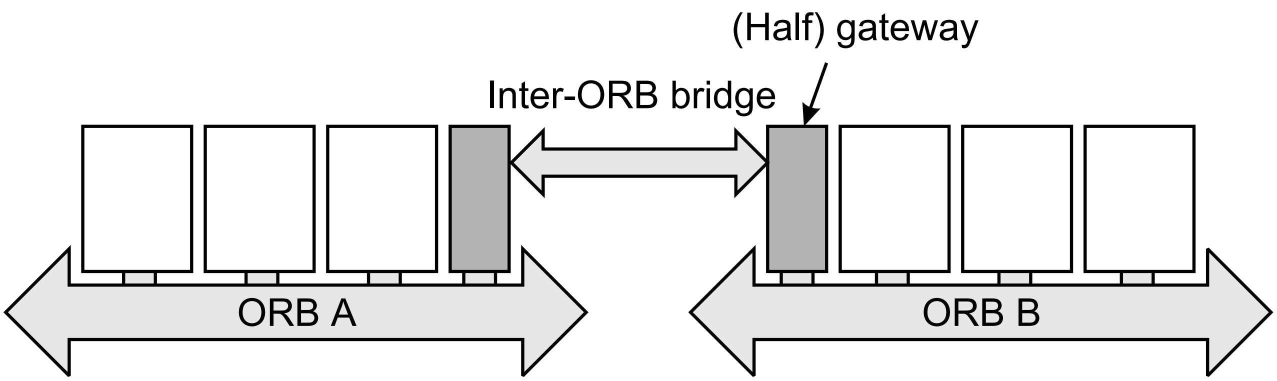
Implementation level: When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client's address space:

$\mathsf{ObjectReference\ objRef;}$
$\mathsf{objRef} = \mathsf{bindTo}(\textit{object O in server S at host H});$

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

Interoperable Object References (1/2)

Observation: Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries

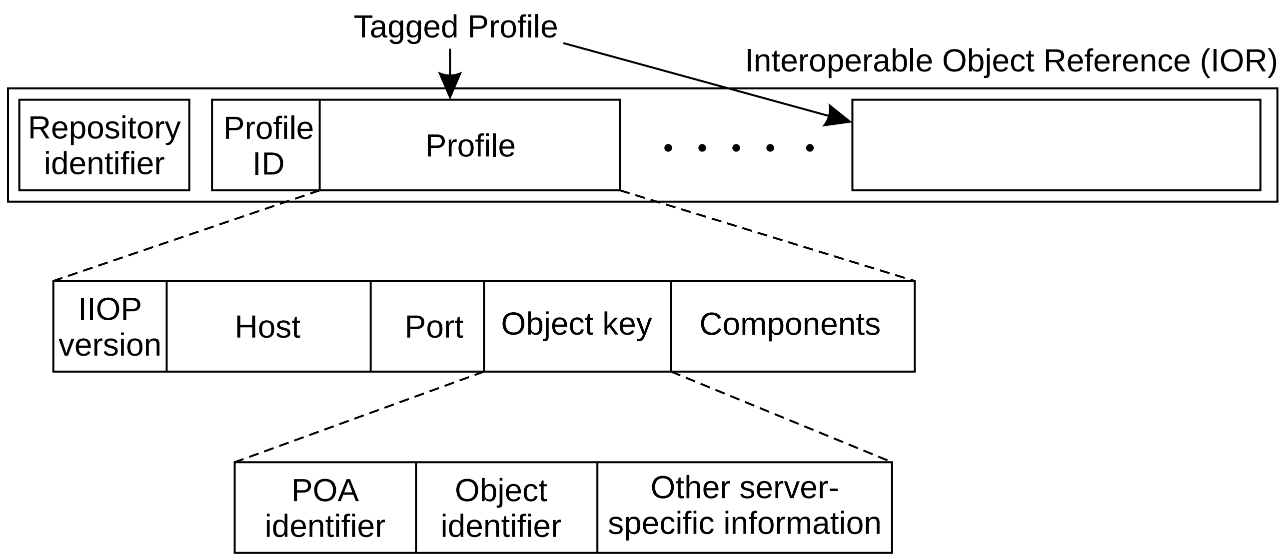


Solution: Object references passed from one ORB to another are transformed by the bridge through which they pass (different transformation schemes can be implemented)

Observation: Passing an object reference refA from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn't understand refA

Interoperable Object References (2/2)

Observation: To allow all kinds of *different* systems to communicate, we standardize the reference that is passed between bridges:



