**Distributed Object based Systems:** CORBA: Overview of CORBA, Communication, Processes, Naming, Synchronization, Caching and Replication, Fault Tolerance, Security, Distributed COM: Overview of DCOM, Communication, Processes, Naming, Synchronization, Replication, Fault Tolerance, Security, GLOBE: Overview of GLOBE, Communication, Process, Naming, Synchronization, Replication, Fault Tolerance, Security, Comparison of CORBA, DCOM, and Globe: Philosophy, Communication, Processes, Naming, Synchronization, Caching and Replication, Fault Tolerance, Security.

**DISTRIBUTED OBJECT-BASED SYSTEMS**

1. CORBA. 2. DCOM. 3. Globe.

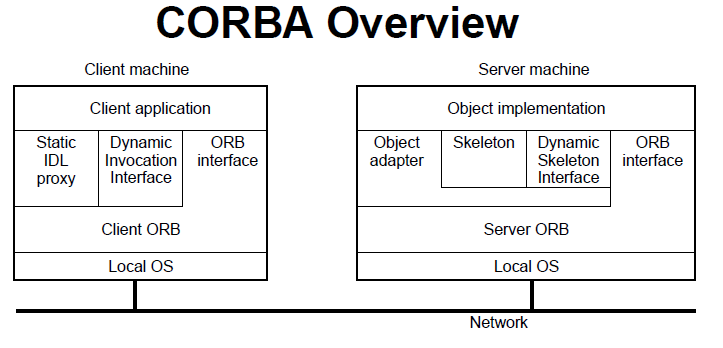
**CORBA (COMMON OBJECT REQUEST BROKER ARCHITECTURE)**

1. Developed by the Object Management Group (OMG) in response to industrial demands for object based middleware and currently in version #2.4 with #3 (almost) done

2. CORBA is a specification: different implementations of CORBA exist

3. 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.



**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.

**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:

1. Not the objects, but interfaces are the really important entities

2. An object may implement one or more interfaces \_

3. Interface descriptions can be stored in an interface repository, and looked up at runtime

4. Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, COBOL, ADA, and Java.

**CORBA SERVICES**

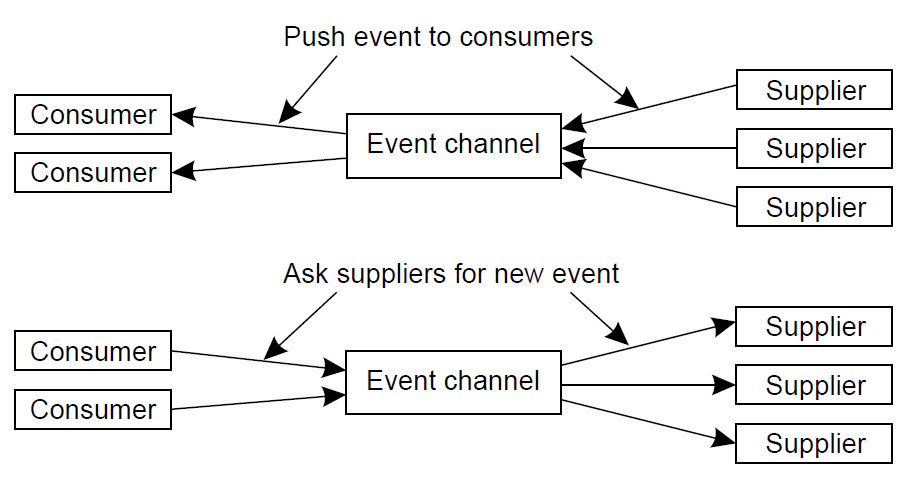
|  |  |
| --- | --- |
| **Service** | **Description** |
| Collection | Facilities for grouping objects into lists, queue, sets, etc. |
| Query | Facilities for querying collections of objects in a declarative manner |
| Concurrency | Facilities to allow concurrent access to shared objects. |
| Transaction | 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 un-marshaling 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 system wide 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**