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

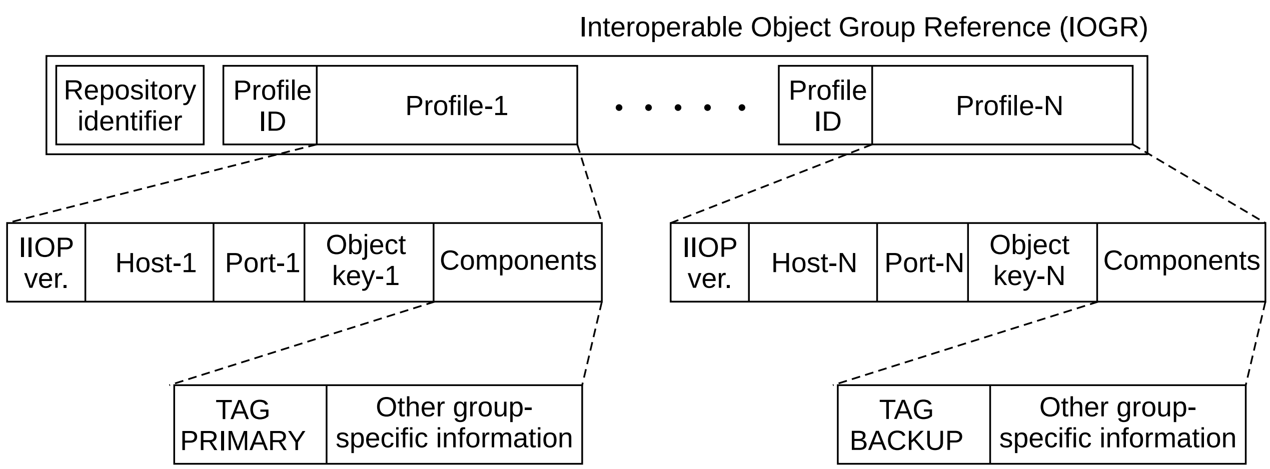
Observation: In the naming graph all nodes are objects; there are no restrictions to binding names to objects $\Rightarrow$ CORBA allows arbitrary naming graphs

Question: How do you imagine cyclic name resolution stops?

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 $\Rightarrow$ **interoperable naming service**

Fault Tolerance

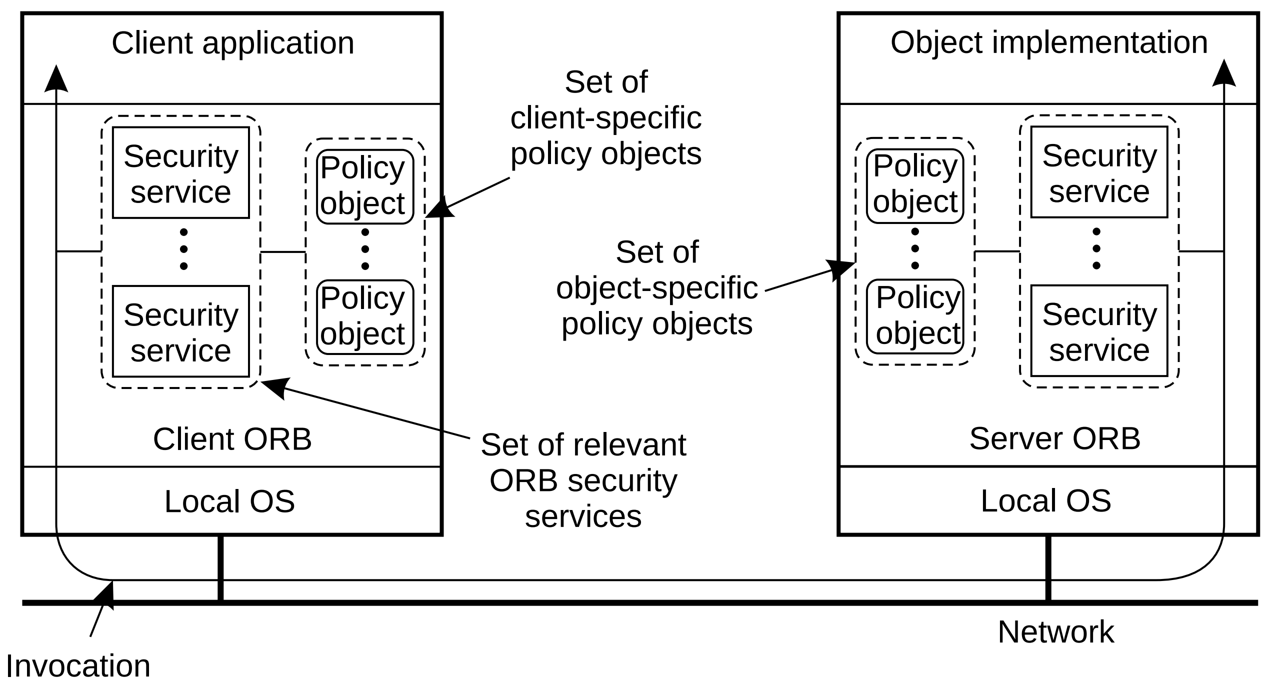
Essence: Mask failures through replication, by putting objects into **object groups**. Object groups are transparent to clients: they appear as “normal” objects. This approach requires a separate type of object reference: **Interoperable Object Group Reference**:



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.

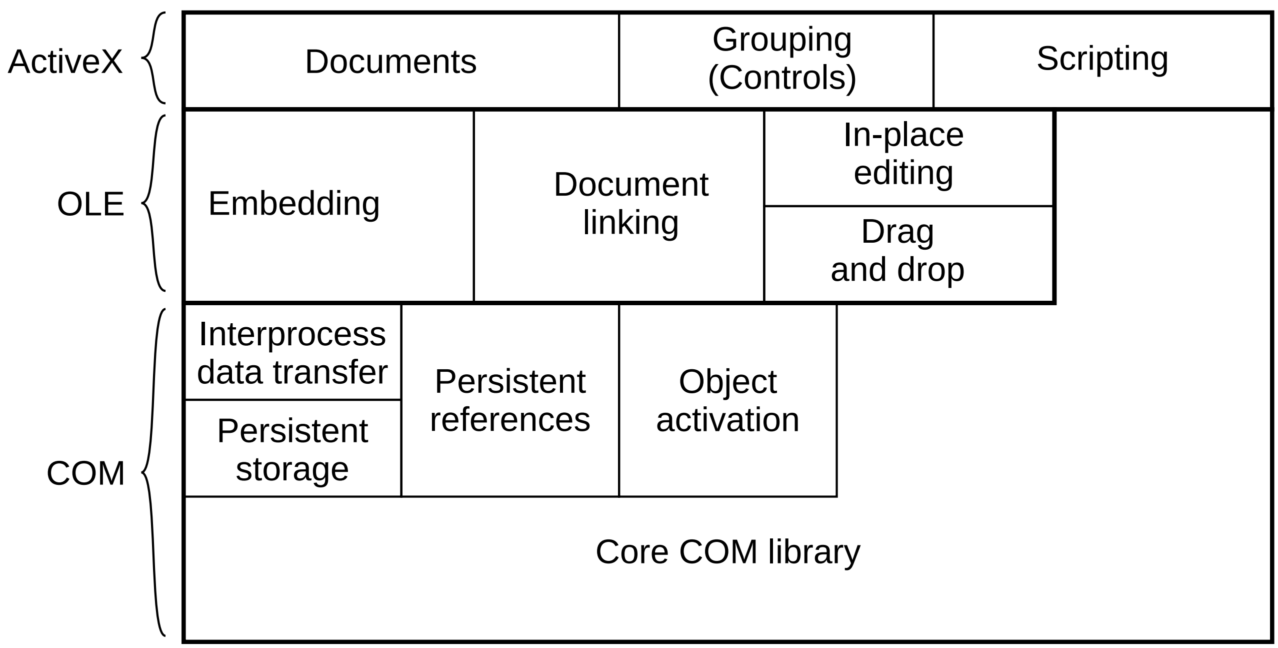
Distributed COM

DCOM: Distributed Component Object Model

- Microsoft's solution to establishing inter-process communication, possibly across machine boundaries.
- Supports a primitive notion of distributed objects
- Evolved from early Windows versions to current NT-based systems (including Windows 2000)
- Comparable to CORBA's object request broker

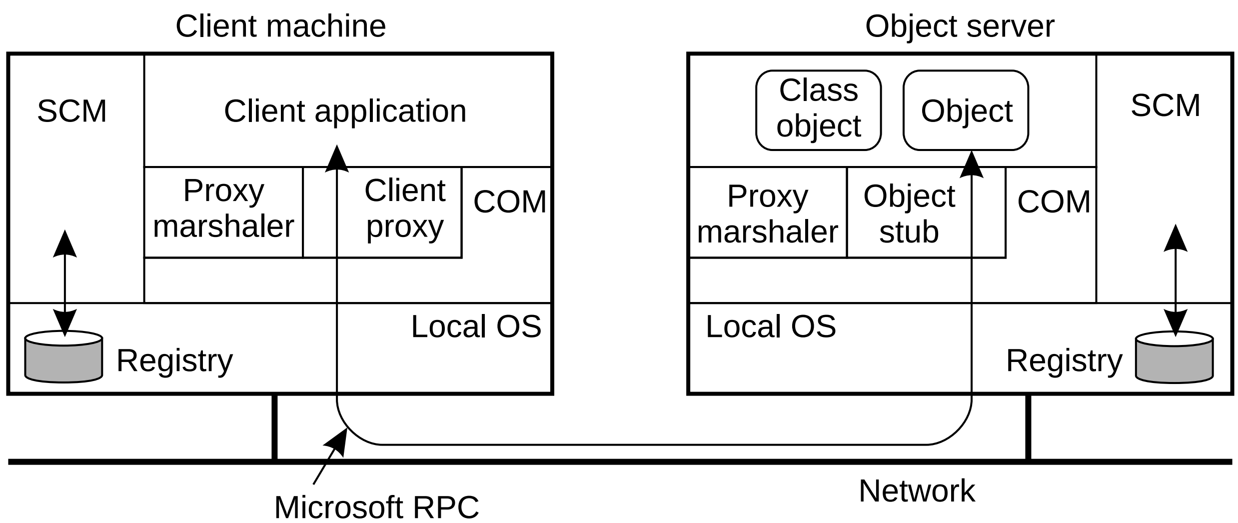
DCOM Overview (1/2)

Somewhat confused? DCOM is related to many things that have been introduced by Microsoft in the past couple of years:



DCOM: Adds facilities to communicate across process and machine boundaries.

DCOM Overview (2/2)



SCM: Service Control Manager, responsible for activating objects (cf., to CORBA's implementation repository).