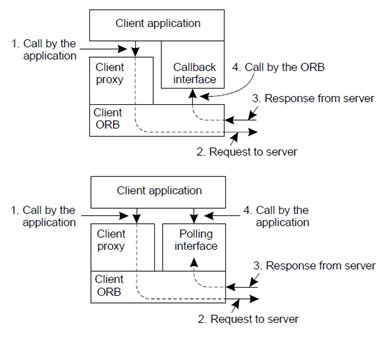
**Object invocations:** CORBA distinguishes three different forms of direct invocations:

|  |  |  |
| --- | --- | --- |
| **Request type** | **Failure sem.** | **Description** |
| Synchronous | At-most-once | Caller blocks |
| One-way | Unreliable | Non-blocking call |
| Deferred synchronous | At-most-once | Non-blocking, but can pickup results later |

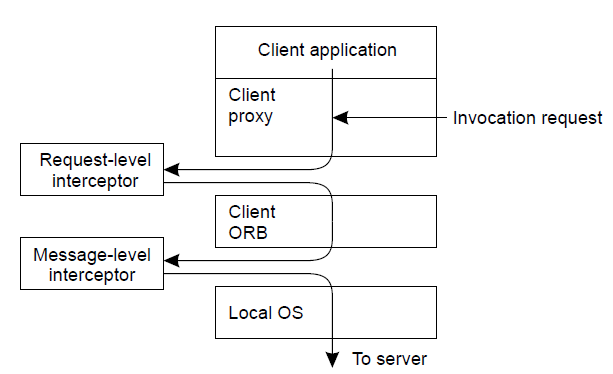
**Event communication:** There are also additional facilities by means of **event channels**:

****

**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)

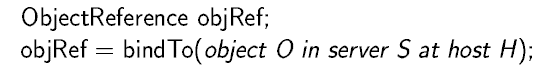
**Message-level:** Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)

**NAMING**

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

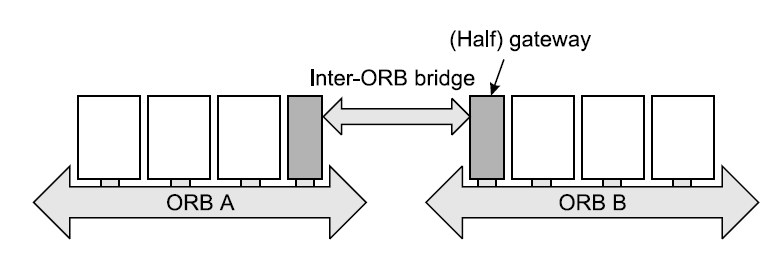
**Implementation level:** When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client address space.



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

**Interoperable Object References**

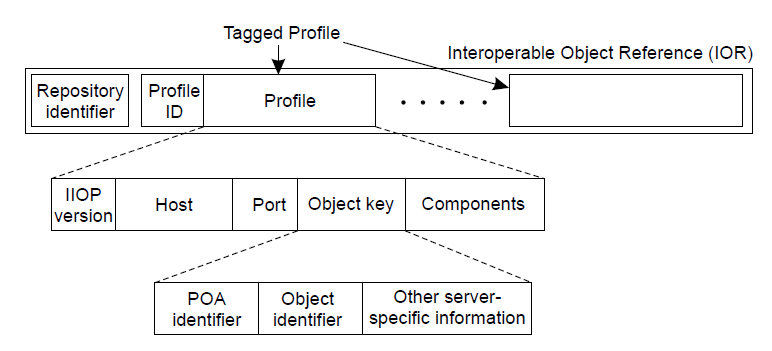
**Observation:** Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries



**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 from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn’t understand ref A.

**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

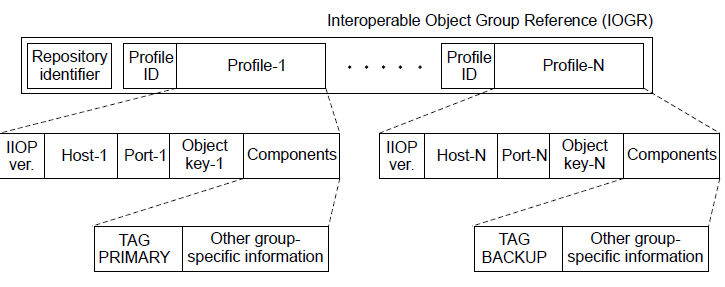
**Observation:** In the naming graph all nodes are objects; there are no restrictions to binding names to objects 􀀀 CORBA allows arbitrary naming graphs