Proxy marshaler: handles the way that object references are passed between different machines

COM Object Model

- An interface is a collection of semantically related operations
- 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

DCOM Services

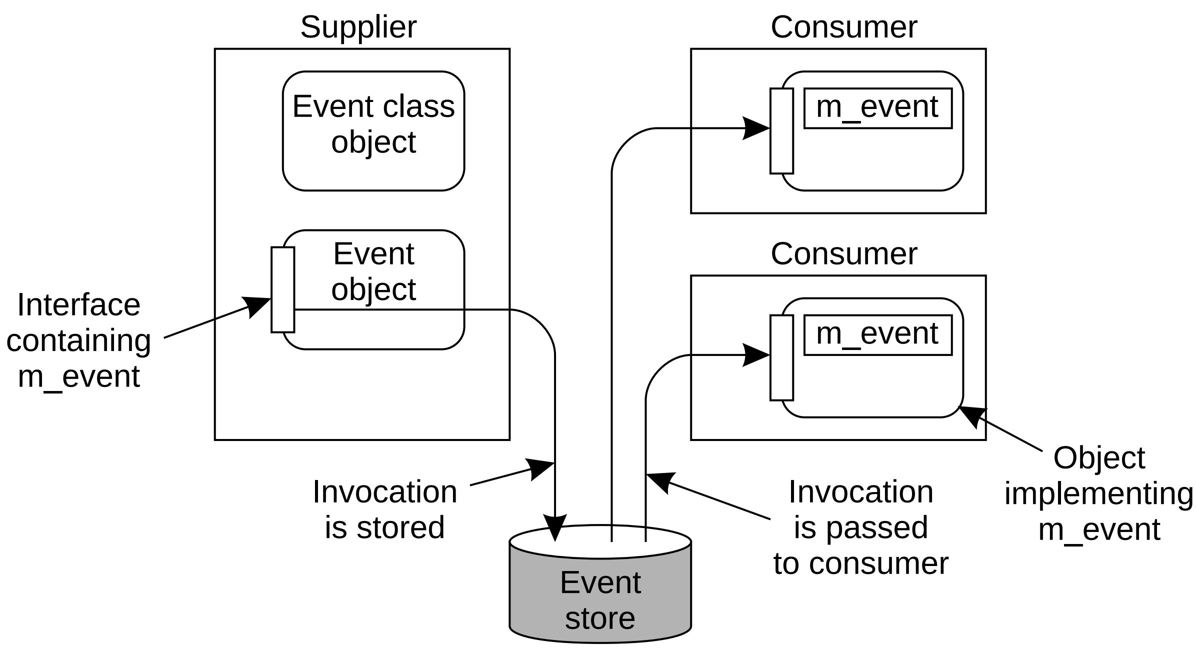
CORBA	**DCOM/COM+**	**Windows 2000**
Collection	ActiveX Data Objects	–
Query	None	–
Concurrency	Thread concurrency	–
Transaction	COM+ Automatic Transactions	Distributed Transaction Coordinator
Event	COM+ Events	–
Notification	COM+ Events	–
Externalization	Marshaling utilities	–
Life cycle	Class factories, 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 Models

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

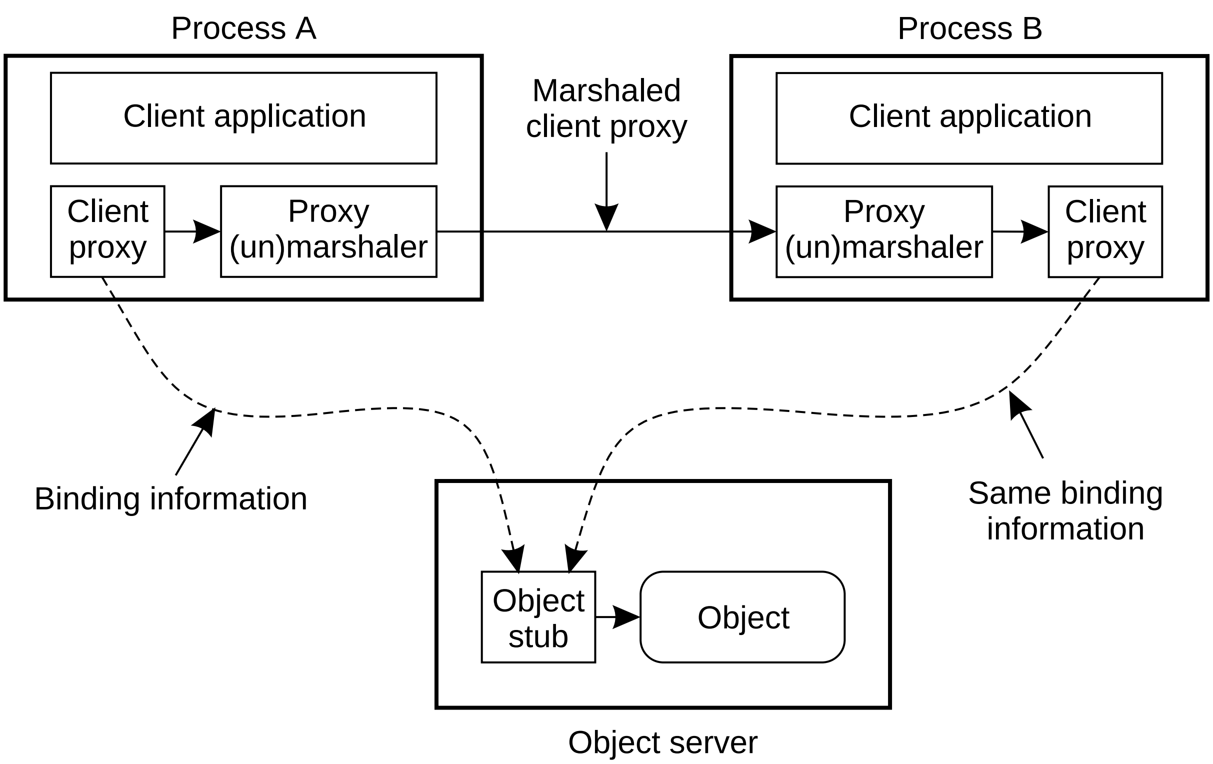
Event communication: Similar to CORBA's push-style model:



Messaging: Completely analogous to CORBA messaging.

Processes: Passing Object References

Observation: Objects are referenced by means of a local interface pointer. The question is how such pointers can be passed between different machines:



Question: Where does the proxy marshaler come from? Do we always need it?

Naming: Monikers

Observation: DCOM can handle only objects as temporary instances of a class. To accommodate objects that can outlive their client, something else is needed.

Moniker: A hack to support real objects

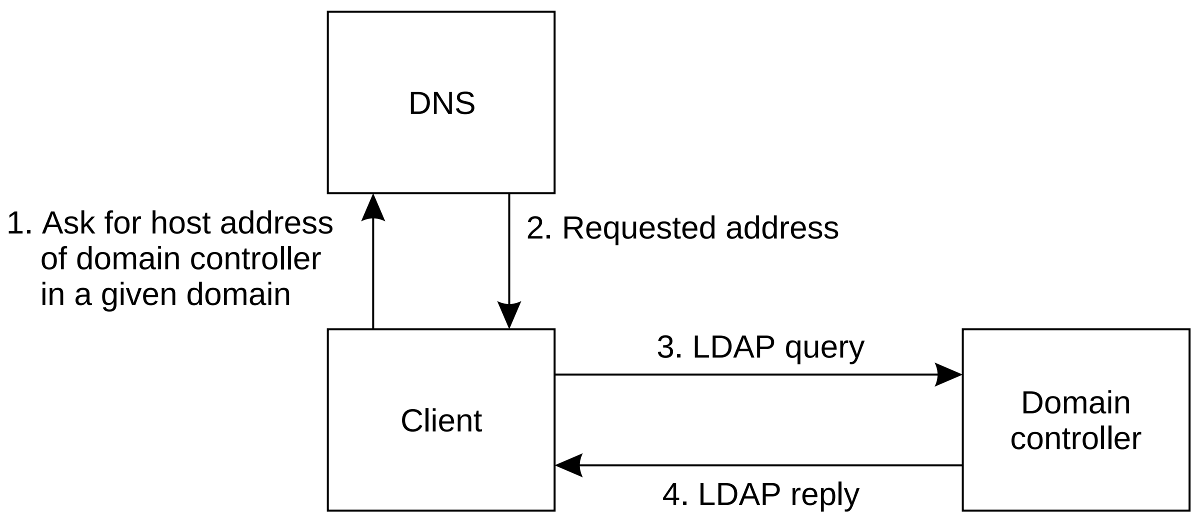
- A moniker associates data (e.g., a file), with an application or program
- Monikers can be stored
- A moniker can contain a **binding protocol**, specifying how the associated program should be "launched" with respect to the data.

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

Essence: 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:



Note: Controller is implemented as an LDAP server

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.

Security (1/2)

Declarative security: Register per object what the system should enforce with respect to authentication. Authentication is associated with users and user groups. There are different authentication levels:

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

Security (2/2)

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

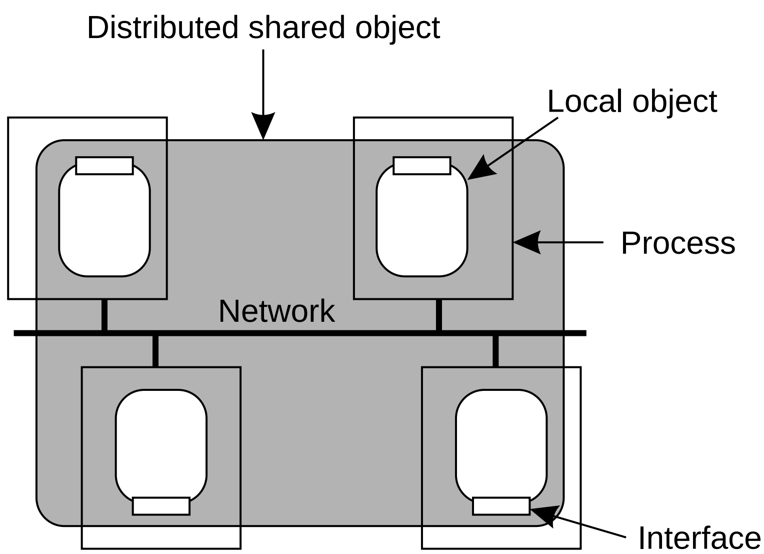
Note: There is also support for **programmatic security** by which security levels can be set by an application, as well as the required security services (see book).