**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🡺 **interoperable naming service**

**FAULT TOLERANCE**

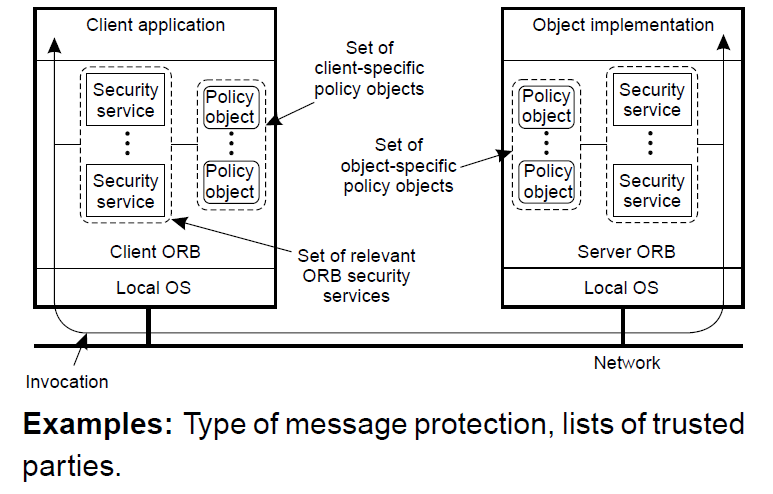
**Essence:** Mask failures through replication, by putting objects into **object groups**. Object groups are transparent to clients: they appear as “normal” objects. This approach requires a separate type of object reference: **Interoperable Object Group Reference**:



**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:



**DISTRIBUTED COM (DCOM: DISTRIBUTED COMPONENT OBJECT MODEL)**

1. Microsoft’s solution to establishing inter-process communication, possibly across machine boundaries.

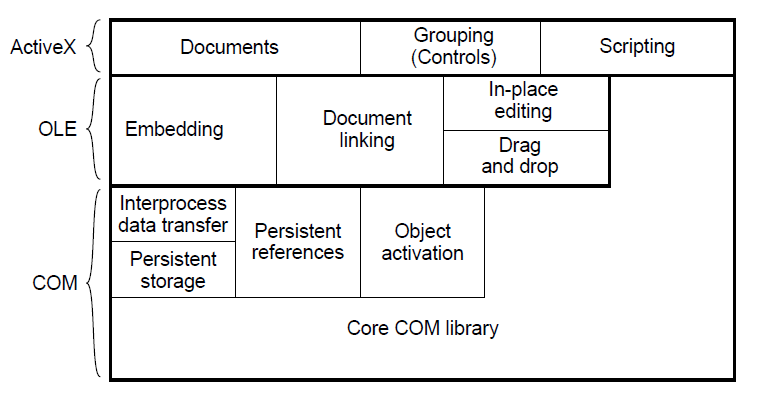
2. Supports a primitive notion of distributed objects

3. Evolved from early Windows versions to current NT-based systems (including Windows 2000)

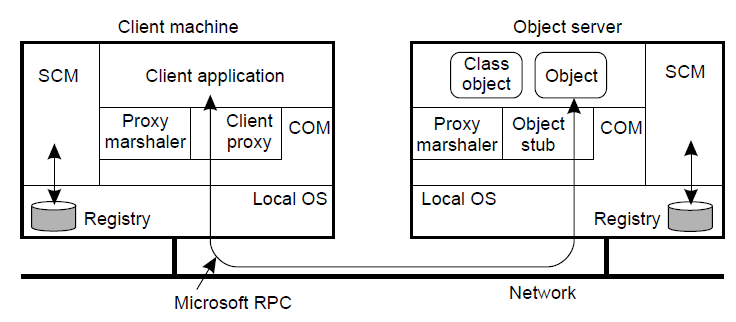
4. Comparable to CORBA’s object request broker

**DCOM OVERVIEW**

**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.



**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**

1. An interface is a collection of semantically related operations.

2. Each interface is typed, and therefore has a globally unique **interface identifier.**

3. A client always requests an implementation of an interface:

a. Locate a class that implements the interface.

b. Instantiate that class, i.e., creates an object.

c. Throw the object away when the client is done.

**DCOM SERVICES**

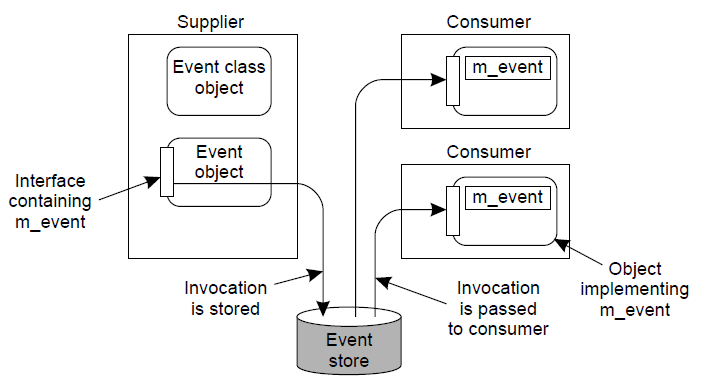
|  |  |  |
| --- | --- | --- |
| **CORBA** | **DCOM/COM+** | **Windows 2000** |
| Collection | ActiveX Data Objects | – |
| Query | None | – |
| Concurrency | Thread concurrency | – |
| Transaction | COM+ Automatic Transactions | Distributed Transaction Coordinator |
| Event | COM+ Events | – |
| Notification | 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**

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

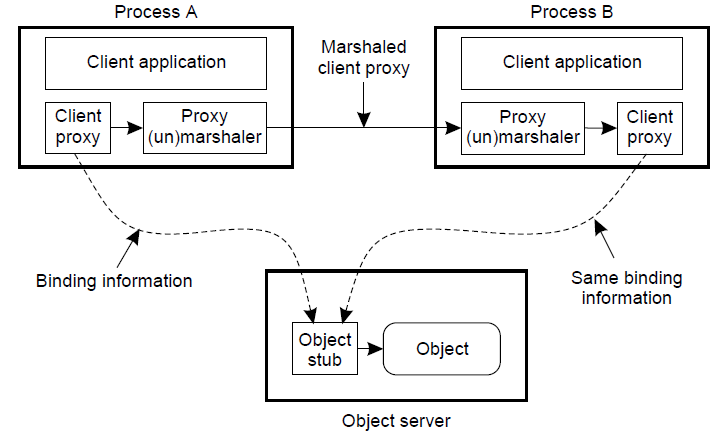
**2. Event communication:** Similar to CORBA’s push style model:



**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

1. A moniker associates data (e.g., a file), with an application or program

2. Monikers can be stored.

3. A moniker can contain a **binding protocol**, specifying how the associated program should be

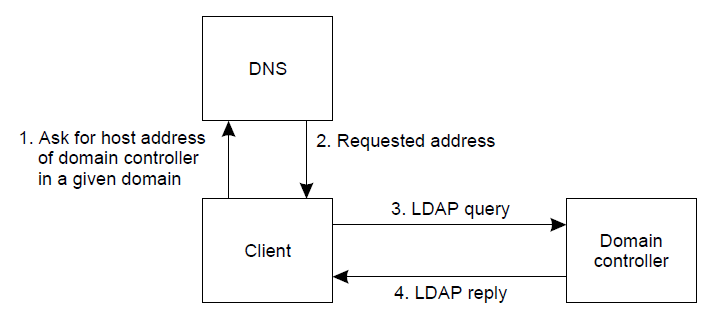
“launched” with respect to the data.

|  |  |  |
| --- | --- | --- |
| 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.

**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**

**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**

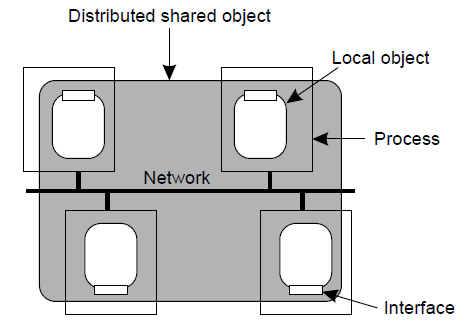
1. Experimental wide-area system currently being developed at Vrije Universiteit