Globe

- Experimental wide-area system currently being developed at Vrije Universiteit
- Unique for its focus on scalability by means of truly distributed objects
- Prototype version up and running across multiple machines distributed in NL and across Europe and the US.

Object Model (1/3)

Essence: A Globe object is a **physically distributed shared object**: the object's state may be physically distributed across several machines

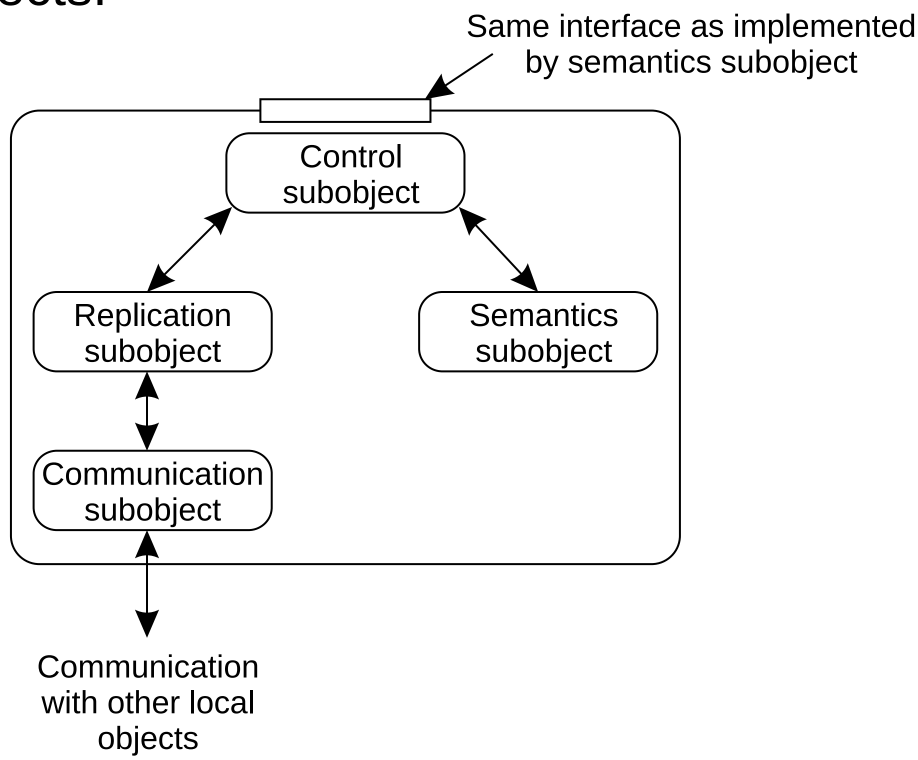


Local object: A nondistributed object residing a single address space, often representing a distributed shared object

Contact point: A point where clients can contact the distributed object; each contact point is described through a **contact address**

Object Model (2/3)

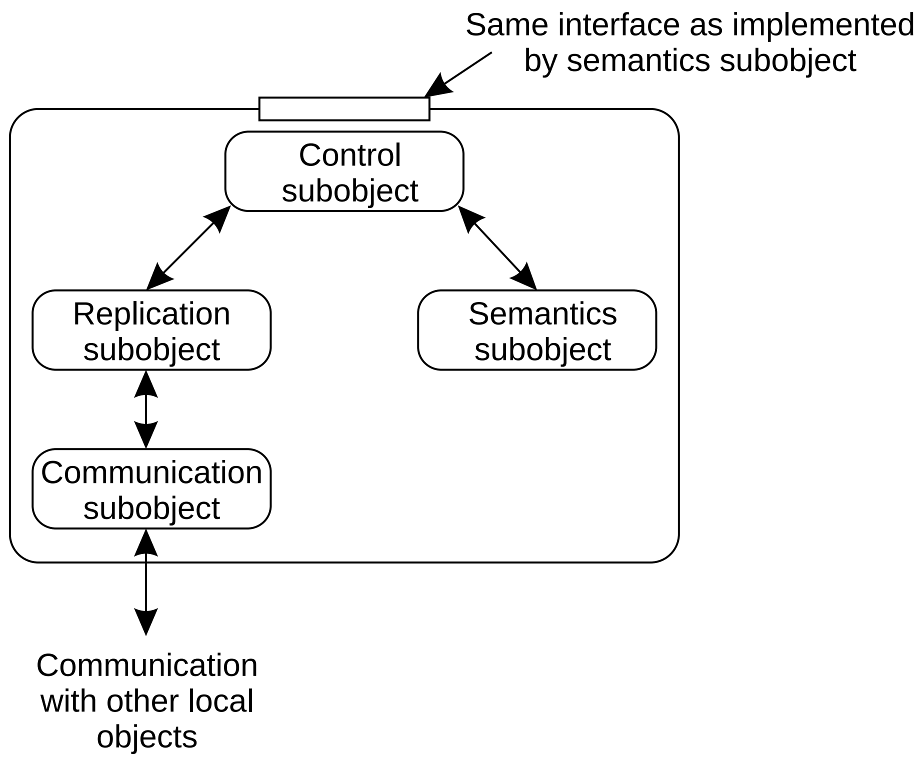
Observation: Globe attempts to separate functionality from distribution by distinguishing different local subobjects:



Semantics subobject: Contains the methods that implement the functionality of the distributed shared object

Communication subobject: Provides a (relatively simple), network-independent interface for communication between local objects

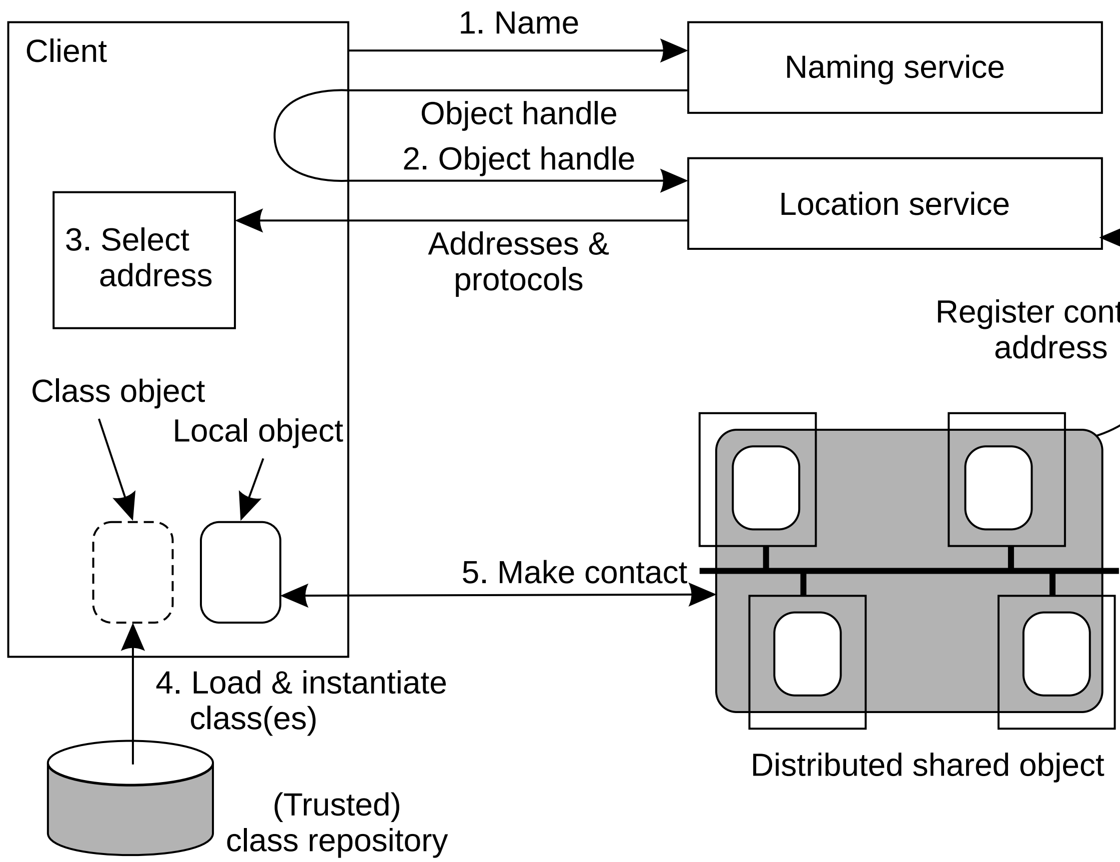
Object Model (3/3)



Replication subobject: Contains the implementation of an **object-specific** consistency protocol that controls exactly when a method on the semantics subobject may be invoked

Control subobject: Connects the user-defined interfaces of the semantics subobject to the generic, predefined interfaces of the replication subobject

Client-to-Object Binding



Observation: Globe's contact addresses correspond to CORBA's object references

Globe Services

Service	Possible implementation	Av?
Collection	Separate object that holds references to other objects	No
Concurrency	Each object implements its own concurrency control strategy	No
Transaction	Separate object representing a transaction manager	No
Event/Notif.	Separate object per group of events (as in DCOM)	No
Externalization	Each object implements its own marshaling routines	Yes
Life cycle	Separate class objects combined with per-object implementations	Yes
Licensing	Implemented by each object separately	No
Naming	Separate service, implemented by a collection of naming objects	Yes
Property	Separate service, implemented by a collection of directory objects	No
Persistence	Implemented on a per-object basis	Yes
Security	Implemented per object, combined with (local) security services	Yes
Replication	Implemented on a per-object basis	Yes
Fault tolerance	Implemented per object combined with fault-tolerant servers	Yes

Object References

Essence: Globe uses location-independent object handles which are to be resolved to **contact addresses** (which describes **where** and **how** an object can be contacted):

- Associated with a contact point of the distributed object
- Specifies (for example) a transport-level network address to which the object will listen
- Contains an **implementation handle**, specifying exactly what the client should implement if it wants to communicate through the contact point:
  - ftp://ftp.globe.org/pub/common/ip/tcp/... ...master-slave/standard/slave.jar
  - "slave/master-slave/tcp/ip"

Observation: Objects in Globe have their own object-specific implementations; there is no "standard" proxy that is implemented for all clients

Naming Objects

Observation: Globe separates naming from locating objects (as described in Chapter 04). The current naming service is based on DNS, using TXT records for storing object handles

Observation: The location service is implemented as a generic, hierarchical tree, similar to the approach explained in Chapter 04.