2. Unique for its focus on scalability by means of truly distributed objects.

3. Prototype version up and running across multiplemachines distributed in NL and across Europe and the US.

**Object Model**

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

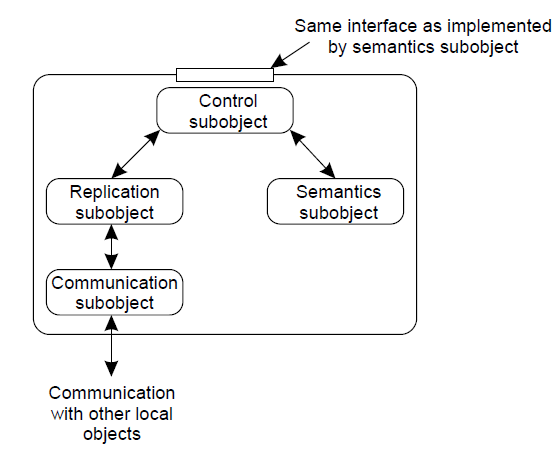


**Local object:** A non-distributed object residing a single address space, often representing a distributed shared object

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

**Object Model**

**Observation:** Globe attempts to separate functionality from distribution by distinguishing different local Sub objects:



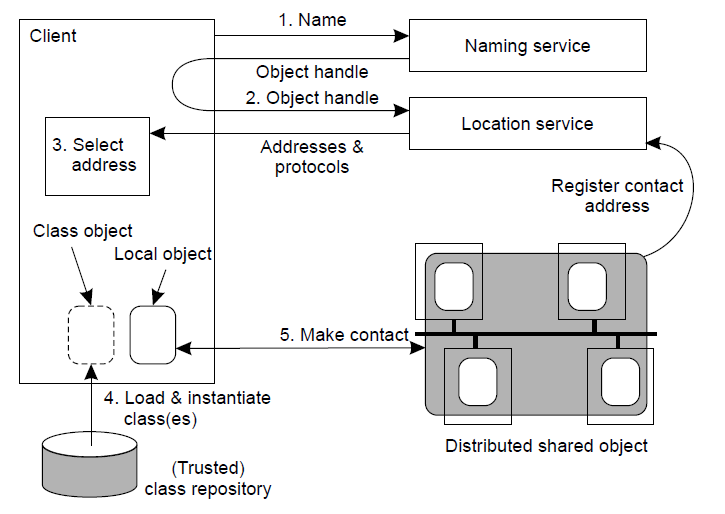
**Semantics sub-object:** Contains the methods that implement the functionality of the distributed shared object

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

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

**Control sub-object:** Connects the user-defined interfaces of the semantics sub-object to the generic, predefined interfaces of the replication sub-object

**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 |

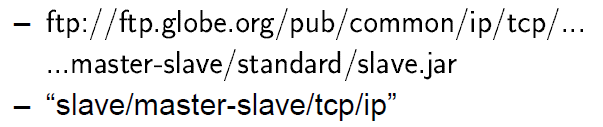
**Object References**

**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):

1. Associated with a contact point of the distributed object

2. Specifies (for example) a transport-level network address to which the object will listen

3. Contains an **implementation handle**, specifying exactly what the client should implement if it wants to communicate through the contact point:



**Observation:** Objects in Globe have their own objectspecific 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:

1. The organization of a local object is such that replication is inherently part of each distributed

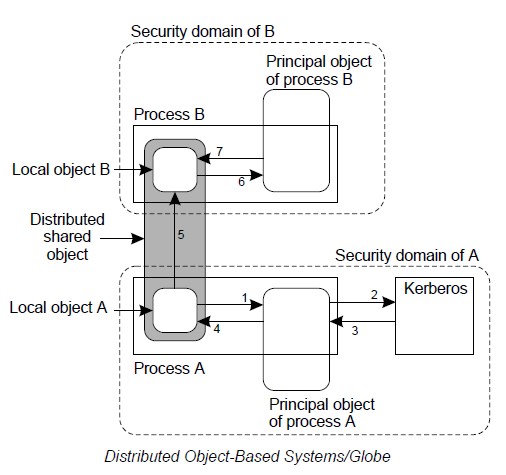
shared object

2. All replication sub-objects 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 sub-object has invoked a specific method at the semantics subobject |

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

**Security: Essence:** Additional security sub-object checks for authorized communication, invocation, and parameter values. Globe can be integrated with existing security services:



**Comparison of CORBA, DCOM, and Globe: Philosophy, Communication, Processes, Naming, Synchronization, Caching and Replication, Fault Tolerance, Security.**