Caching and Replication

Observation: Here's where Globe differs from many other systems:

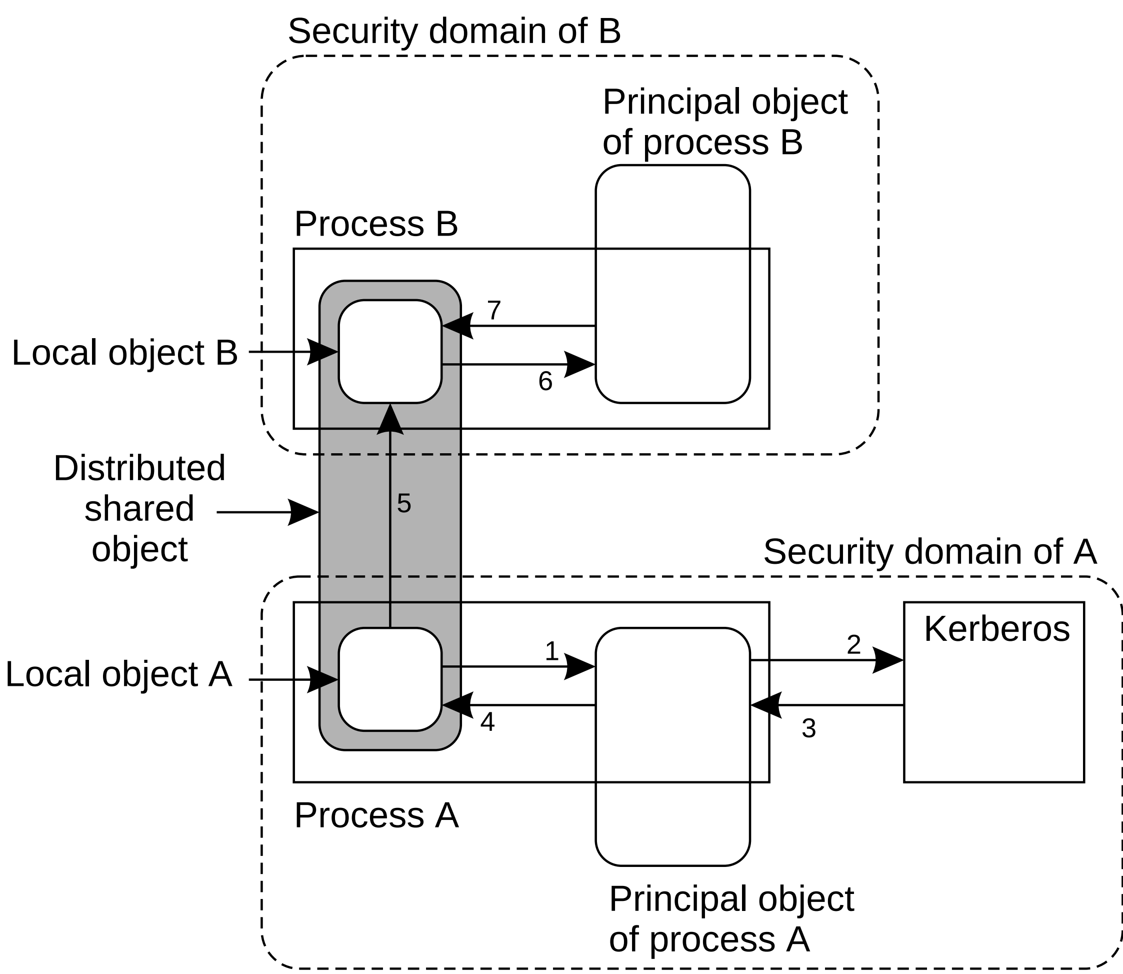
- The organization of a local object is such that replication is inherently part of each distributed shared object
- All replication subobjects have the same interface:

Method	**Description**
start	Called to synchronize replicas of the semantics subobjects, obtain locks if necessary, etc.
send	Provide marshaled arguments of a specific method, and pass invocation to local objects in other address spaces
invoked	Called after the control subobject has invoked a specific method at the semantics subobject

- This approach allows to implement any **object-specific** caching/replication strategy

Security

Essence: Additional security subobject checks for authorized communication, invocation, and parameter values. Globe can be integrated with existing security services:



Comparison

Issue	CORBA	DCOM	Globe
Design goals	Interoperability	Functionality	Scalability
Object model	Remote objects	Remote objects	Distributed objects
Services	Many of its own	From environment	Few
Interfaces	IDL based	Binary	Binary
Sync. communication	Yes	Yes	Yes
Async. communication	Yes	Yes	No
Callbacks	Yes	Yes	No
Events	Yes	Yes	No
Messaging	Yes	Yes	No
Object server	Flexible (POA)	Hard-coded	Object dependent
Directory service	Yes	Yes	No
Trading service	Yes	No	No
Naming service	Yes	Yes	Yes
Location service	No	No	Yes
Object reference	Object's location	Interface pointer	True identifier
Synchronization	Transactions	Transactions	Only intra-object
Replication support	Separate server	None	Separate subobject
Transactions	Yes	Yes	No
Fault tolerance	By replication	By transactions	By replication
Recovery support	Yes	By transactions	No